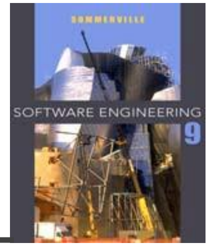
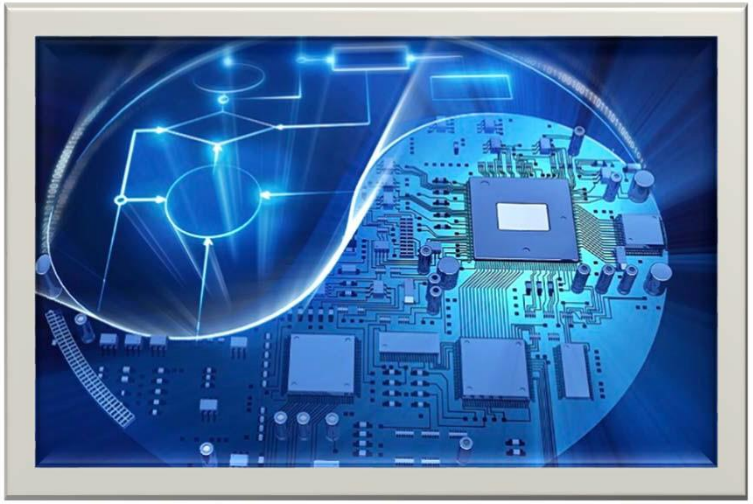
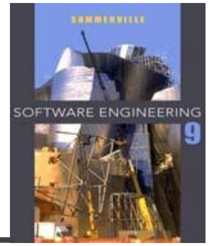


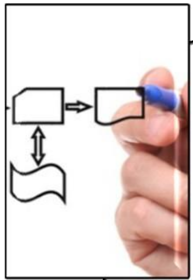
Chapter 5 – System Modeling



Topics covered



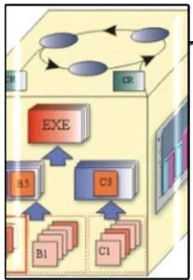
Lecture-1



Modeling with UML

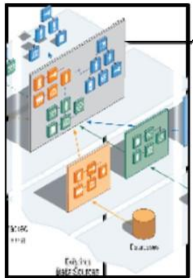
- What is Modeling?
- Use case diagram
- Class diagram
- Sequence diagram
- State diagram
- Activity diagram

Lecture-2



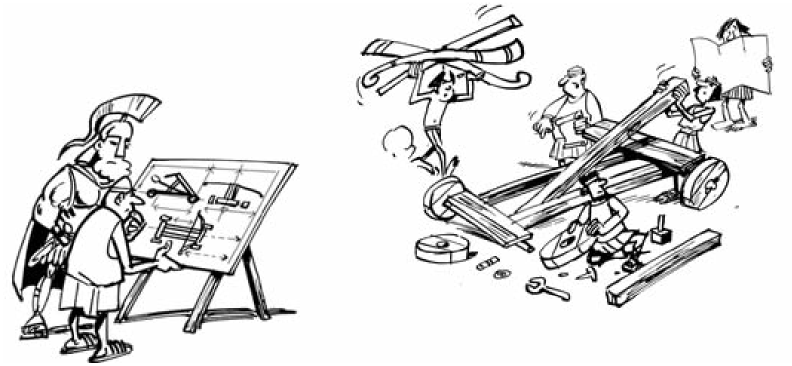
System perspectives

- Context models
- Interaction models
- Structural models
- Behavioral models



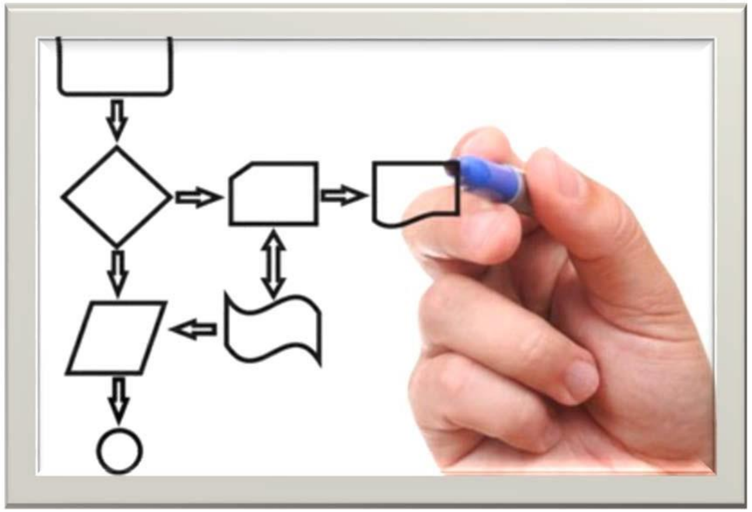
Model-driven Engineering

- Model-driven Architecture – MDA
- MDA Model Levels
- MDA Transformations
- Executable UML



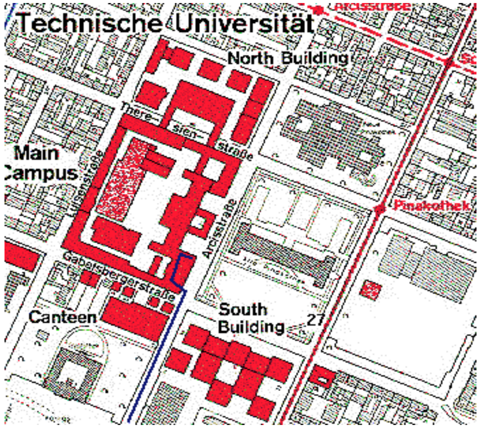
Modeling with UML

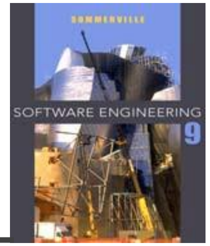
Lecture 1



What is modeling?

- Modeling consists of building an *abstraction* of reality.
- Abstractions are simplifications because:
  - They ignore irrelevant details and
  - They only represent the relevant details.
- What is *relevant* or *irrelevant* depends on the purpose of the model.
- Example: Street map



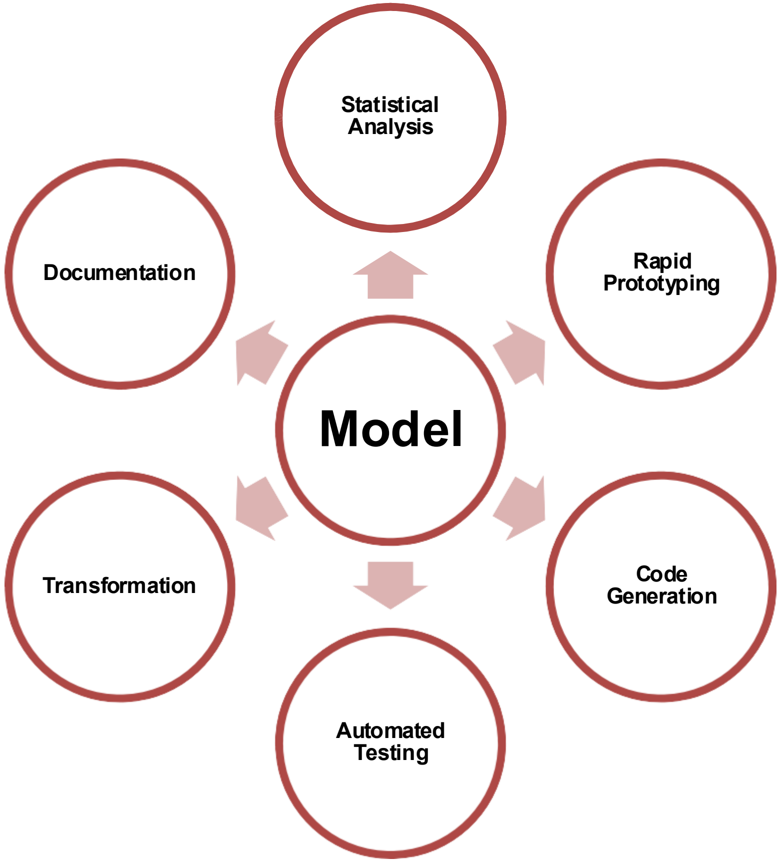


Why model software?

- Software is getting increasingly more complex
  - Windows XP > 40 million lines of code
  - A single programmer cannot manage this amount of code in its entirety.
- Code is not easily understandable by developers who did not write it
- We need simpler representations for complex systems
  - **Modeling** is a mean for dealing with complexity e.g. Flight simulator

What is Model?

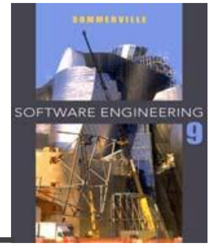
- Central artifact of software development
- Helps the analyst to understand the functionality of the system
- Represented by Unified Modeling Language – UML notations



The potential uses of UML-models

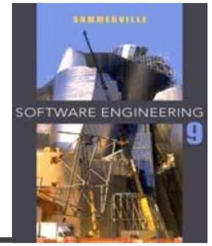
Rumpe, Bernhard. "Agile modeling with the UML." Radical Innovations of Software and Systems Engineering in the Future. Springer Berlin Heidelberg, 2004. 297-309.

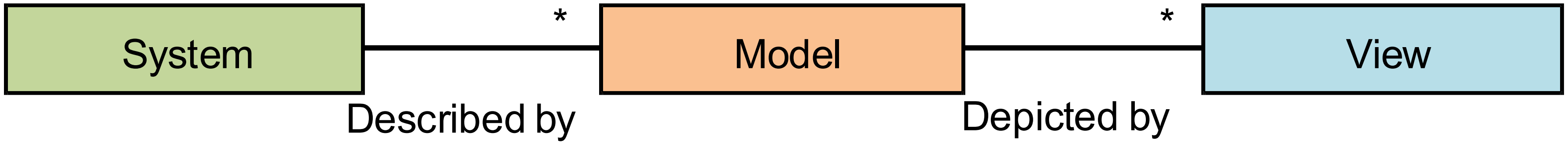
Systems, Models and Views

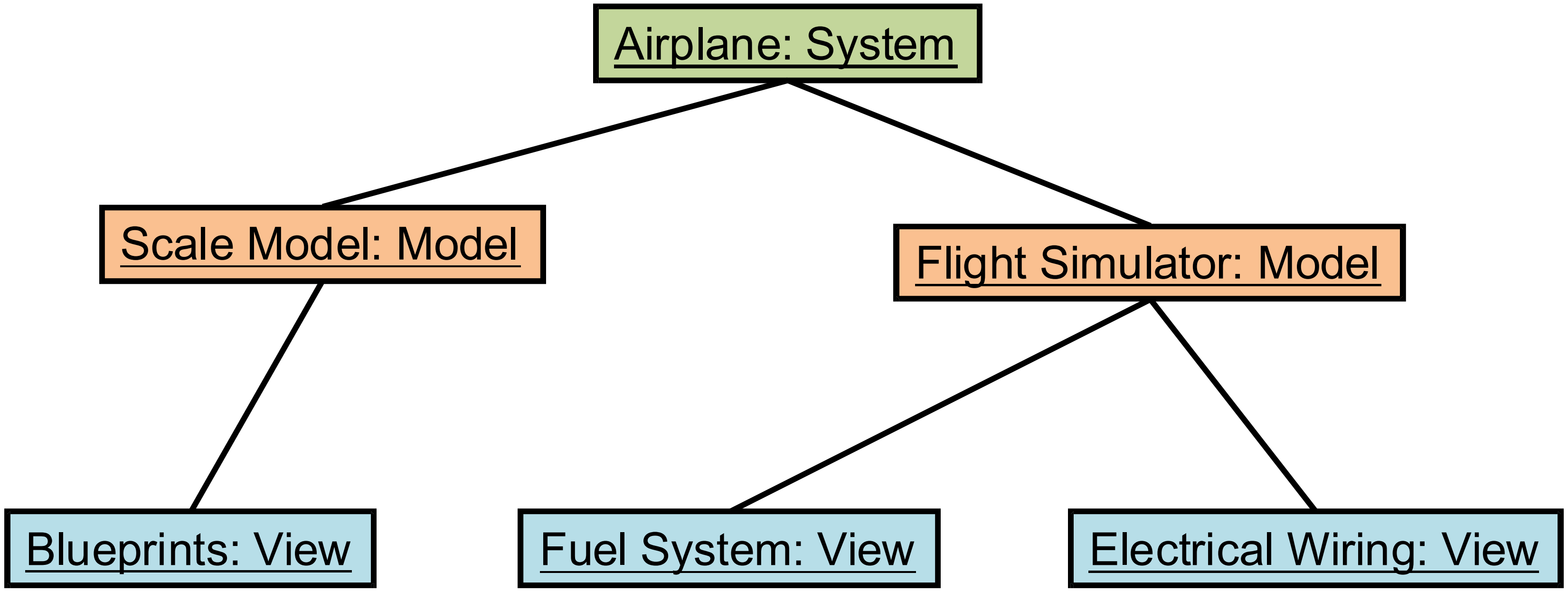


- A model is an abstraction describing a subset of a system
- A view depicts selected aspects of a model
- A notation is a set of graphical or textual rules for depicting views
- Views and models of a single system may overlap each other

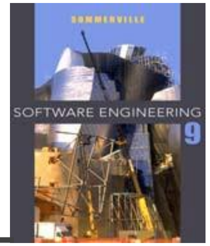
Systems, Models, and Views (UML)



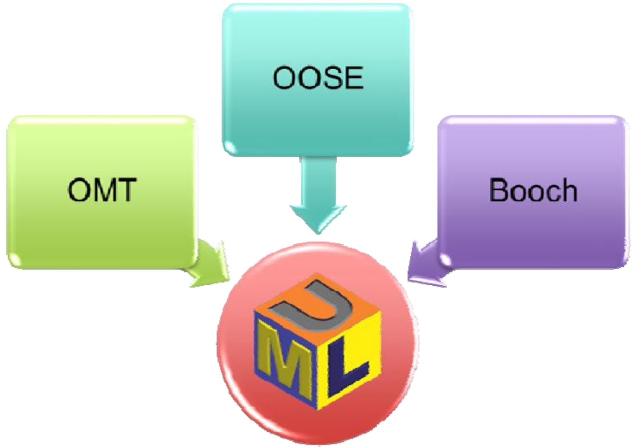




What is UML?



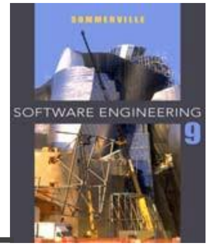
- UML (Unified Modeling Language)
  - An emerging standard for modeling object-oriented software.
  - Resulted from the convergence of notations from three leading object-oriented methods:
    - OMT (James Rumbaugh)
    - OOSE (Ivar Jacobson)
    - Booch (Grady Booch)
- Supported by several CASE tools
  - Rational ROSE
  - TogetherJ

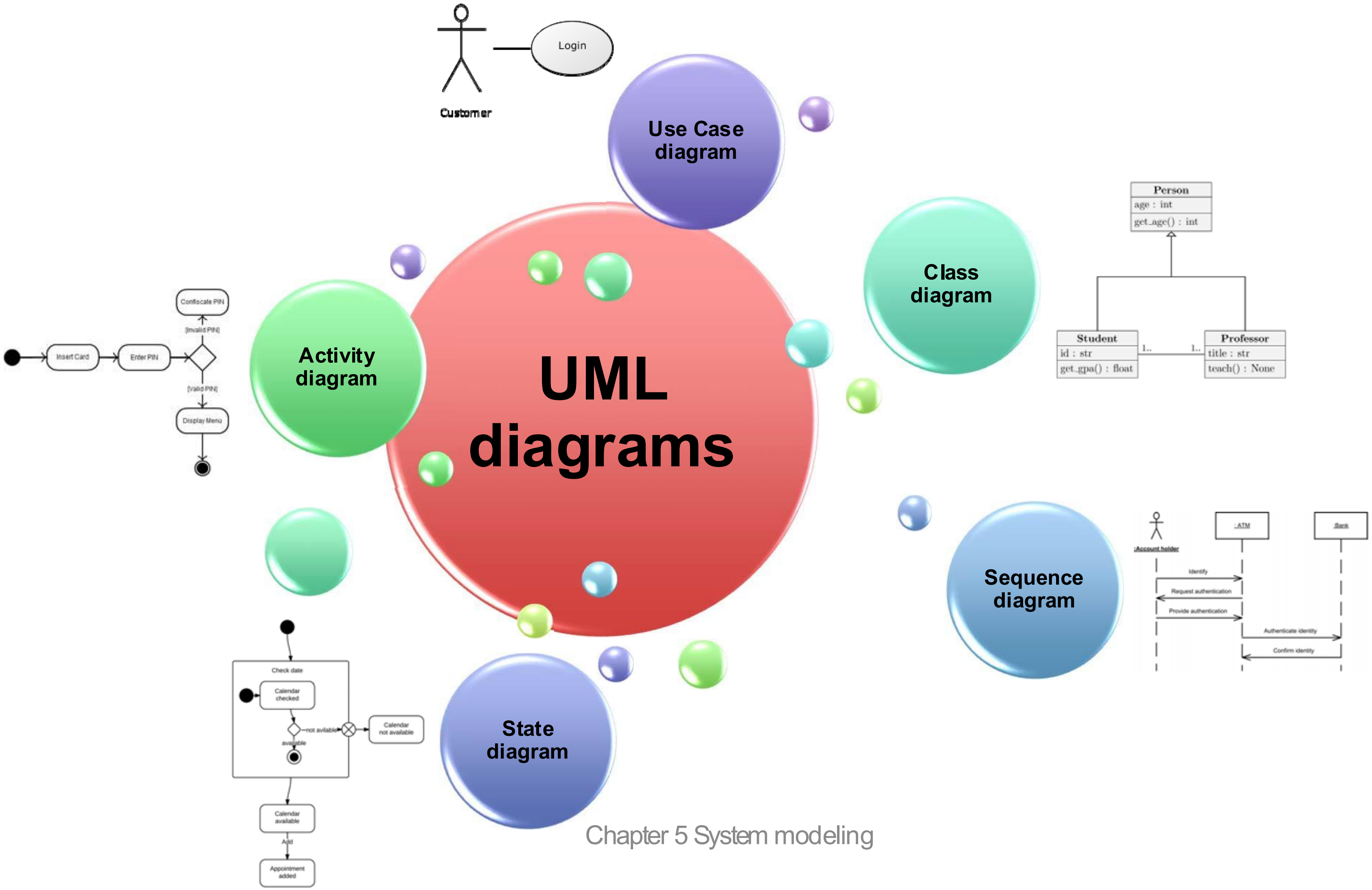


UML diagram types

- 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.
- State diagrams, which show how the system reacts to internal and external events.
- Activity diagrams, which show the activities involved in a process or in data processing .

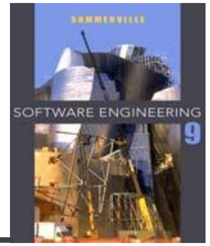
UML diagram types



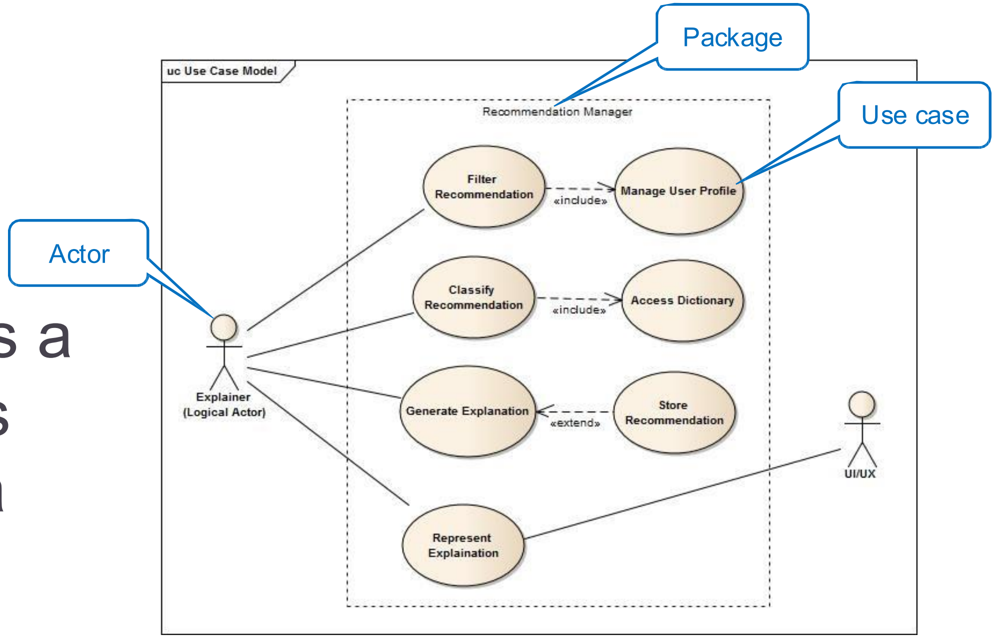


What is a Use Case?

Use Case diagrams



- Developed originally to support requirements elicitation
- Each use case represents a discrete task that involves external interaction with a system
- Actors in a use case may be people or other systems

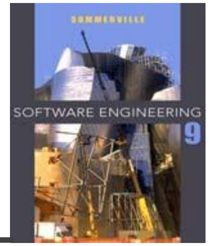


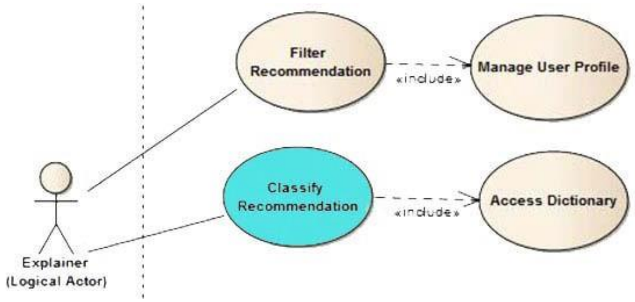
Use Case Model - Example
(Recommendation Manager)

Use Case Diagrams

- Actors
  - Represent roles i.e. a type of user of the system
- Use cases
  - Represent a sequence of interaction for a type of functionality
- Use case model
  - The set of all use cases
  - A complete description of the functionality of the system and its environment

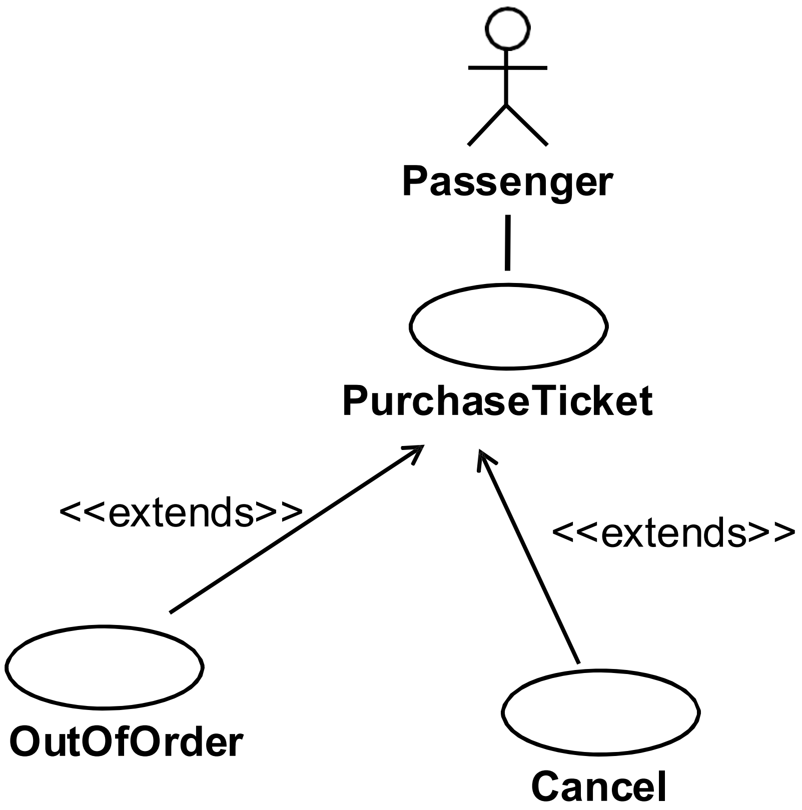
Tabular description of the '*Classify Recommendation*' use-case





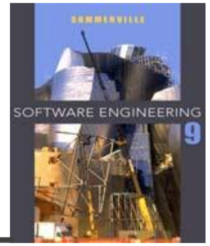
ID:	**UC-2**
Title:	Classify Recommendation
Primary Actor:	Explainer
Preconditions:	• Recommendation is filtered
Post conditions:	• Recommendation is classified
Cross Reference:	UC-1: Filter Recommendation
Main Success Scenario:	1. Explainer gets the filtered recommendation from Filter recommendation component. 2. It accesses to concept dictionary for classification 3. It classifies the recommendation into either every-day or scientific 4. It provides the classified recommendation to Explanation module.

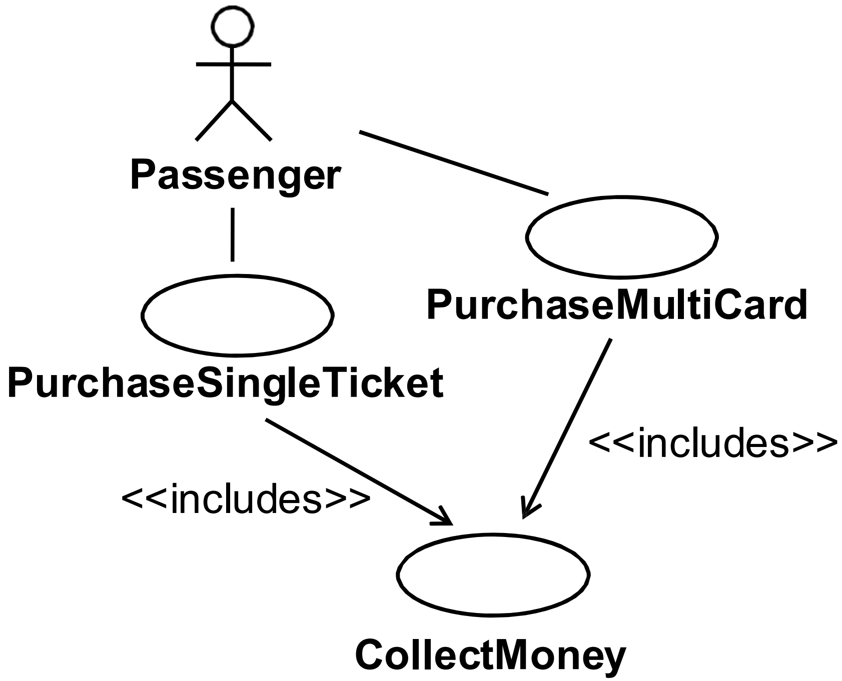
The <<extends>> Relationship



- <<extends>> relationships represent exceptional or seldom invoked cases.
- Example:
  - Cancel
  - OutofOrder

The <<includes>> Relationship



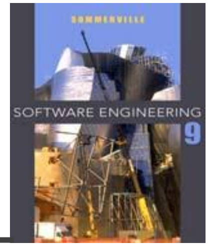


- <<includes>> relationship represents behavior that is factored out of the use case.
- <<includes>> behavior is factored out for reuse.
- Example:
  - CollectMoney

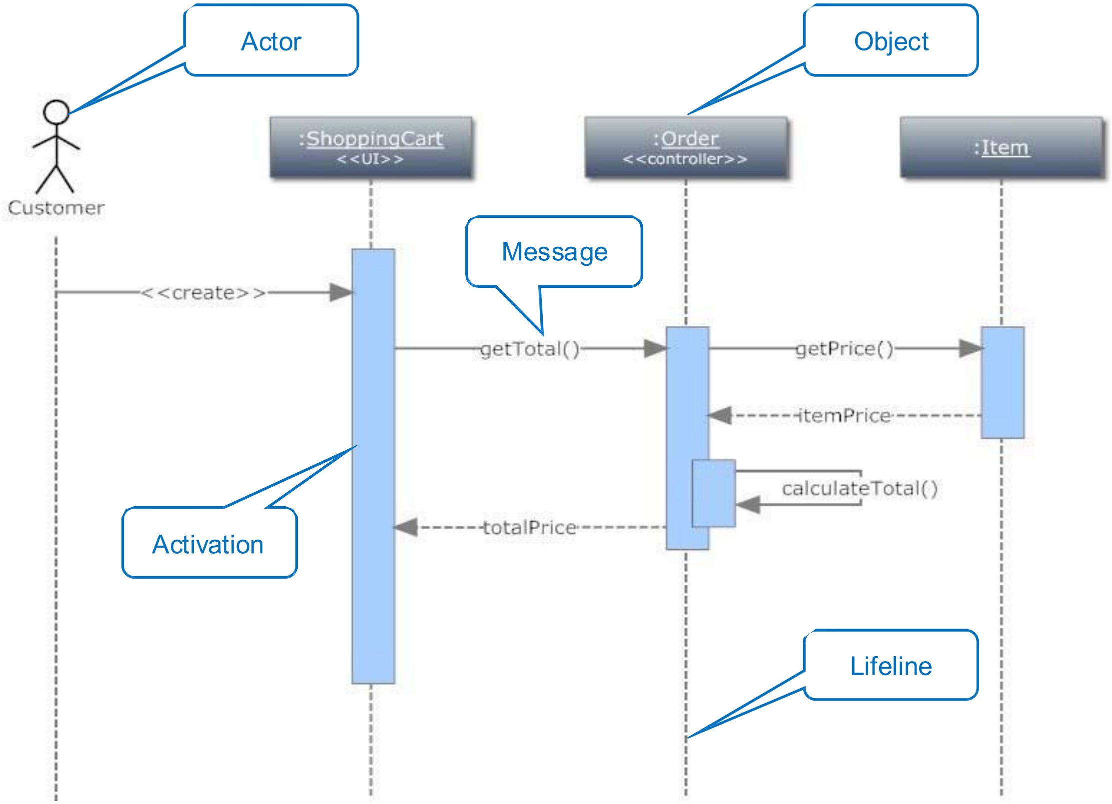
Sequence diagrams

- Shows the sequence of interactions that take place during a particular use case or use case instance
- Used to model the interactions between the actors and the objects within a system
- The objects and actors involved are listed along the top of the diagram, with a dotted line drawn vertically from these.
- Useful to find missing objects

Sequence diagram for Shopping Cart

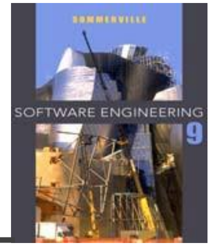


- Represent behavior in terms of interactions between system's components
- Interactions between objects are indicated by annotated arrows.



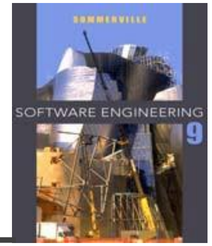
http://wc1.smartdraw.com/examples/content/examples/06_software_design/uml_diagrams/shopping_cart_sequence_diagram_l.jpg

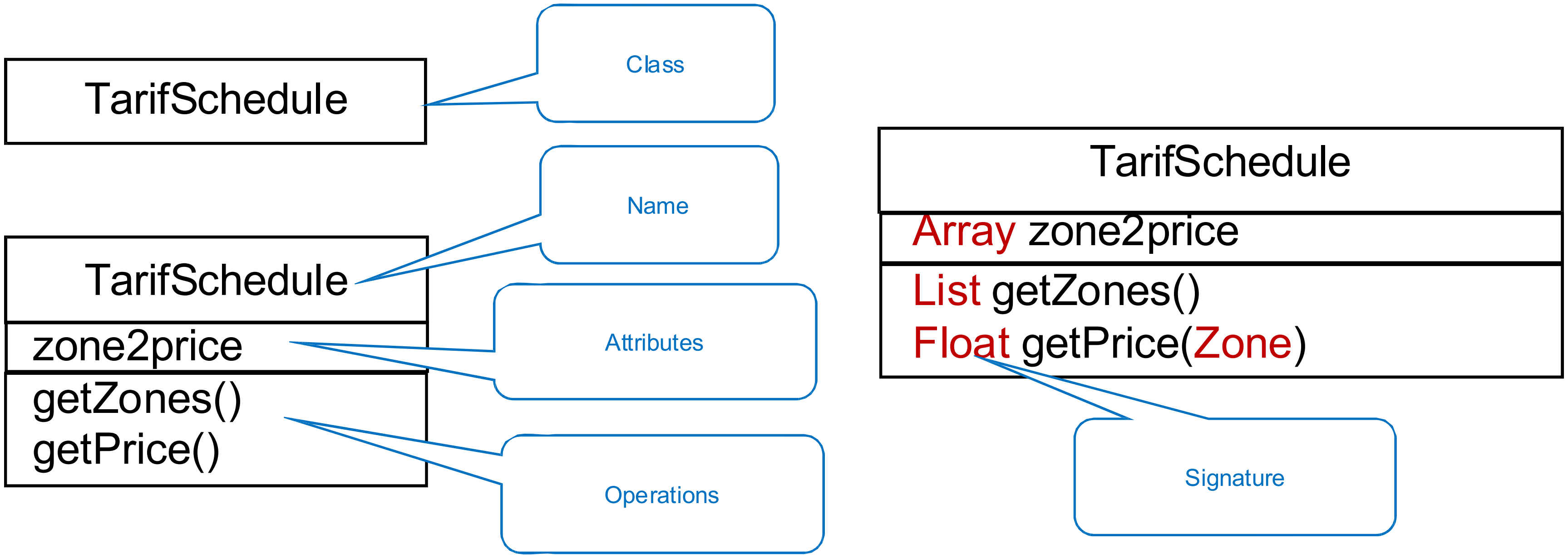
Class diagrams



- Represents the static structure of the system: objects, attributes, associations
- Shows the classes in a system and the associations between these classes
- Used during :
  - requirements analysis to model problem domain concepts
  - system design to model subsystems and interfaces
  - object-oriented design to model classes
- An object class can be thought of as a general definition of one kind of system object.
  - The objects represent something in the real world, such as a patient, a prescription, doctor, etc.

Classes

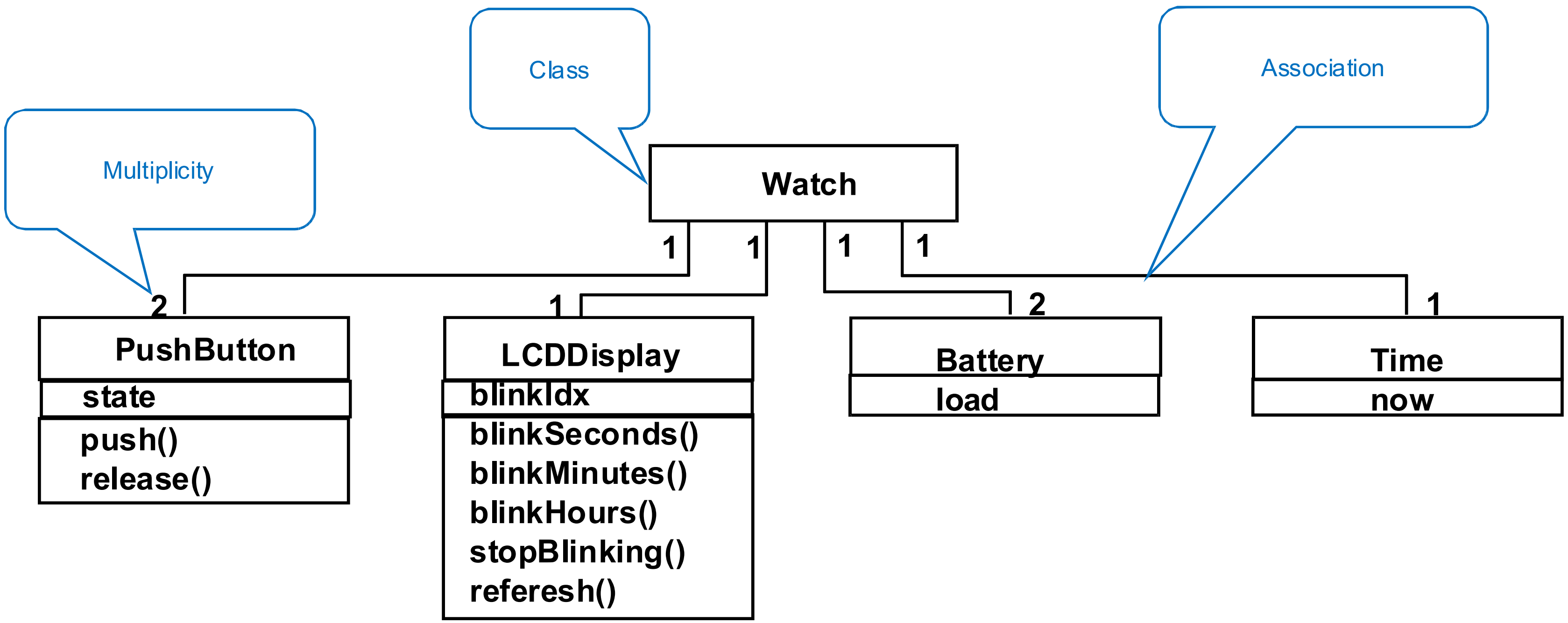




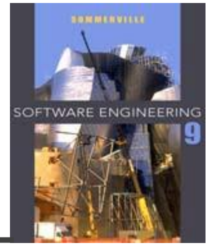
- A class represent a concept
- A class encapsulates state (attributes) and behavior (operations).
- Each attribute has a type.
- Each operation has a signature.
- The class name is the only mandatory information

Class and Associations

- Associations denote relationships between classes.
- The multiplicity of an association end denotes how many objects the source object can legitimately reference.

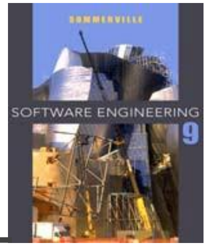


Generalization

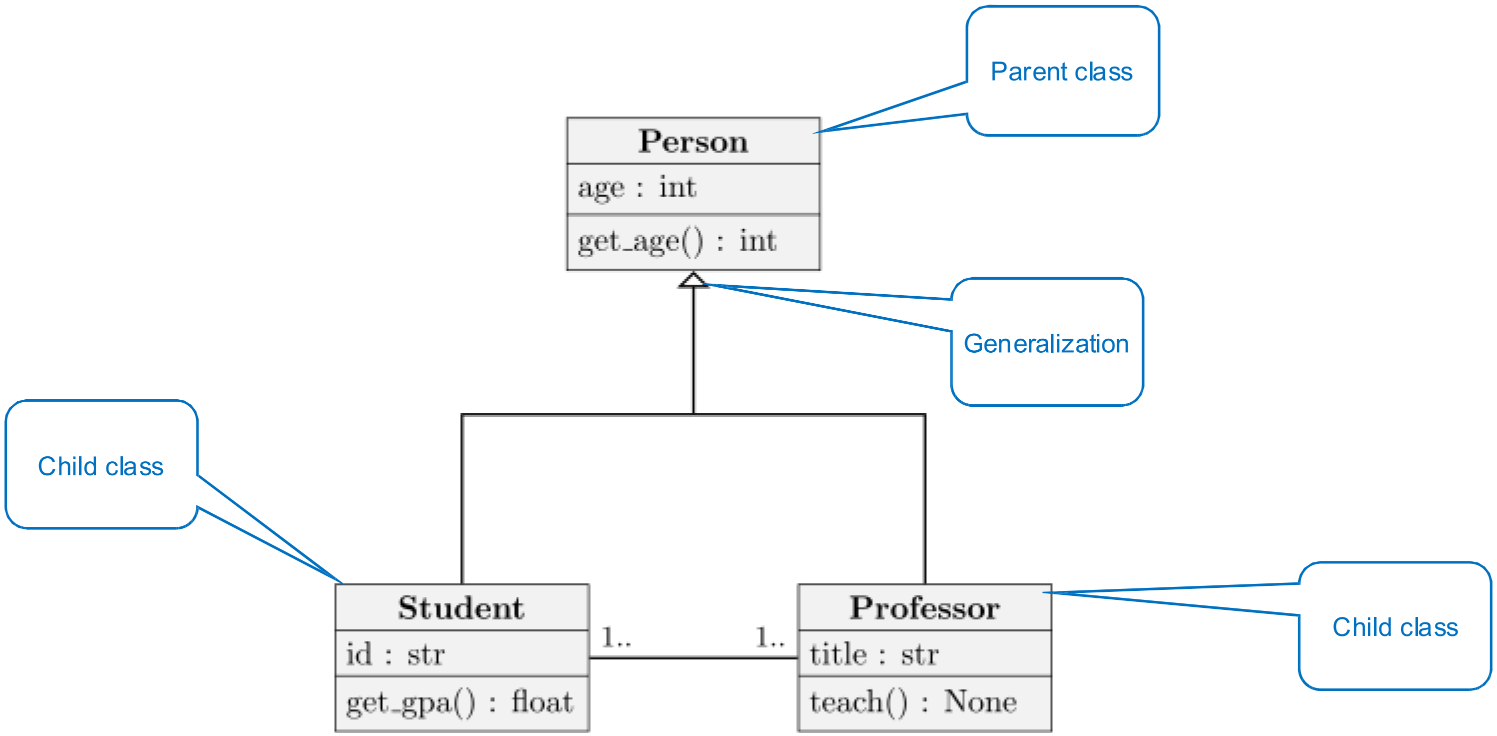


- Use to manage complexity by placing entities in more general form (animals, cars, houses, etc)
- Implemented using the class inheritance mechanisms like in Java language
- Inheritance simplifies the model by eliminating redundancy.
- The children classes inherit the attributes and operations of the parent class. These children classes then add more specific attributes and operations.
- Attributes and operations that are associated with higher-level classes are also associated with the lower-level classes

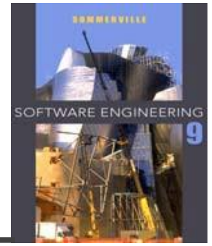
A generalization hierarchy with added detail



- Allows to infer common characteristics e.g. Student and Professor are Person

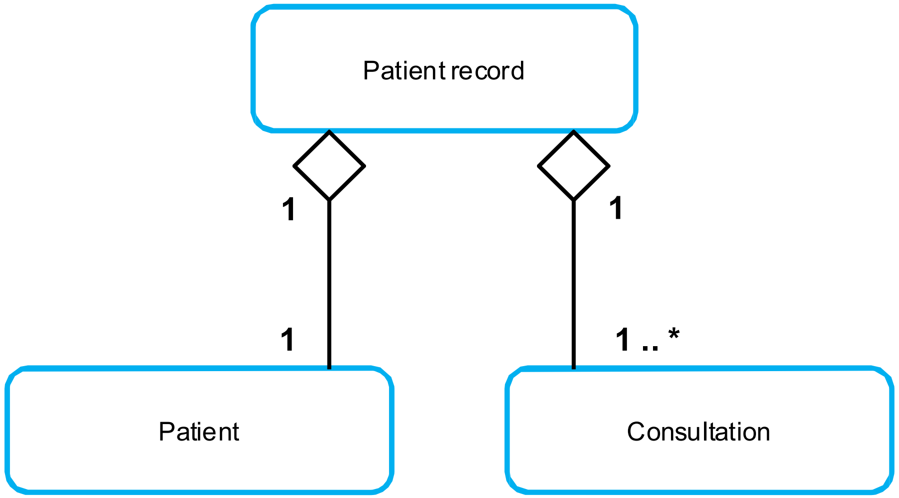


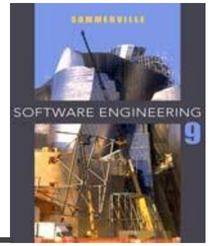
Object class aggregation models



- An aggregation model shows how classes that are collections are composed of other classes.
- Aggregation models are similar to the part-of relationship in semantic data models.

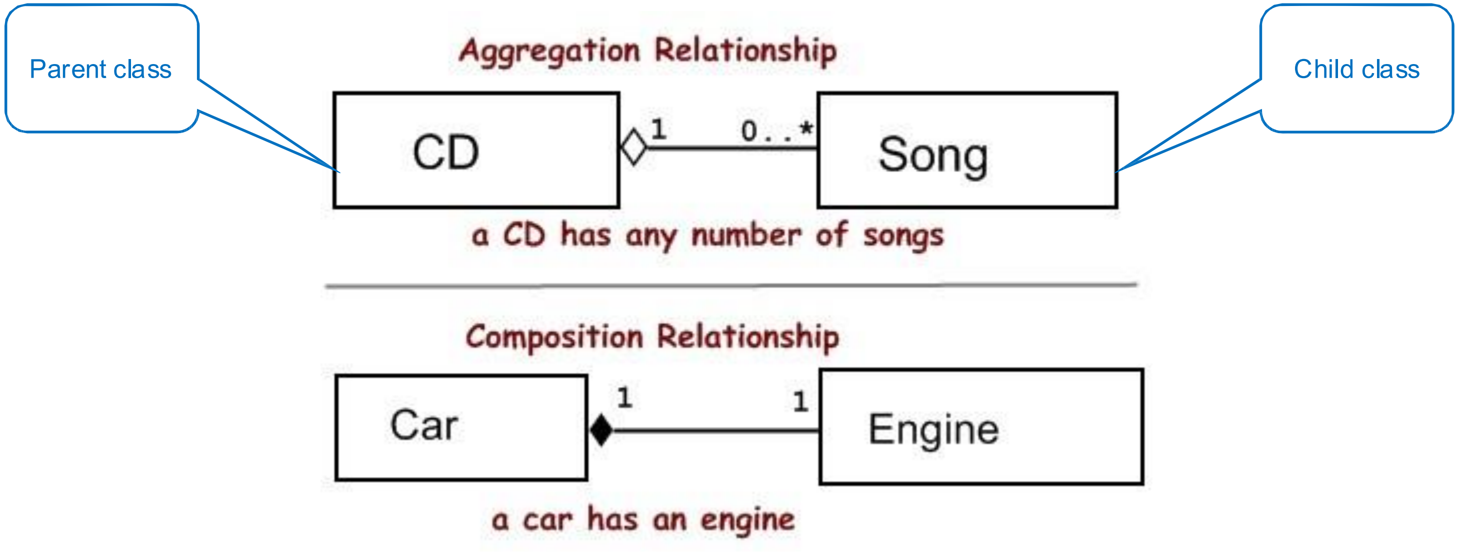
The aggregation association





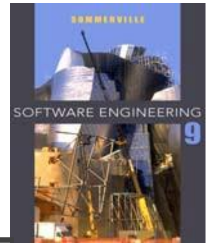
Aggregation and Composition

- An aggregation is a special case of association denoting a "consists of" hierarchy.
- A strong form of aggregation where components cannot exist without the aggregate.

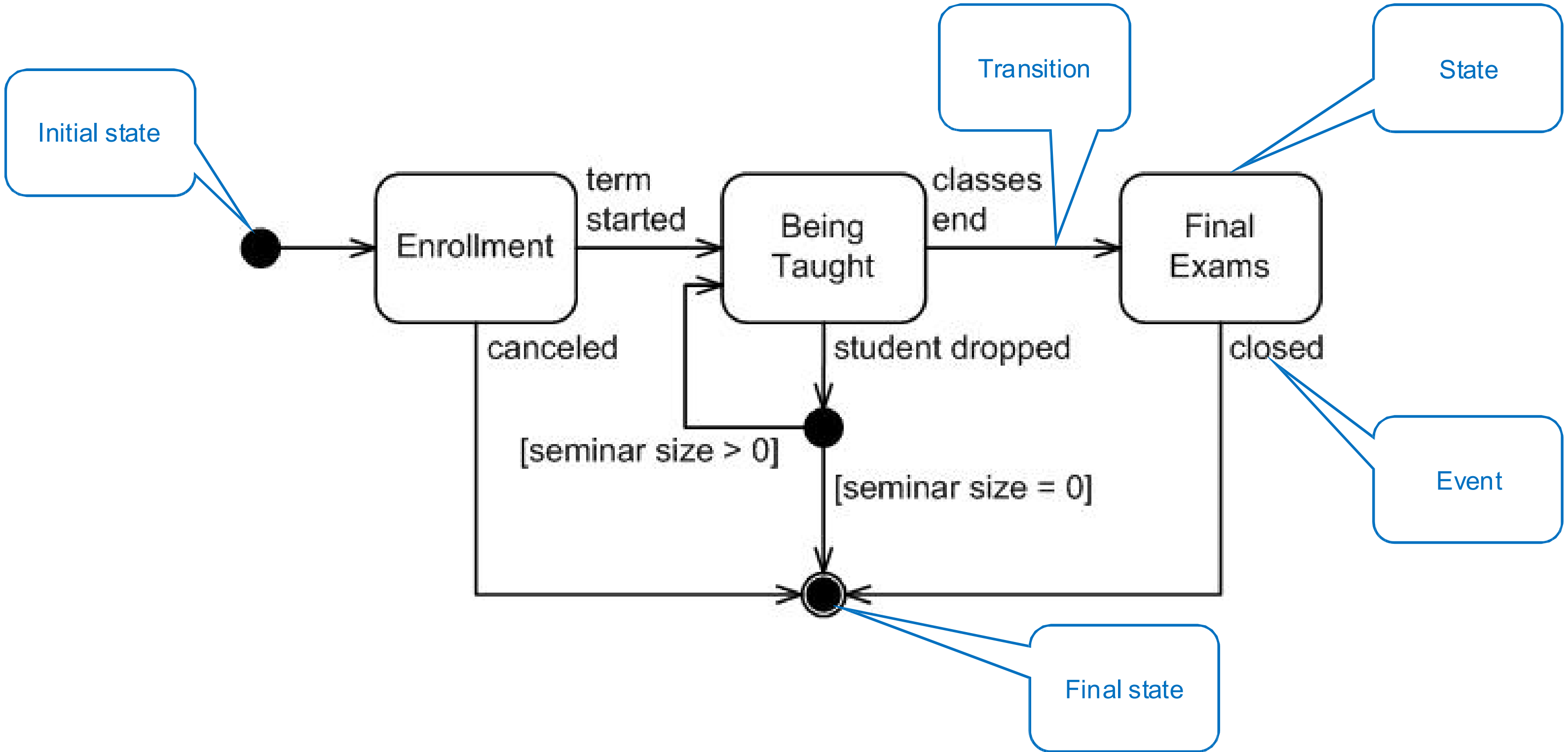


http://www-acad.sheridanc.on.ca/~jollymor/prog24178/images/compAgg.jpg

State diagrams



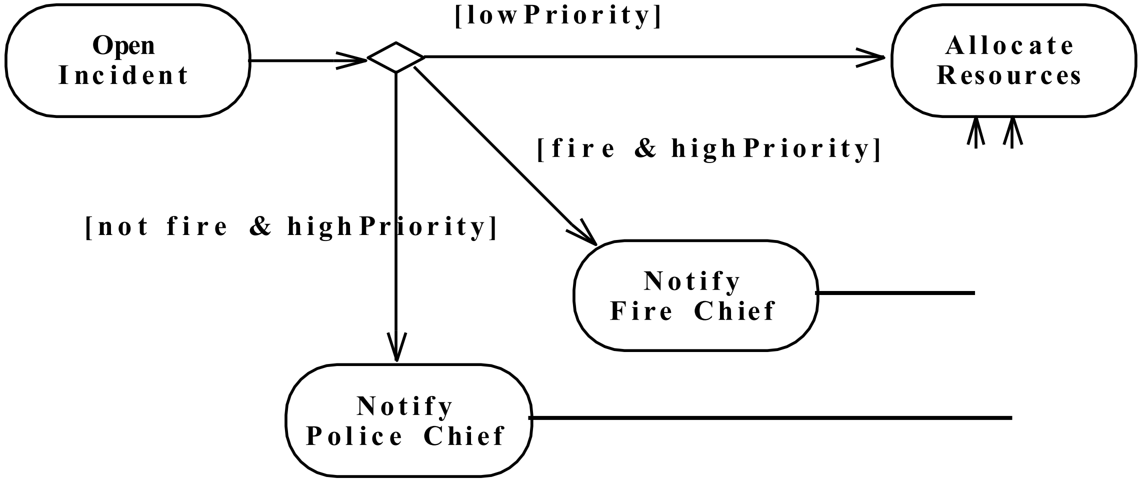
- Describe the dynamic behavior of an individual object
- Show how the system reacts to internal and external events



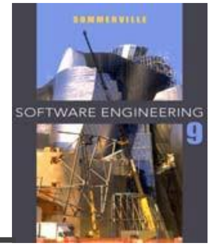
http://agilemodeling.com/images/style/
stateChartDiagramTopLevel.gif

Activity diagrams

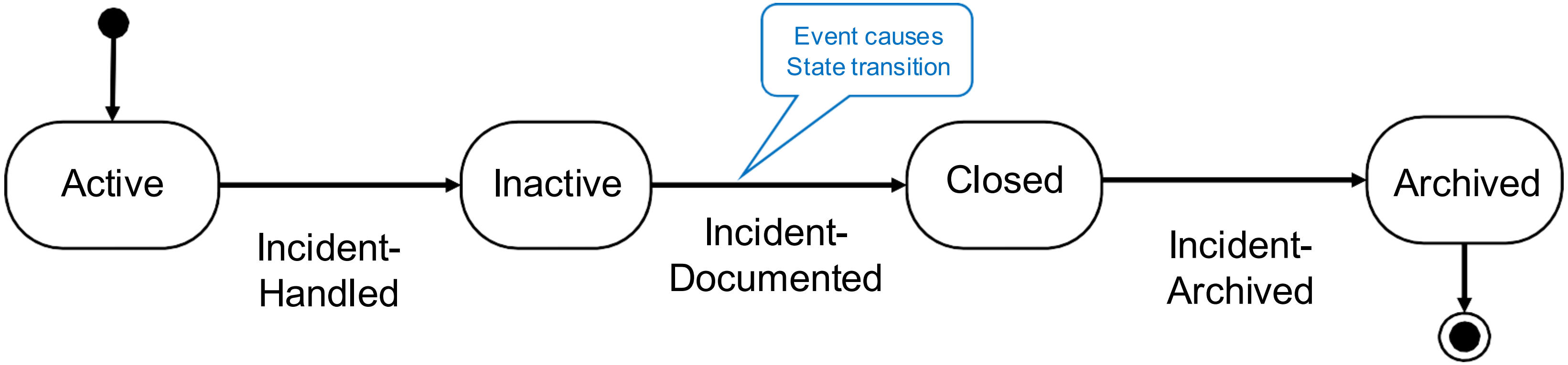
- Describes the dynamic behavior of a system
- Shows the flow control within a system or
- Represents the activities involved in a process
- Special case of a state chart diagram in which states are activities (“functions”)



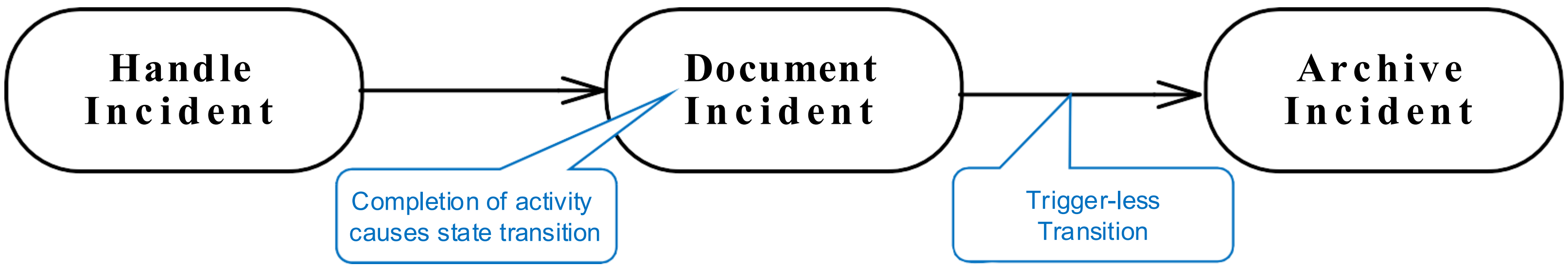
State Diagram vs. Activity Diagram



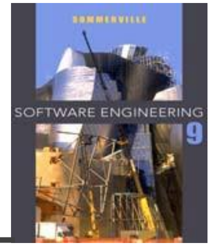
- State Diagram for Incident:
  - State: Attribute or Collection of Attributes of object of type Incident



- Activity Diagram for Incident:
  - State: Operation or Collection of Operations

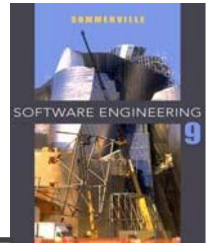


Key points



- A **model** is an abstract view of a system that ignores system details.
- **UML** is an emerging standard for modeling object-oriented software.
- **Use cases** describe interactions between a system and external actors.
- **Use case diagrams** and **Sequence diagrams** are used to describe the interactions between users and systems.
- **Class diagrams** are used to define the static structure of classes in a system and their associations.
- **State diagrams** are used to model a system’s behavior in response to internal or external events.
- **Activity diagrams** may be used to model the processing of data.

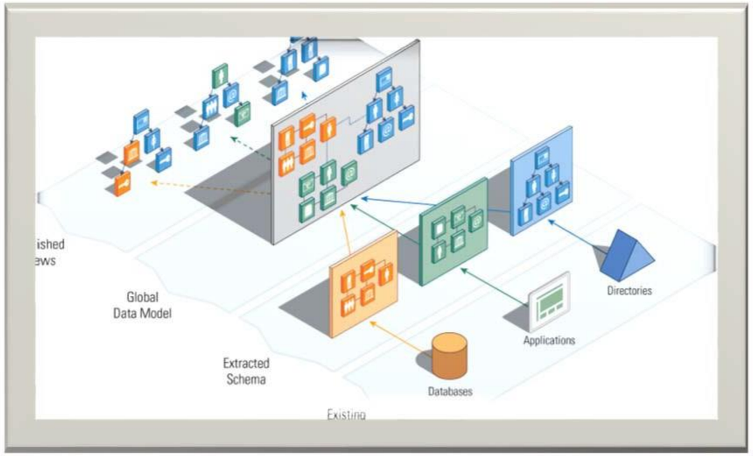
Further Readings

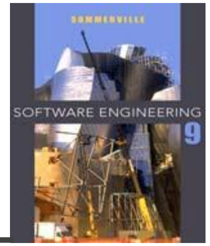


- UML Tutorial - Use Case, Activity, and Sequence Diagrams - Essential Software Modeling
  - http://www.youtube.com/watch?v=RMuMz5hQMf4

System perspectives and Model-driven Engineering

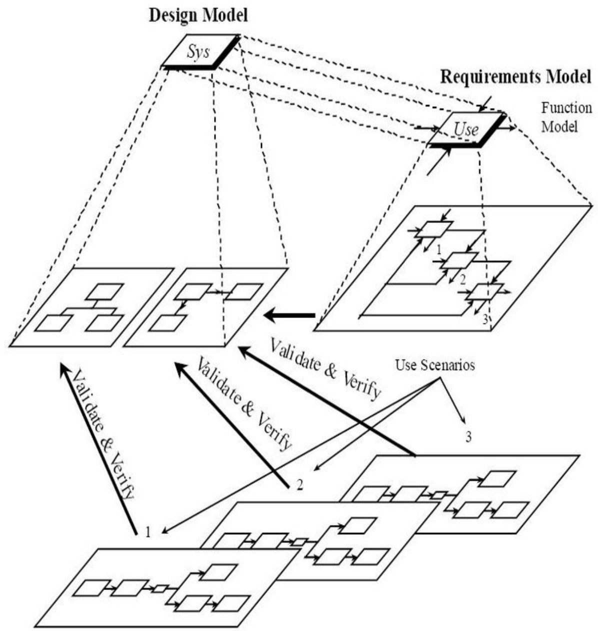
Lecture 2



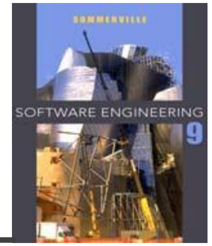


What is System modeling?

- System modeling is the process of developing abstract models of a system
- Each model presenting a different view or perspective of that system
- System modeling helps the analyst to understand the functionality of the system and models are used to communicate with customers

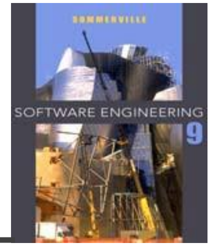


Existing and planned system models



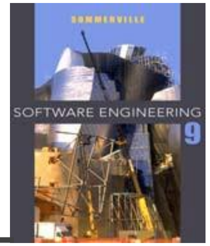
- Models of the existing system are used during requirements engineering.
  - They help clarify what the existing system does and can be used as a basis for discussing its strengths and weaknesses. These then lead to requirements for the new system.
- Models of the new system are used during requirements engineering to explain the proposed requirements to other system stakeholders.
  - Engineers use these models to discuss design proposals and to document the system for implementation.
- In a model-driven engineering process, it is possible to generate a complete or partial system implementation from the system model.

Use of graphical models



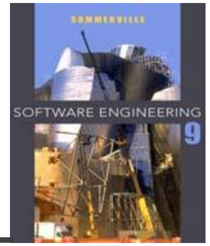
- As a means of facilitating discussion about an existing or proposed system
  - Incomplete and incorrect models are OK as their role is to support discussion.
- As a way of documenting an existing system
  - Models should be an accurate representation of the system but need not be complete.
- As a detailed system description that can be used to generate a system implementation
  - Models have to be both correct and complete.

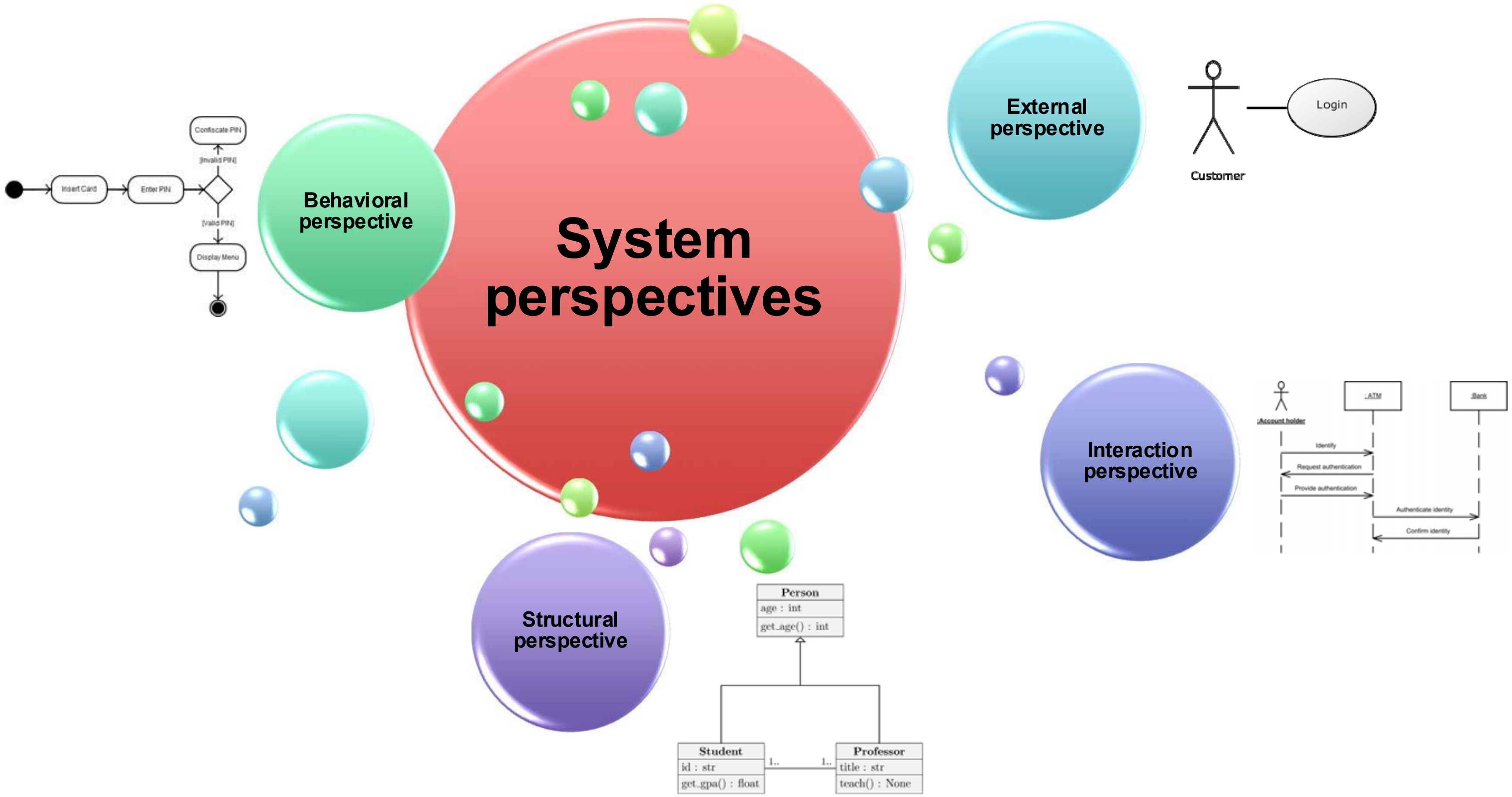
System perspectives



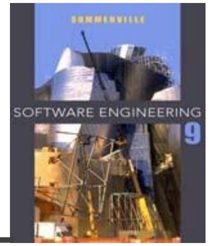
External perspective	- Model the context or environment of the system	- Context models
Interaction perspective	- Model the interactions between a system and its environment, or between the components of a system	- Interaction models
Structural perspective	- Model the organization of a system or the structure of the data that is processed by the system	- Structural models
Behavioral perspective	- Model the dynamic behavior of the system and how it responds to events	- Behavioral models, State machine models

System perspectives





Context models



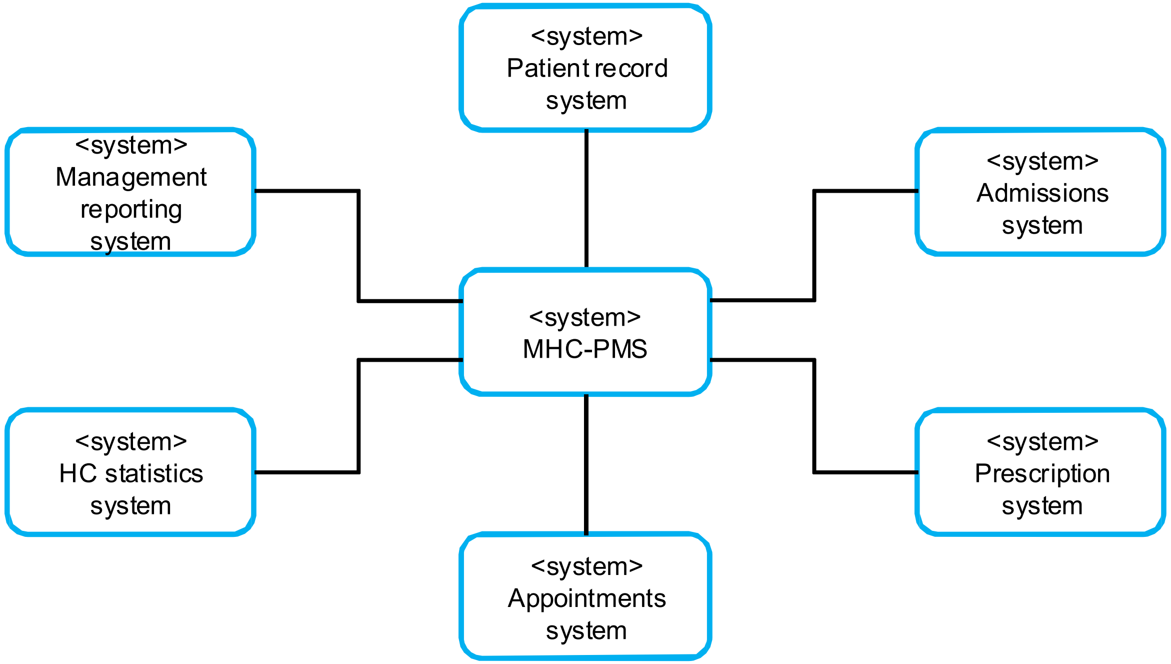
- Used to illustrate the operational context of a system
  - They show what lies outside the system boundaries
- Defines the physical scope of the system:
  - what is part of the system (under your control)
  - what is external to the system
- Social and organisational concerns may affect the decision on where to position system boundaries
- Architectural models show the system and its relationship with other systems

http://www.aconventional.com/2010/05/towards-working-theory-of-learning.html

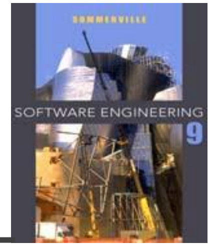
System boundaries

- System boundaries are established to define what is inside and what is outside the system.
  - They show other systems that are used or depend on the system being developed.
- The position of the system boundary has a profound effect on the system requirements.
- Defining a system boundary is a political judgment
  - There may be pressures to develop system boundaries that increase / decrease the influence or workload of different parts of an organization.

The context of the MHC-PMS

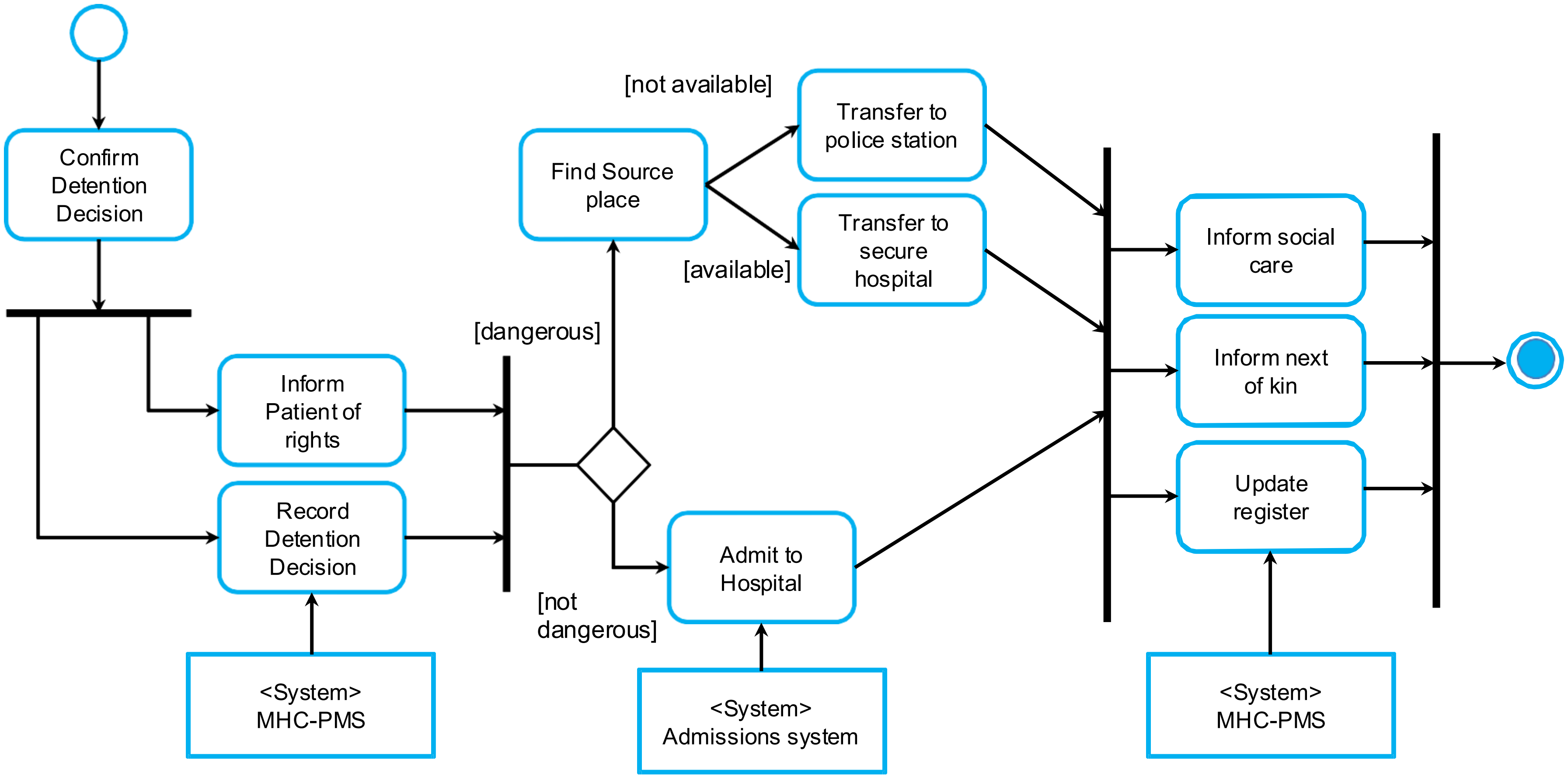


Process perspective

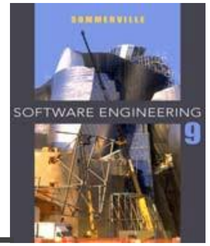


- Context models simply show the other systems in the environment, not how the system being developed is used in that environment.
- Process models reveal how the system being developed is used in broader business processes.
- UML activity diagrams may be used to define business process models.

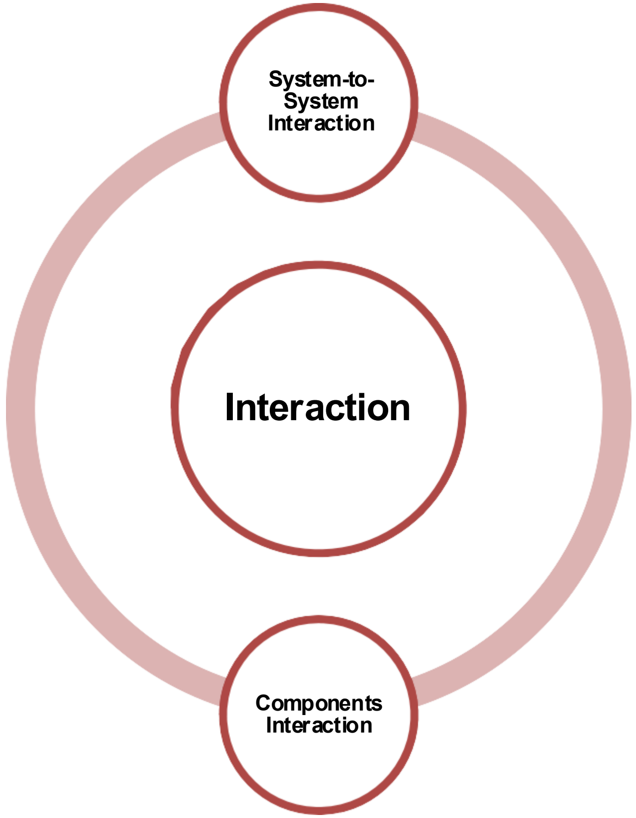
Process model of involuntary detention



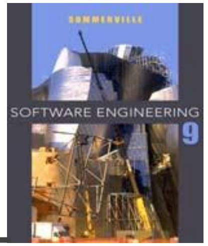
Interaction models



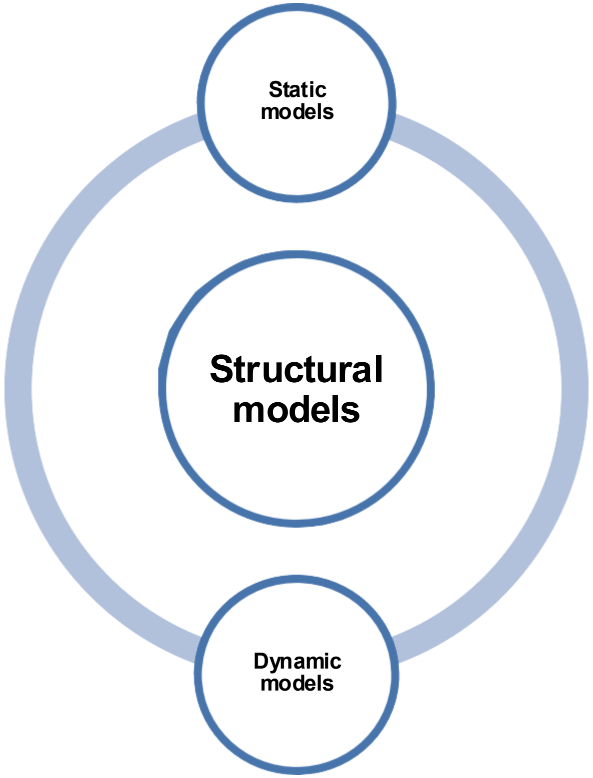
- Modeling user interaction is important as it helps to identify user requirements.
- Modeling system-to-system interaction highlights the communication problems that may arise.
- Modeling component interaction helps us understand if a proposed system structure is likely to deliver the required system performance and dependability.
- Use case diagrams and sequence diagrams may be used for interaction modeling.



Structural models



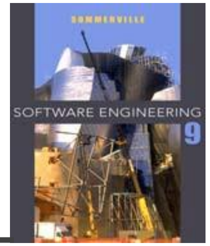
- Display the organization of a system in terms of the components that make up that system and their relationships
- Structural models are created when system architecture is discussed and designed.
- Models may be static or dynamic model
- Static Model
  - Show the structure of the system design
- Dynamic Model
  - Show the organization of the system when it is executing



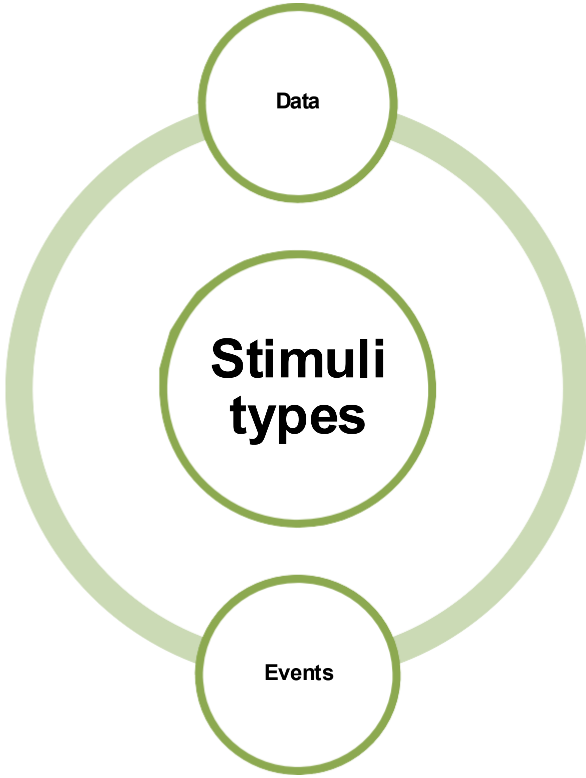
Behavioral models

- Represents the dynamic behavior (execution) of a system
  - which is specified declaratively using the object constraint language, or expressed using UML's action language.
- Shows what happens or what is supposed to happen when a system responds to a stimulus from its environment.
- Allows the analyst to capture when and how the system functionality is available.

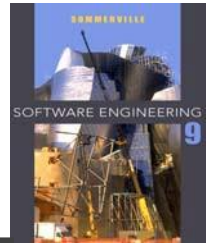
Behavioral models



- Stimuli can be think of two types
- Data
  - Some data arrives that is to be processed by the system
- Events
  - Some event happens that triggers system processing
  - Events may have associated data, although this is not always the case

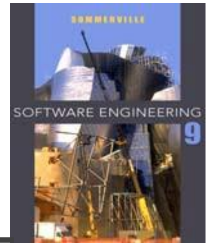


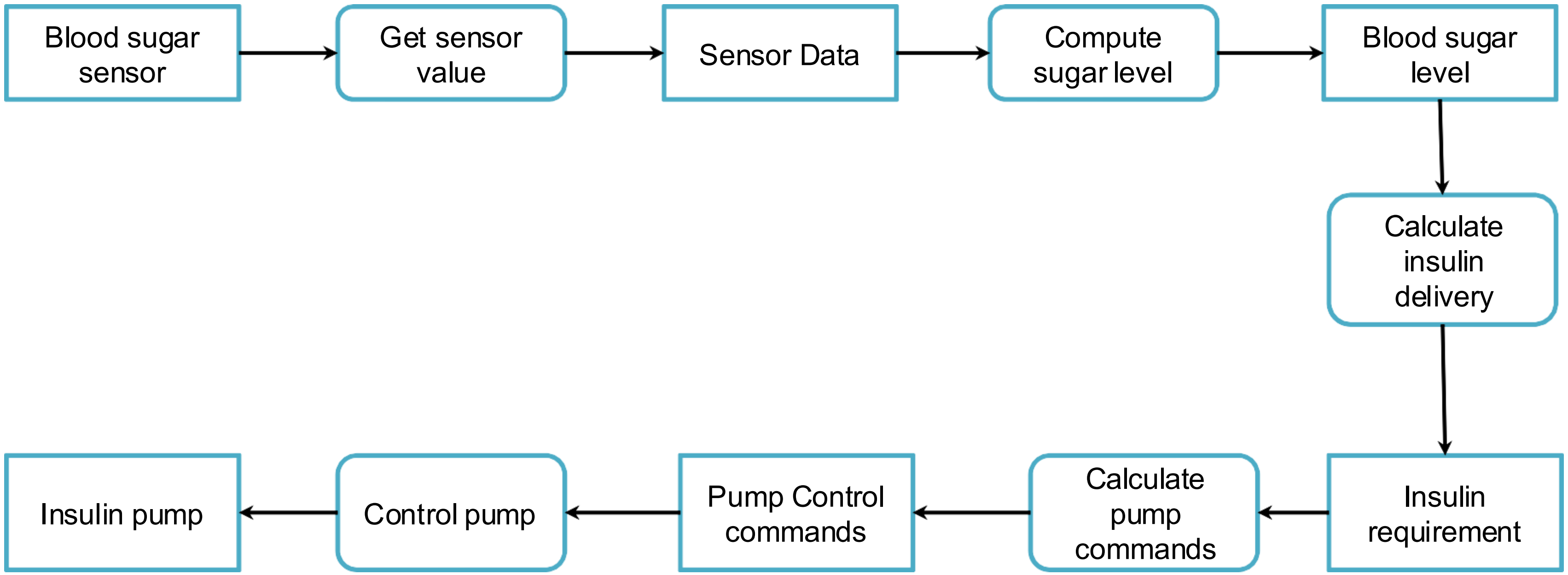
Data-driven modeling



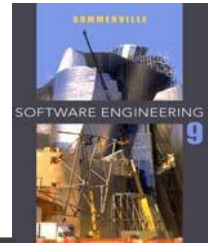
- Many business systems are data-processing systems that are primarily driven by data. They are controlled by the data input to the system, with relatively little external event processing.
- Data-driven models show the sequence of actions involved in processing input data and generating an associated output.
- Show end-to-end processing in a system
- Useful during the analysis of requirements

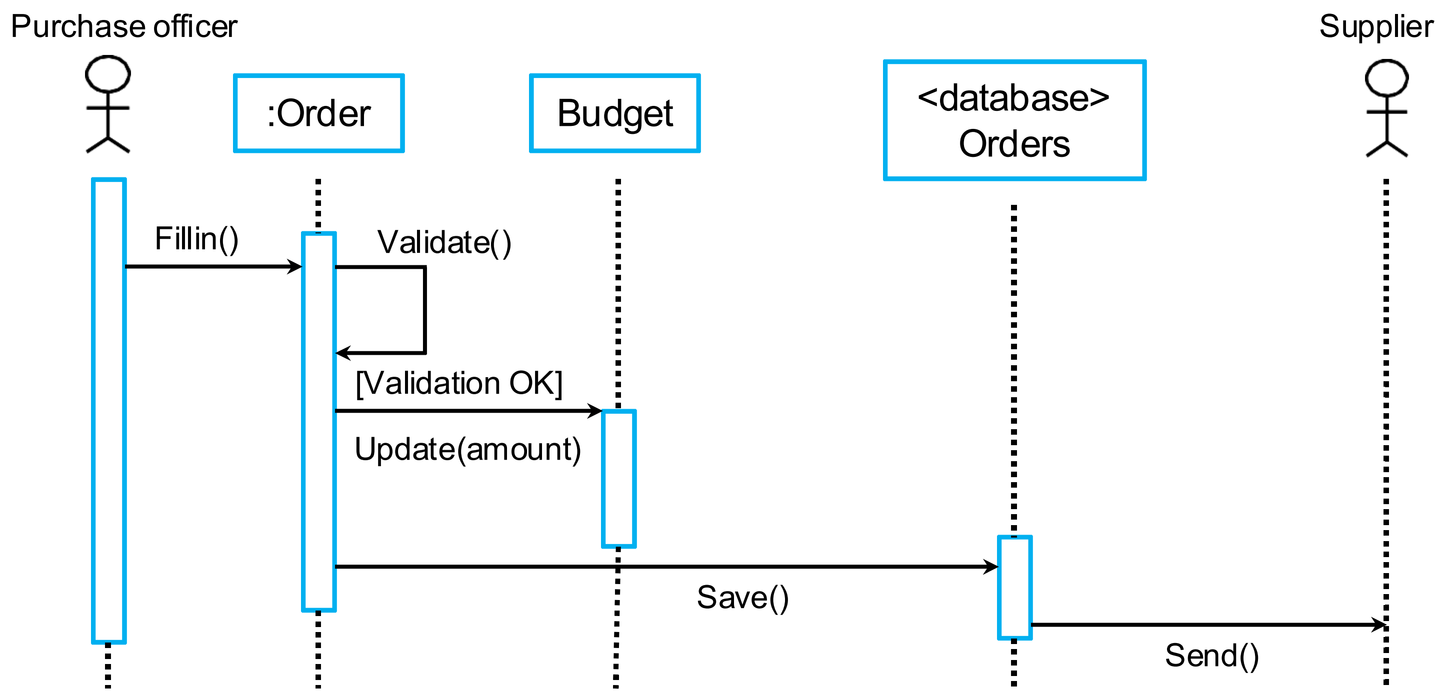
An activity model of the insulin pump's operation



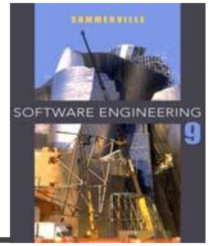


Order processing





Event-driven modeling

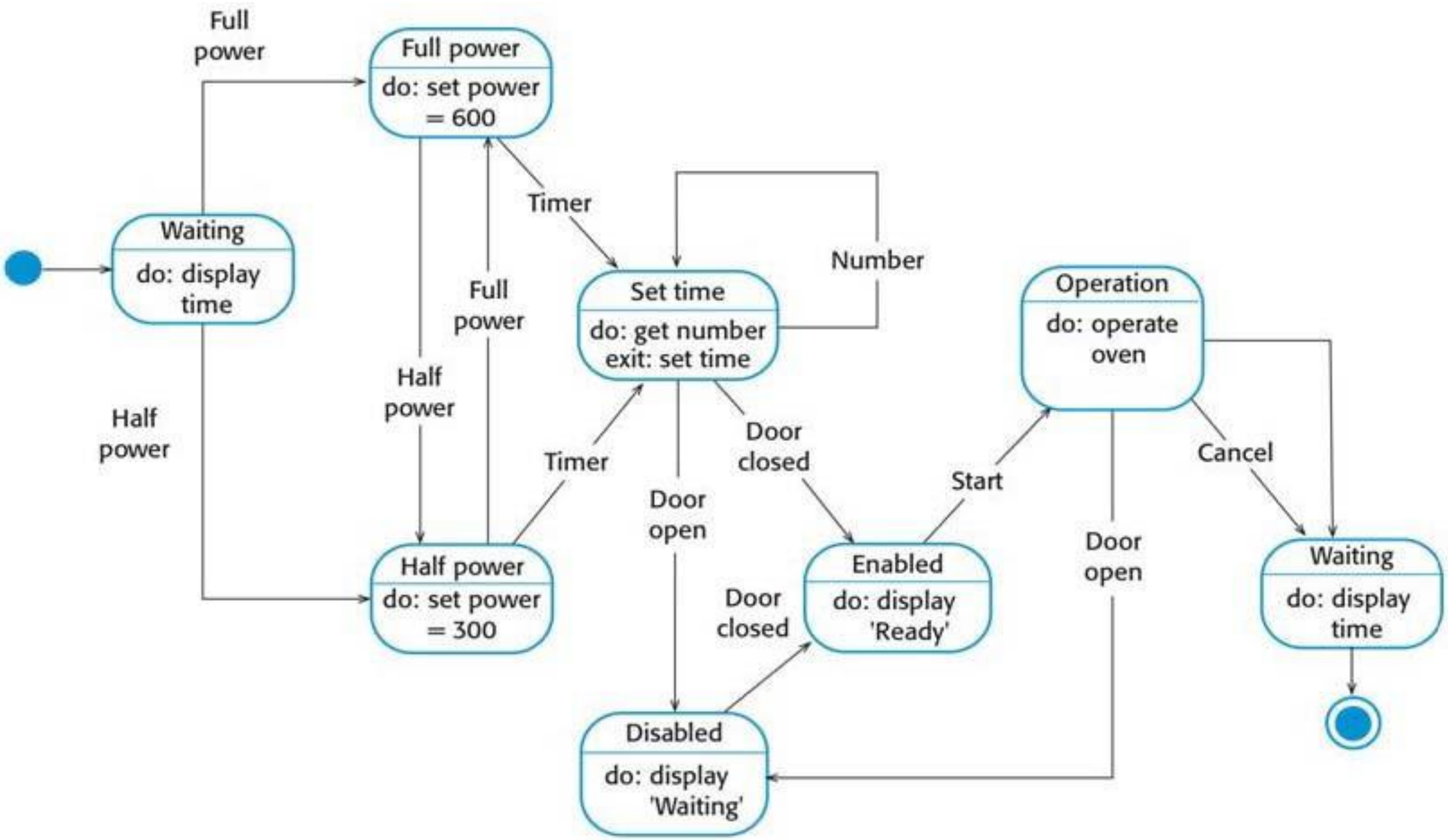


- Real-time systems are often event-driven, with minimal data processing.
  - For example, a landline phone switching system responds to events such as ‘receiver off hook’ by generating a dial tone.
- Event-driven modeling shows how a system responds to external and internal events.
- It is based on the assumption that a system has a finite number of states and that events (stimuli) may cause a transition from one state to another.

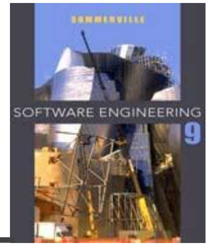
State machine models

- Model the behaviour of the system in response to external and internal events
- Show the system’s responses to stimuli
- Used for modelling real-time systems
- Represented by UML Statechart diagrams
  - System states represents nodes
  - Events represents arcs between these nodes
- When an event occurs, the system moves from one state to another

State diagram of a microwave oven

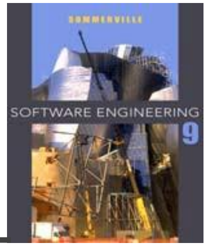


States and stimuli for the microwave oven (a)



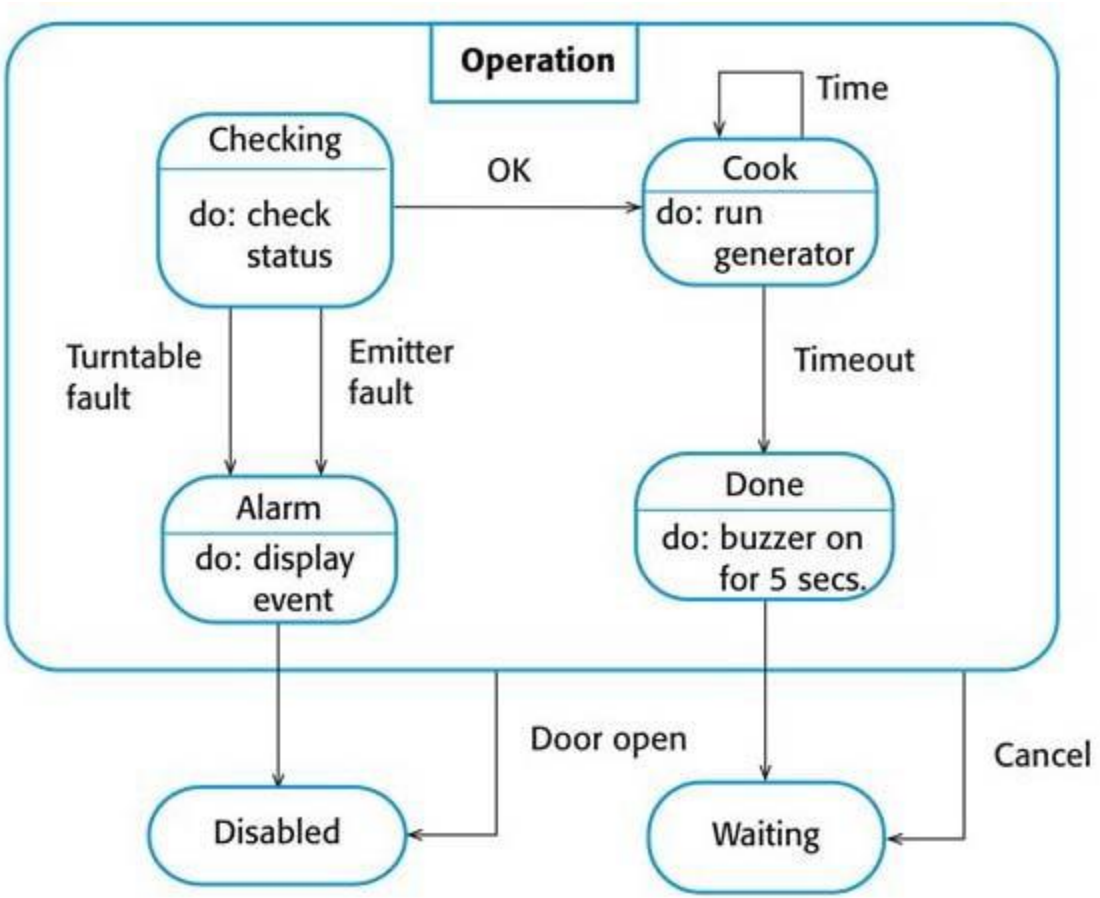
State	Description
Waiting	The oven is waiting for input. The display shows the current time.
Half power	The oven power is set to 300 watts. The display shows ‘Half power’.
Full power	The oven power is set to 600 watts. The display shows ‘Full power’.
Set time	The cooking time is set to the user’s input value. The display shows the cooking time selected and is updated as the time is set.
Disabled	Oven operation is disabled for safety. Interior oven light is on. Display shows ‘Not ready’.
Enabled	Oven operation is enabled. Interior oven light is off. Display shows ‘Ready to cook’.
Operation	Oven in operation. Interior oven light is on. Display shows the timer countdown. On completion of cooking, the buzzer is sounded for five seconds. Oven light is on. Display shows ‘Cooking complete’ while buzzer is sounding.

States and stimuli for the microwave oven (b)



Stimulus	Description
Half power	The user has pressed the half-power button.
Full power	The user has pressed the full-power button.
Timer	The user has pressed one of the timer buttons.
Number	The user has pressed a numeric key.
Door open	The oven door switch is not closed.
Door closed	The oven door switch is closed.
Start	The user has pressed the Start button.
Cancel	The user has pressed the Cancel button.

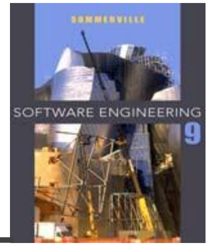
Microwave oven operation



Model-driven engineering - MDE

- An approach to software development where models rather than programs are the principal outputs
- The programs are generated automatically from models
- MDE raises the level of abstraction
  - No need to concern with programming language details or the specifics of execution platforms

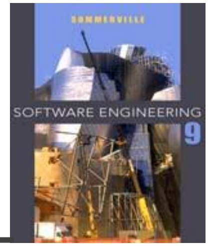
Usage of model-driven engineering



- Still at an early stage of development and unclear whether or not it will have a significant effect on software engineering practice.

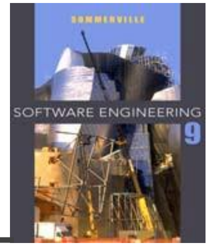
+	−
→ Allows systems to be considered at higher levels of abstraction	→ Abstracted models are not necessarily right for implementation
→ Generating code automatically	→ Savings from generated code may be outweighed by the costs of developing translators for new platforms
→ Cheaper to adapt systems to new platforms	

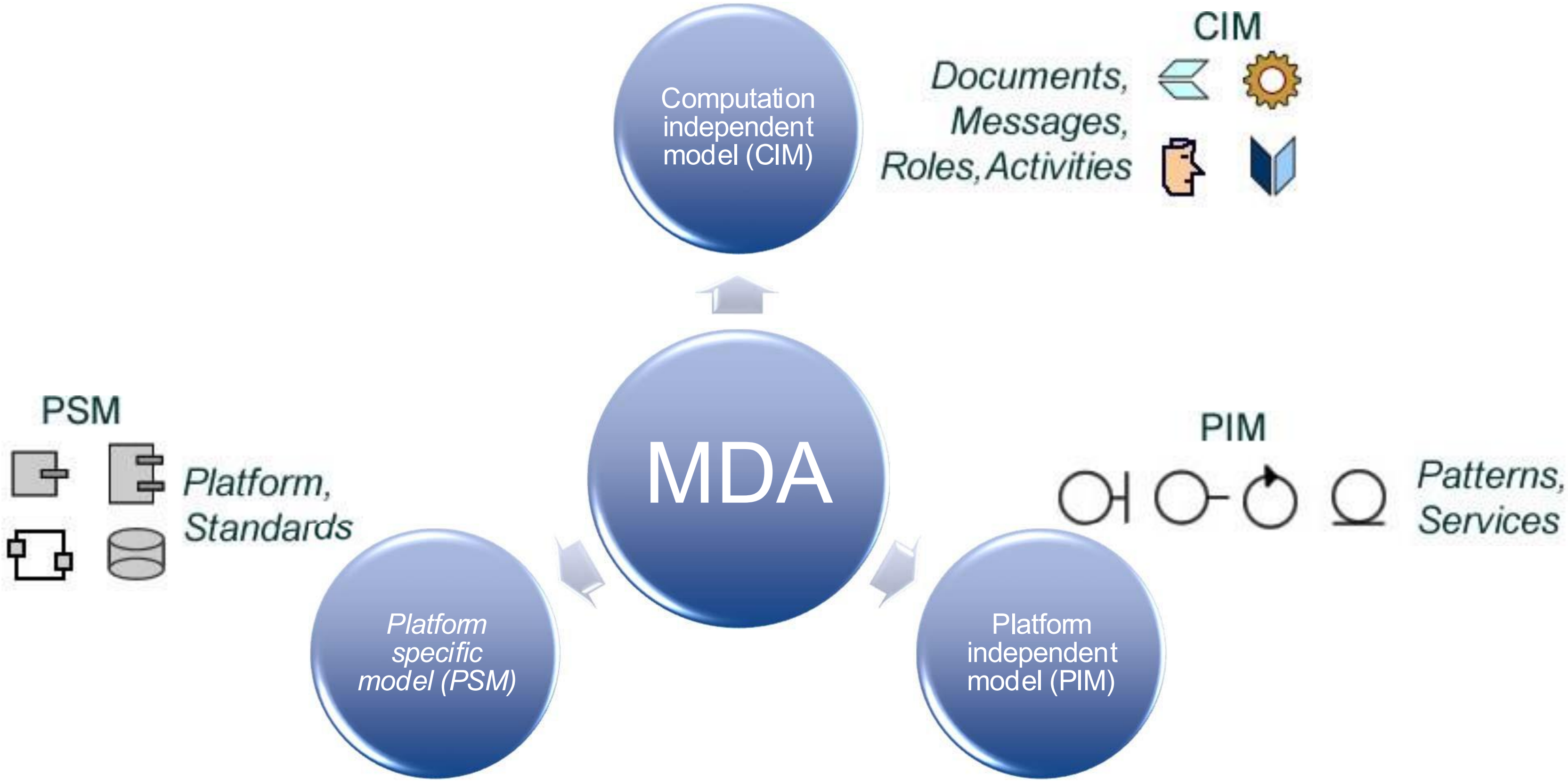
Model-driven Architecture - MDA



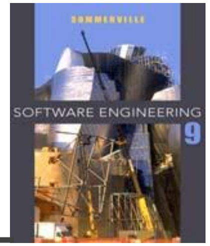
- Precursor of model-driven engineering
- A model-focused approach to software design and implementation
- Models at different levels of abstraction are created.
- From a high-level, platform independent model, it is possible, in principle, to generate a working program without manual intervention.

Types of model



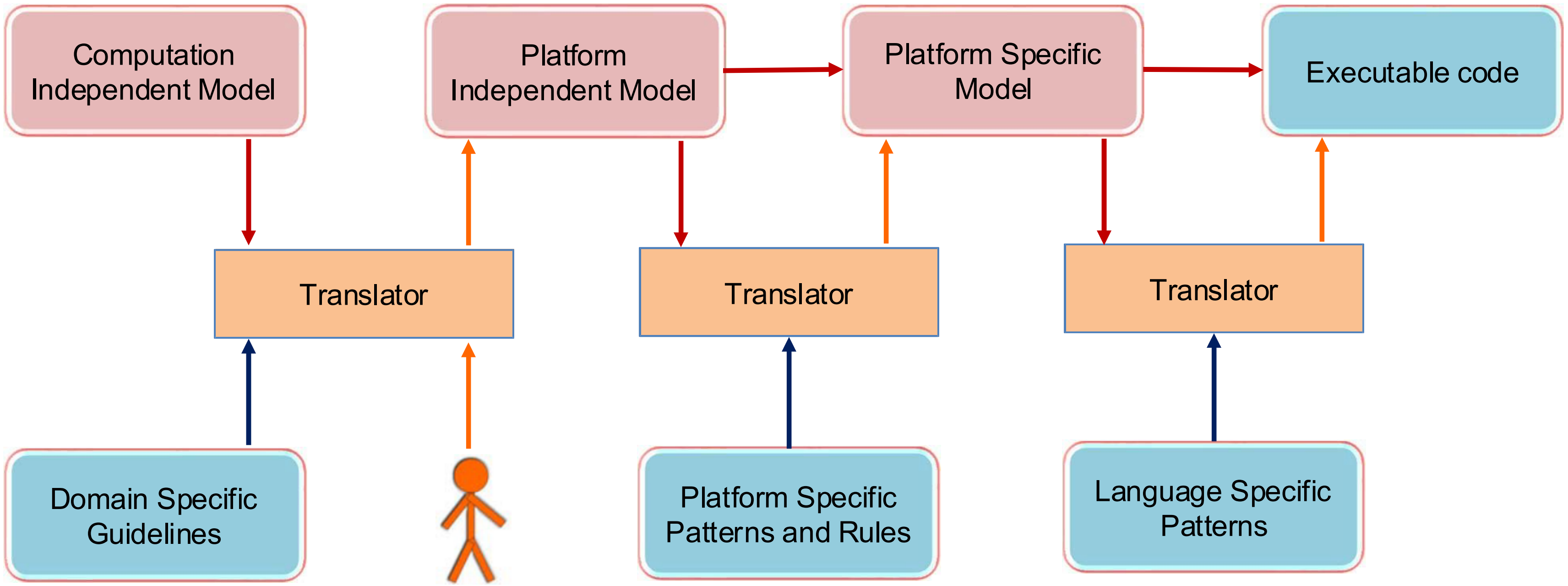


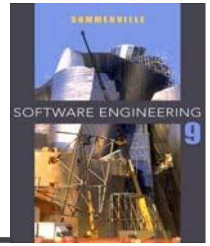
Types of model



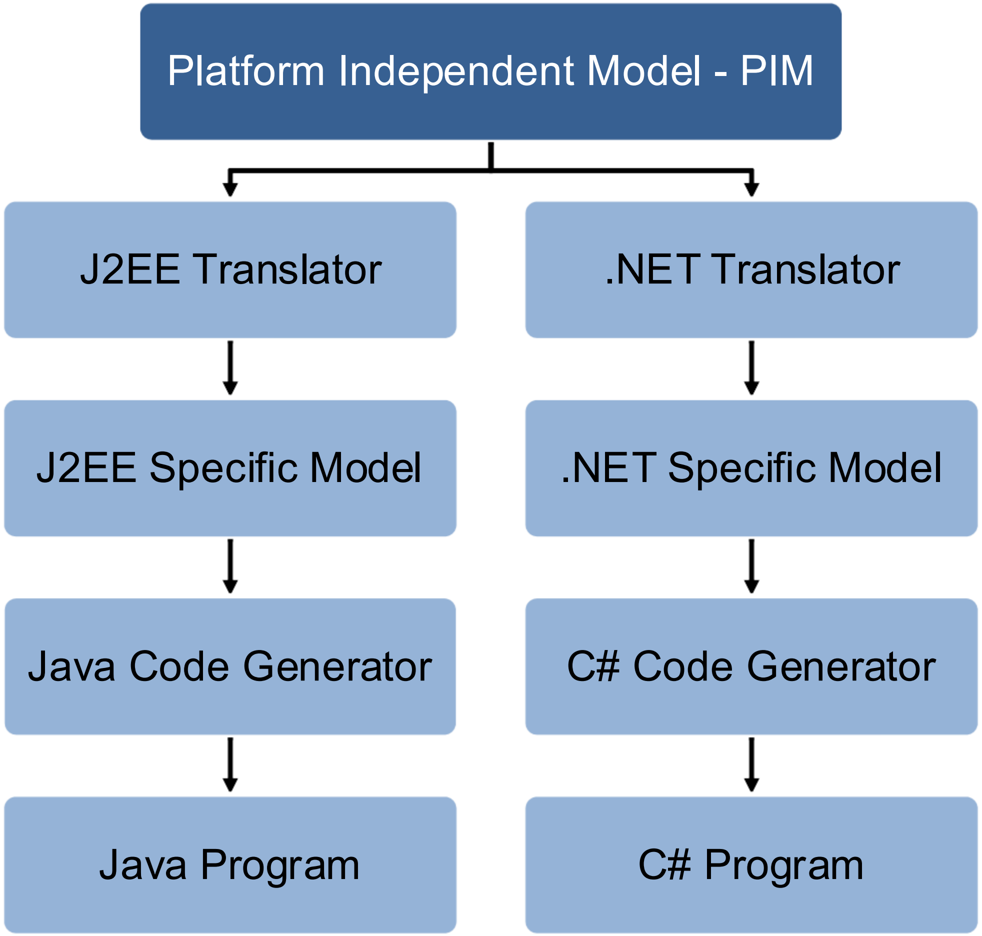
Computation independent model (CIM)	- Model the important domain abstractions used in a system
	- Also called Domain model / Business Model
	- Used in requirement gathering
Platform independent model (PIM)	- Model the operation / functionality of the system without implementation
	- Shows the static system structure and its reactions against events
	- Used in functional (aka analysis) patterns
Platform specific model (PSM)	- Transformation of PIM into multiple application platform PSM layers
	- Each layer of PSM adds some platform-specific detail
	- Used in technical (aka Design / Implementation) patterns

MDA Transformations



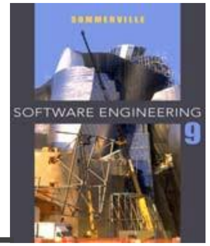


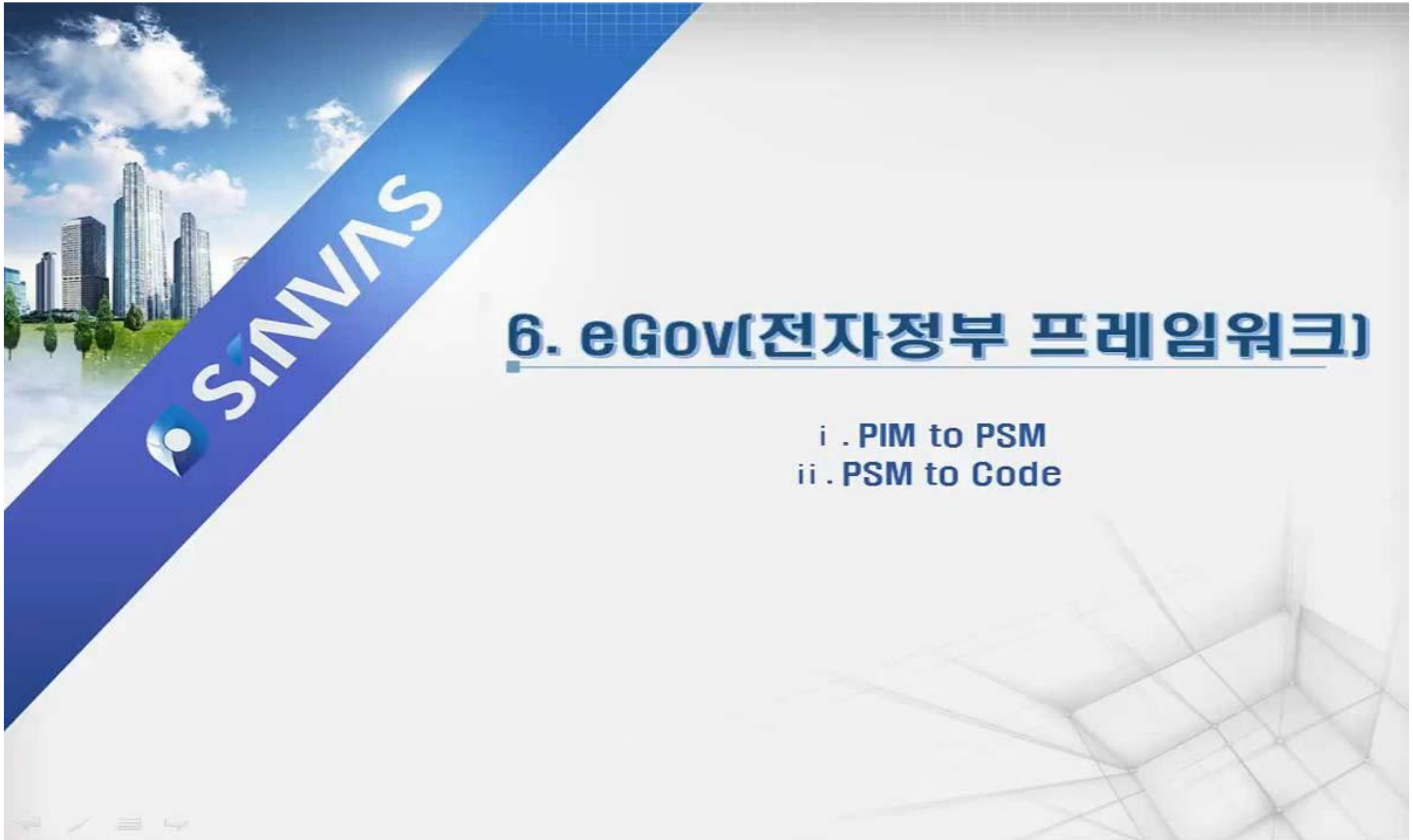
Multiple platform-specific models



Model Driven Development

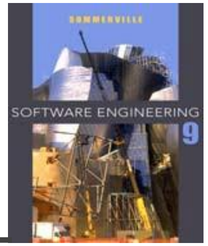
PIM to PSM and PSM to Code



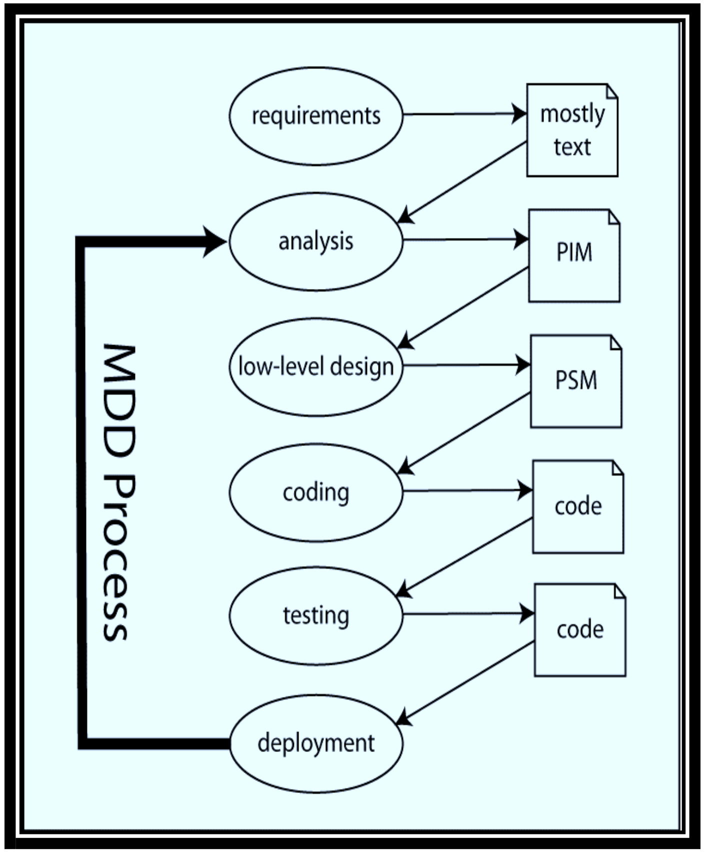


http://www.youtube.com/watch?v=7q7kKk_eaJg

Agile methods and MDA

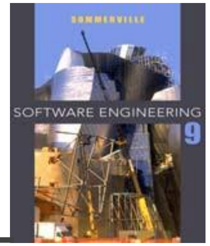


- The MDA supports iterative approach → Agile methods
- But notions of MDA contradicts with the fundamental ideas in the agile manifesto
- If transformations can be completely automated and a complete program generated from a PIM, then, in principle, MDA could be used in an agile development process as no separate coding would be required.



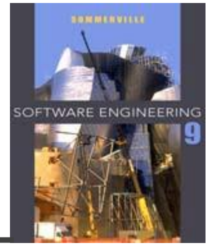
http://www.idt.mdh.se/utbildning/exjobb/files/
TR0962_5_Model_Based_Testing_final.ppt

Executable UML



- The fundamental notion behind model-driven engineering is that completely automated transformation of models to code should be possible.
- Possible using a subset of UML 2, called Executable UML or xUML
- Helps in automated transformation of models to code
- Notions of xUML are used in model-driven engineering
- The dynamic behavior of the system may be specified declaratively using the object constraint language (OCL), or may be expressed using UML's action language.

Types of Executable UML



Domain models

- Identify the principal concerns in a system
- Defined using UML class diagrams
- Include objects, attributes, and associations

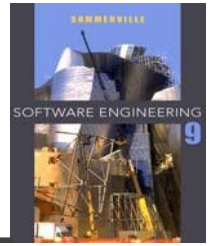
Class models

- Define classes, attributes, and operations

State models

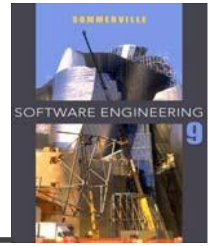
- Associated with each class
- Describe the life cycle of the class

Key points



- **Complementary system models** can be developed to show the system's context, interactions, structure, and behaviour.
- **Context models** show how a system is positioned in an environment with other systems and processes.
- **Interaction models** show the system to system and components interactions.
- **Structural models** show the organization / architecture of a system.
- **Behavioral models** are used to describe the dynamic behavior of an executing system.
- **Model-driven engineering** is an approach to software development in which a system is represented as a set of models that can be automatically transformed to executable code.

Further Readings



- Systems Modelling Overview
  - http://www.youtube.com/watch?v=ayP5Ey-djgw
- Effective SE Communication through Models and Representations
  - http://www.youtube.com/watch?v=7-JCgxNgX40
- Requirements Analysis Models - Systems perspectives
  - https://blog.feabhas.com/tag/context-model/