Software Engineering

(1) Overview

Sokendai / National Institute of Informatics
Fuyuki Ishikawa / 石川　冬樹
f-ishikawa@nii.ac.jp / @fyufyu
http://research.nii.ac.jp/~f-ishikawa/

大学共同利用機関法人 情報・システム研究機構
国立情報学研究所
National Institute of Informatics

From Syllabus

- Learn software engineering techniques for efficient development and operation of large-scale and high-quality software systems
- Overview activities and techniques in each phases of development process
- Also discuss various development paradigms and the state-of-the-art topics

Evaluation

- Contributions to the lecture (40%)
- Report (60%)

Lecture Content

- Overview of Software Engineering
- Go though the development process
  - Requirements Engineering
  - System Analysis and Architecture
  - Detail Design and Reuse
  - Formal Methods
  - Testing and Debugging
  - Maintenance
  - Project Management
- Discuss different paradigms and latest research
  - Agile Software Development
  - Various Development Paradigms
  - State-of-the-art Industrial Applications and Research Topics

TOC

- Overview of Software Engineering
- Modeling and Process
- UML

Software Engineering

- 1968 at NATO Science Committee
  - Response to “software crisis”
  - Just over 50 years [http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF]

- Definition in SWEBOK (to be detailed later)

> *the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software*

Causes of Difficulties

- Famous phrase : “No Silver Bullet”

[F. P. Brooks, Jr., No Silver Bullet – Essence and Accident in Software Engineering, 1987]

  - Complexity : has essential complexity, where details are not trivial, that increases non-linearly over size
  - Conformity : required to conform a variety of principles, every time, such as human perception and interfaces
  - Changeability : required to deal with various changes of requirements and environments
  - Invisibility : difficult to effectively visualize with interleaving dependencies

Other Points (vs. Programming Exercise)

- Develop what other people want
  - Example : lecture management system in Sokendai

    What mutual understanding and decision making were done by people who ordered and who constructed??
- Develop a large product by many people, possibly crossing multiple organizations
  - Maybe no “genius”, people changing
- Develop with an agreement or contract
  - Budget, delivery dateline, scope of responsibility

SWEBOK

- Software Engineering Body Of Knowledge (V 3.0, 2014, IEEE)

[https://www.computer.org/education/bodies-of-knowledge/software-engineering]

1. Software Requirements
2. Software Design
3. Software Construction
4. Software Testing
5. Software Maintenance
6. Software Configuration Management
7. Software Engineering Management
8. Software Engineering Process
9. Software Engineering Models and Methods
10. Software Quality
11. Software Engineering Professional Practice
12. Software Engineering Economics
13. Computing Foundations
14. Mathematical Foundations
15. Engineering Foundations

Examples inside Software Engineering

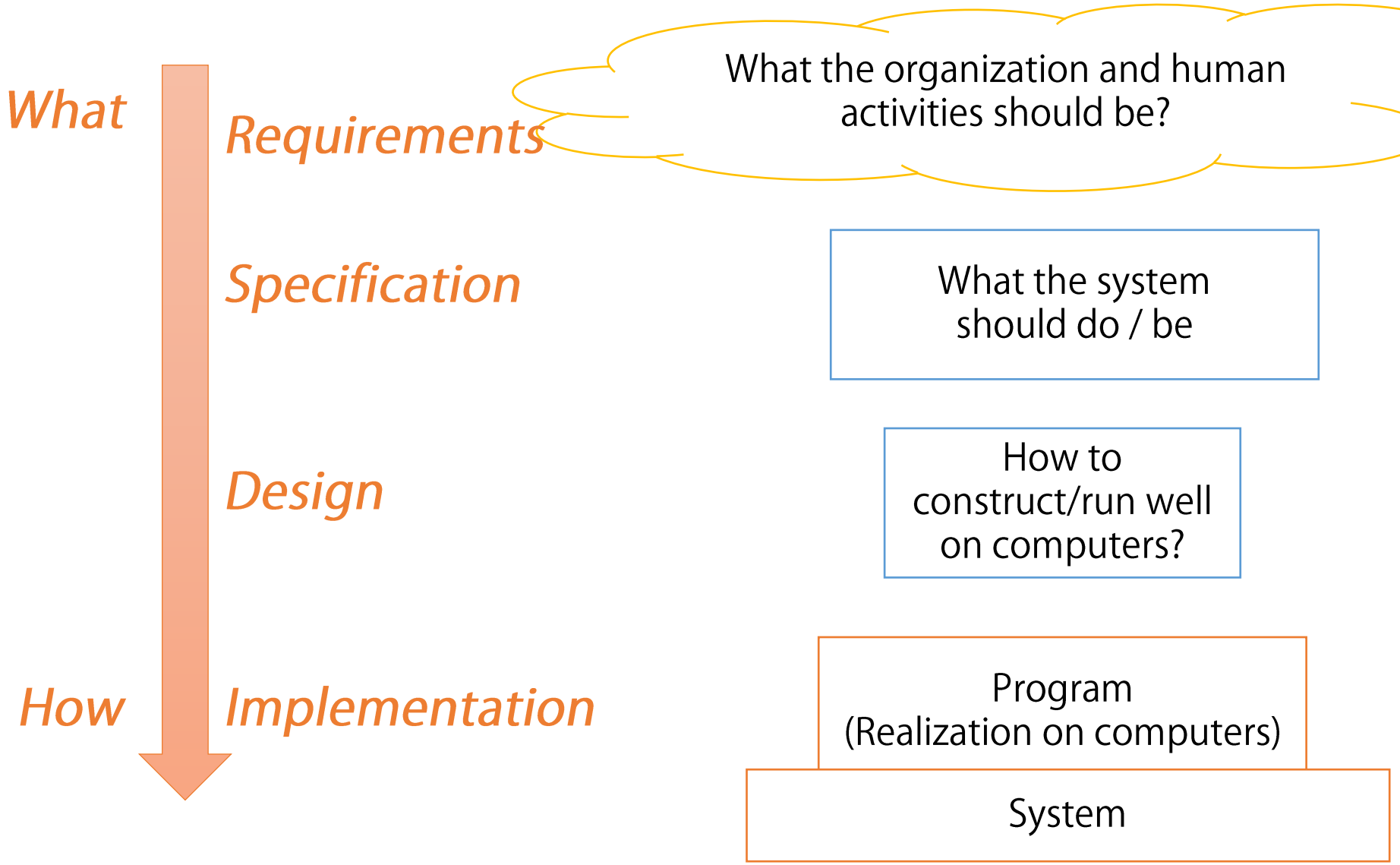
(Not only program related issues)

- Elicit “what should be done”
  - Techniques : interviews, ethnography
- Estimate how much cost will be required
  - Techniques : measurement of complexity, statistics
- Communicate for effective collaboration
  - Techniques : morning meeting, Kanban

TOC

- Overview of Software Engineering
- **Modeling and Process**
- UML

Abstraction Level in Software Development



Model

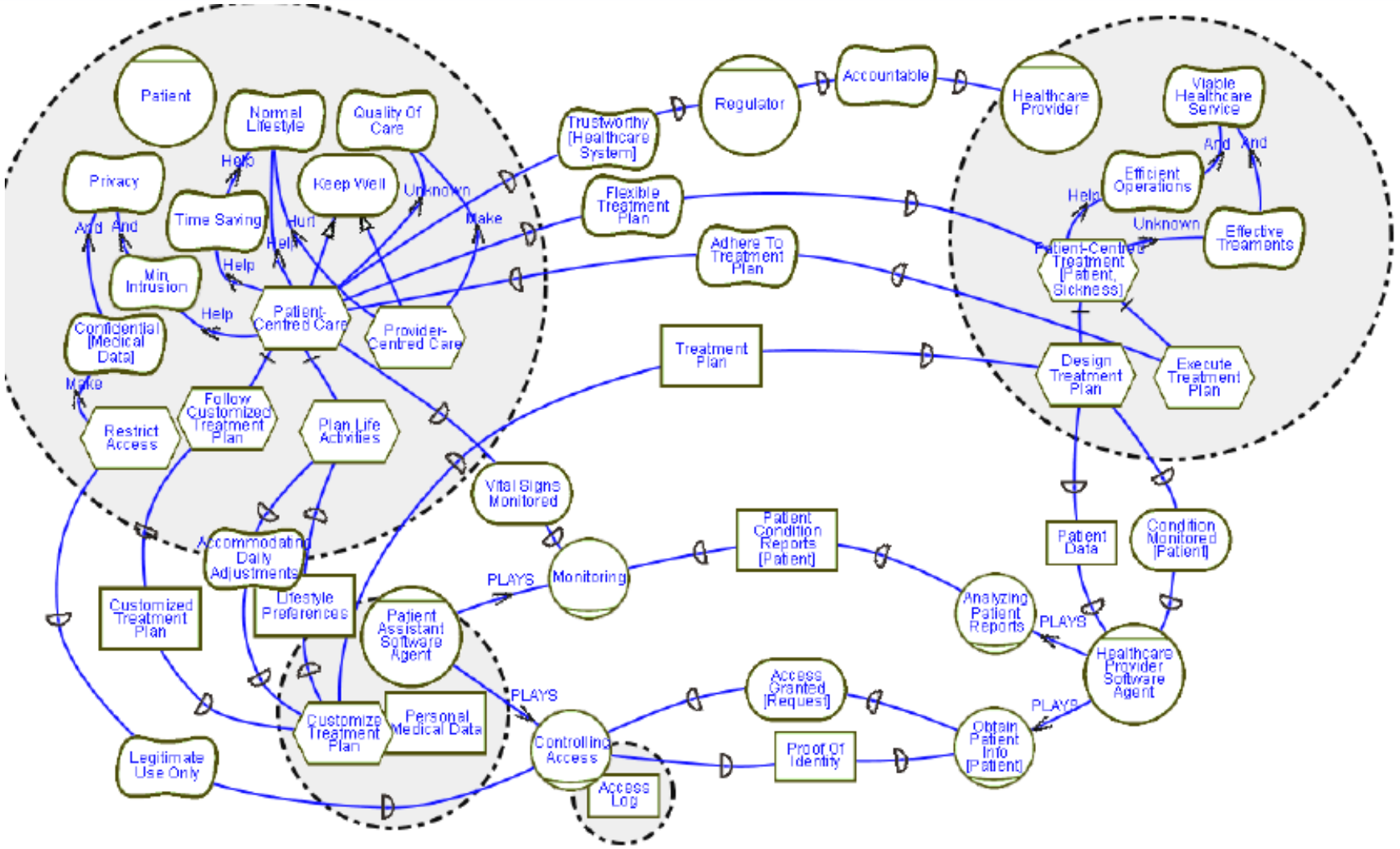
- What is a “model”?

> A simplified description, especially a mathematical one, of a system or process, to assist calculations and predictions
>
> [Oxford English Dictionary]

  - Abstraction and simplification by focusing on specific aspects (abstracting away unnecessary details)
  - Efficient, effective analysis and verification

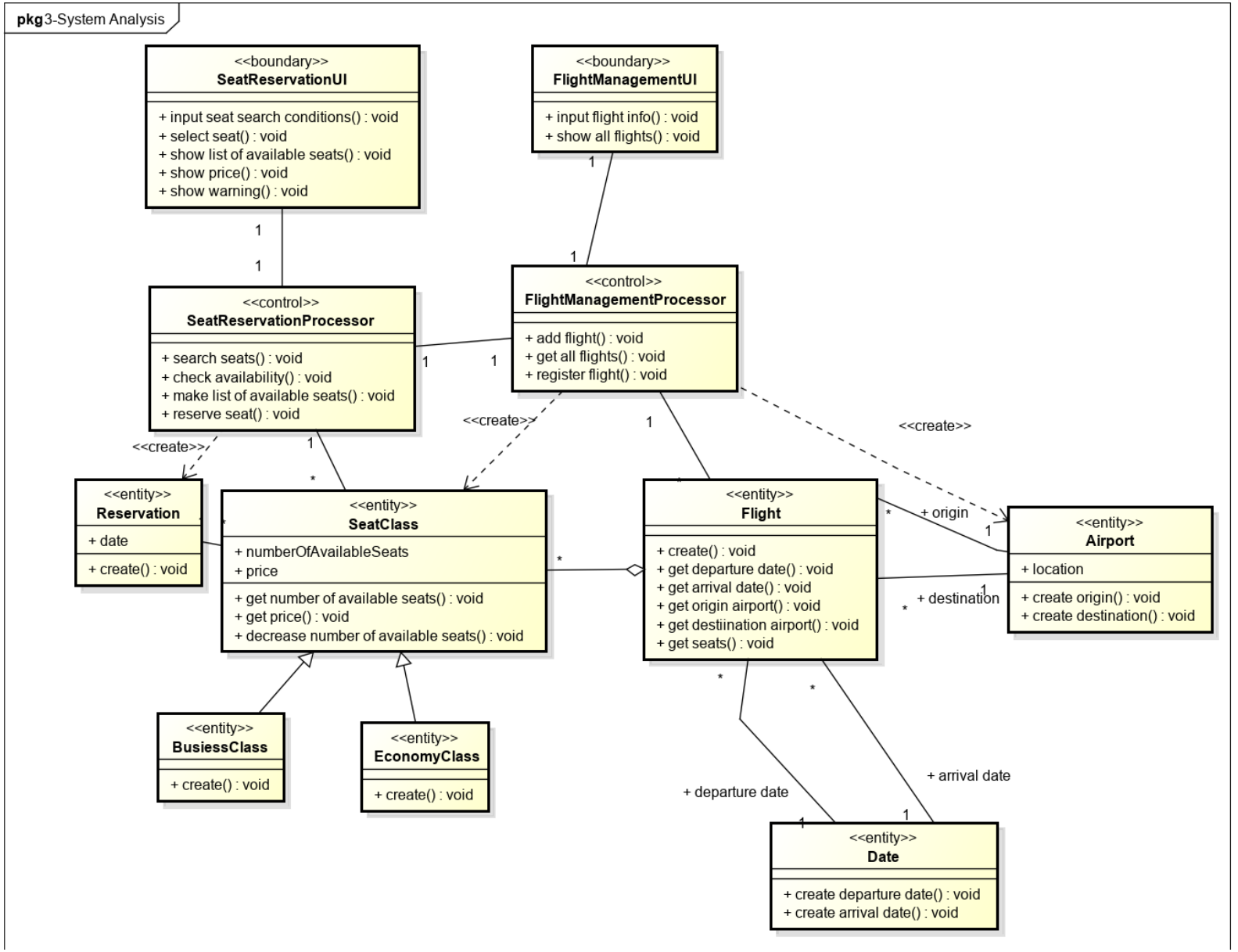
Essential to capture systems to be developed, existing systems, and development activities

Example of Models: Stakeholders and Goals

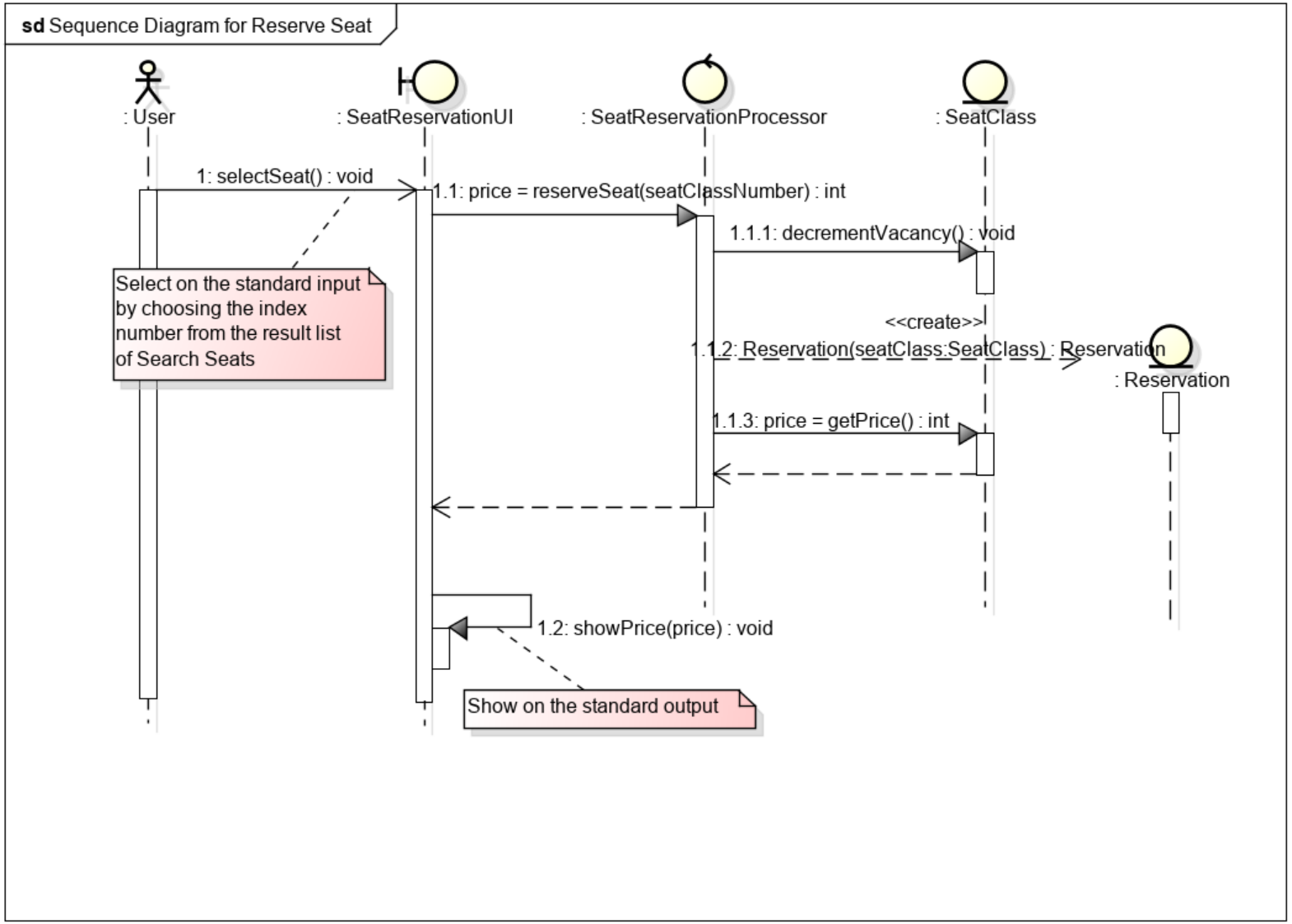


[cited from E. Yu, Social Modeling and i*]

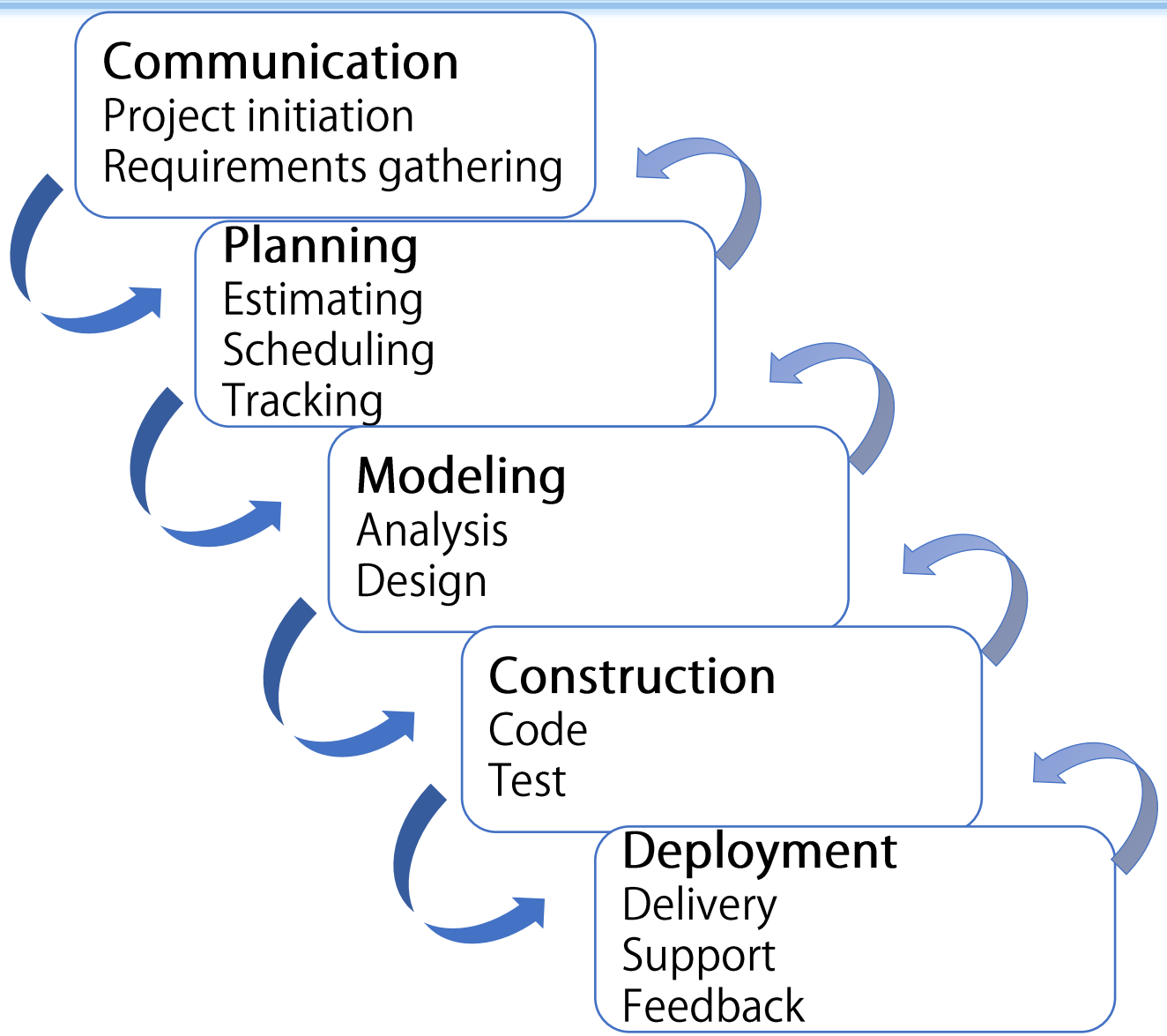
Example of Models: System Architecture



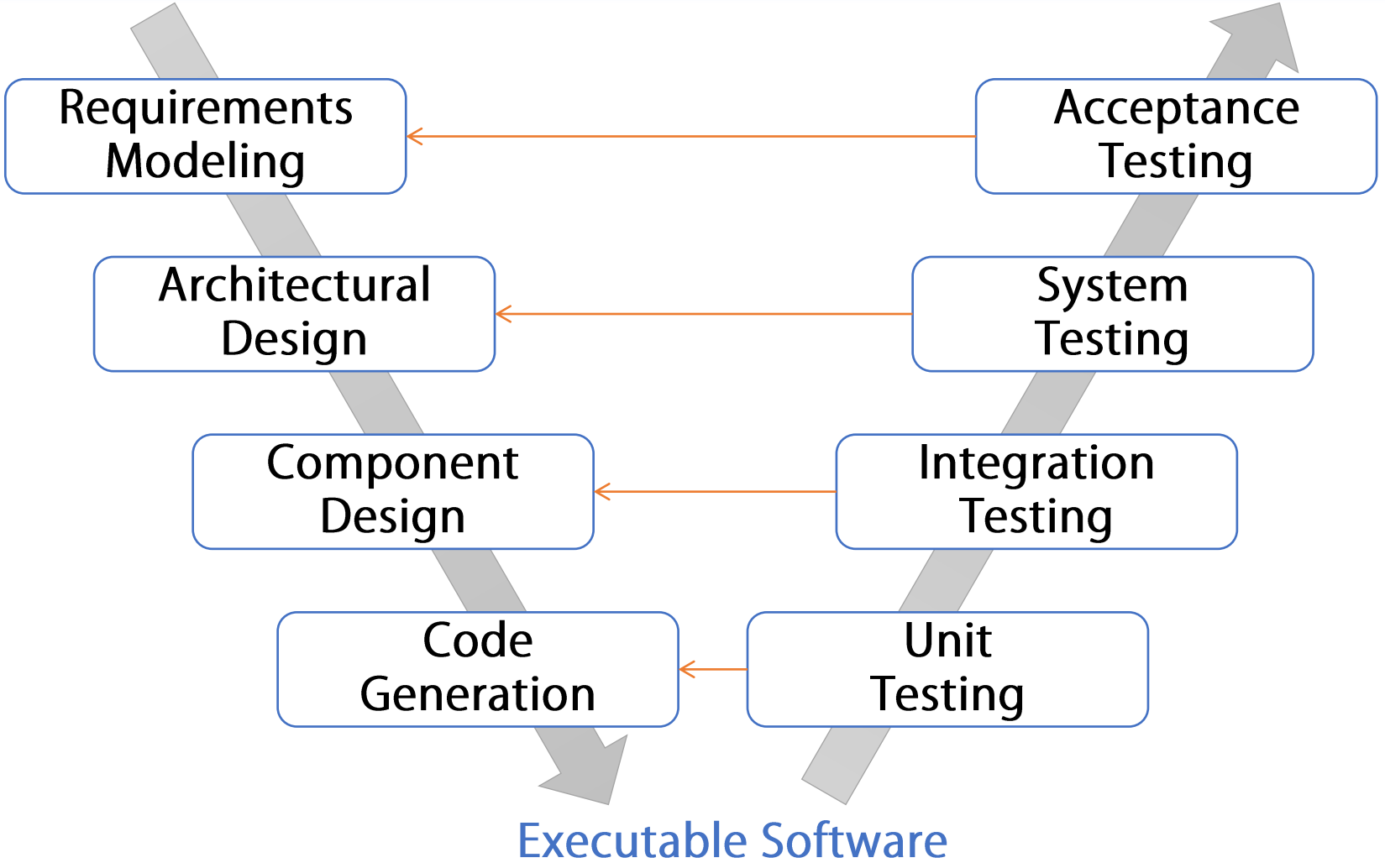
Example of Models: Behavior and Interaction



Example of Models: Process (Waterfall Model)



Example of Models: Process (V-Model)

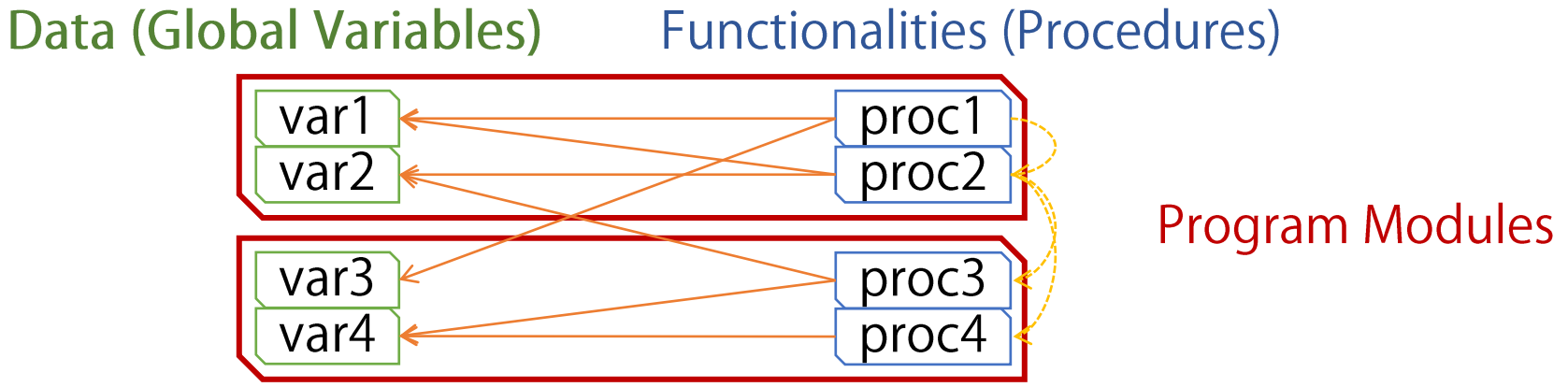


TOC

- Overview of Software Engineering
- Modeling and Process
- UML

Before OO (1)

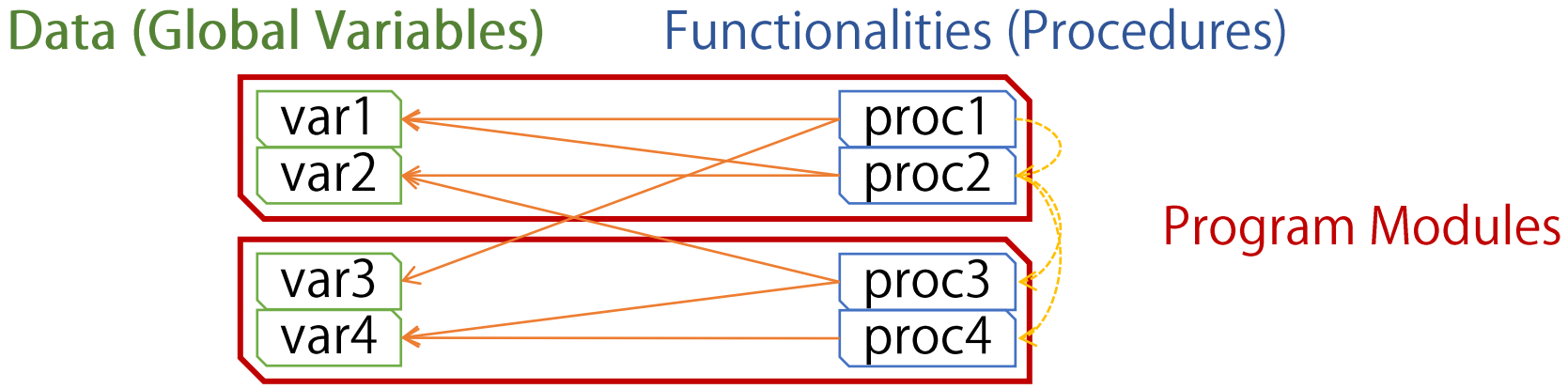
- Structured Programming (Procedure-Oriented)



  - Dependencies through global variables
  - Difficulties in preventing destructive modification of variable values, or unexpected side effects

Before OO (2)

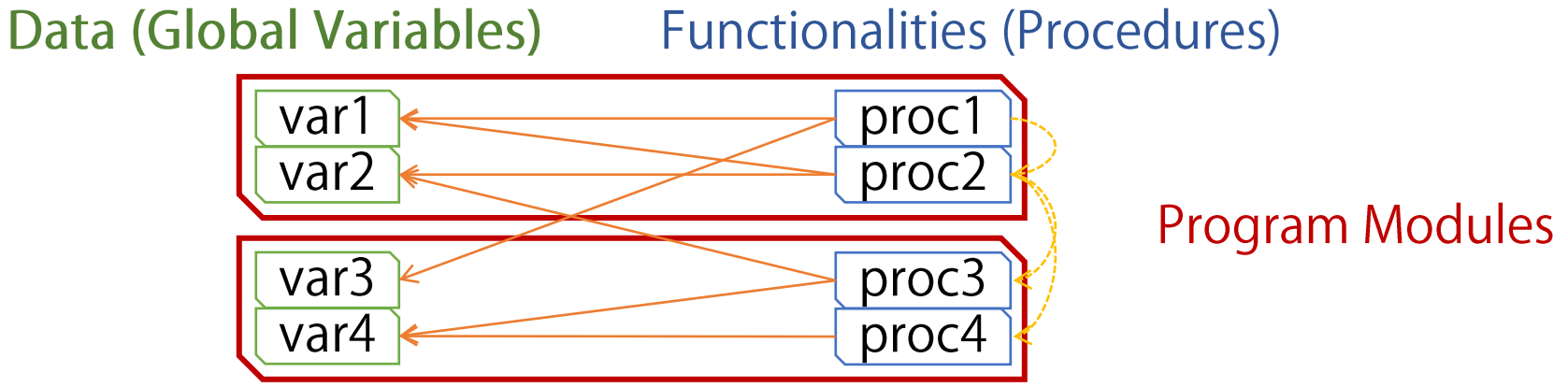
- Structured Programming (Procedure-Oriented)



  - Decomposition of the application functionality into procedures
  - Procedures understandable and meaningful only from the global viewpoint of the application, not reusable in a more general context

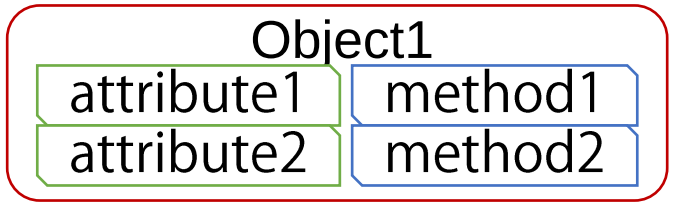
Before OO (3)

- Structured Programming (Procedure-Oriented)



  - Implicit interdependencies among data and functionalities
  - Side effects of modification to other parts
  - Difficulties in reusing meaningful combinations of data and functionalities, especially extending them

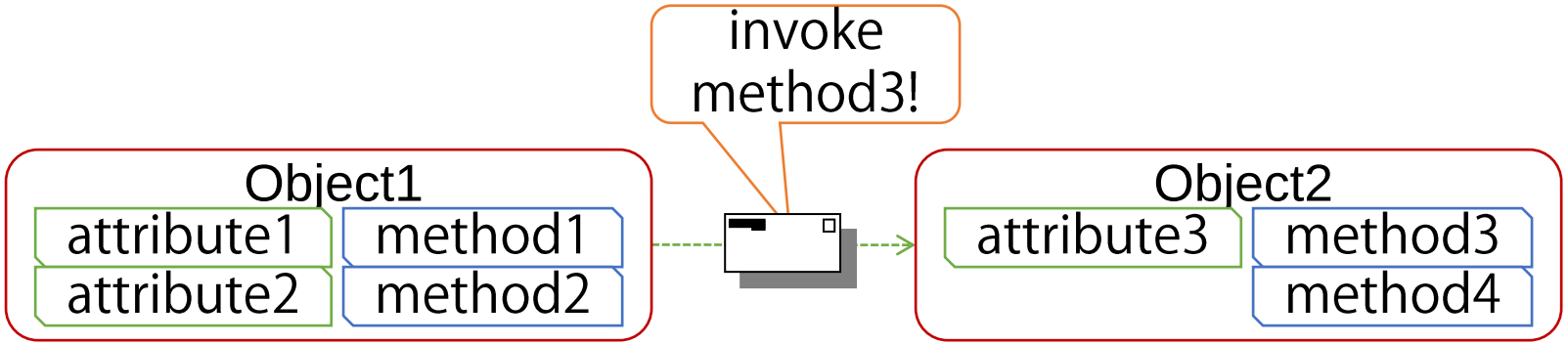
OO Approach : Object (1)



- Combine related data and functionalities as a unit of modularization
- ➡ Explicitly model the relationships among data and functionalities
- Move from functionality-oriented to object-oriented
- ➡ Modularization dependent on not the application but the natural concepts in the real world

OO Approach : Object (2)

- Message passing or method invocation
  - Objects interact with each other by sending messages (requests)
  - The result can differ depending on the state of the receiving object, even if the same messages are sent



OO Approach : Object (3)

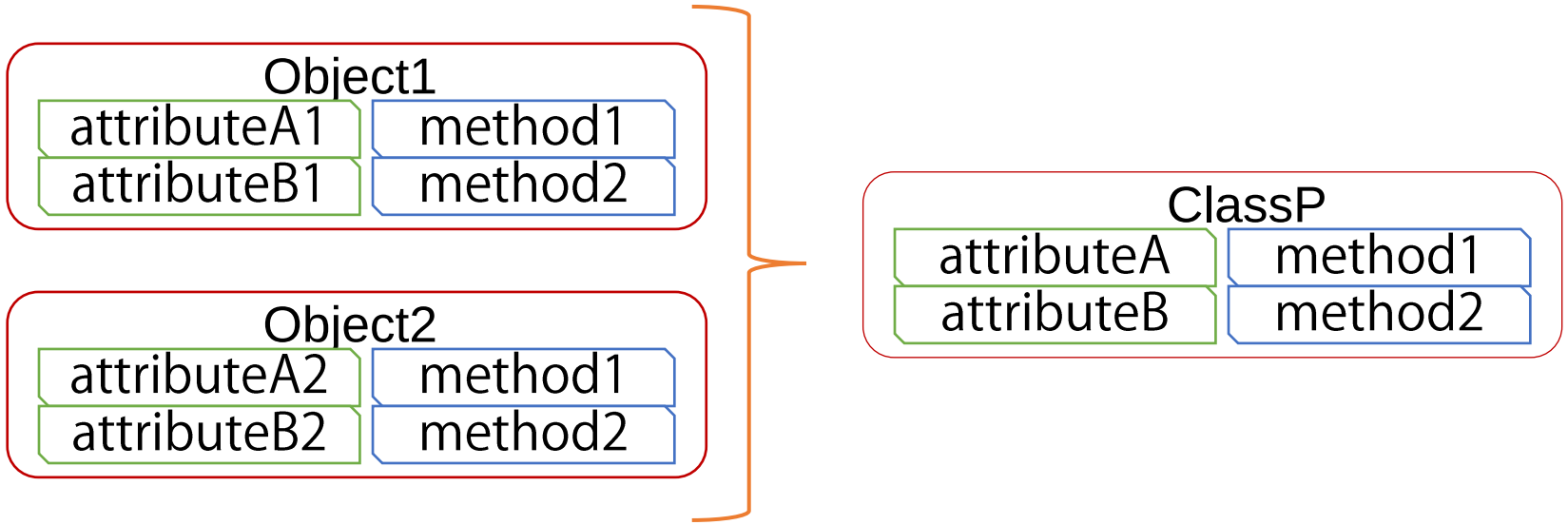
- Encapsulation

  As objects interact with each other only by message passing,
  not by manipulating data directly,

  - The requester object does not need to know inside of the provider object but need only to know the interface
  - The provider object can change its inside without affecting its requester objects
  - The provider object can be sure its data can be modified only in a way it defines in the object

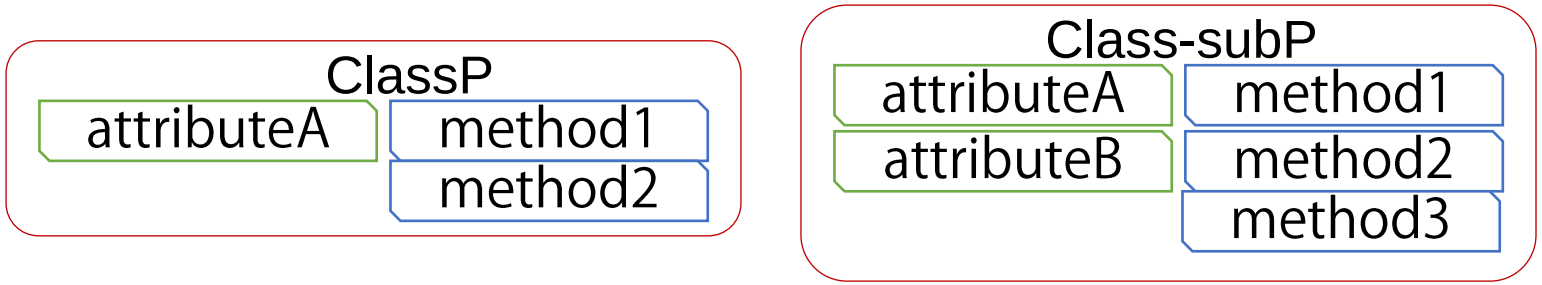
OO Approach: Class (1)

- Class: abstract characteristics that define a thing (object), including its attributes and methods
  - Provide a notion to group objects that have same types of attributes, and same methods



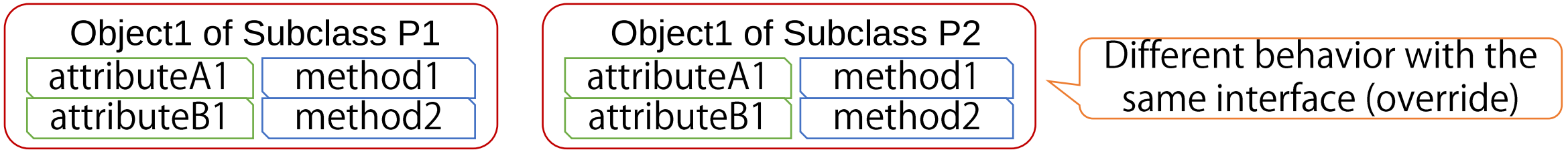
OO Approach: Class (2)

- Inheritance
  - A class (subclass) can inherit attributes and methods from another (superclass), and have more:
  - Reusability is improved based on specialization relationships: creating a new subclass, or generalization relationships: creating a new superclass
  - Common parts and specialized parts are separated clearly



OO Approach: Class (3)

- Polymorphism
  - Different functionalities can be obtained by invoking a method of the same name, appropriately according to the context
  - A subclass can define a method of the same name as in the superclass, and give it a different functionality (override)
  - A class can define multiple methods of the same name with different arguments (overload)



Object-Oriented XXX

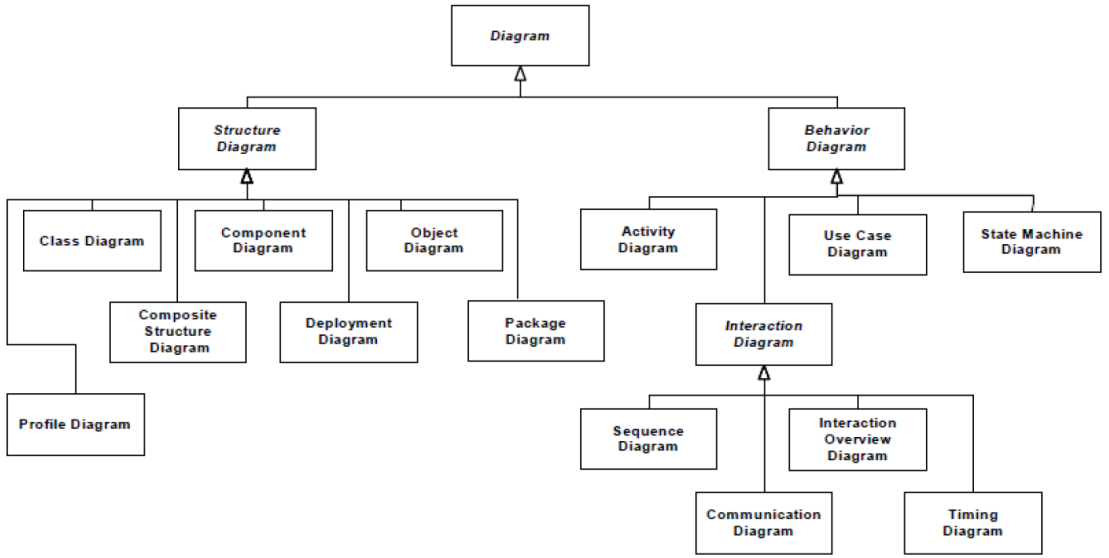
- Objected-Orientation is not limited to Programming
  - Object-oriented model-centric software development
  - Object-oriented analysis and design process
  - Object-oriented domain analysis
  - …

UML

UML: Unified Modeling Language

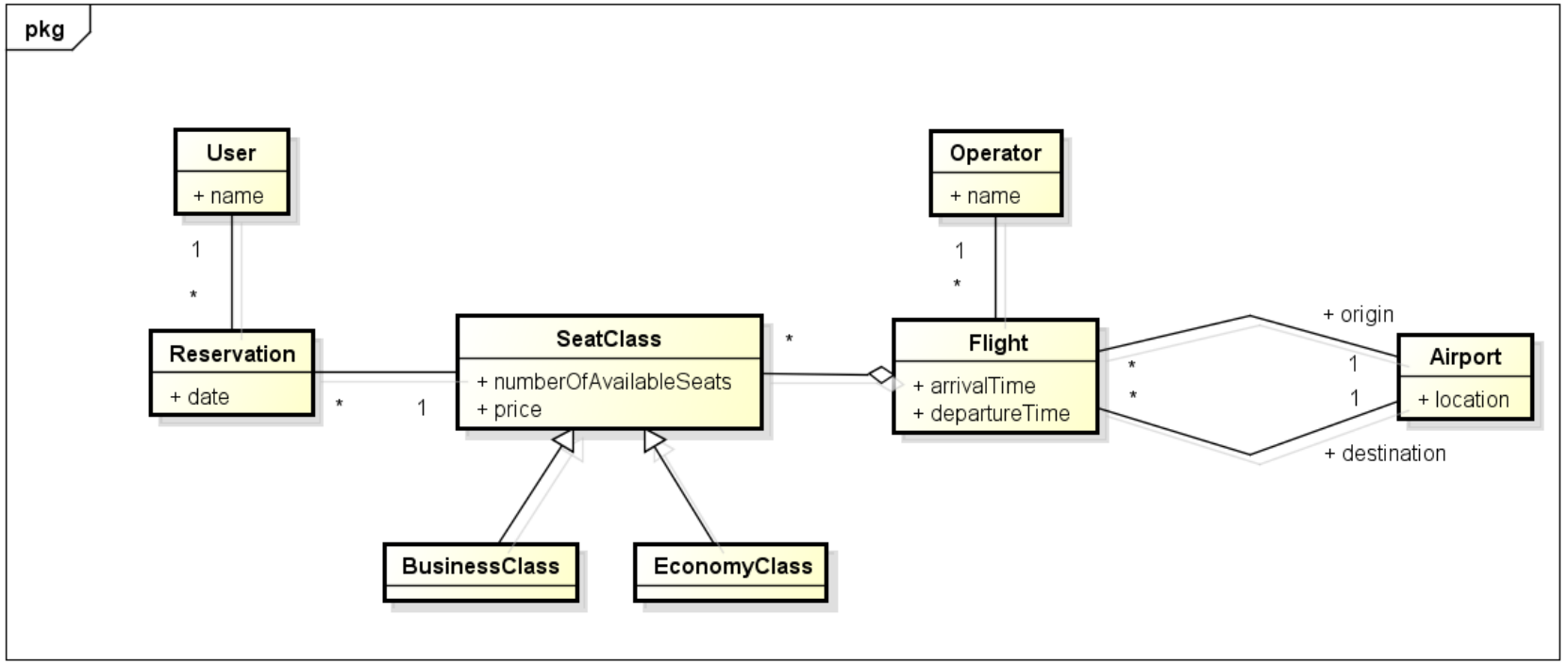
[https://www.omg.org/spec/UML/]

- Object-oriented modeling language (diagrams)
- 14 diagrams with different roles
- Standard by OMG (Object Management Group)
  (Ver. 1.0 in 1996, Ver. 2.5.1 in 2017)



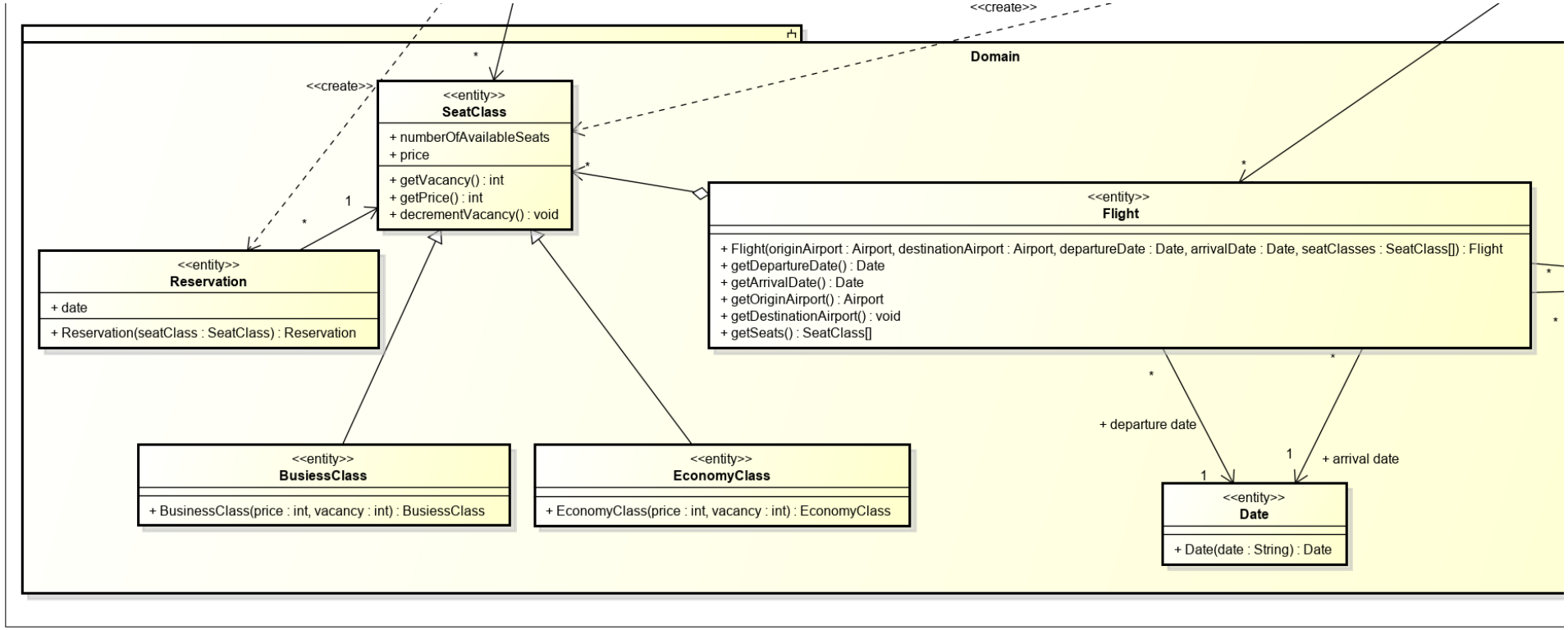
Example: Class Diagram (Conceptual Analysis)

- Concepts in flight reservation



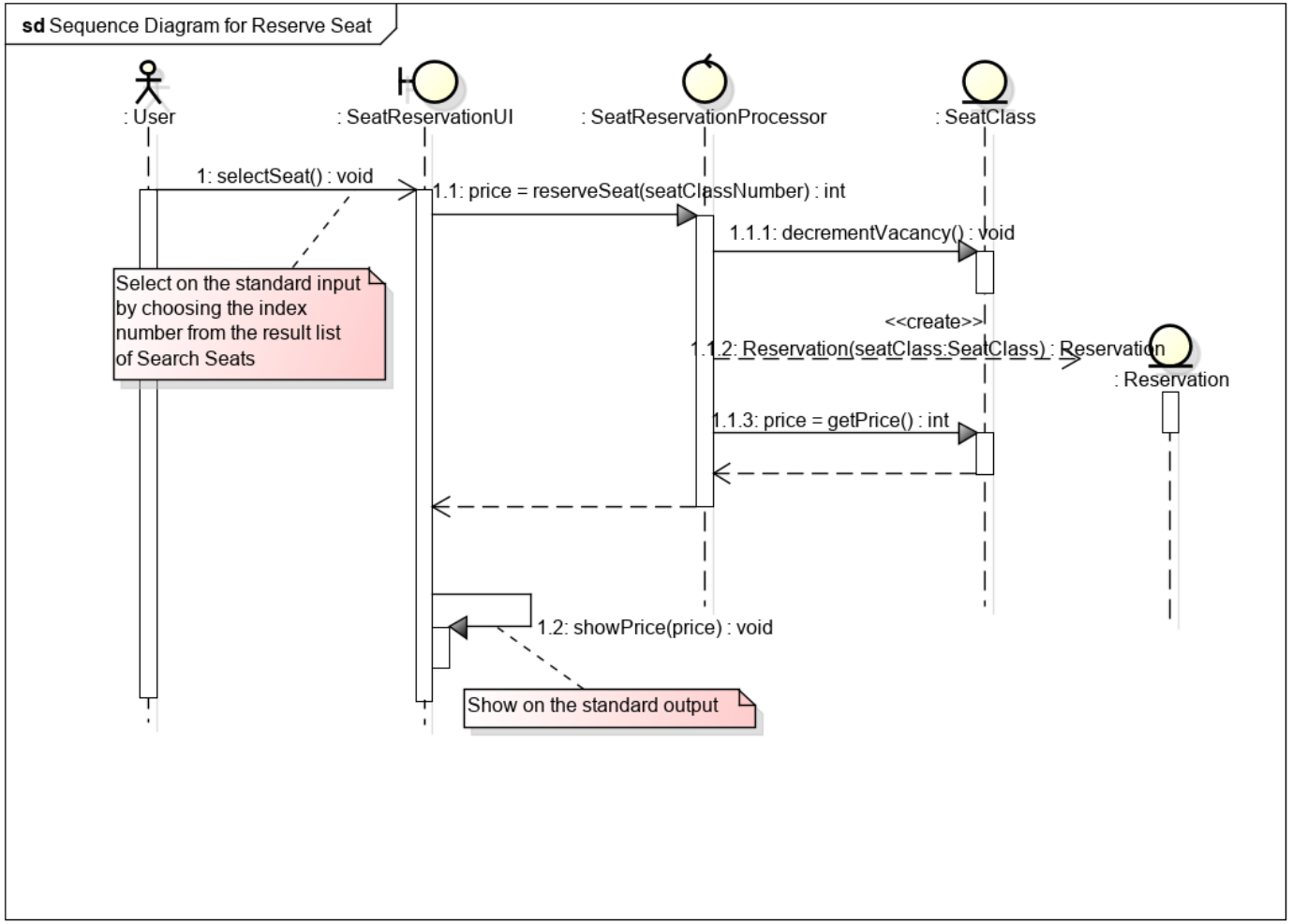
Example: Class Diagram (Program Design)

- Design of a flight reservation system



Example: Sequence Diagram

- Interaction for flight reservation



Summary

- Software engineering
  - Deals with everything about software development, operation, and maintenance
- Models
  - Essential to capture, share, analyze, and evaluate abstract (intermediate) deliverables, complex systems, or development activities
  - Effectively support the process from abstract goals and problems to software-based solutions