322 371 Software Engineering

Chapter 6: System Modeling

By

Chitsutha Soomlek, Ph.D.

Department of Computer Science

Faculty of Science, KKU

What is Software Design?

- A software design is like the blueprint of a software that are sufficient for a programmer to build the required system.
- A software design is a representation, or model, of the software to be built.
- It is a set of documents containing *text and diagrams* to serve as the base on which an application can be fully programmed.

The Goals of Software Design

- To be sufficient for satisfying the requirements
- To help programmers to implement the requirements by designating the projected parts of the implementation

- A software design is sufficient if it provides the components for an implementation that satisfies the requirements.
- **High-level design** -> software architecture
- **Detailed design** -> short of being actual code

Design Goals

- **Sufficiency**: handles the requirements
- **Understandability**: can be understood by intended audience
- **Modularity**: divided into well-defined parts
- **Cohesion**: organized so like-minded elements are grouped together
- **Coupling**: organized to minimize dependence between elements
- **Robustness**: can deal with wide variety of input

Design Goals

- **Flexibility**: can be readily modified to handle changes in requirements
- **Reusability**: can use parts of the design and implementation in other applications
- **Information hiding**: module internals are hidden from others
- **Efficiency**: executes within acceptable time and space limits
- **Reliability**: executes with acceptable failure rate

What is 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 uses some kind of graphical notation, e.g. UML and mathematical models, to present a system.

Models

- Used during the requirements engineering process to help derive the requirements for a system
- Used during the design process to describe the system to engineers implementing the system
- Used after implementation to document the system's structure and operation

Models of the Existing Systems

- Used during requirements engineering
- Help clarify what the existing system does and can be used as a basis for discussing its strengths and weaknesses
- Lead to requirements for the new system

Models of the New Systems

- Used during requirements engineering to help explain the proposed requirements to other system stakeholders
- Used by engineers to discuss design proposals and to document the system for implementation
- It is possible to generate a complete or partial system implementation from the system model

Perspectives

- **External perspective** - models the context or environment of the system
- **Interaction perspective** - model the interactions between a system and its environment or between the components of a system แต่ละระบบทำงานร่วมกันยังไง อธิบายความสัมพันธ์ระหว่างระบบ
- **Structural perspective** - models the organization of a system or the structure of the data that is processed by the system
- **Behavioral perspective** - model the dynamic behavior of the system and how it responds to events แสดงถึงพฤติกรรม อง component ในระบบ ถ้ามีข้อมูลเข้ามา component จะตอบสนองยังไง

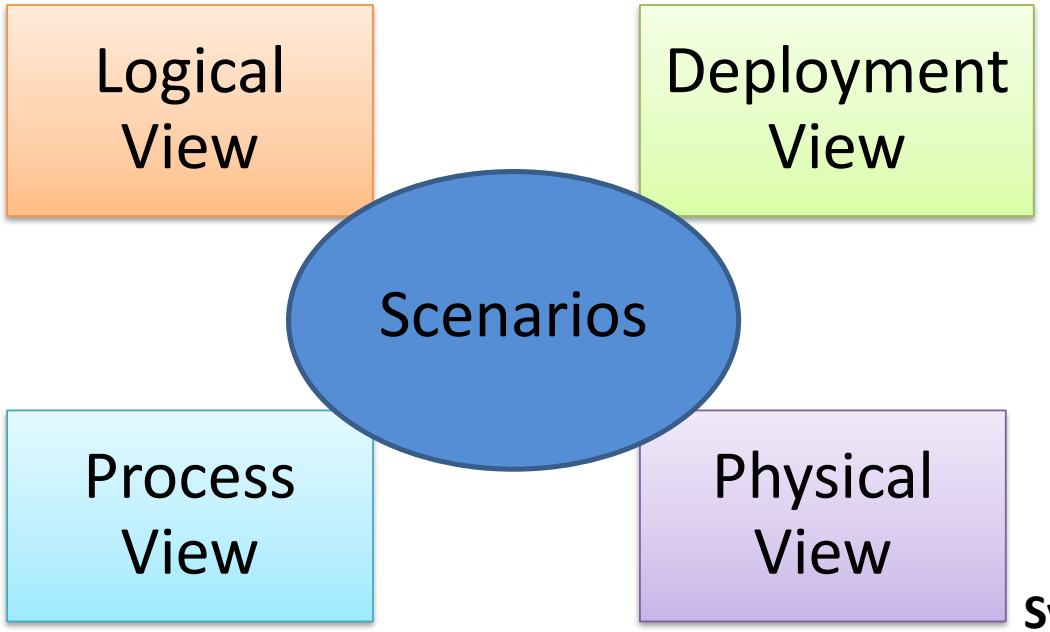
UML Diagrams

- Use case diagram
- Activity diagram
- Sequence diagram
- Class diagram
- State diagram
- Component diagram
- Composite structure diagram
- Deployment diagram
- Object diagram
- Package diagram
- Communication diagram
- Interaction overview diagram
- Timing diagram

Krutchen 4+1 View Model

End users - Functionality

Programmers- Software management



System integrators
-Performance
-Scalability
-Throughput

System engineers
-System topology
-Delivery
-Installation
-Telecommunication

Modified from P. B. Krutchen, The 4+1 View Model of Architecture, IEEE Journal of Software, Vol. 12, No. 6, 1995

Mapping With The UML Diagram

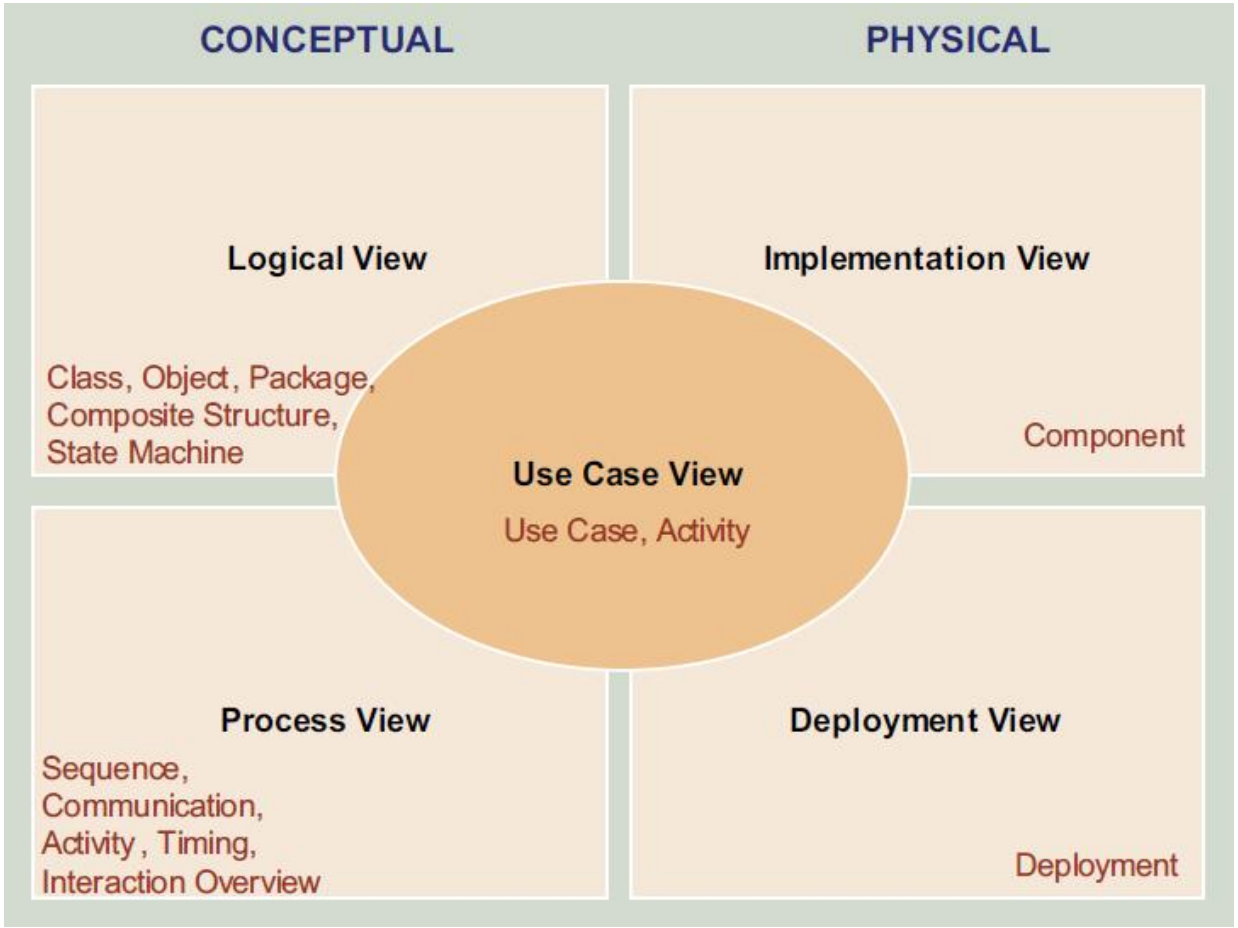


Figure from http://programmers.stackexchange.com/questions/233257/mapping-between-41-architectural-view-model-uml

Diagram ที่มักใช้บ่อยๆ

Essential UML Diagrams

- **Activity diagrams**
  - show the activities involved in a process or in data processing แสดง Process การทำงานของตัวระบบ
- **Use case diagrams** เห็นภาพรวมของตัวระบบ ว่าระบบเรามีความสัมพันธ์กับระบบอื่นไหม
  - show the interactions between a system and its environment
- **Sequence diagrams** Component ข้างในระบบ มีการคุยกันยังไง
  - show interactions between actors and the system and between system components
- **Class diagrams** เห็นโครงสร้างของระบบ
  - show the object classes in the system and the associations between these classes
- **State diagrams** บอกสถานะของระบบว่าเปลี่ยนไปยังไงถ้ามีข้อมูลเข้ามา
  - show how the system reacts to internal and external events

How to Model a System

- **Context Modeling**
  - defines the system boundaries, context, and dependencies
- **Interaction Modeling**
  - defines user interaction, the system's interaction with other systems, and the interaction between the system's components
- **Structural Modeling**
  - specifies the organization of the system's components and their relationship
- **Behavioral Modeling**
  - defines what happens or supposed to happen when a system is executing

Context Modeling

- decide on the system boundaries
- decide what functionality should be included in the system and what is provided by the system's environment
- look at possible overlaps in functionality with existing systems and decide where new functionality should be implemented
- decide that automated support for some business processes should be implemented but others should be manual processes or supported by different systems

System Boundary

- 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.

Context Models

- Context models are used to illustrate the operational context of a system.
- Context models show what lies outside the system boundaries.
- Context models are architectural models that show the system and its relationship with other systems.
- Context models do not show the types of relationships between the systems in the environment and the system that is being specified.
- Therefore, context models are used along with other models, such as business process model.

Example of Context Models

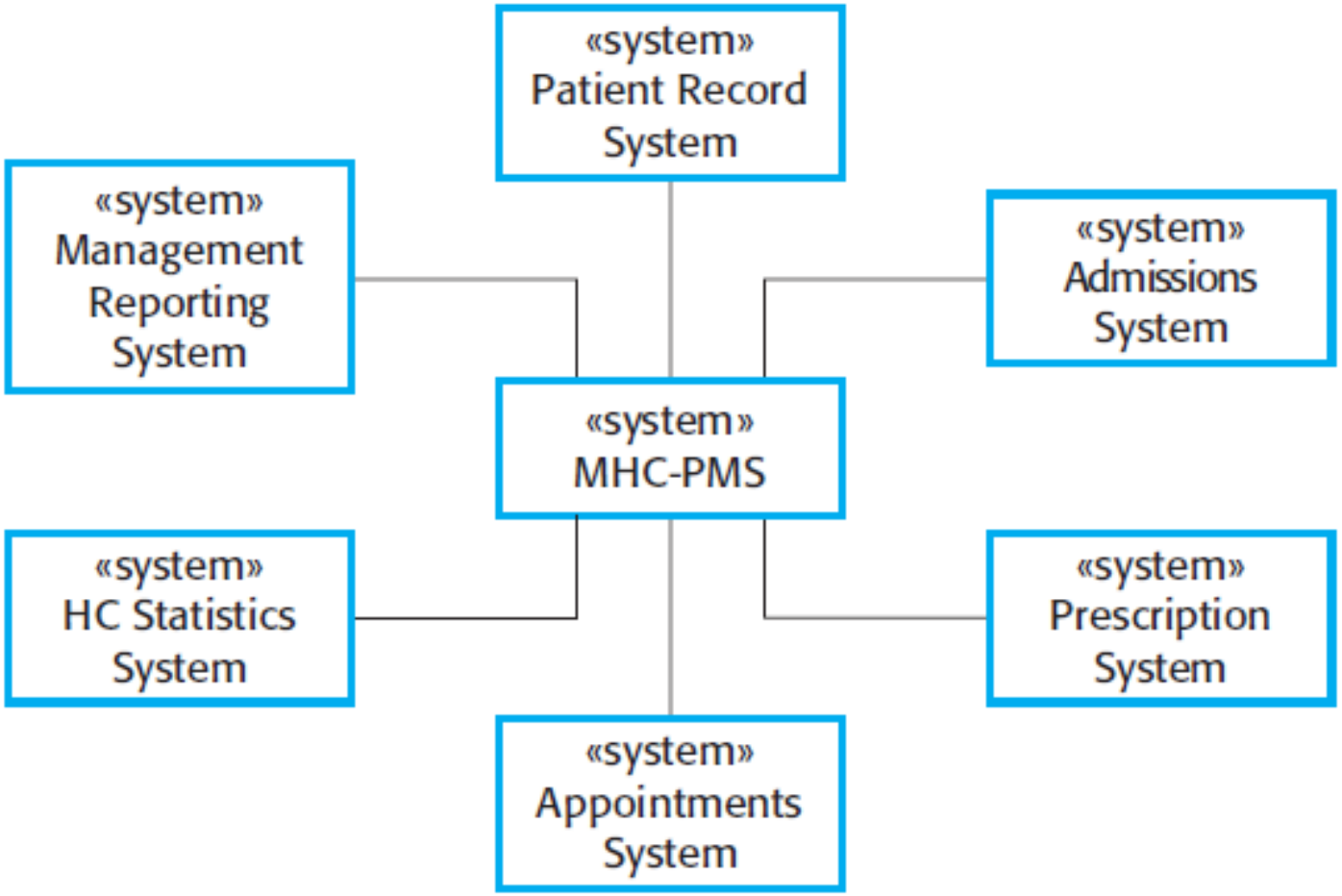


Figure from I. Sommerville, Software Engineering, 9th ed.

Limitations of Context Models

- Context models simply show the other systems in the environment, not how the system being developed is used in that environment.
- Therefore, simple context models are used along with other models, such as business process models and UML activity diagram.

Example

Activity Diagram

ระบบ PMS บันทึกข้อมูลคนไข้

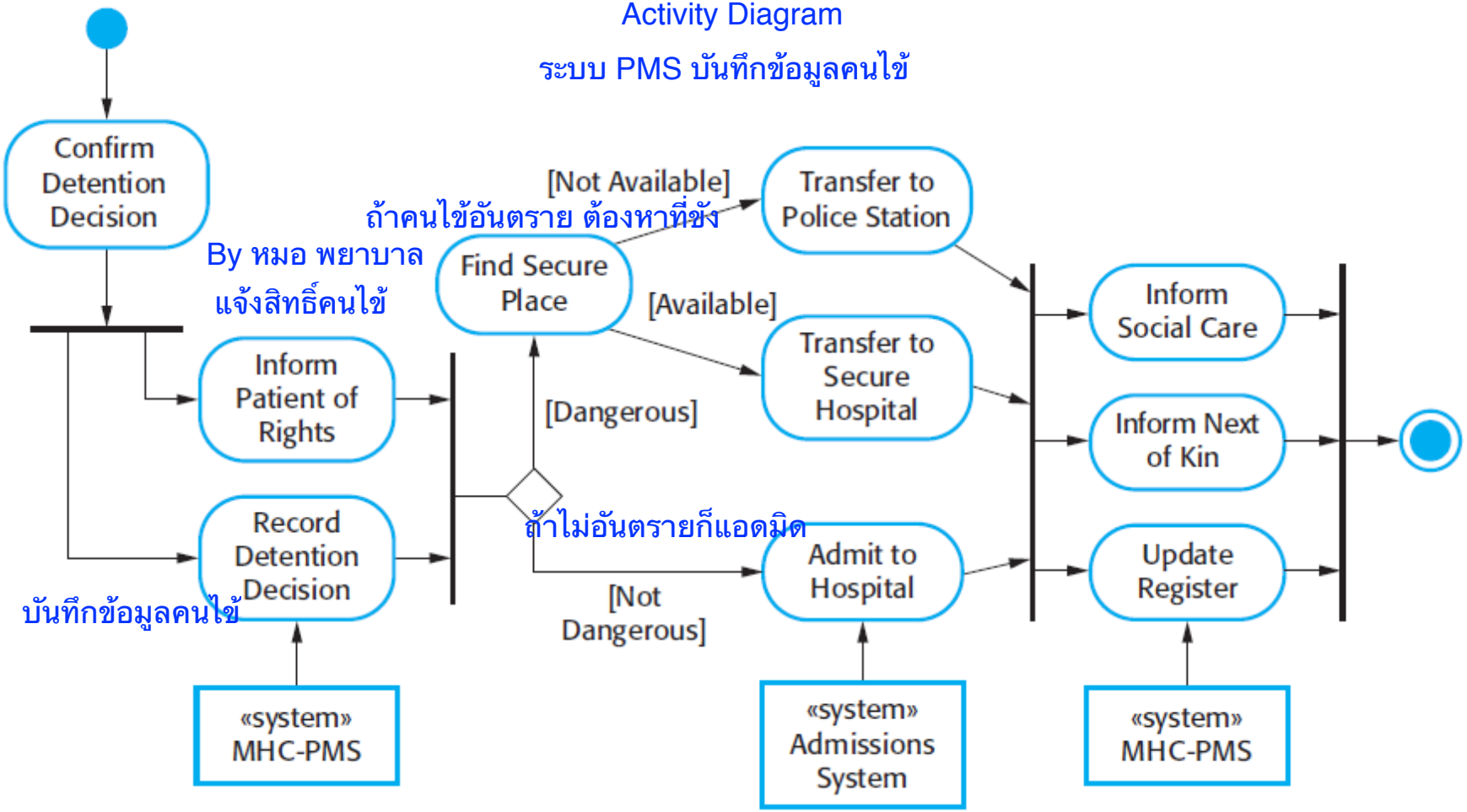


Figure from I. Sommerville, Software Engineering, 9th ed.

Interaction Modeling

User ให้ input อะไรกับระบบเรา และระบบให้ output อะไร

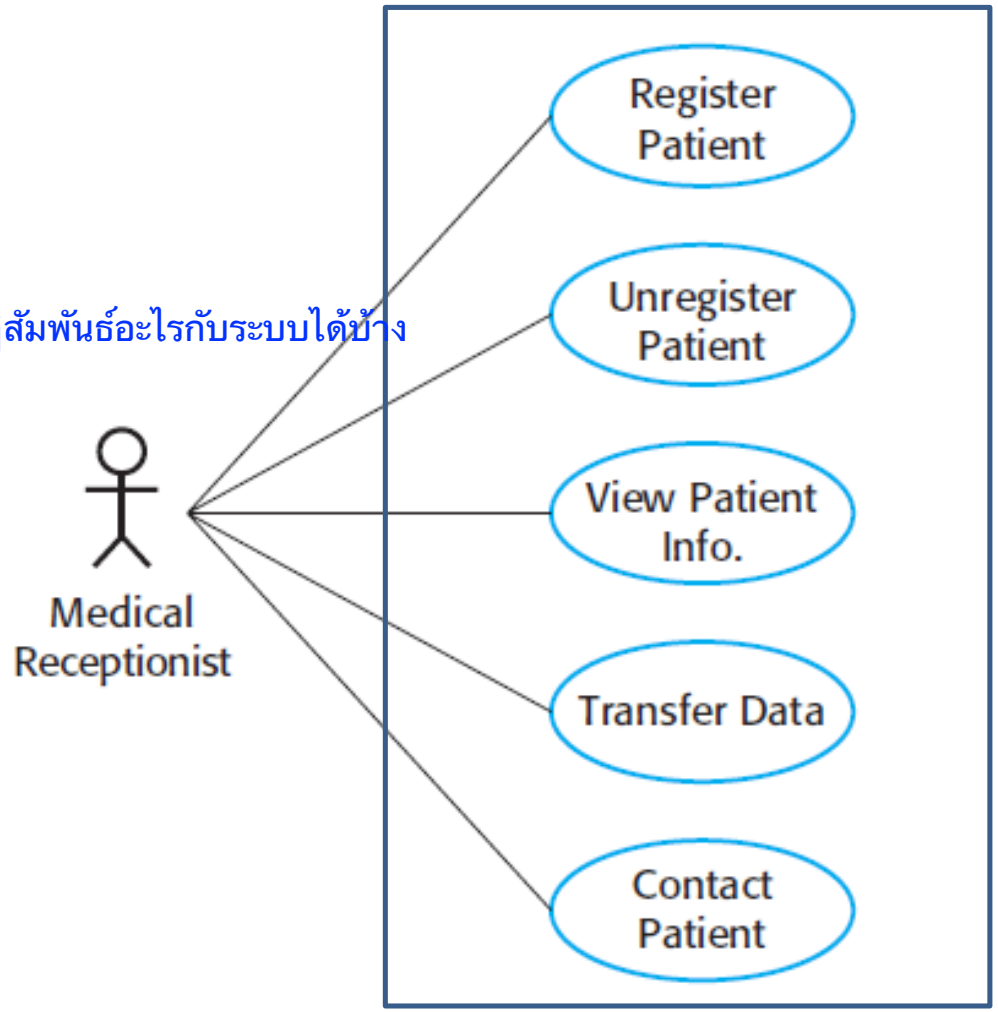
- All systems involve interaction of some kind:
  - **User interaction** - involves user inputs and outputs
  - **System interaction** - interaction between the system being developed and other systems or interaction between the components of the system
- There are two major approaches to interaction modeling:
  - **Use case modeling** - mostly used to model interactions between a system and external actors (users or other systems)
  - **Sequence diagrams** - used to model interactions between system components, although external agents may also be included

Use Case Modeling

- Use case modeling is widely used to support requirements elicitation.
- A use case can be taken as a simple scenario that describes what a user expects from a system.
- Each use case represents a discrete task that involves external interaction with a system.

Example

แสดงว่า user คนนี้ปฏิสัมพันธ์อะไรกับระบบได้บ้าง



หยิบยูสเคสมาเจาะลึกว่ายูสเคสนั้นต้องมีคอมโพแนนอะไรมาเกี่ยวข้องบ้างและทำงานร่วมกันยังไง

Sequence Diagrams

- Use case models and sequence diagrams present interaction at different levels of detail and so may be used together.
- The details of the interactions involved in a high-level use case may be documented in a sequence diagram.
- Communication diagram (also called collaboration diagram) is an alternative representation of interaction

Sequence Diagrams

- Sequence diagrams are part of the UML and are used to model the interactions between the actors and the objects within a system and the interactions between the objects themselves.
- A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance.

Example

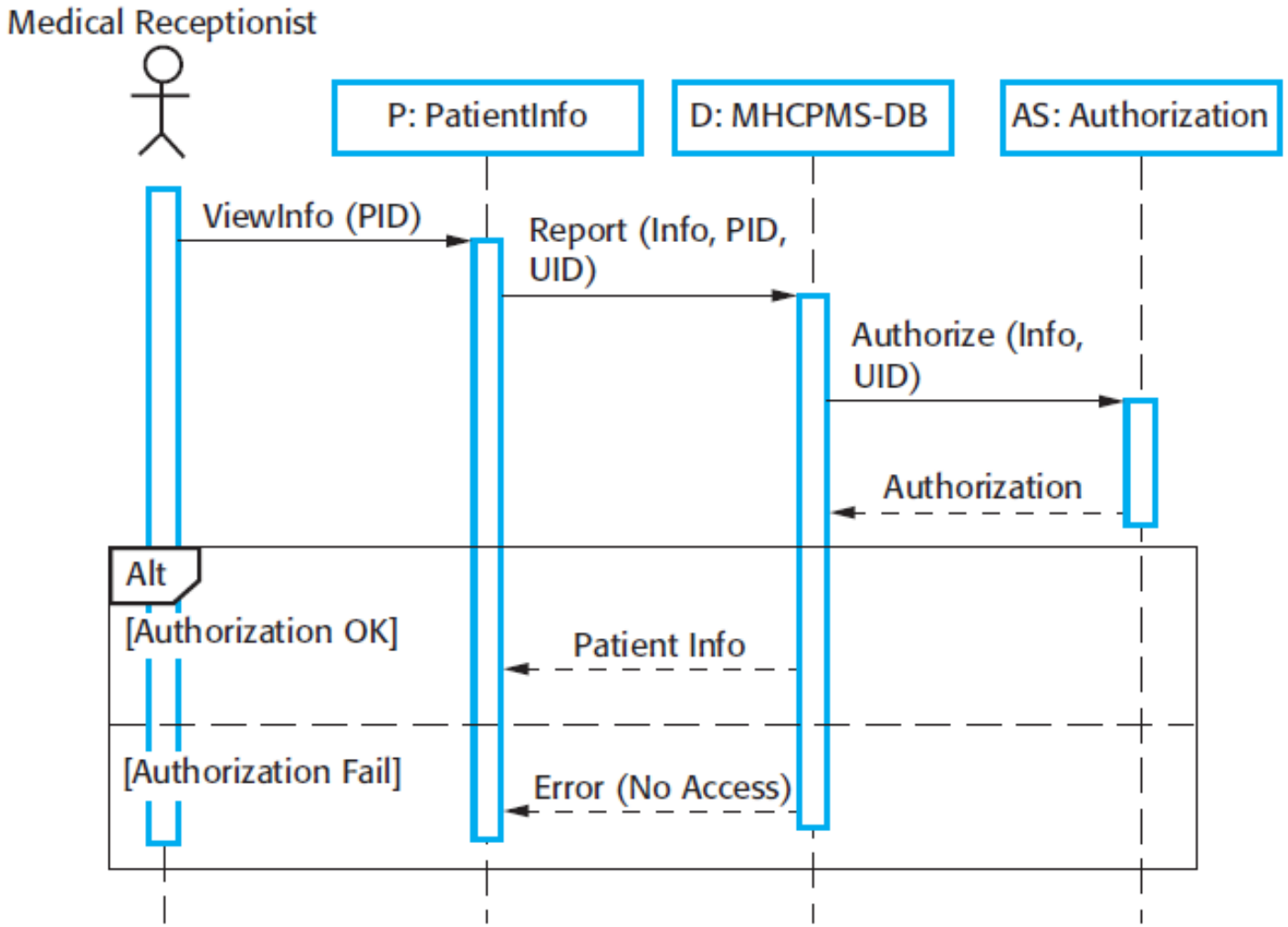


Figure from I. Sommerville, Software Engineering, 9th ed.

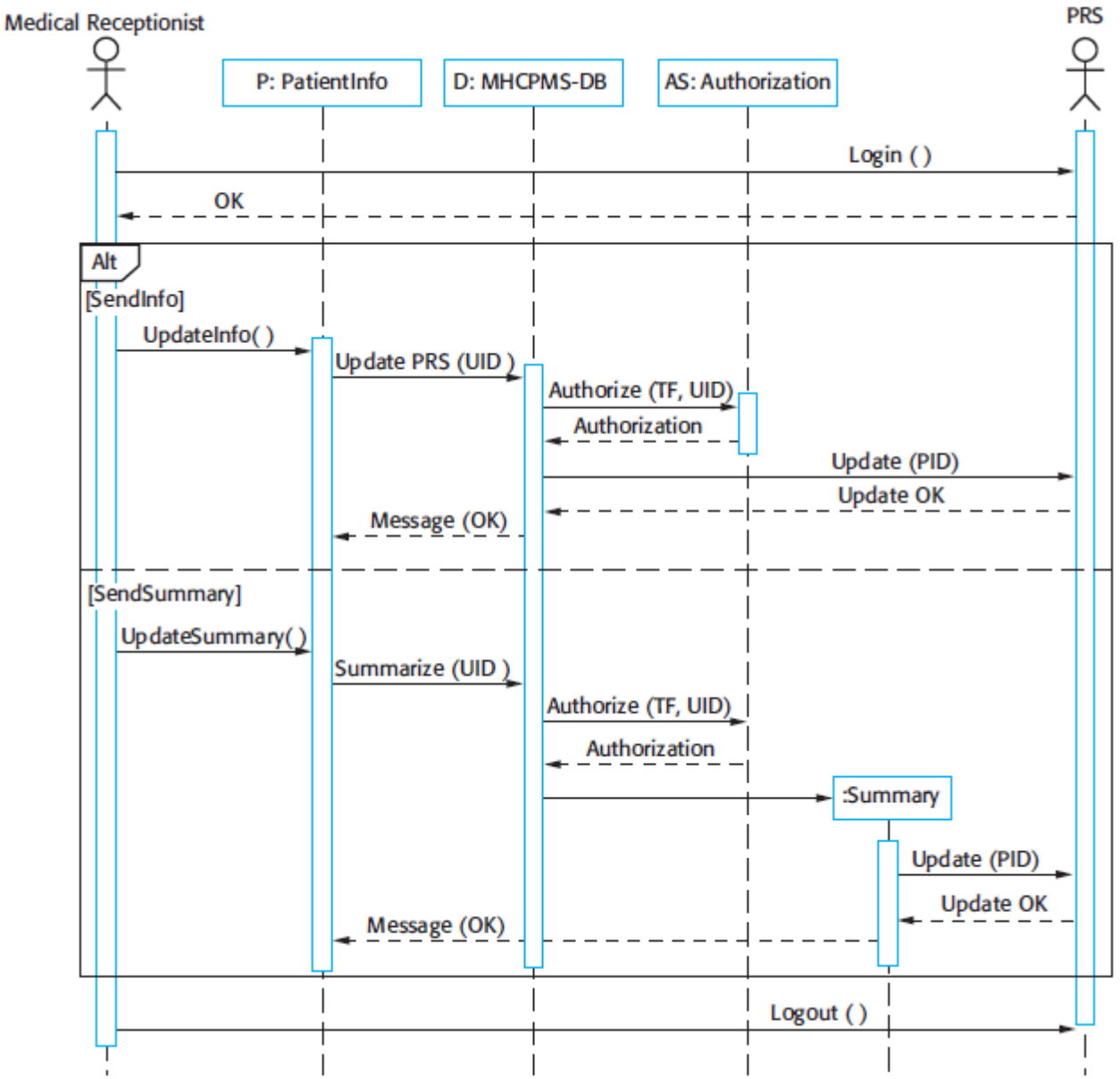
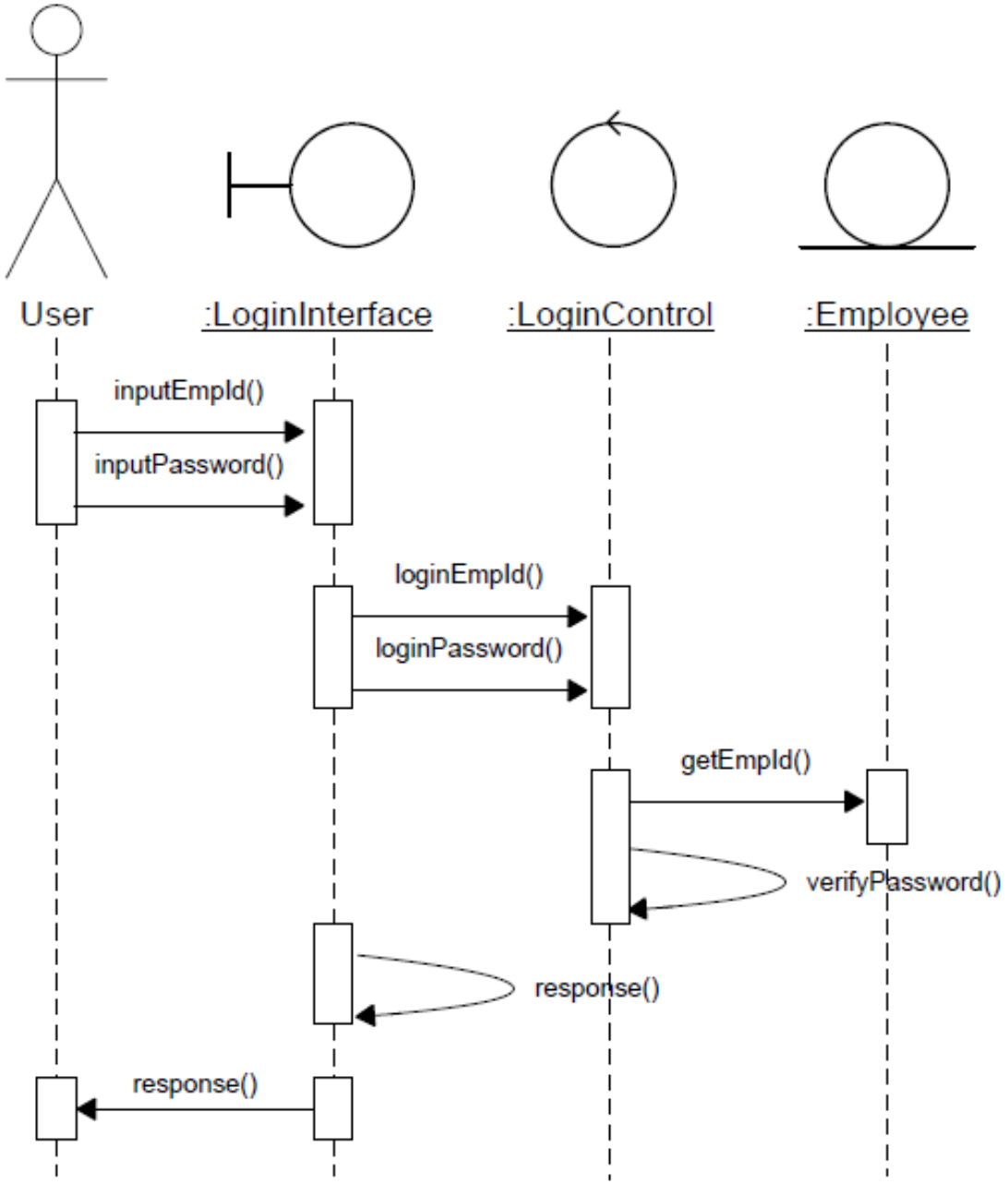


Figure from I. Sommerville, Software Engineering, 9th ed.


User
:LoginInterface
:LoginControl
:Employee
inputEmpId()
inputPassword()
loginEmpId()
loginPassword()
getEmpId()
verifyPassword()
response()
response()

Note!!

- To present the sequential data processing in the system using a sequence diagram, you have to draw a sequence diagram containing messages are only sent from left to right.

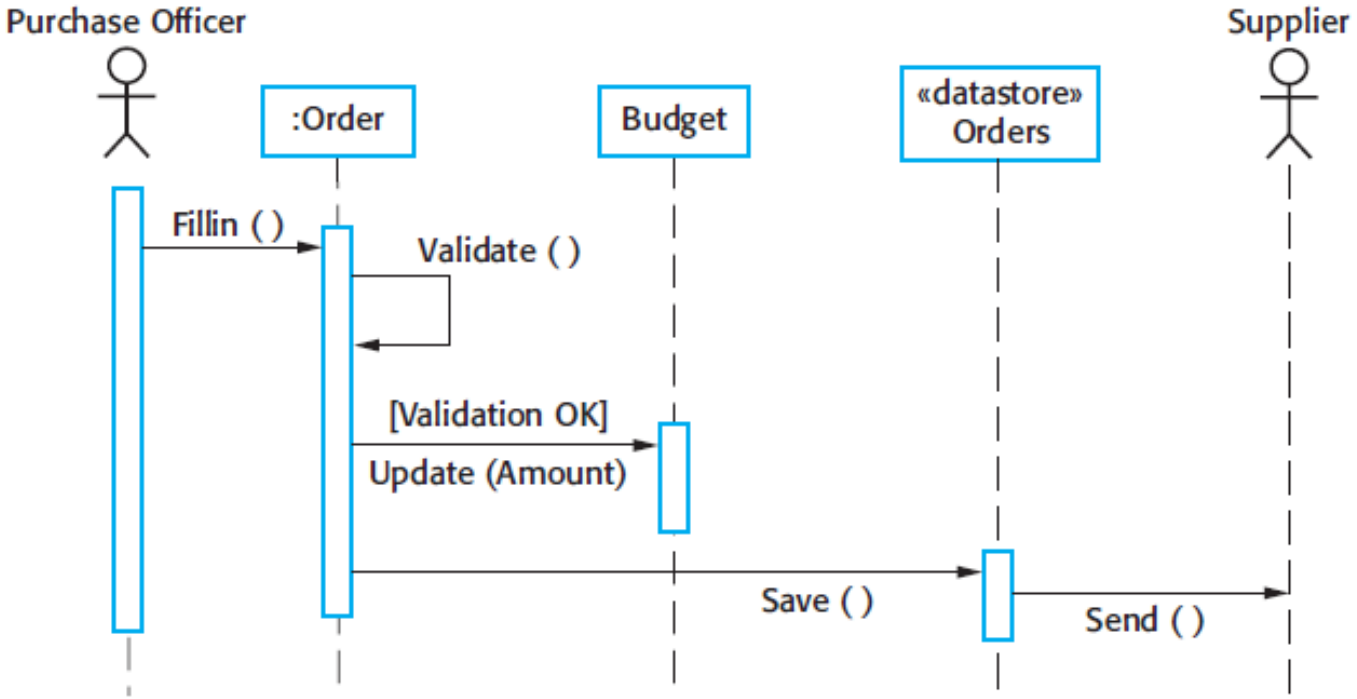


Figure from I. Sommerville, Software Engineering, 9th ed.

Structure Modeling

- Structural models of software display the organization of a system in terms of the components that make up that system and their relationships.
- Structural models may be static models, which show the structure of the system design, or dynamic models, which show the organization of the system when it is executing.
- You create structural models of a system when you are discussing and designing the system architecture.
- Examples are UML class, component, package, and deployment diagrams.

Class Diagrams

- Class diagrams can be used to model the static structure of the object classes in a software system.
- Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes.
- An object class can be thought of as a general definition of one kind of system object.
- An association is a link between classes that indicates that there is some relationship between these classes.

Class Diagrams

- Class diagrams can be expressed in different levels of detail.
- When you are developing a model, the first stage is usually to look at the world, identify the essential objects, and represent these as classes.

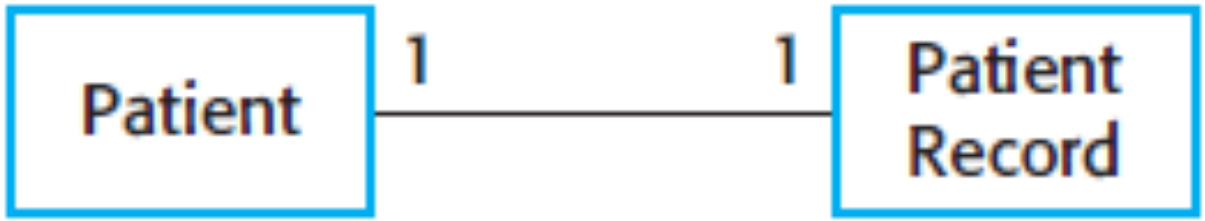


Figure from I. Sommerville, Software Engineering, 9th ed.

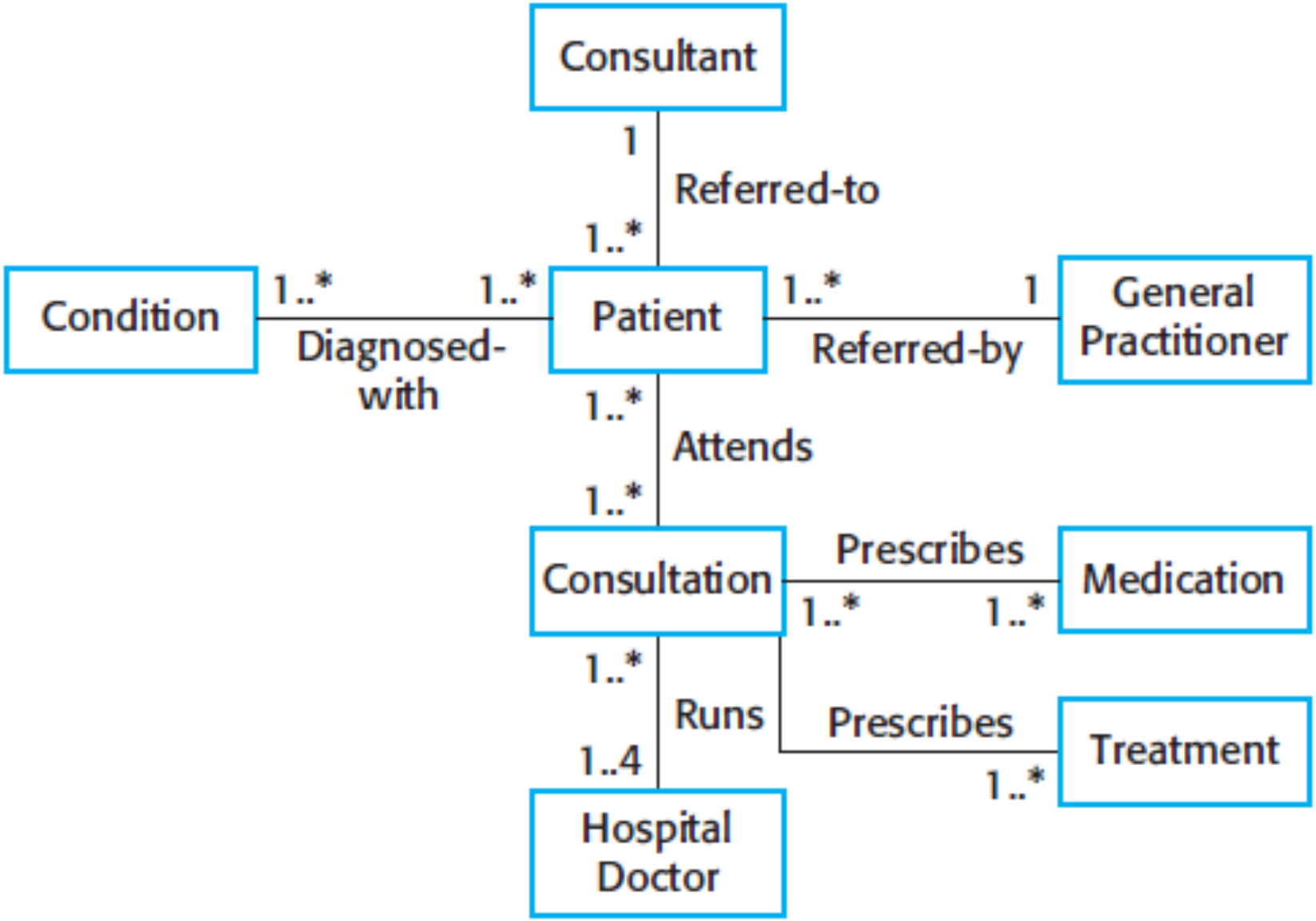


Figure from I. Sommerville, Software Engineering, 9th ed.

- To define classes in more detail, add their attributes and operations.

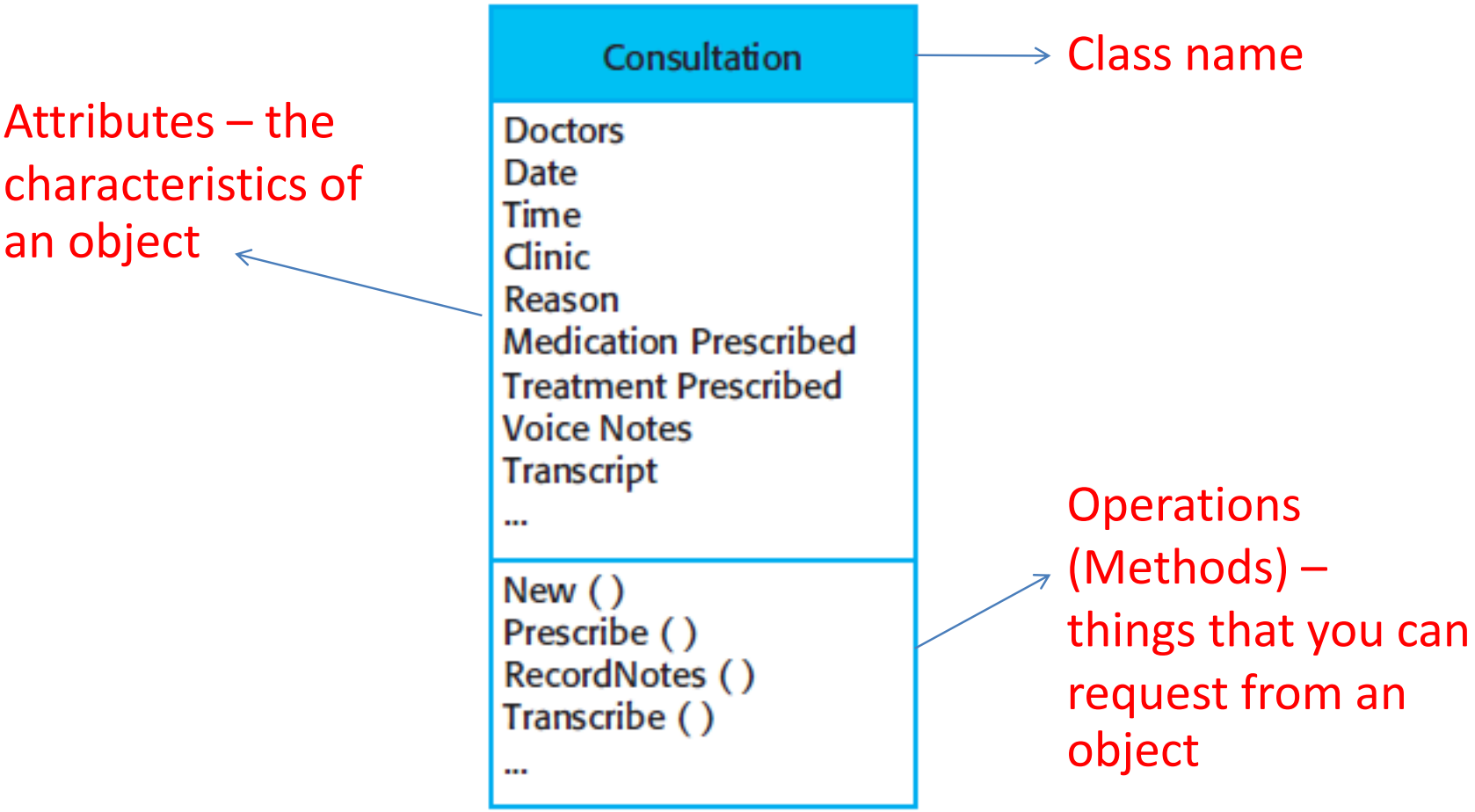


Figure from I. Sommerville, Software Engineering, 9th ed.

Generalization

- Rather than learn the detailed characteristics of every entity that we experience, we place these entities in more general classes (animals, cars, houses, etc.) and learn the characteristics of these classes.
- Generalization allows us to infer that different members of these classes have some common characteristics, e.g., squirrels and rats are rodents. We can make general statements that apply to all class members, e.g., all rodents have teeth for gnawing.

Advantages of Generalization

- If changes are proposed, then you do not have to look at all classes in the system to see if they are affected by the change.
- The attributes and operations associated with higher-level classes are also associated with the lower-level classes.
- The lower-level classes are subclasses inherit the attributes and operations from their superclasses. These lower-level classes then add more specific attributes and operations.

Example

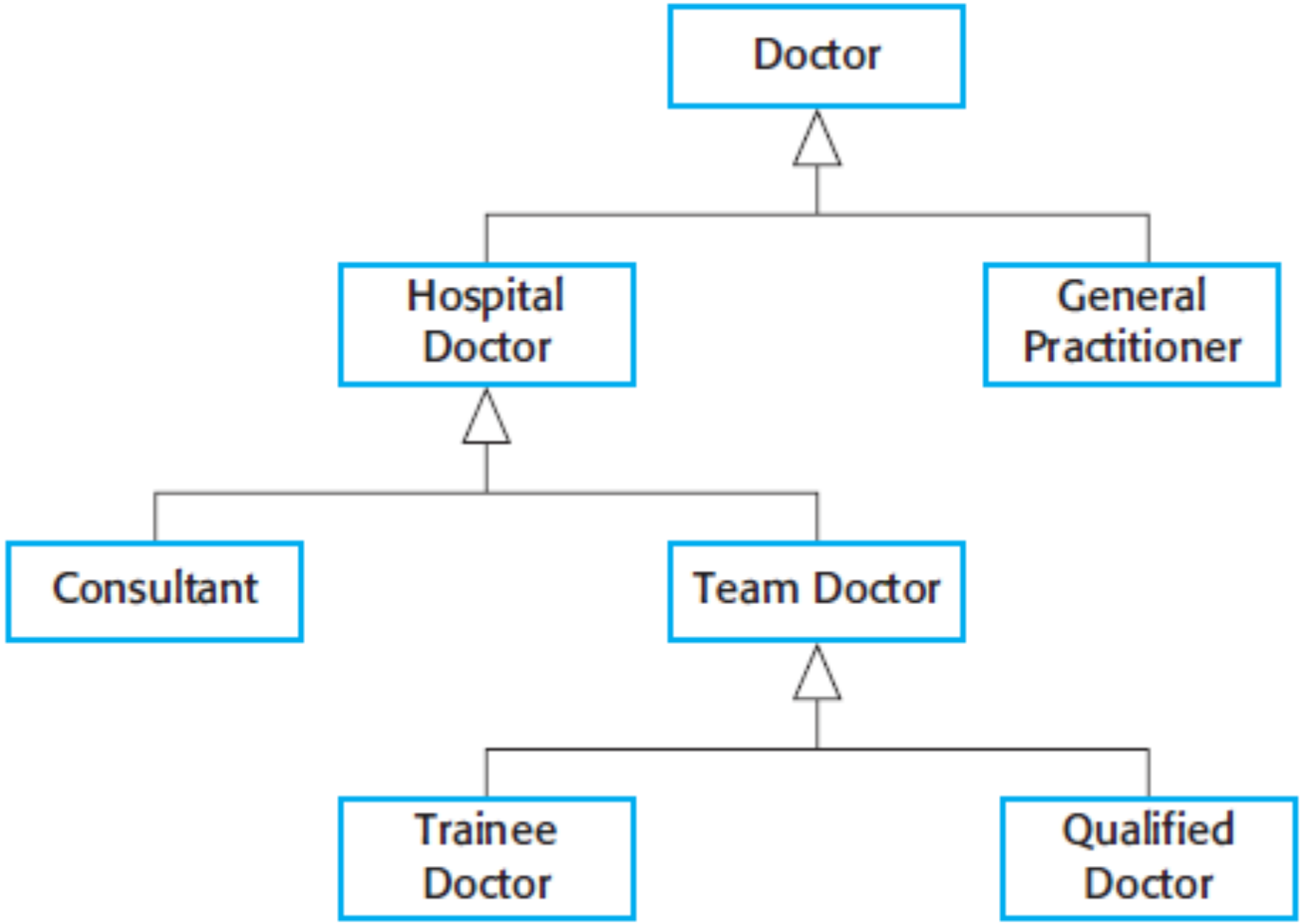


Figure from I. Sommerville, Software Engineering, 9th ed.

Example

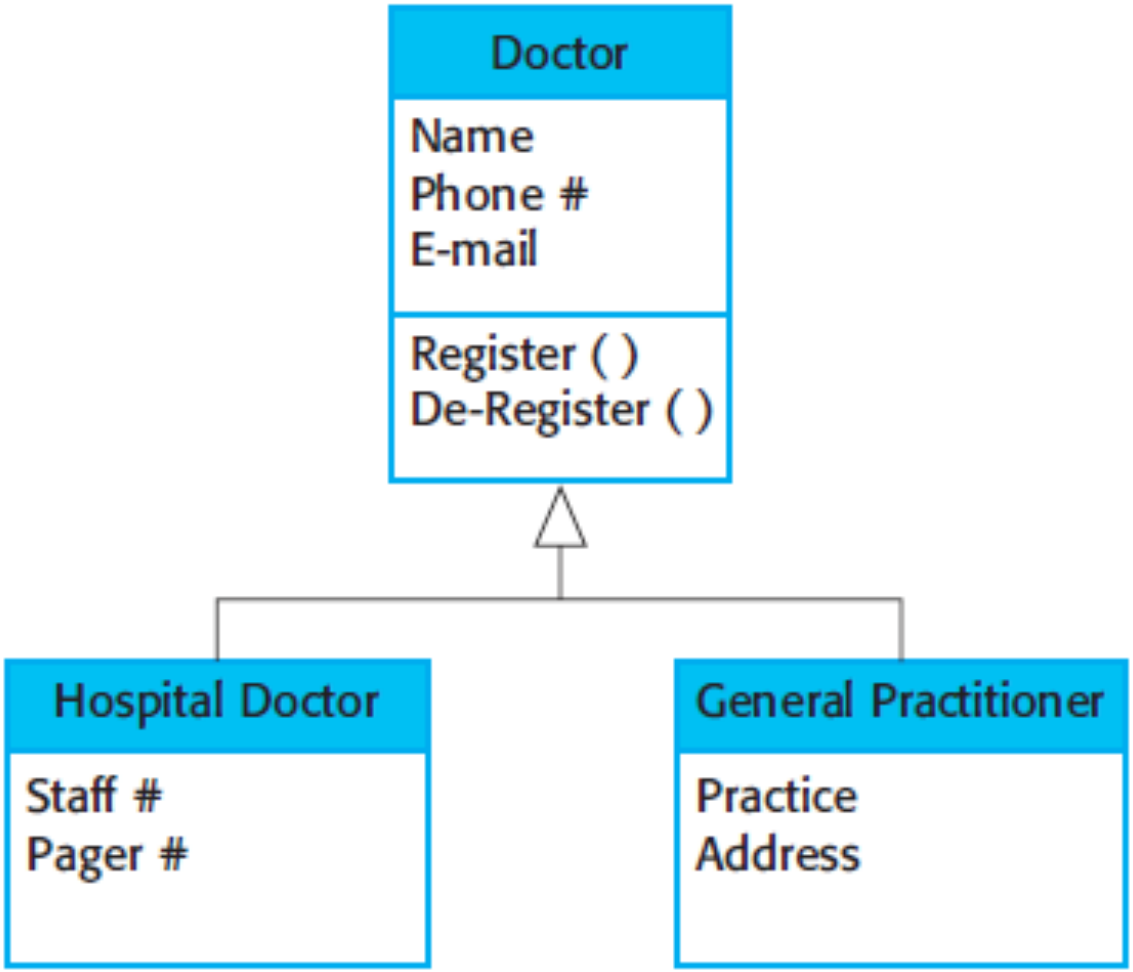


Figure from I. Sommerville, Software Engineering, 9th ed.

Aggregation

- An aggregation means that one object (the whole) is composed of other objects (the parts).
- Aggregation models are similar to the part-of relationship in semantic data models.

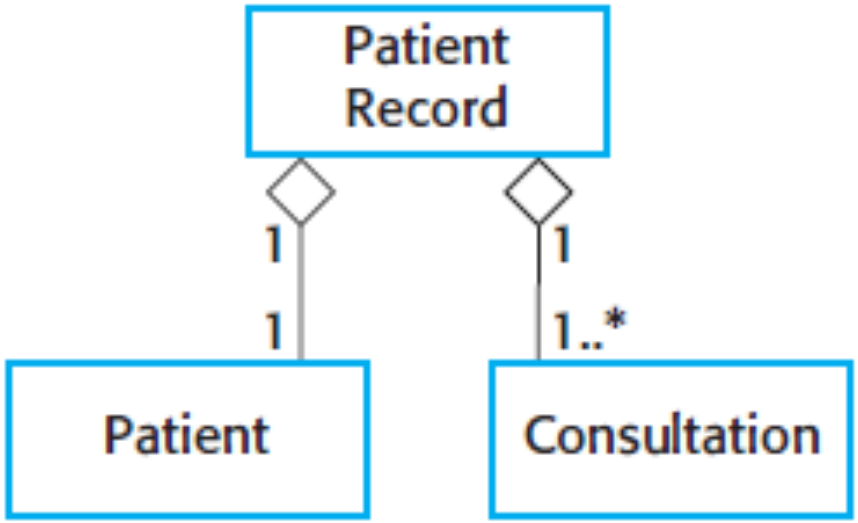


Figure from I. Sommerville, Software Engineering, 9th ed.

Behavioral Modeling

- Behavioral models are models of the dynamic behavior of a system as it is executing.
- Behavioral models show what happens or what is supposed to happen when a system responds to a stimulus from its environment.
- There are two types of stimuli:
  - **Data** - Some data arrives that has 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.
- They are particularly useful during the analysis of requirements as they can be used to show end-to-end processing in a system.

Data-Driven Modeling

- Data-driven models show the entire sequence of actions that take place from an input being processed to the corresponding output, which is the system's response.
- Example – Data Flow Diagram (DFD) introduced in Demarco's Structured Analysis, UML activity diagram, and UML sequence diagram.
- DFDs are used to show how data flows through a sequence of processing steps.

Example of DFDs

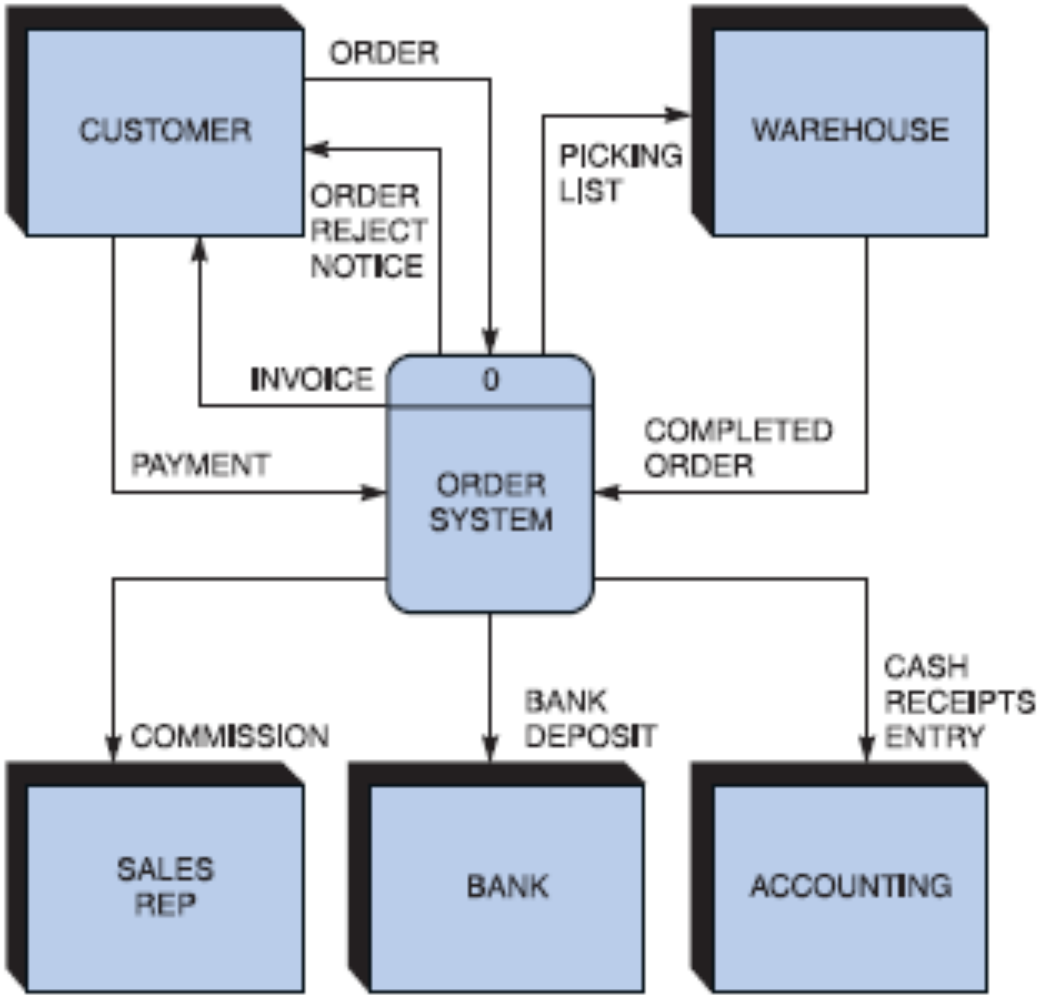


Figure from H. J. R. Gary B. Shelly, System Analysis and Design

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 Diagrams

- State machine, state, or statechart diagrams model the behaviour of the system in response to external and internal events.
- State diagrams show the system's responses to stimuli so are often used for modelling real-time systems.
- State diagrams show system states and events that cause transitions from one state to another.
- State diagrams do not show the flow of data within the system but may include additional information on the computations carried out in each state.

Example of State Diagrams

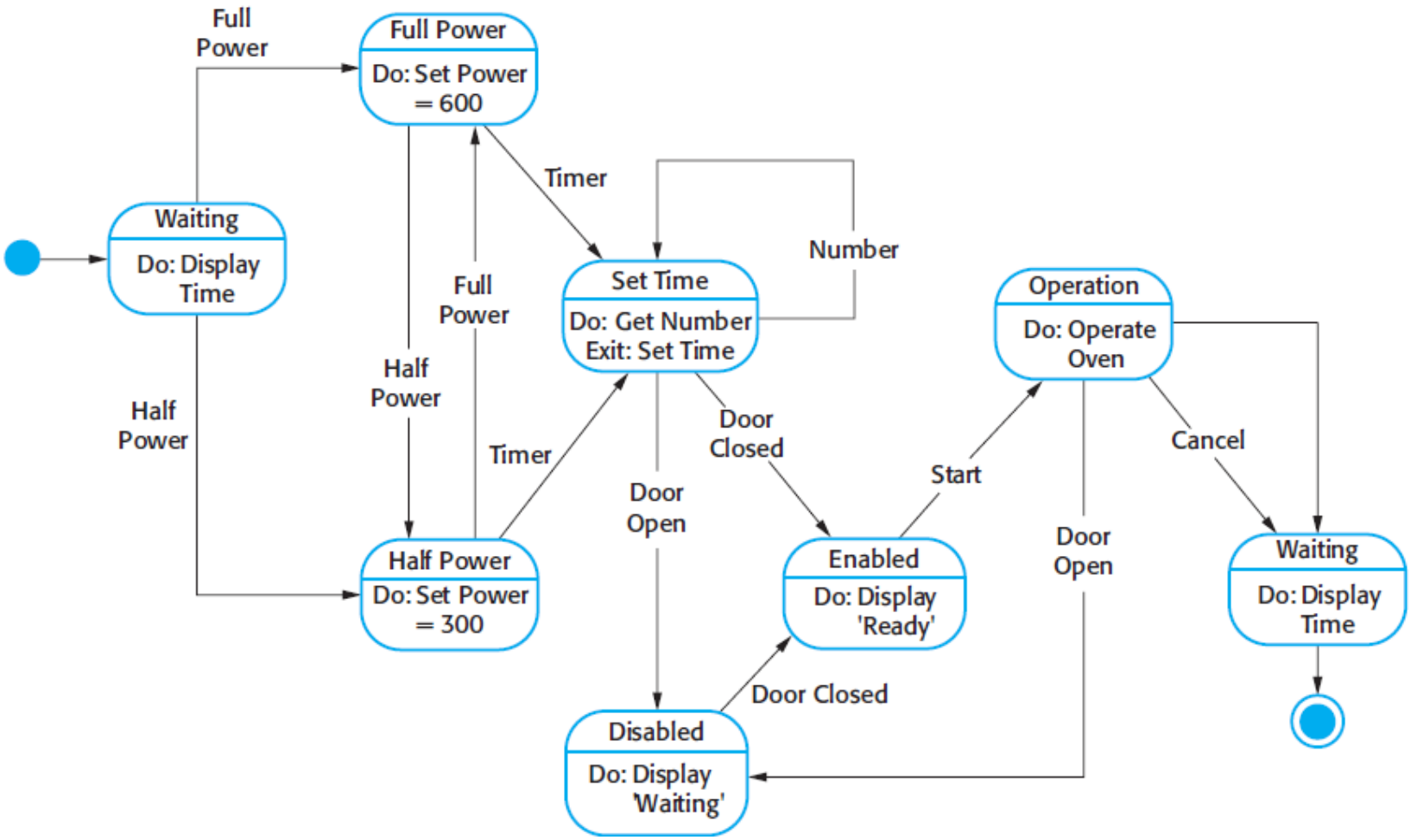


Figure from I. Sommerville, Software Engineering, 9th ed.

Example of Superstate

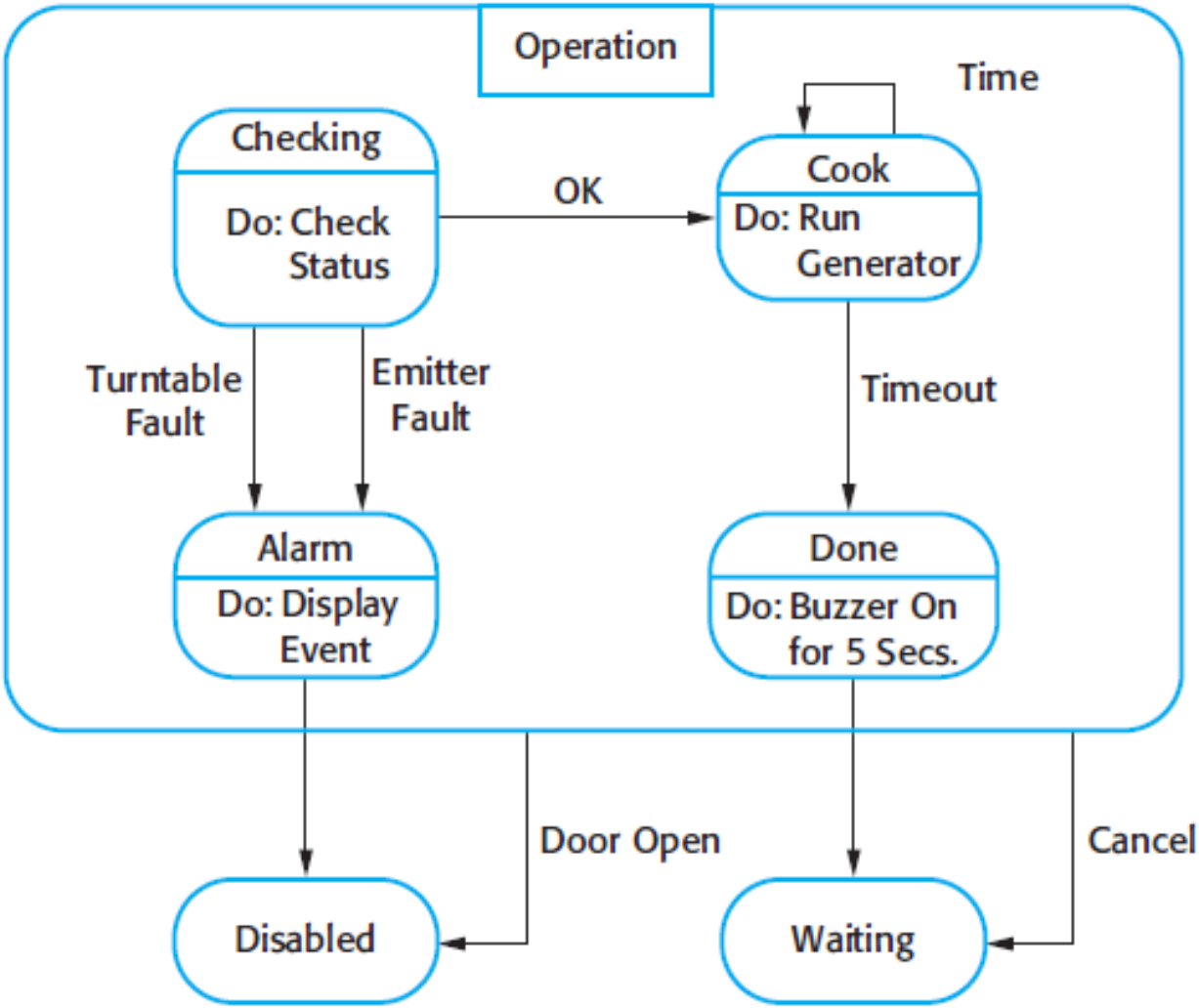


Figure from I. Sommerville, Software Engineering, 9th ed.

IEEE Software Design Document (SOD) standard 1016- 1998

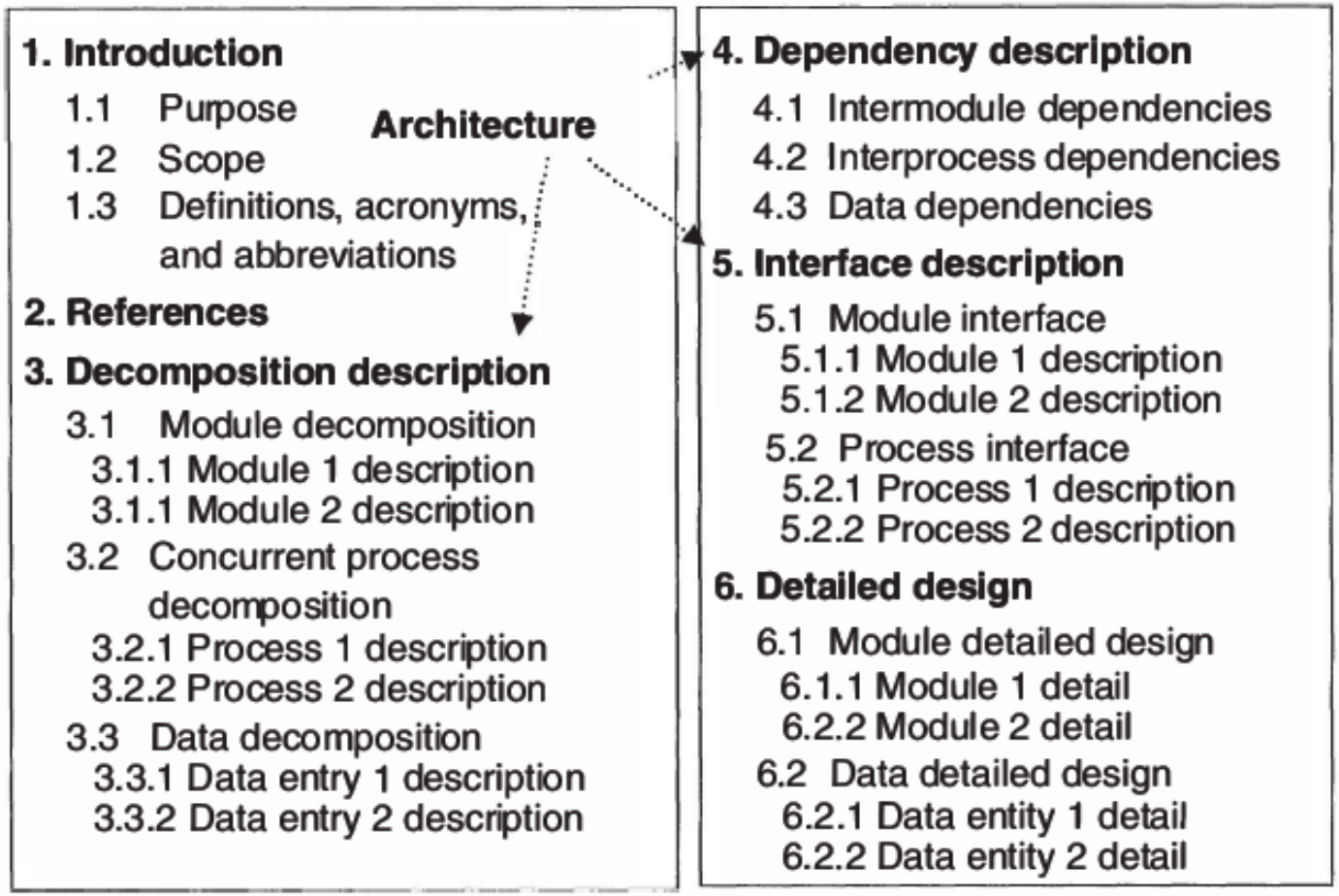


Figure from E. J. Braude and M. E. Bernstein, "Software Engineering: Modern Approaches, 2nd ed.

Model-Driven Engineering (MDE)

- Model-driven engineering (MDE) is an approach to software development where models rather than programs are the principal outputs of the development process.
- The programs that execute on a hardware/software platform are then generated automatically from the models.
- Proponents of MDE argue that this raises the level of abstraction in software engineering so that engineers no longer have to be concerned with programming language details or the specifics of execution platforms.

Usage of Model Driven Engineering

- Model-driven engineering is still at an early stage of development, and it is unclear whether or not it will have a significant effect on software engineering practice.
- Pros
  - Allows systems to be considered at higher levels of abstraction
  - Generating code automatically means that it is cheaper to adapt systems to new platforms.
- Cons
  - Models for abstraction and not necessarily right for implementation.
  - Savings from generating code may be outweighed by the costs of developing translators for new platforms.