Software Design and Architecture

System Decomposition

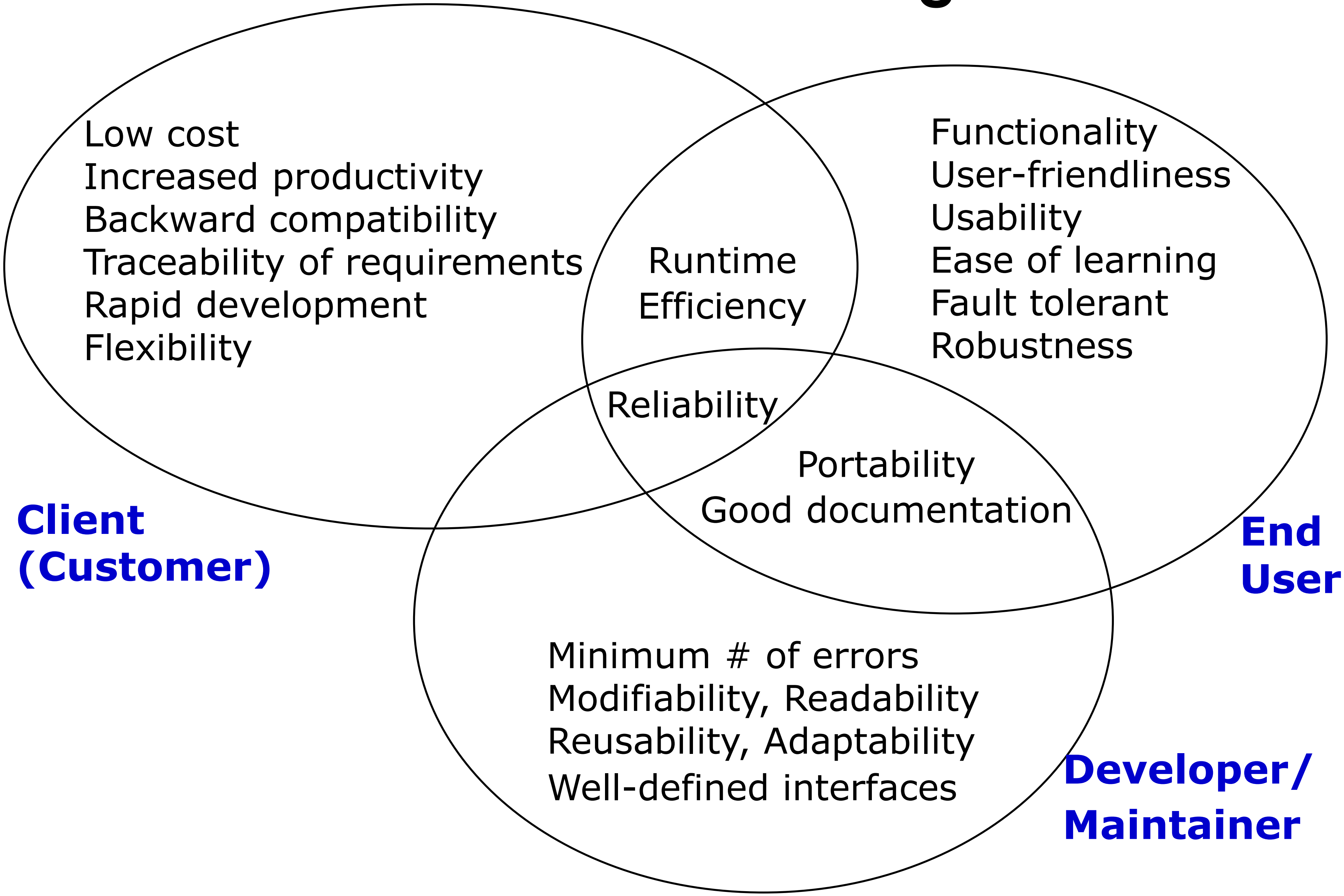
Soha Makady

Example of Design Goals

- Reliability
- Modifiability
- Maintainability
- Understandability
- Adaptability
- Reusability
- Efficiency
- Portability
- Traceability of requirements
- Fault tolerance
- Backward-compatibility
- Cost-effectiveness
- Robustness
- High-performance

- Good documentation
- Well-defined interfaces
- User-friendliness
- Reuse of components
- Rapid development
- Minimum number of errors
- Readability
- Ease of learning
- Ease of remembering
- Ease of use
- Increased productivity
- Low-cost
- Flexibility

Stakeholders have different Design Goals



Low cost
Increased productivity
Backward compatibility
Traceability of requirements
Rapid development
Flexibility

Runtime
Efficiency

Functionality
User-friendliness
Usability
Ease of learning
Fault tolerant
Robustness

Reliability

Portability
Good documentation

Client (Customer)

End User

Minimum # of errors
Modifiability, Readability
Reusability, Adaptability
Well-defined interfaces

Developer/ Maintainer

Typical Design Trade-offs

- Functionality v. Usability
- Cost v. Robustness
- Efficiency v. Portability
- Rapid development v. Functionality

System Design Concepts

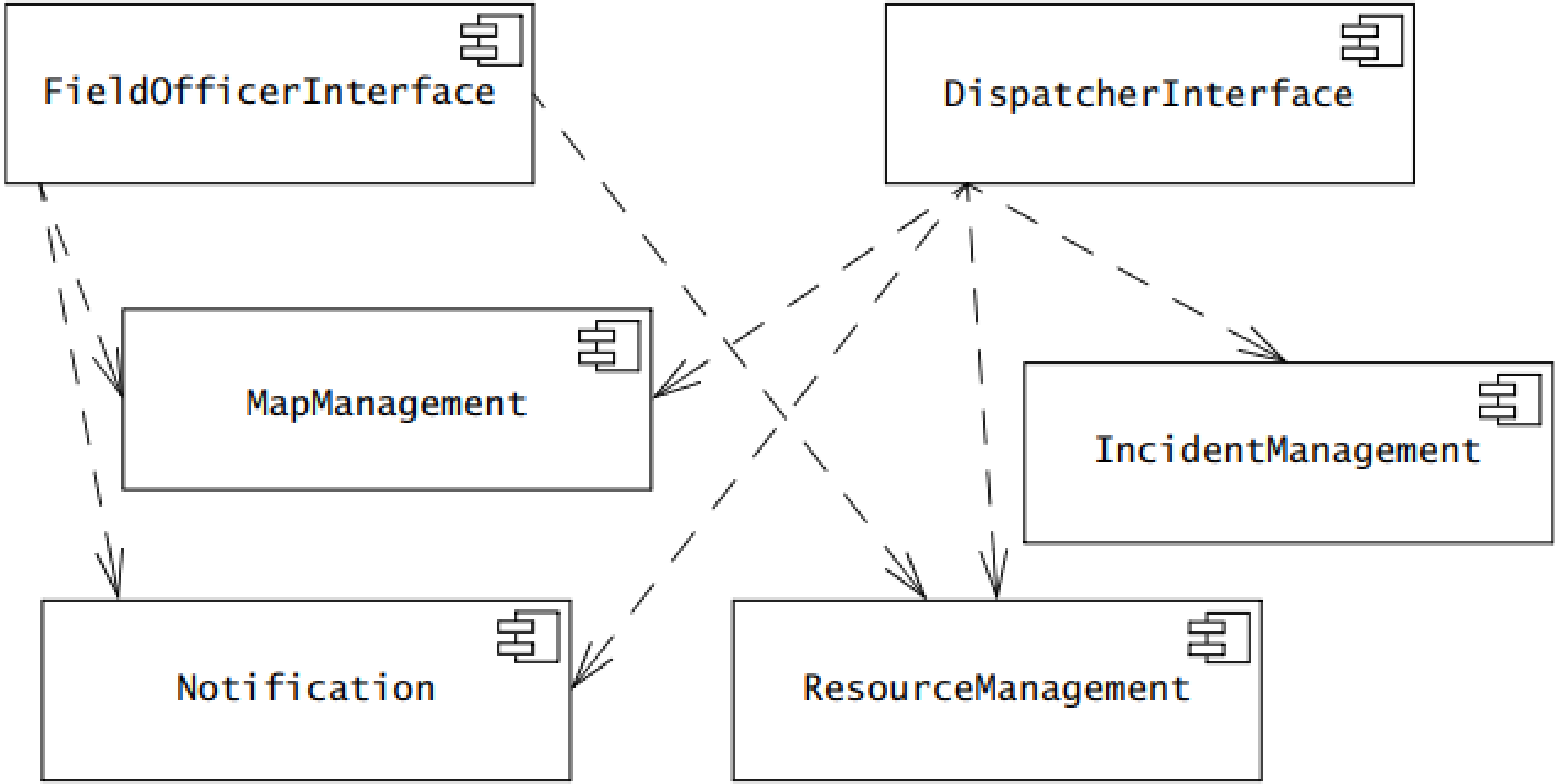
- Subsystem decomposition
- Services and subsystem interfaces
- Coupling and Coherence

Services and subsystem interfaces

- Subsystem
  - Subsystems provide services to other subsystems
  - The objects and classes from the object model are the “seeds” for the subsystems
  - In UML subsystems are modeled as packages
- Service
  - A set of related operations that share a common purpose
  - The origin (“seed”) for services are the use cases from the functional model

Services and subsystem interfaces

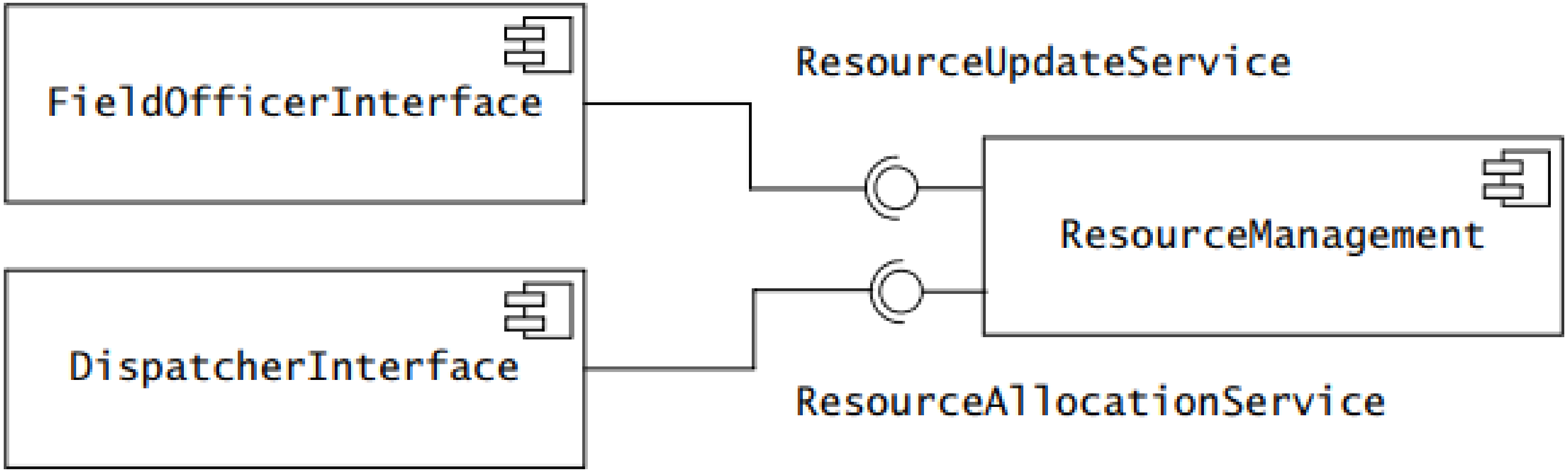
- During the early stages of system design, we do not know the exact services provided by each subsystem.
- In such cases, we use the dependency notation.



Services and subsystem interfaces

- Subsystem services: During **system design**, we define the subsystem in terms of the services it provides.
- A subsystem consists of a collection of classes, associations, operations, events and constraints that are closely interrelated with each other
- Subsystem interface: **During the object design**, we define the subsystem interface in terms of the operations it provides.

Services and subsystem interfaces Provided and Required Interfaces in UML

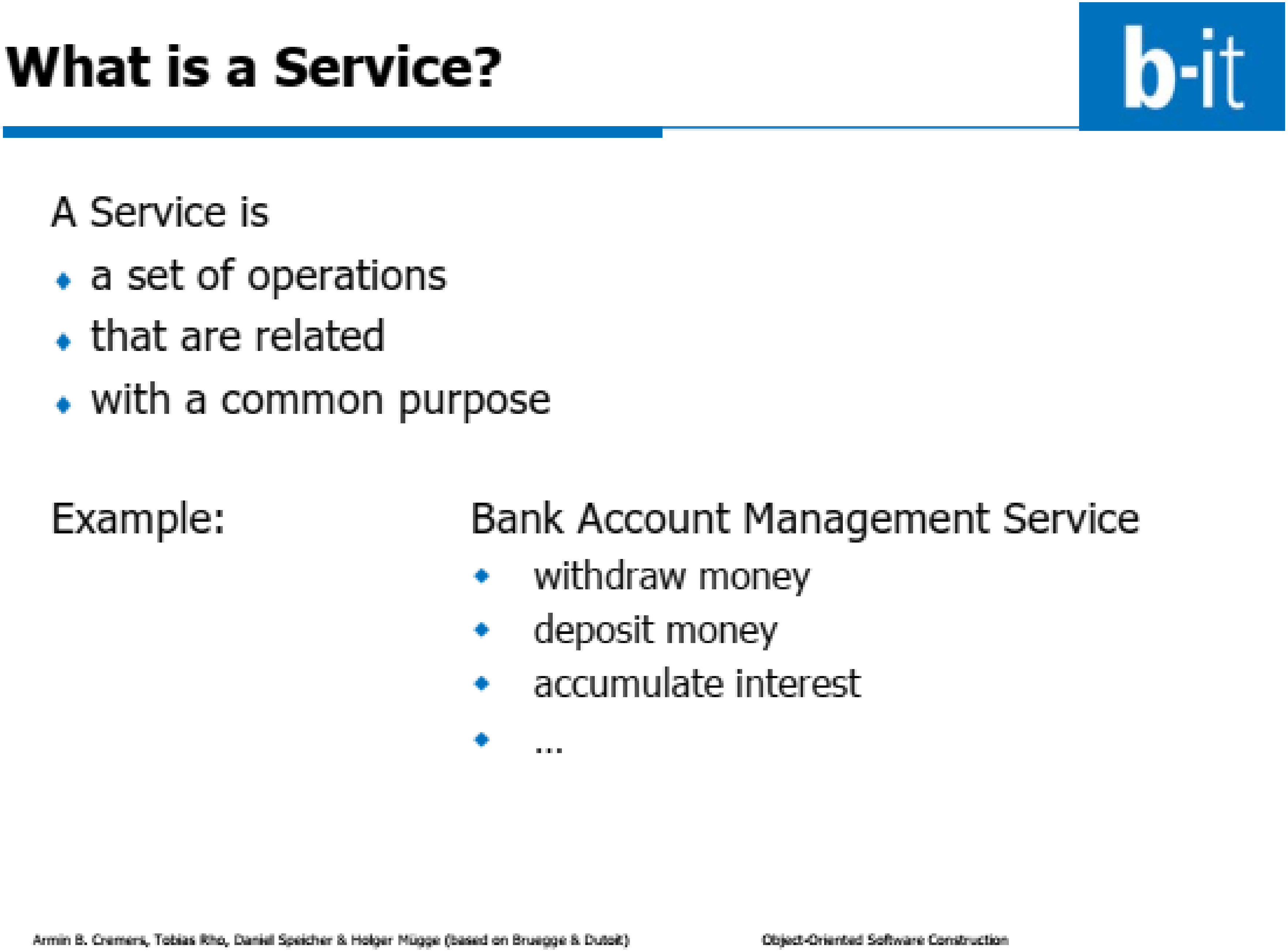


Services and subsystem interfaces

- Application programmer's interface (API)
  - The API is the specification of the subsystem interface in a specific programming language
  - APIs are defined during implementation
- The terms subsystem interface and API are often confused with each other
  - *The term API should not be used during system design and object design, but only during implementation.*

Example: Notification subsystem

- Service provided by Notification Subsystem
    - LookupChannel()
    - SubscribeToChannel()
    - SendNotice()
    - UnscubscribeFromChannel()
- Subsystem Interface of Notification Subsystem
    - Set of fully typed UML operations
- API of Notification Subsystem
    - Implementation in Java



What is a Service?

A Service is

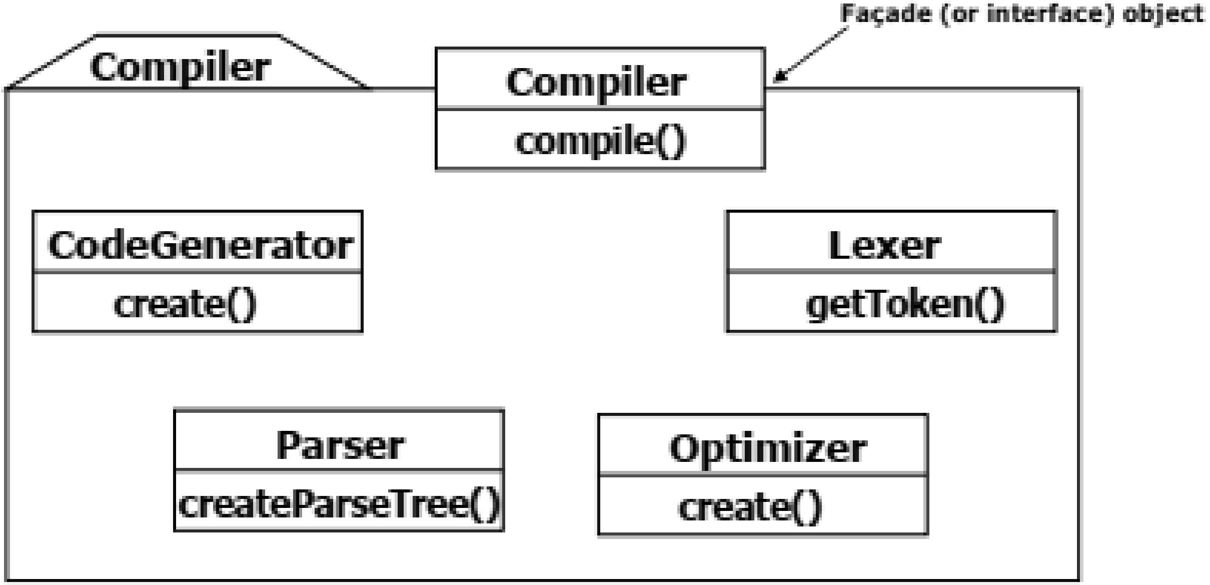
- a set of operations
- that are related
- with a common purpose

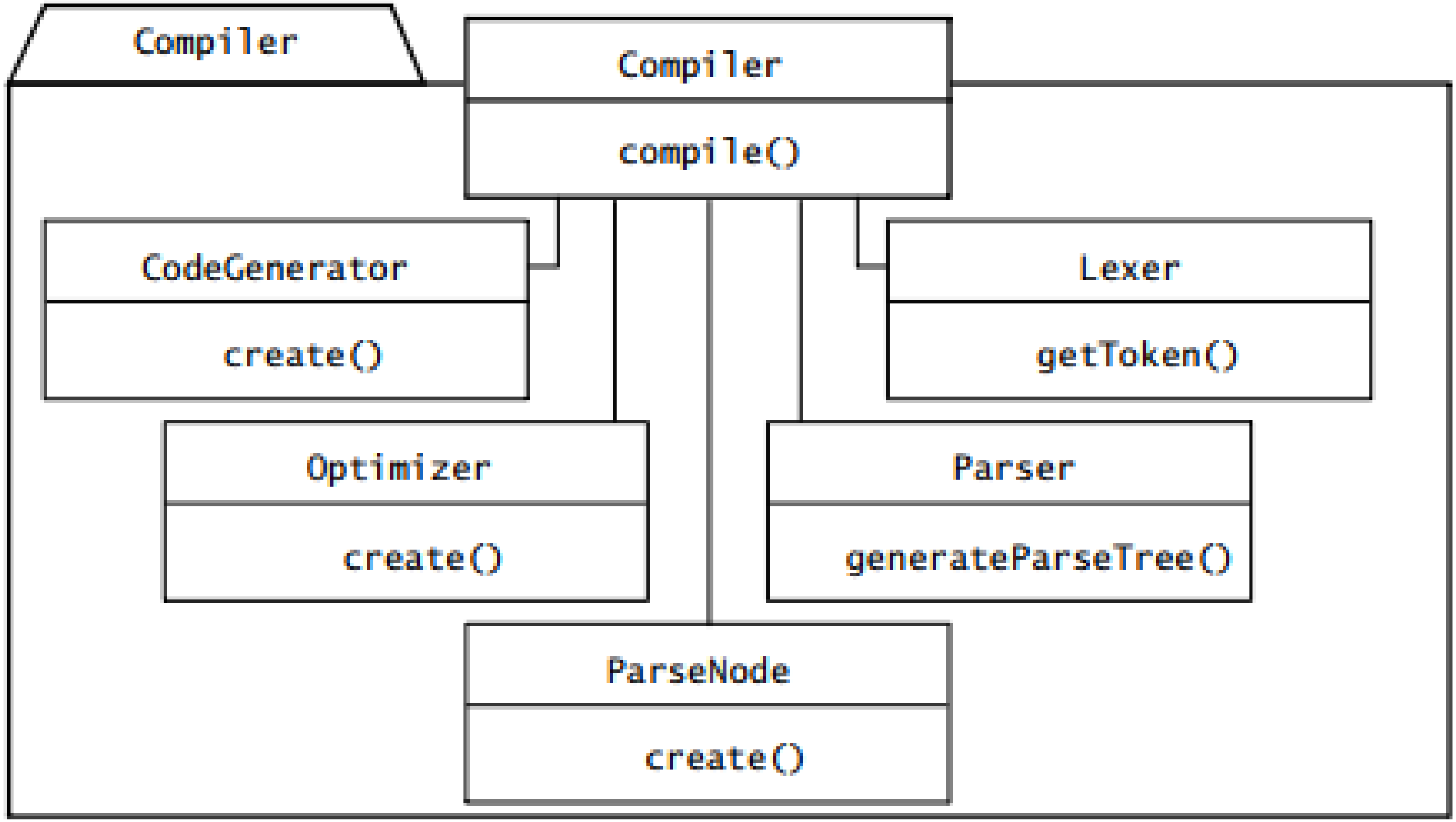
Example: Bank Account Management Service

- withdraw money
- deposit money
- accumulate interest
- …

Definition: Subsystem Interface Object

- A Subsystem Interface Object provides a service
  - This is the set of public methods provided by the subsystem
  - The Subsystem interface describes all the methods of the subsystem interface object
- Use a Facade pattern for the subsystem interface object




Compiler
Compiler
compile()
CodeGenerator
create()
Lexer
getToken()
Optimizer
create()
Parser
generateParseTree()
ParseNode
create()

Subsystem Interface Object

- Good design: The subsystem interface object describes *all* the services of the subsystem interface

- Subsystem Interface Object
    - The set of public operations provided by a subsystem

Subsystem Interface Objects can be realized with the Façade pattern (=> lecture on design patterns).

Coupling and Coherence of Subsystems

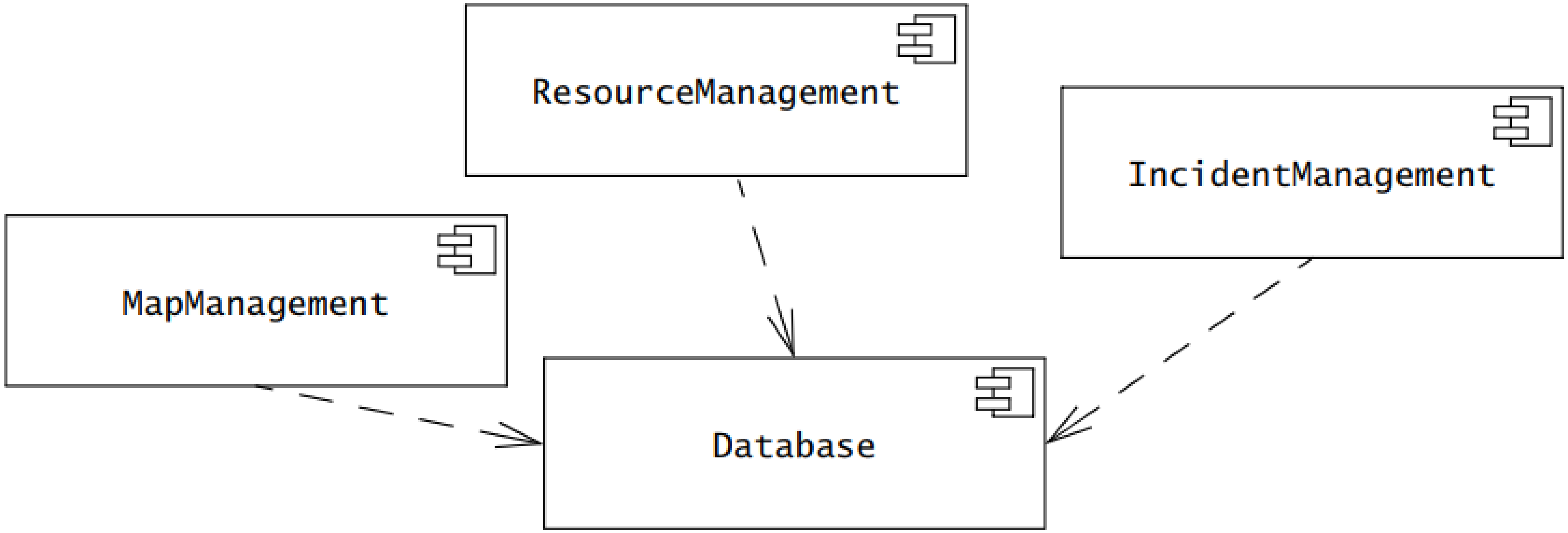
- Goal: Reduce system complexity while allowing change
- Coherence measures dependency among classes
  - High coherence: The classes in the subsystem perform similar tasks and are related to each other via many associations
  - Low coherence: Lots of miscellaneous and auxiliary classes, almost no associations
- Coupling measures dependency among subsystems
  - High coupling: Changes to one subsystem will have high impact on the other subsystem
  - Low coupling: A change in one subsystem does not affect any other subsystem.

Coupling and Coherence of Subsystems

Good Design

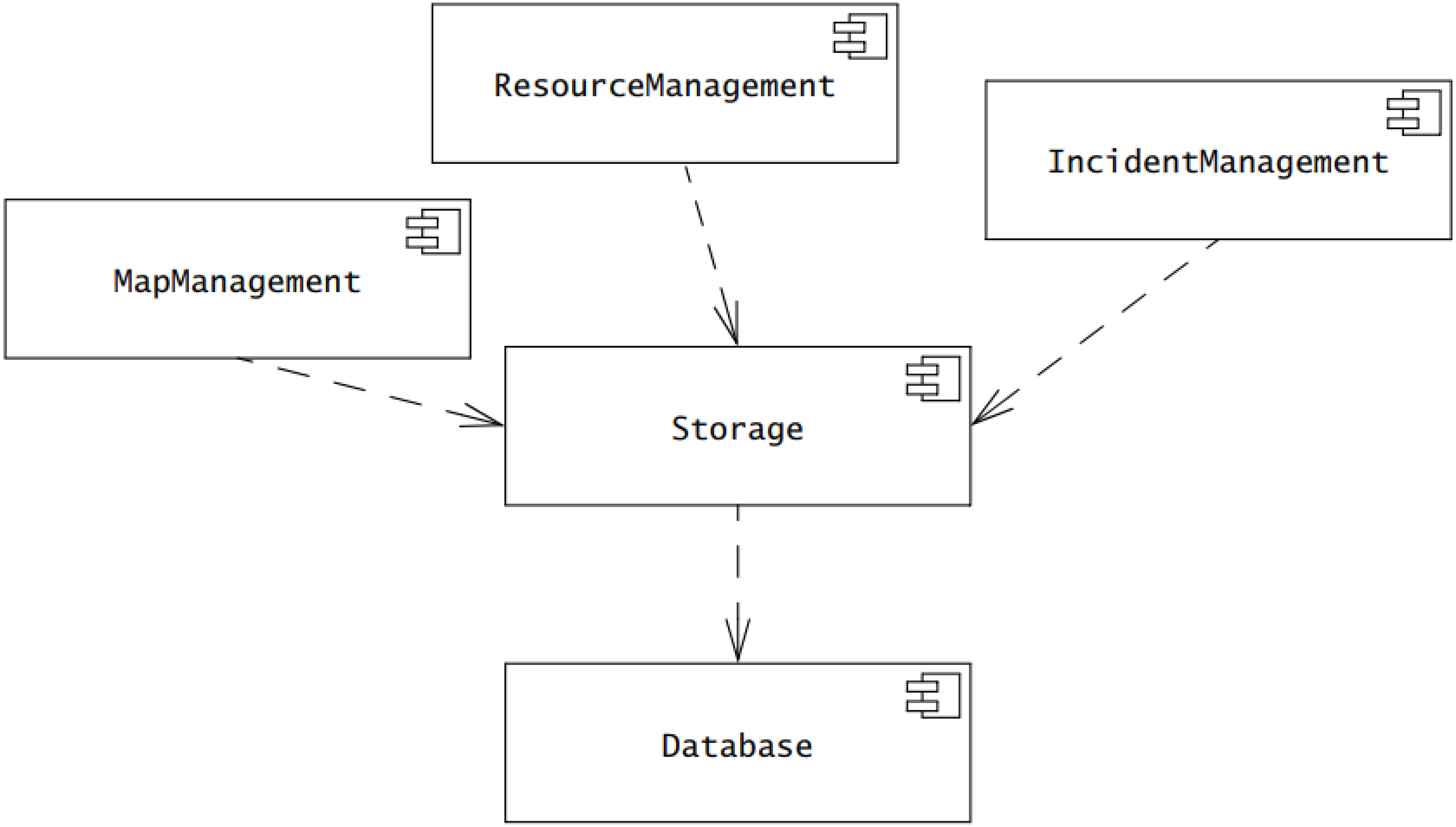
- Goal: Reduce system complexity while allowing change
- Coherence measures dependency among classes
  - ➡ High coherence: The classes in the subsystem perform similar tasks and are related to each other via associations
  - Low coherence: Lots of miscellaneous and auxiliary classes, no associations
- Coupling measures dependency among subsystems
  - High coupling: Changes to one subsystem will have high impact on the other subsystem
  - ➡ Low coupling: A change in one subsystem does not affect any other subsystem

Coupling and Coherence of Subsystems

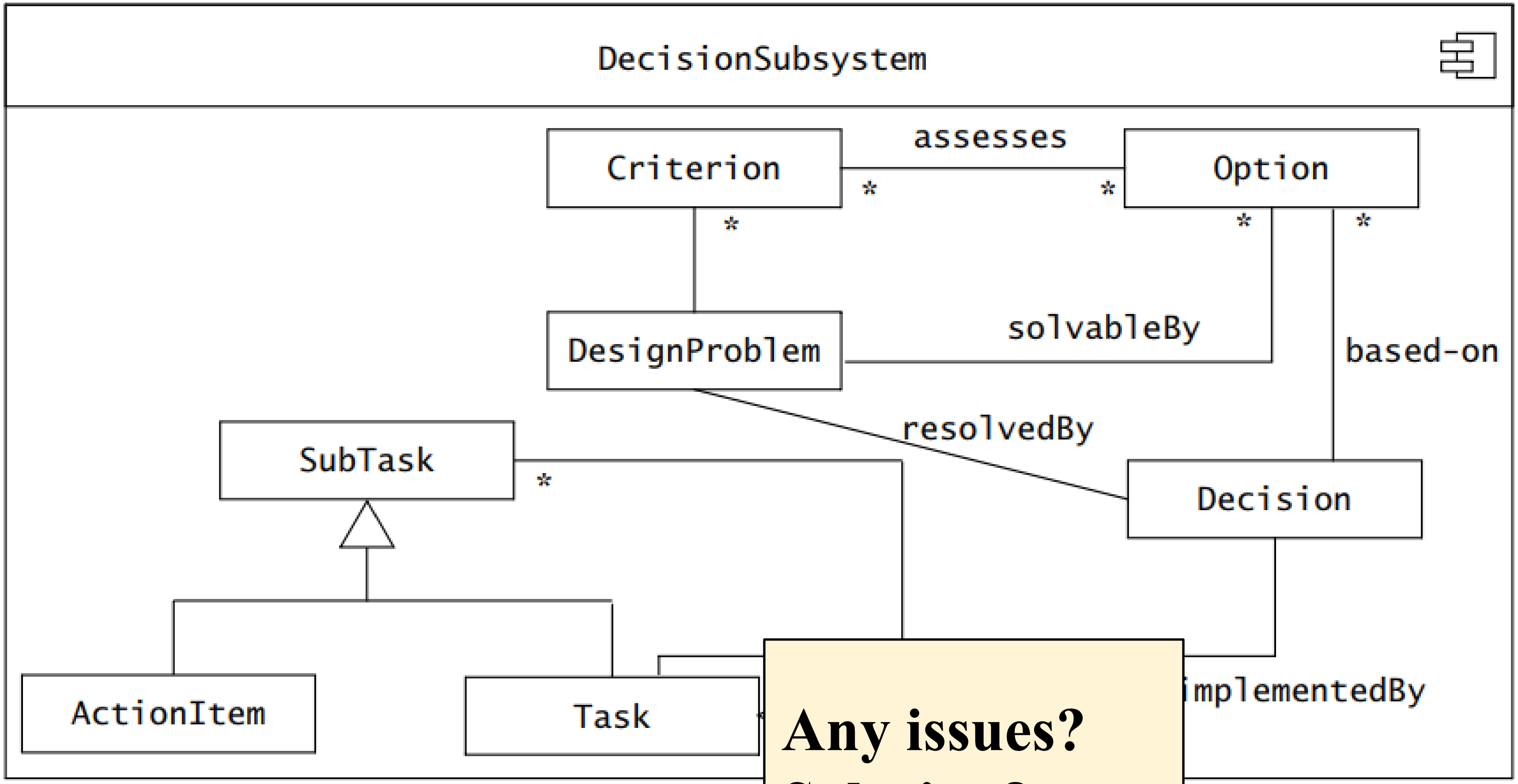


Any issues?
Solution?

Coupling and Coherence of Subsystems

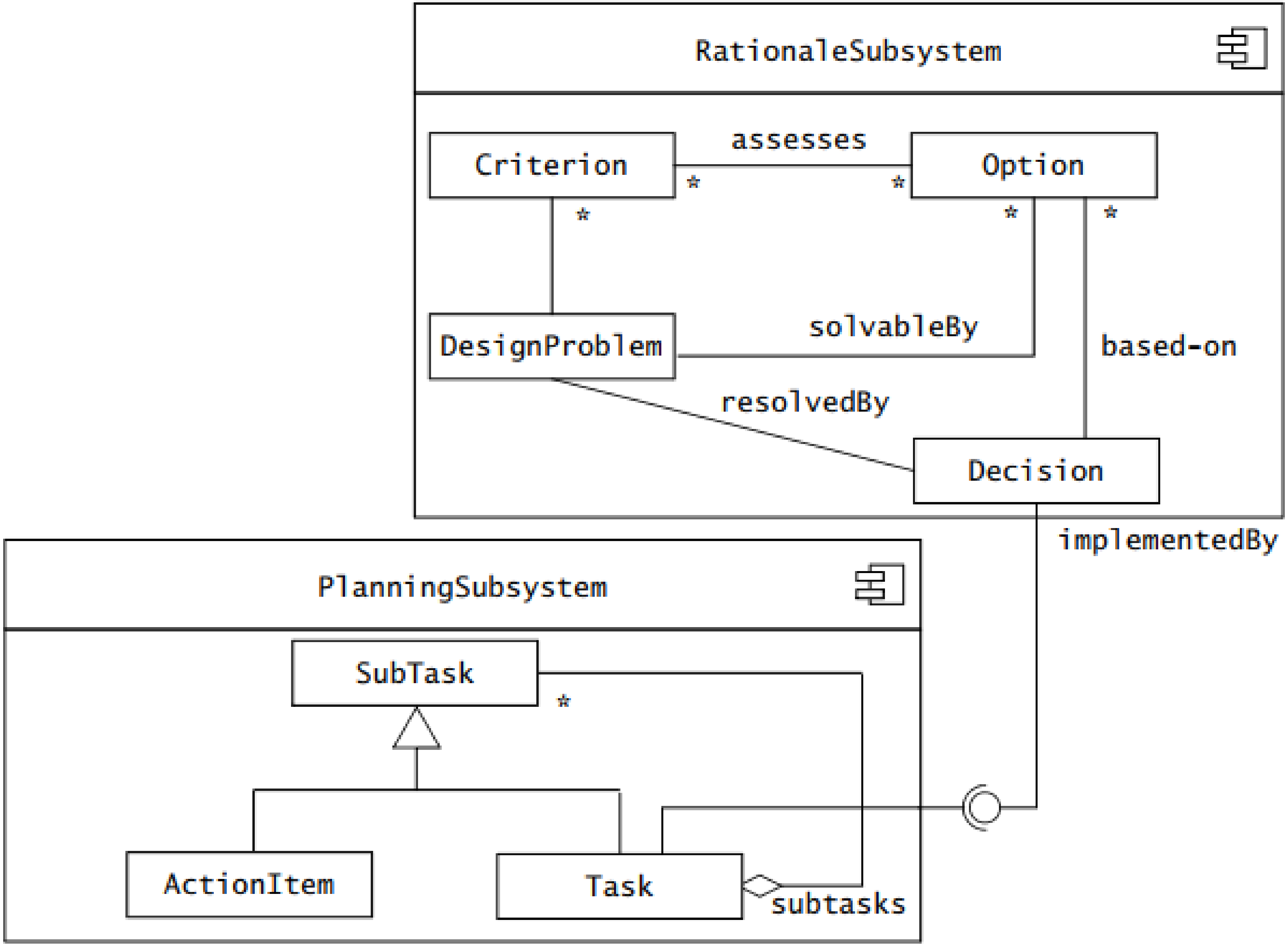


Coupling and Coherence of Subsystems



Any issues?
Solution?

Coupling and Coherence of Subsystems



Required Readings

- Chapter 6 from Bruegge’s OOSE textbook:
  Bruegge, Bernd, and Allen H. Dutoit. "Object-oriented software Engineering." *ed: Prentice Hall* , Third Edtiton