Southern
Illinois University

System Design

Tessema M. Mengistu
Department of Computer Science
Southern Illinois University Carbondale
tessema.mengistu@cs.siu.edu
Office - EGRA 409G

Outline

- Overview of System Design
- System design concepts
- System Design Activities
- System Design Document

Overview of System Design

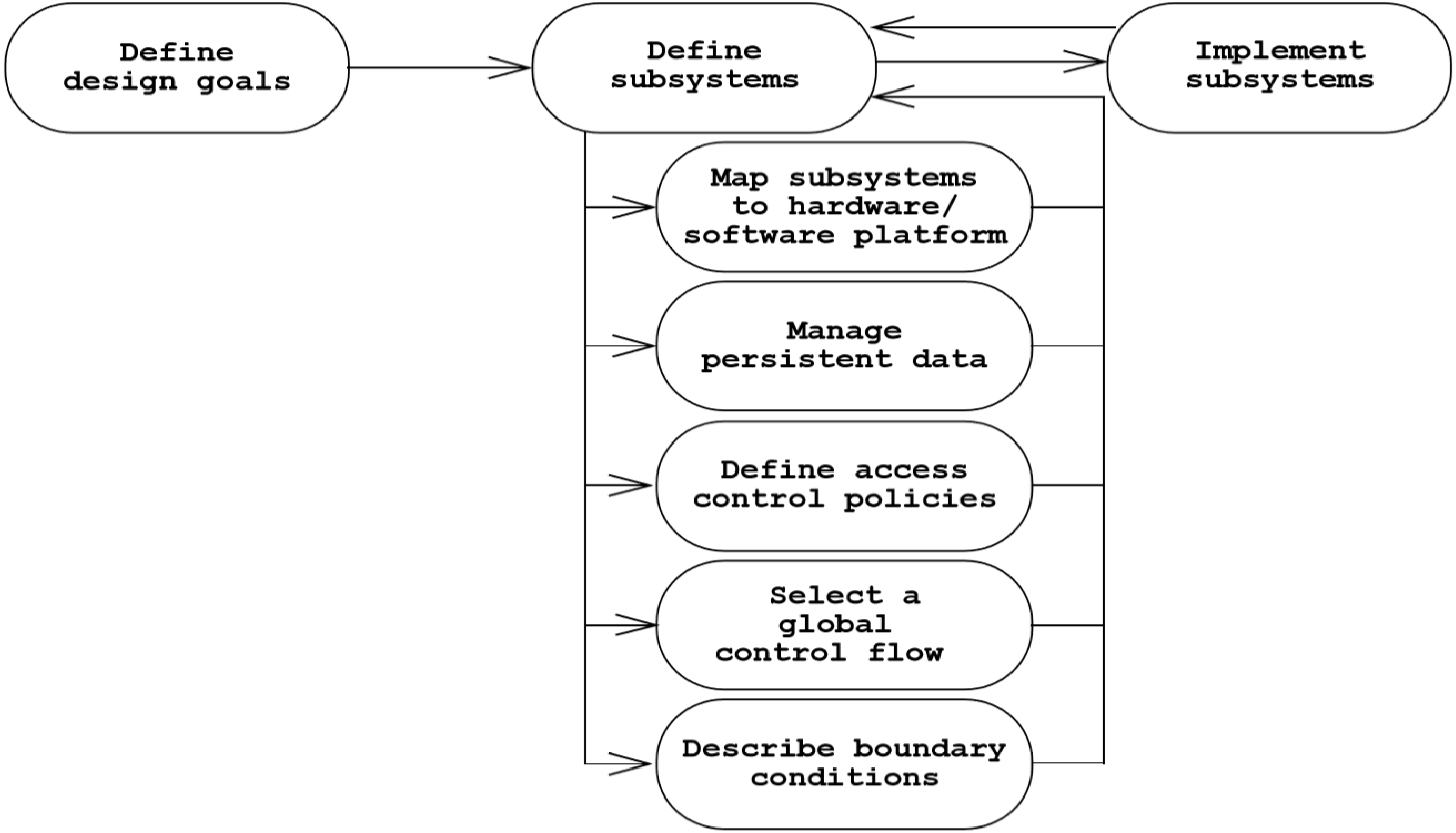
- Requirements analysis results in the requirements model described by:
  - A set of nonfunctional requirements and constraints, such as maximum response time, minimum throughput, reliability, operating system platform, and so on
  - A use case model, describing the functionality of the system from the actors’ point of view
  - An object model, describing the entities manipulated by the system
  - A sequence diagram for each use case, showing the sequence of interactions among objects participating in the use case

Overview of System Design

- System design, object design, and implementation constitute the construction of the system
- During system design:
  - We focus on the processes, data structures, and software and hardware components necessary to implement it
  - Developers define the design goals of the project and decompose the system into smaller subsystems that can be realized by individual teams
- System design is the transformation of the analysis model into a system design model

Overview of System Design

- System design includes:
  - The definition of design goals
  - The decomposition of the system into subsystems
  - The selection of off-the-shelf and legacy components
  - The mapping of subsystem to hardware
  - The selection of a persistent data management infrastructure
  - The selection of an access control policy
  - The selection of a global control flow mechanism
  - The handling of boundary conditions


Define design goals
Define subsystems
Implement subsystems
Map subsystems to hardware/ software platform
Manage persistent data
Define access control policies
Select a global control flow
Describe boundary conditions

System Design Concepts

- In order to reduce the complexity of the application domain, smaller parts called **classes** are identified and organized them into **packages**
- To reduce the complexity of the solution domain, we decompose a system into simpler parts, called **subsystems**
- Subsystems are made of a number of solution domain classes

System Design Concepts

- A subsystem is characterized by the **services** it provides to other subsystems
- A service is a set of related operations that share a common purpose
  - E.g.
    - A subsystem providing a notification service:
      - Defines operations to send notices, look up notification channels, and subscribe and unsubscribe to a channel

System Design Concepts

- The set of operations of a subsystem that are available to other subsystems form the **subsystem interface**
  - a.k.a application programmer interface (**API**)
  - includes the name of the operations, their parameters, their types, and their return values
- System design focuses on defining the services provided by each subsystem, that is, enumerating the operations, their parameters, and their high-level behavior

System Design Concepts

- Coupling and coherence are two important properties of subsystems
- **Coupling** is the strength of dependencies between two subsystems
  - A desirable property of a subsystem decomposition is that subsystems are as loosely coupled as possible
- **Coherence** is the strength of dependencies within a subsystem
  - A desirable property of a subsystem decomposition is that it leads to subsystems with high coherence

System Design Concepts

- The goal of system design is to manage complexity by dividing the system into smaller, manageable pieces
- There are two techniques for relating subsystems to each other
  - **Layering** - allows a system to be organized as a hierarchy of subsystems
  - **Partitioning** - organizes subsystems as peers

System Design Concepts

- Layer
  - A hierarchical decomposition in which each subsystem, or layer, provides higher level services, using services provided from lower level subsystems
  - Each layer can only depend on lower level layers and has no knowledge of the layers above it
  - Two architectures:
    - **Closed** architecture - each layer can only depend on the layers immediately below it
    - **Open** architecture - a layer can also access layers at deeper levels

System Design Concepts

- Partition
  - Decompose the system into **peer** subsystems, each responsible for a different class of services
  - Peers mutually provide different services to each other
- In general, a subsystem decomposition is the result of both partitioning and layering

System Design Concepts

- Software Architecture
  - A set of rules and methods that describe the functionality, organization, and implementation of computer systems
  - Is the conceptual model that defines the structure, behavior, and more views of a system
  - A software architecture includes the system decomposition, the global control flow, error-handling policies and inter-subsystem communication protocols
  - Different types
    - Client-Server (2-tier, 3-tier, n-tier, cloud computing)
      - MVC
    - Peer-to-peer

System Design Activities

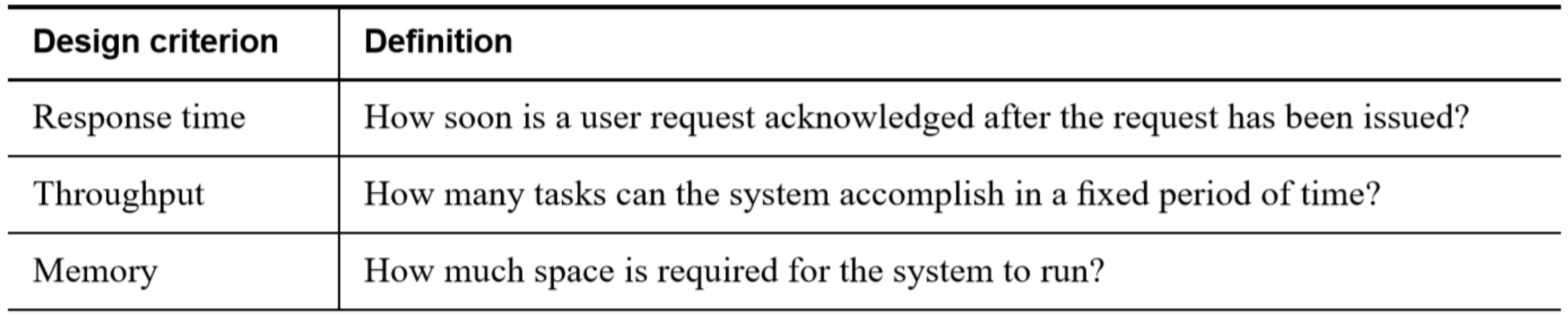
- System design consists of transforming the analysis model into the design model
- It takes into account the nonfunctional requirements and constraints described in the requirements analysis document
- It includes:
  - Identify design goals
  - Design an initial subsystem decomposition
  - Map subsystems to processors and components
  - Decide storage
  - Define access control policies
  - Select a control flow mechanism
  - Identify boundary conditions

Design Goals

- The definition of design goals is the first step of system design
- It identifies the qualities that the system should focus on
- Many design goals can be inferred from the nonfunctional requirements or from the application domain
- Organized into five groups:
  - Performance
  - Dependability
  - Cost
  - Maintenance
  - End user criteria

Design Goals

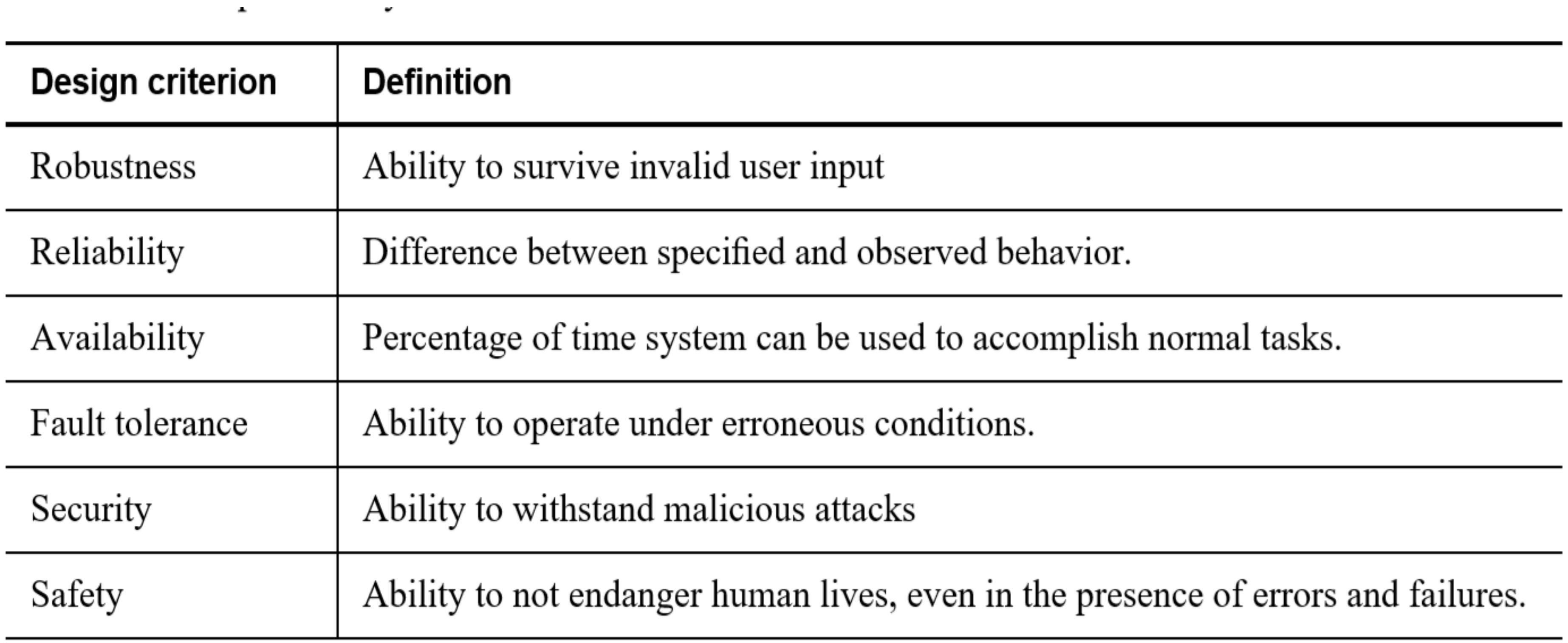
- Performance:



Design criterion	Definition
Response time	How soon is a user request acknowledged after the request has been issued?
Throughput	How many tasks can the system accomplish in a fixed period of time?
Memory	How much space is required for the system to run?

Design Goals

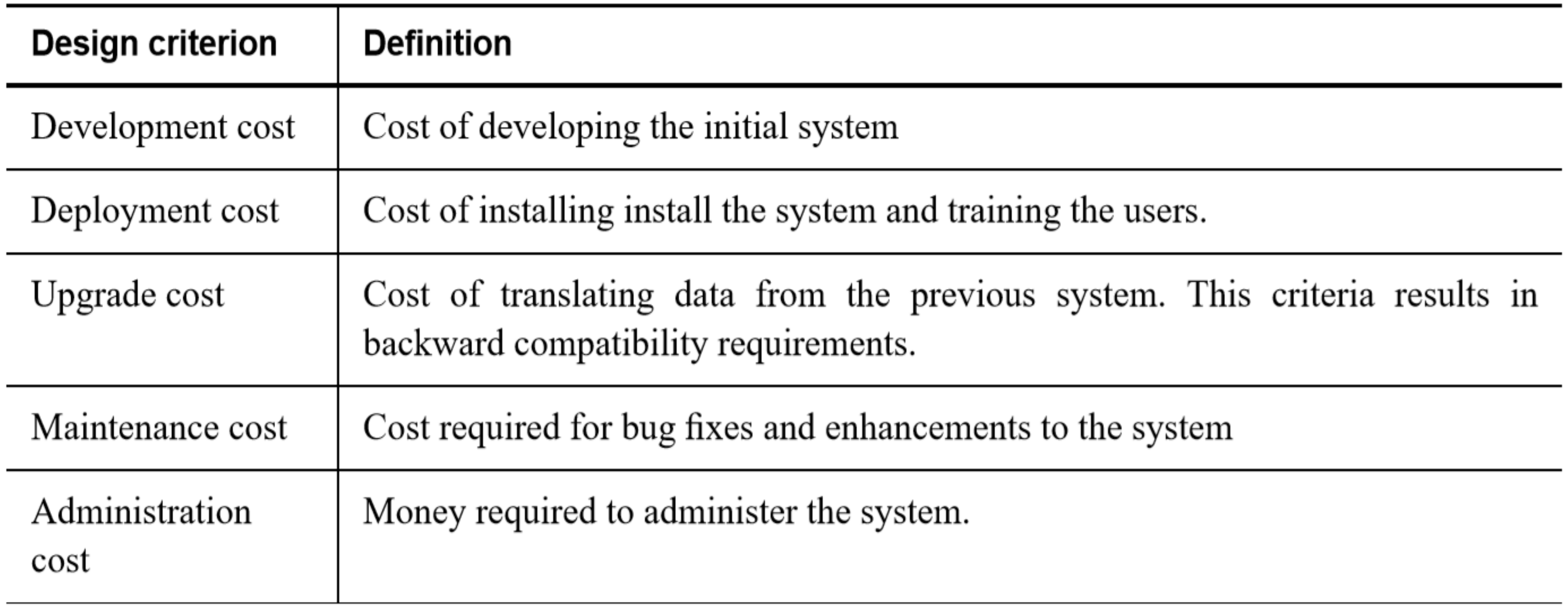
- Dependability:



Design criterion	Definition
Robustness	Ability to survive invalid user input
Reliability	Difference between specified and observed behavior.
Availability	Percentage of time system can be used to accomplish normal tasks.
Fault tolerance	Ability to operate under erroneous conditions.
Security	Ability to withstand malicious attacks
Safety	Ability to not endanger human lives, even in the presence of errors and failures.

Design Goals

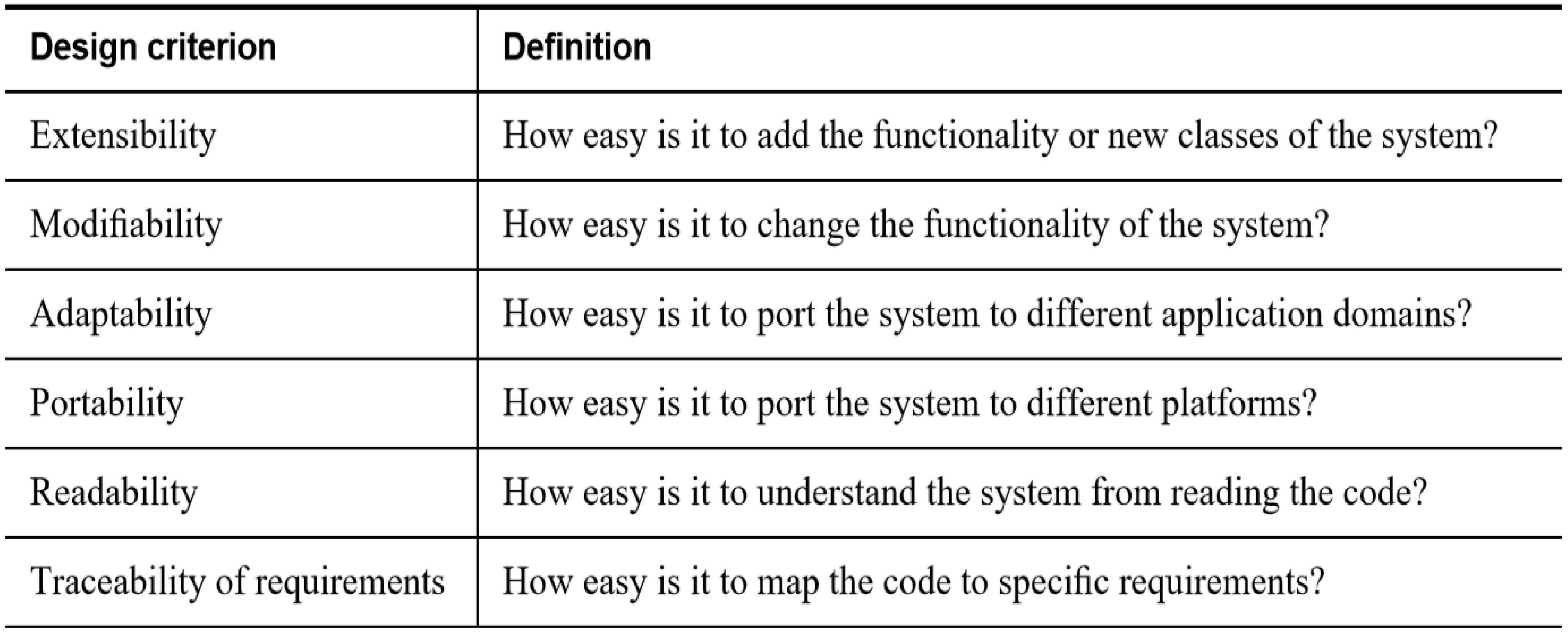
- Cost:



Design criterion	Definition
Development cost	Cost of developing the initial system
Deployment cost	Cost of installing install the system and training the users.
Upgrade cost	Cost of translating data from the previous system. This criteria results in backward compatibility requirements.
Maintenance cost	Cost required for bug fixes and enhancements to the system
Administration cost	Money required to administer the system.

Design Goals

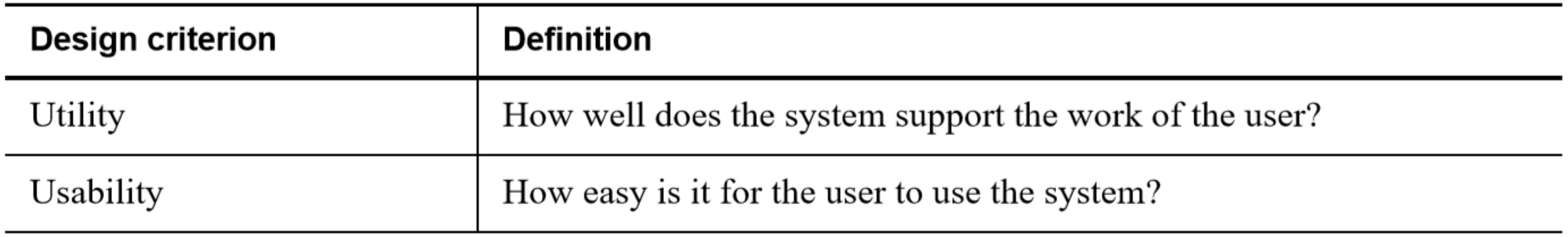
- Maintenance:



Design criterion	Definition
Extensibility	How easy is it to add the functionality or new classes of the system?
Modifiability	How easy is it to change the functionality of the system?
Adaptability	How easy is it to port the system to different application domains?
Portability	How easy is it to port the system to different platforms?
Readability	How easy is it to understand the system from reading the code?
Traceability of requirements	How easy is it to map the code to specific requirements?

Design Goals

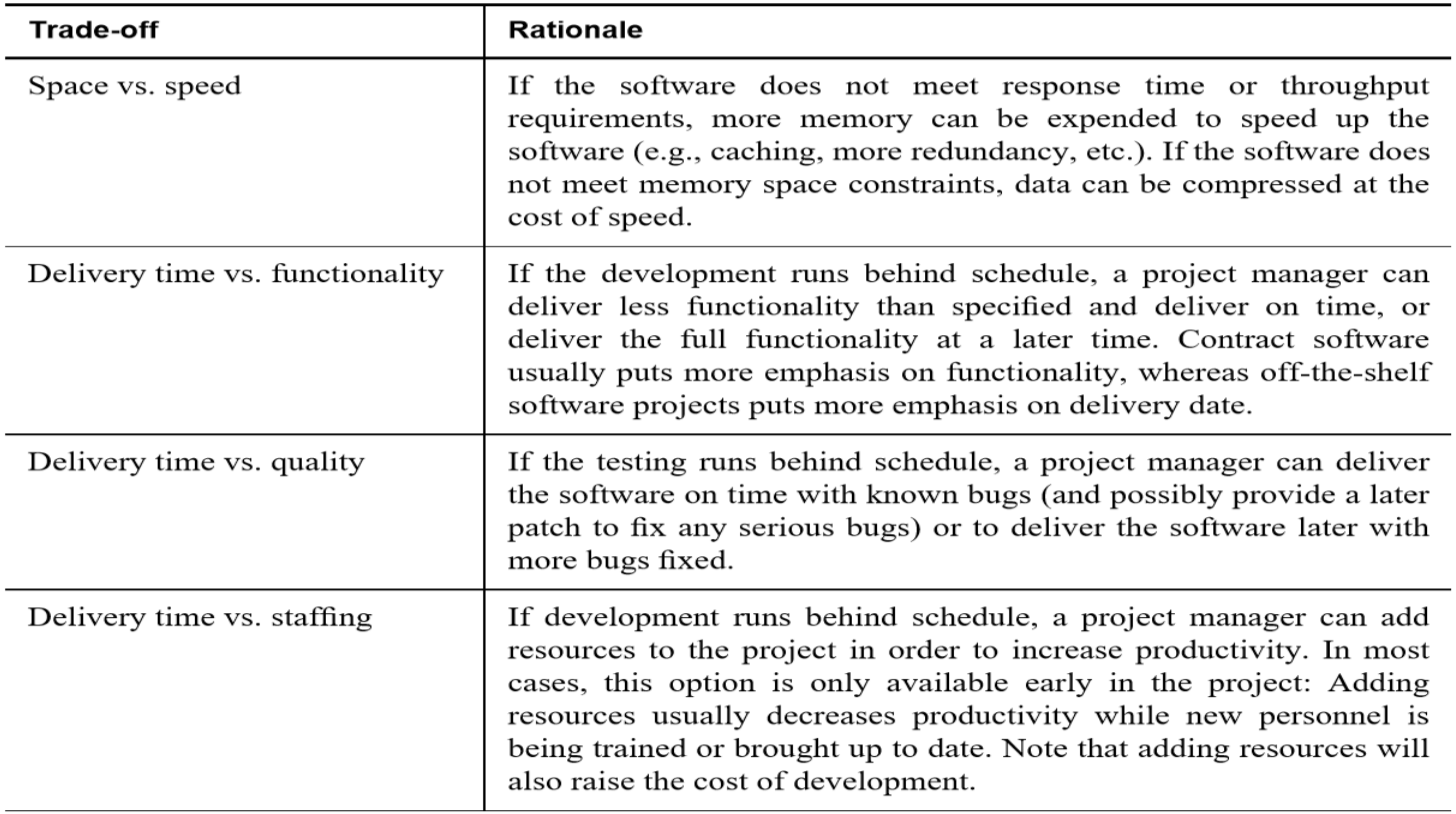
- End User Criteria



Design criterion	Definition
Utility	How well does the system support the work of the user?
Usability	How easy is it for the user to use the system?

Design Goals

- When defining design goals, only a small subset of these criteria can be simultaneously taken into account



Trade-off	Rationale
Space vs. speed	If the software does not meet response time or throughput requirements, more memory can be expended to speed up the software (e.g., caching, more redundancy, etc.). If the software does not meet memory space constraints, data can be compressed at the cost of speed.
Delivery time vs. functionality	If the development runs behind schedule, a project manager can deliver less functionality than specified and deliver on time, or deliver the full functionality at a later time. Contract software usually puts more emphasis on functionality, whereas off-the-shelf software projects puts more emphasis on delivery date.
Delivery time vs. quality	If the testing runs behind schedule, a project manager can deliver the software on time with known bugs (and possibly provide a later patch to fix any serious bugs) or to deliver the software later with more bugs fixed.
Delivery time vs. staffing	If development runs behind schedule, a project manager can add resources to the project in order to increase productivity. In most cases, this option is only available early in the project: Adding resources usually decreases productivity while new personnel is being trained or brought up to date. Note that adding resources will also raise the cost of development.

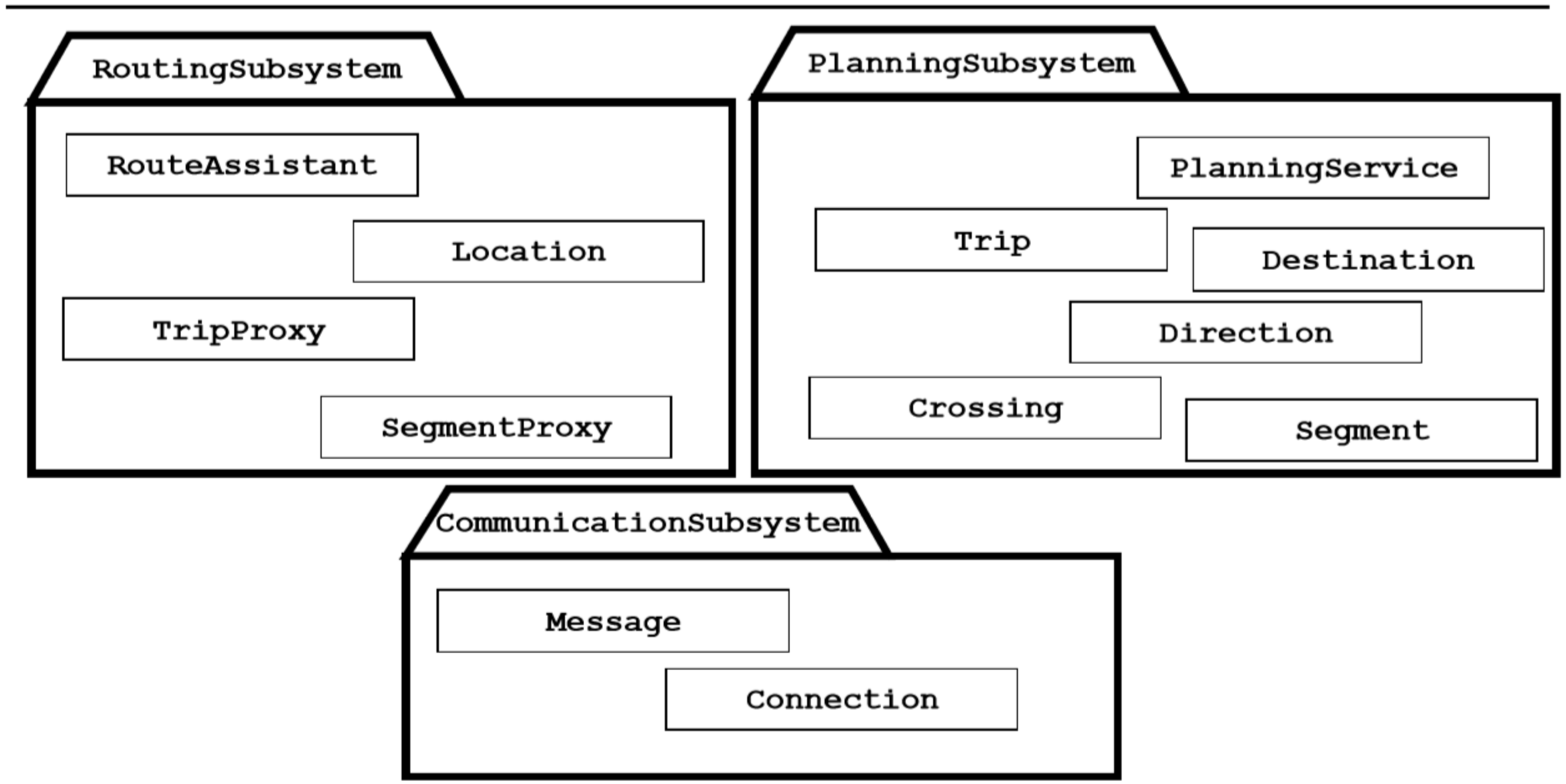
Subsystem Decomposition

- Subsystem decomposition constitutes the bulk of system design
- Developers divide the system into manageable pieces to deal with complexity
- The initial subsystem decomposition should be derived from the functional requirements
- Subsystem decomposition reduces the complexity of the solution domain by minimizing dependencies among classes

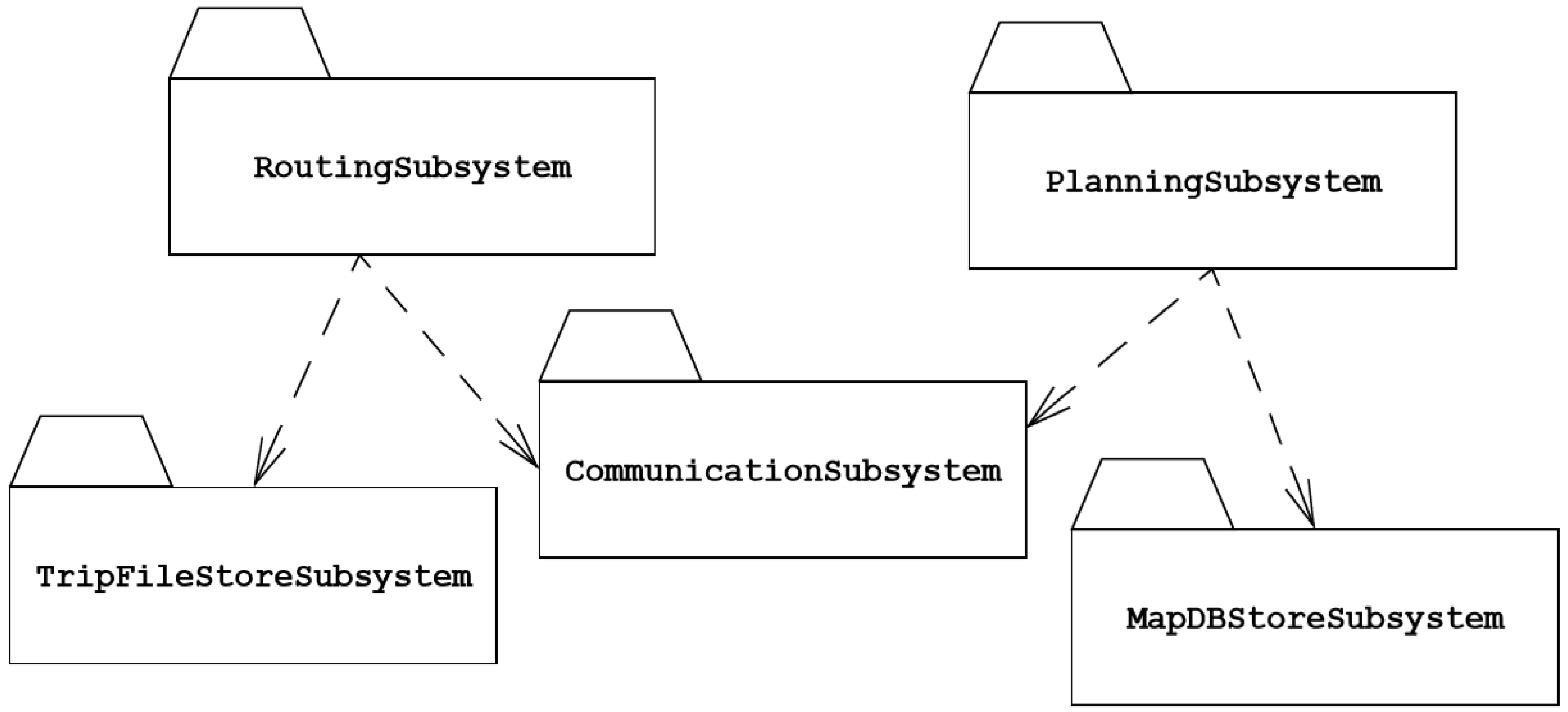
Subsystem Decomposition

- Heuristics for grouping objects into subsystems
  - Assign objects identified in one use case into the same subsystem
  - Create a dedicated subsystem for objects used for moving data among subsystems
  - Minimize the number of associations crossing subsystem boundaries
  - All objects in the same subsystem should be functionally related

Subsystem Decomposition



Subsystem Decomposition

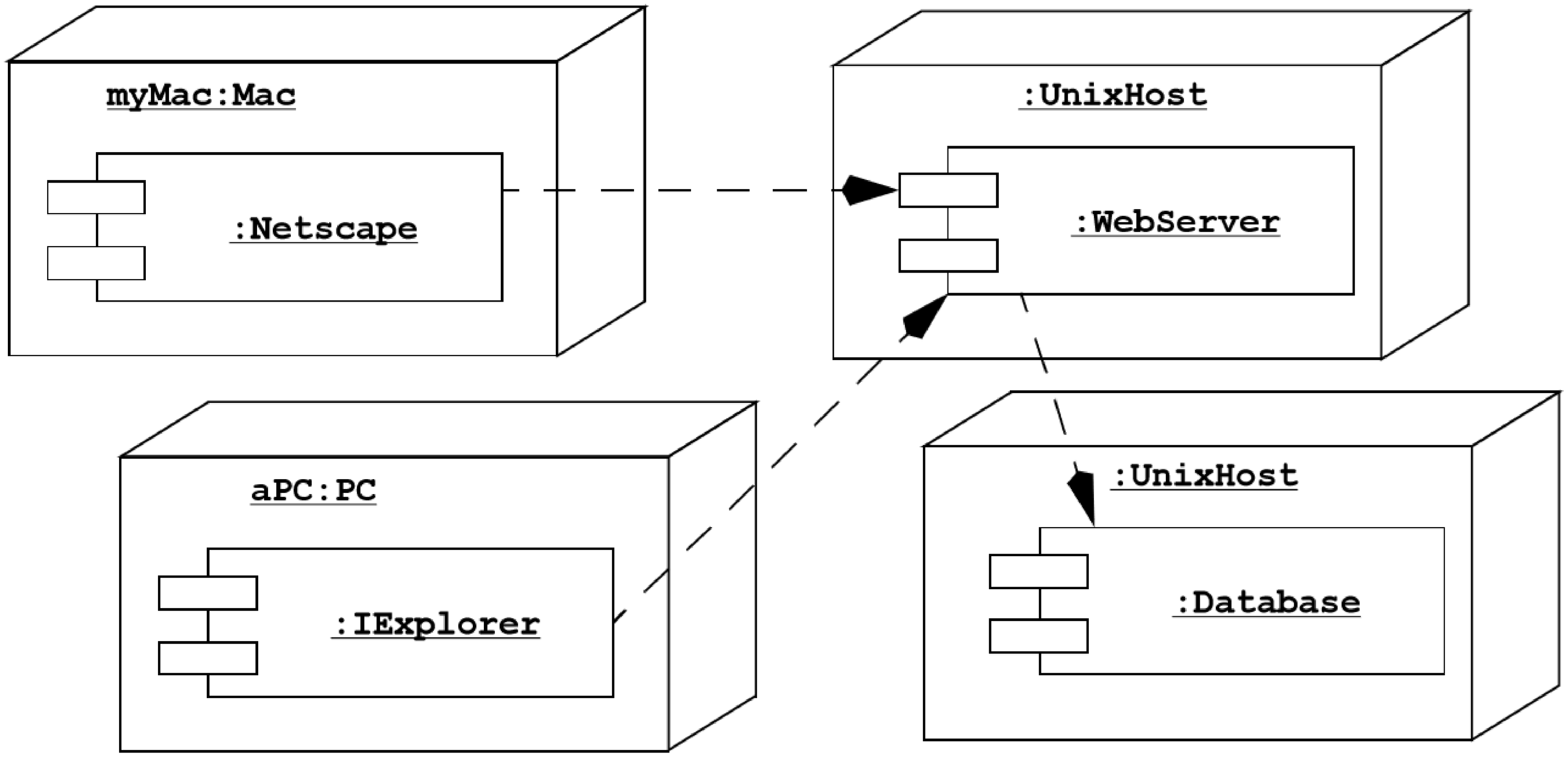


Hardware/software mapping

- What is the hardware configuration of the system?
- Which node is responsible for which functionality?
- How is communication between nodes realized?
- Which services are realized using existing software components?
- Addressing hardware/software mapping issues often leads to the definition of additional subsystems dedicated to moving data from one node to another, dealing with concurrency, and reliability issues
- Off-the-shelf components enable developers to realize complex services more economically

Deployment

- UML deployment diagrams are used to depict the relationship among run-time components and hardware nodes
- Components are self-contained entities that provide services to other components or actors
- In UML deployment diagrams, nodes are represented by boxes containing component icons


myMac:Mac
:Netscape
:UnixHost
:WebServer
aPC:PC
:IExplorer
:UnixHost
:Database

Data management

- Most functionality in system is concerned with creating or manipulating persistent data
  - For this reason, access to the data should be fast and reliable
- This leads to the selection of a database management system and of an additional subsystem dedicated to the management of persistent data
- Which data need to be persistent?
- Where should persistent data be stored?
- How are they accessed?

Data management

- Different options for data management:
  - – Flat file
  - – Relational database
  - – Object Oriented database
  - – Graph database
  - – No-sql database

Access control

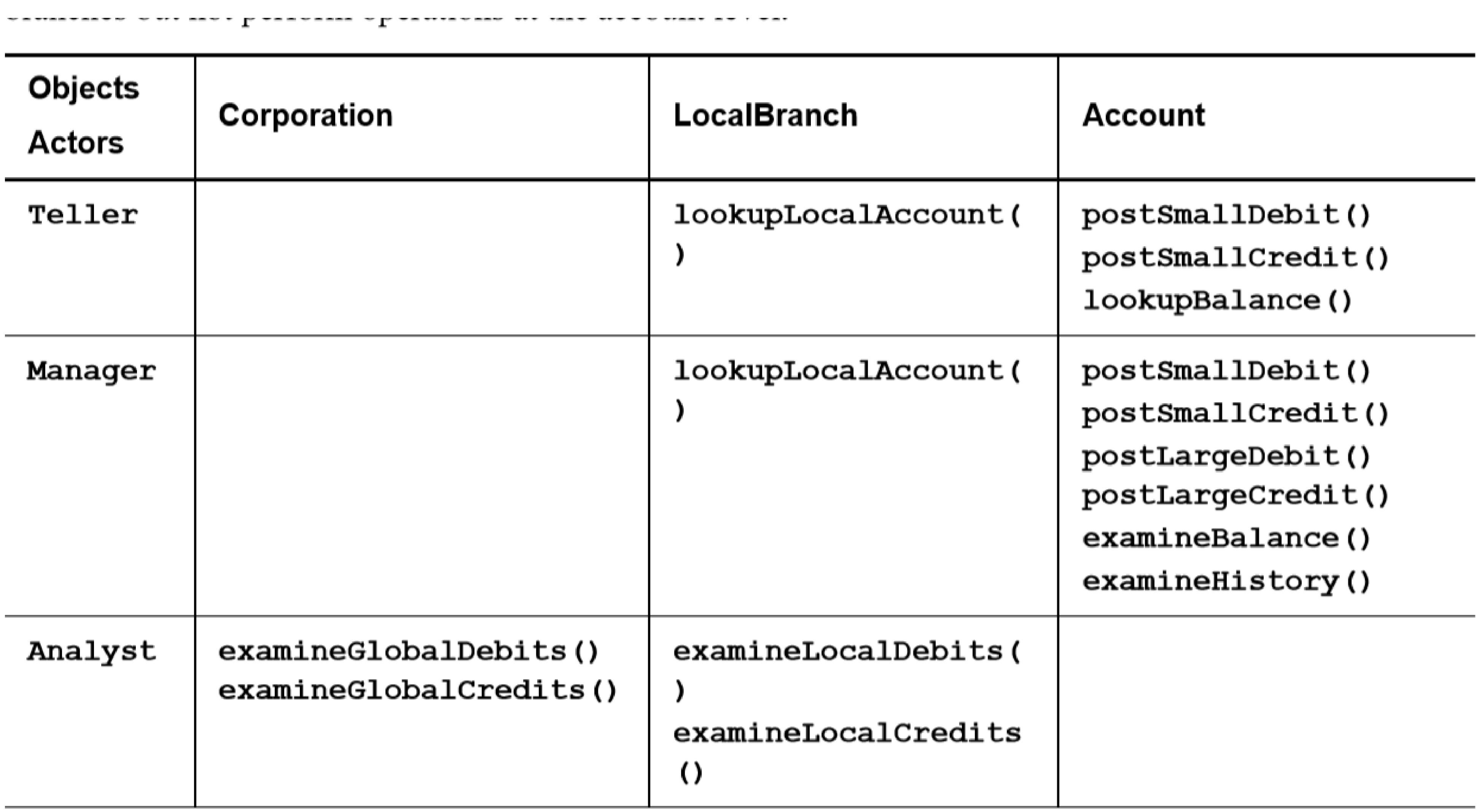
- In multiuser systems, different actors have access to different functionality and data
- Access control specify who can and cannot access certain data
- It is a system-wide issue and must be consistent across the system
- Who can access which data?
- Can access control change dynamically?
- How is access control specified and realized?

Access control

- Access control can be represented using one of three different approaches:
  - Global access table
  - Access control list
  - Capabilities

Access control

- A global access table represents explicitly every cell in the matrix as a (actor, class, operation) tuple
- Determining if an actor has access to a specific object requires looking up the corresponding tuple
  - If no such tuple is found, access is denied
- Global access tables require a lot of space



[illegible]

Objects Actors	Corporation	LocalBranch	Account

`Teller`		`lookupLocalAccount()`	`postSmallDebit()`
			`postSmallCredit()`
			`lookupBalance()`

`Manager`		`lookupLocalAccount()`	`postSmallDebit()`
			`postSmallCredit()`
			`postLargeDebit()`
			`postLargeCredit()`
			`examineBalance()`
			`examineHistory()`

`Analyst`	`examineGlobalDebits()`
	`examineGlobalCredits()`
		`examineLocalDebits()`
		`examineLocalCredits()`	

Access control

- An access control list associates a list of (actor, operation) pairs with each class to be accessed
- Every time an object is accessed, its access list is checked for the corresponding actor and operation
- Access control lists make it faster to answer the question, “Who has access to this object?”

Access control

- A capability associates a (class, operation) pair with an actor.
- A capability provides an actor to gain control access to an object of the class described in the capability
- Denying a capability is equivalent to denying access
- Capability lists make it faster to answer the question, “Which objects has this actor access to?”

Control Flow

- Control flow is the sequencing of actions in a system
- How does the system sequence operations?
- Is the system event driven?
- Can it handle more than one user interaction at a time?
- There are three possible control flow mechanisms:
  - Procedure - driven
  - Event - driven
  - Threads

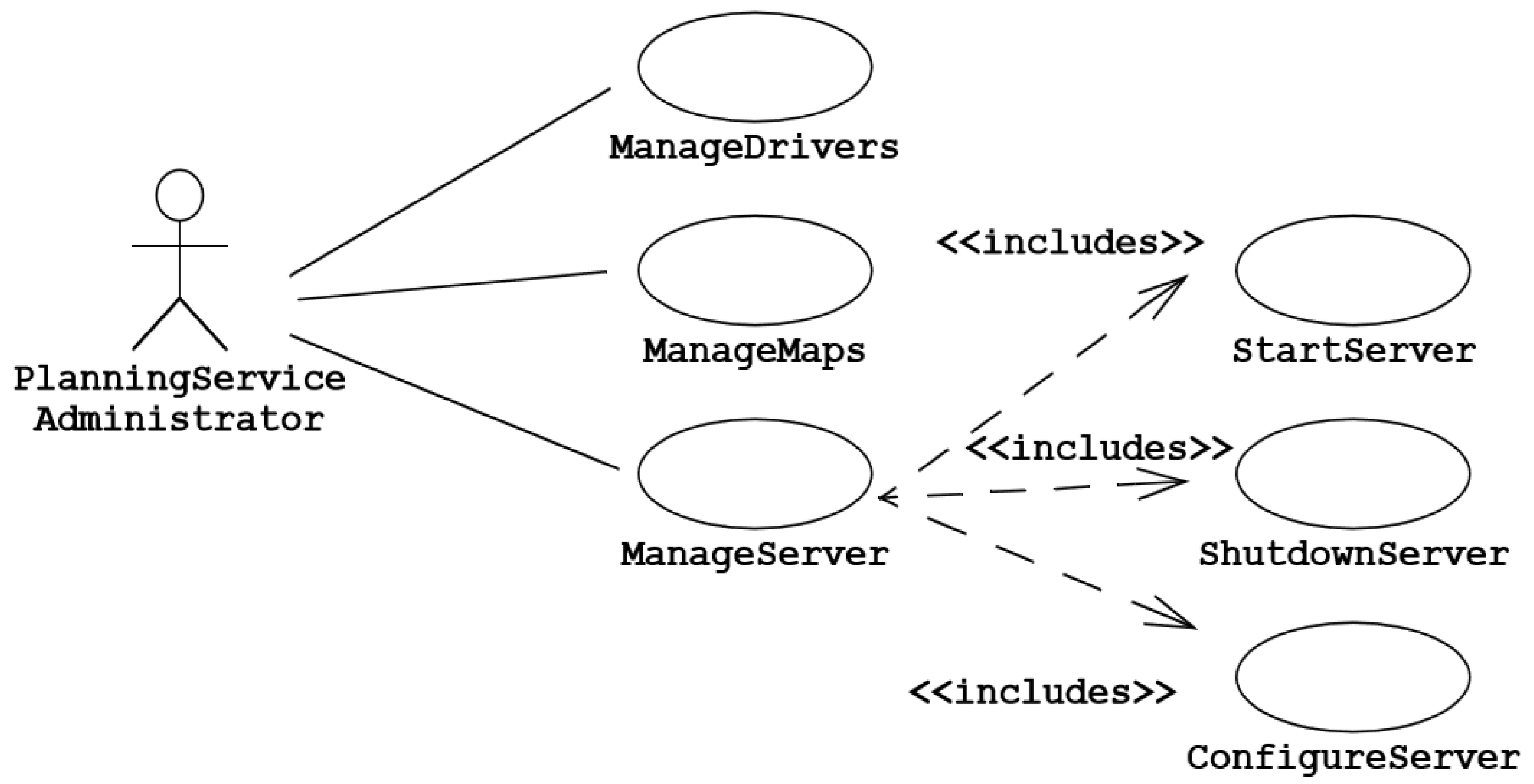
Control Flow

- The choice of control flow has an impact on the interfaces of subsystems
  - If an event-driven control flow is selected, subsystems will provide event handlers
  - If threads are selected, subsystems need to guarantee mutual exclusion in critical sections

Boundary Conditions

- Initialization, shutdown, and exception handling have an impact on the interface of all subsystems
- How is the system initialized?
- How is it shut down?
- How are exceptional cases detected and handled?

Boundary Conditions



Correctness of System Design

- The system design model is correct if the analysis model can be mapped to the system design model
- Questions to determine if the system design is correct:
  - Can every subsystem be traced back to a use case or a nonfunctional requirement?
  - Can every use case be mapped to a set of subsystems?
  - Can every design goal be traced back to a nonfunctional requirement?
  - Is every nonfunctional requirement addressed in the system design model?
  - Does each actor have an access policy?
  - Is it consistent with the nonfunctional security requirement?

System Design Document

- 1. Introduction
  - 1.1 Purpose of the system
  - 1.2 Design goals
  - 1.3 Definitions, acronyms, and abbreviations
  - 1.4 References
  - 1.5 Overview
- 2. Current software architecture
- 3. Proposed software architecture
  - 3.1 Overview
  - 3.2 Subsystem decomposition
  - 3.3 Hardware/software mapping
  - 3.4 Persistent data management
  - 3.5 Access control and security
  - 3.6 Global software control
  - 3.7 Boundary conditions
- 4. Subsystem services