Fundamental of Software Engineering
CSE 3205

Chapter Four

Software Design

School of Electrical Engineering and Computing

Department of Computer Science and Engineering

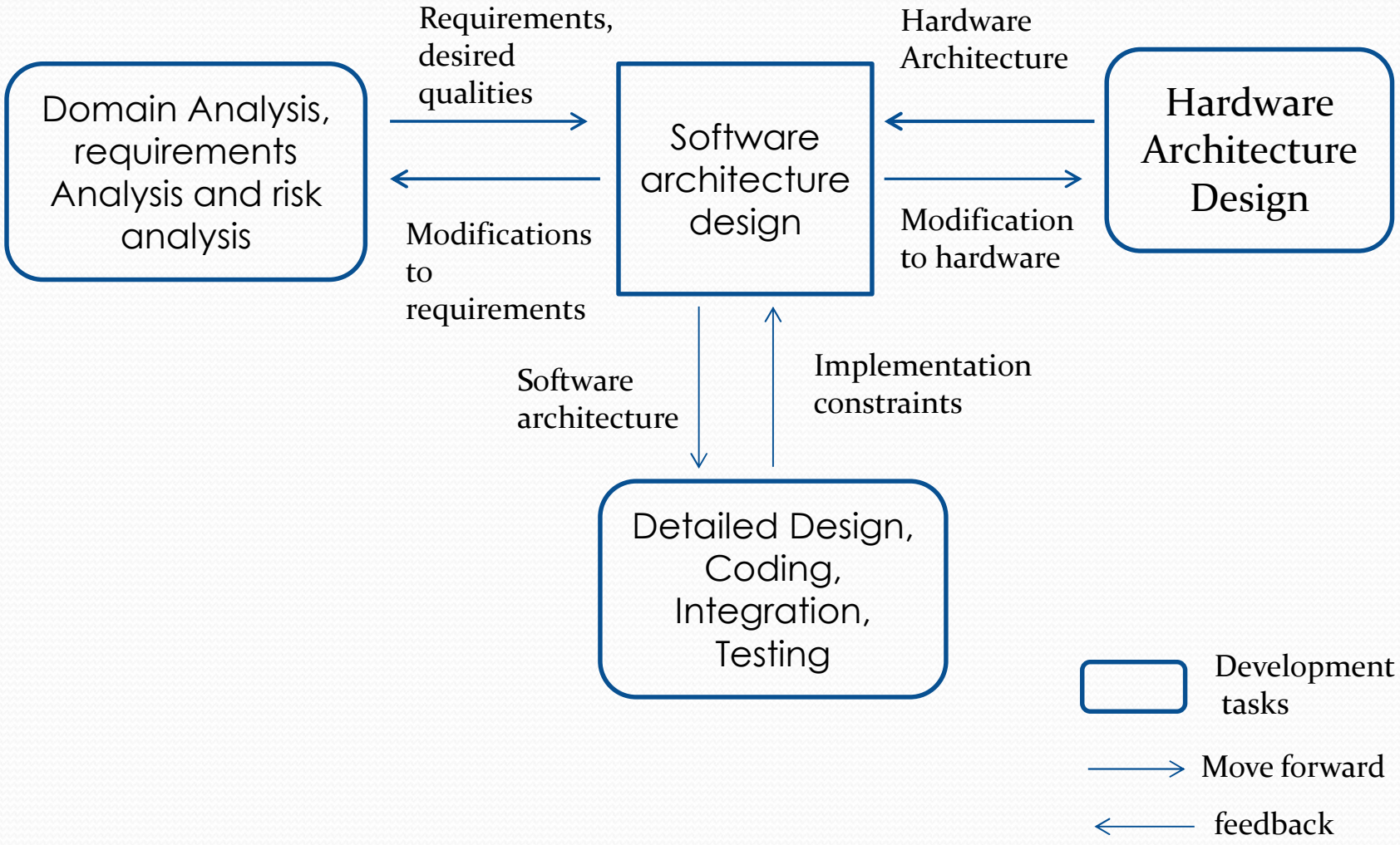
ASTU

9 February 2024

Software Design

- The design is the *blueprint* of the system to be implemented in order to produce quality software
- Software design involves a set of **principles**, **concepts** and **practices** that lead to the development of high quality software
- For design, software engineers are looking more at the ***solution domain*** (rather than the problem domain when doing requirements)
- The output is ***models that are used for system implementation***, i.e. models that are architectural, interface, component-level and Data and deployment representations of the system

Relation of Software architecture to other development tasks



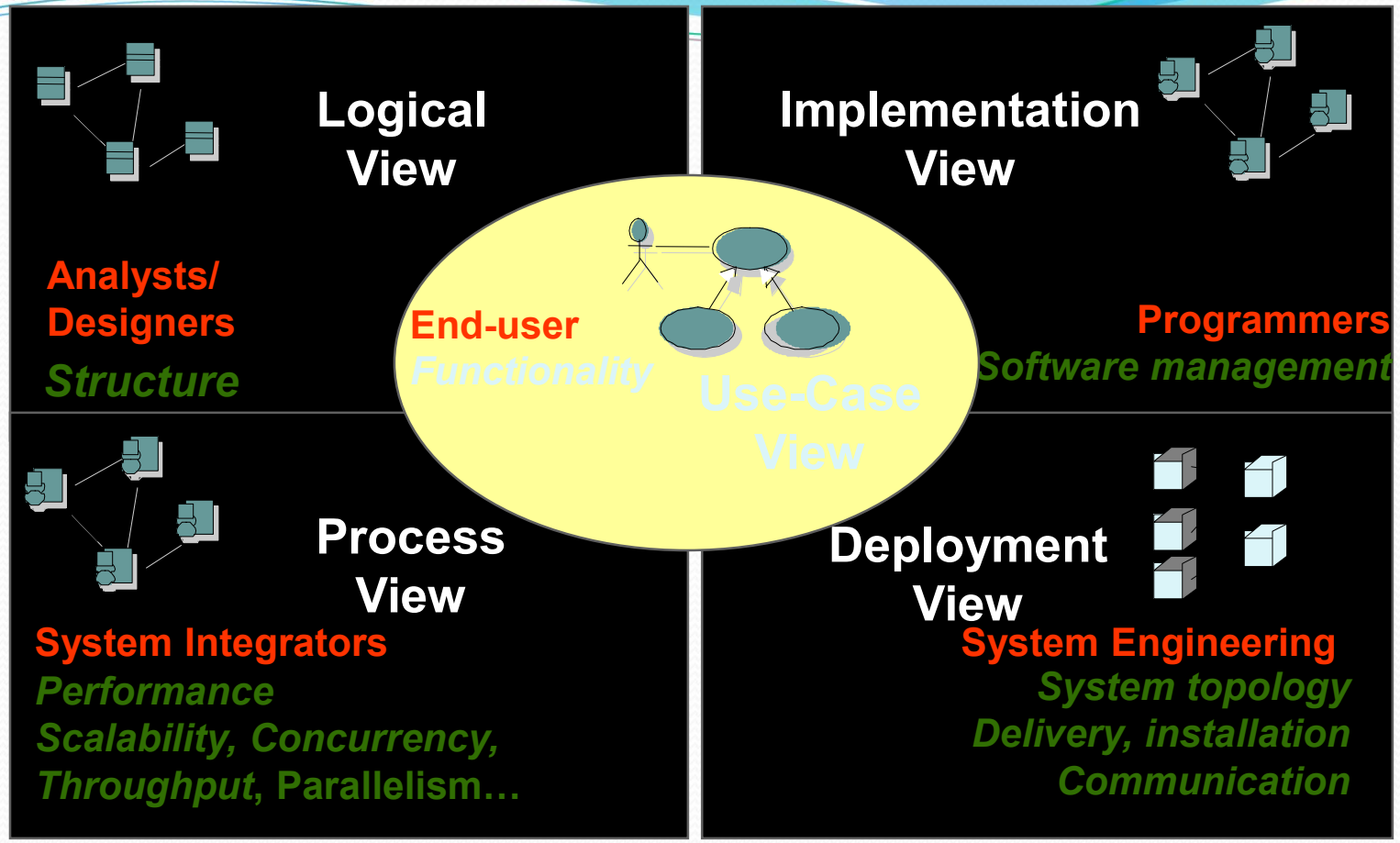
4 + 1 view model of software architecture

- A logical view, which shows the key abstractions in the system as objects or object classes.
- A process view, which shows how, at run-time, the system is composed of interacting processes.
- A development view, which shows how the software is decomposed for development.
- A physical view, which shows the system hardware and how software components are distributed across the processors in the system.
- Related using use cases or scenarios (+1)

Logical View
Functional Requirements –

Deals with design, packages, sub-systems, and classes, layers, …

Implementation View – deals mostly with programming and organization of the static software modules & unit test



A ***View*** is a complete description (an abstraction) of a system **from a particular view-point or perspective – covering particular concerns and omitting others not relevant to this perspective.**

Different 'views' from different 'stakeholders; different concerns.

A *Model* is a complete representation.

Software Design

- Software design model consists of 4 major designs:
  - Architectural Design
  - Data/Class Design
  - Interface Design
  - Component Design

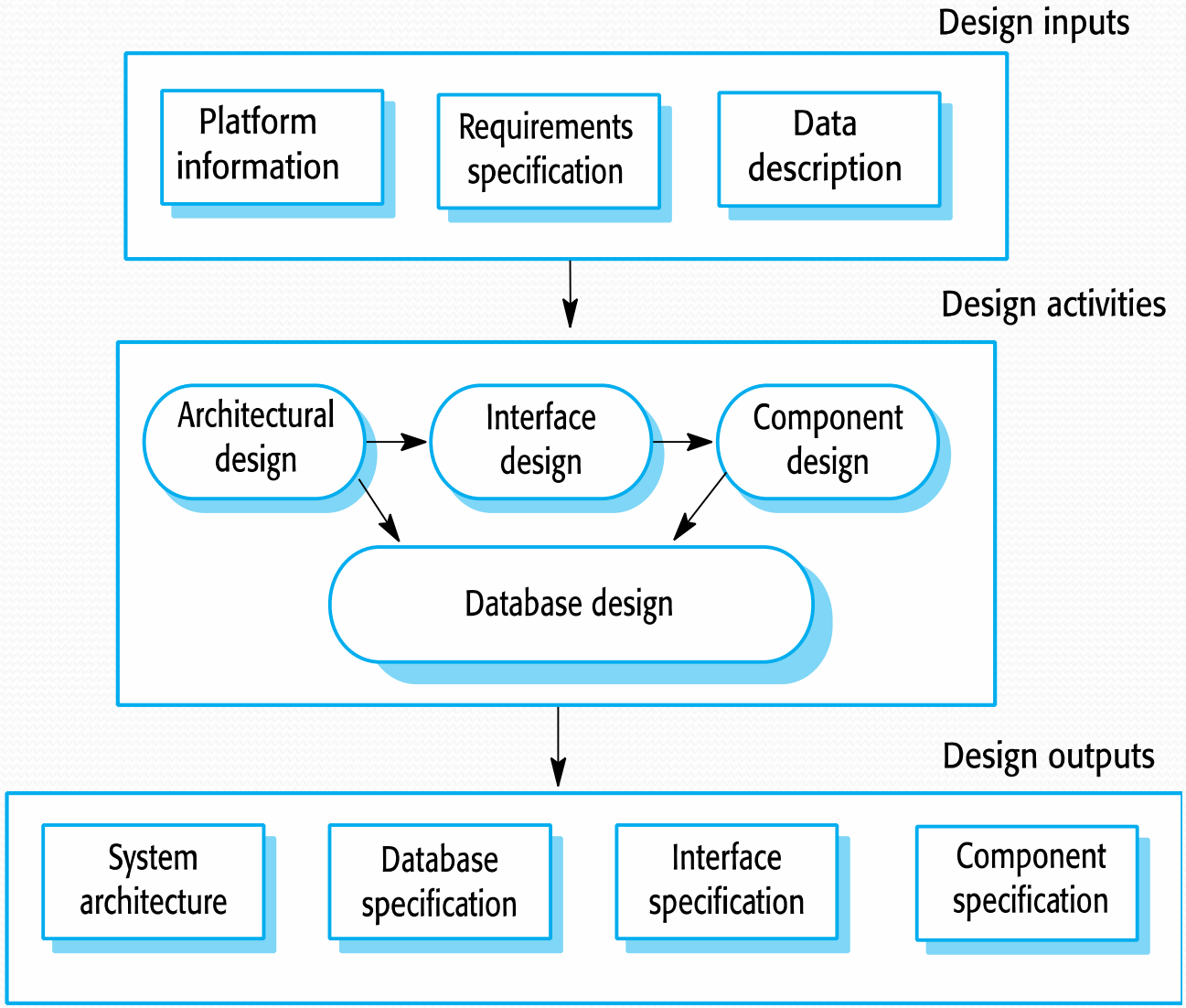
Cont....

- **Architectural Design** - defines the relationships among the major structural elements of the software, it is derived from the class-based elements and flow-oriented elements of the analysis model
- **Data/class Design** - Created by transforming the analysis model class-based elements into classes and data structures required to implement the software
- **Interface Design** - describes how the software elements, hardware elements, and end-users communicate with one another, it is derived from the analysis model ,scenario-based elements, flow-oriented elements, and behavioral elements
- **Component-level Design** - created by transforming the structural elements defined by the software architecture into a procedural description of the software components using information obtained from the analysis model , class-based elements, flow-oriented elements, and behavioral elements

The Design Process

- The design is the *blueprint* of the system to be implemented in order to produce quality software
  - The **quality guidelines** used are:
    - The design must implement all of the explicit requirements contained in the analysis model, and it must accommodate all of the implicit requirements desired by the customer.
    - The design must be a readable, understandable guide for those who generate code and for those who test and subsequently support the software.
    - The design should **provide a complete picture** of the software, addressing the data, functional, and behavioral domains from an implementation perspective.

General Model of the Design Process



Why design is so important?

- It is place where quality is fostered.
- It provides us with representation of software that can be assessed for quality.
- Only way that can accurately translate a customer's requirements into a finished software product.
- It serves as foundation for all software engineering activities.
- Without design difficult to assess:
  - Risk
  - Test
  - Quality

Design Process and Design Quality

- S/w design is an iterative process through which requirements are translated into a “blueprint” for constructing the s/w.
- As design iteration occur, subsequent refinement leads to design representation at much lower levels of abstraction(solution domain).

Design quality attributes

- Acronym FURPS –
  - Functionality
  - Usability
  - Reliability
  - Performance
  - Supportability

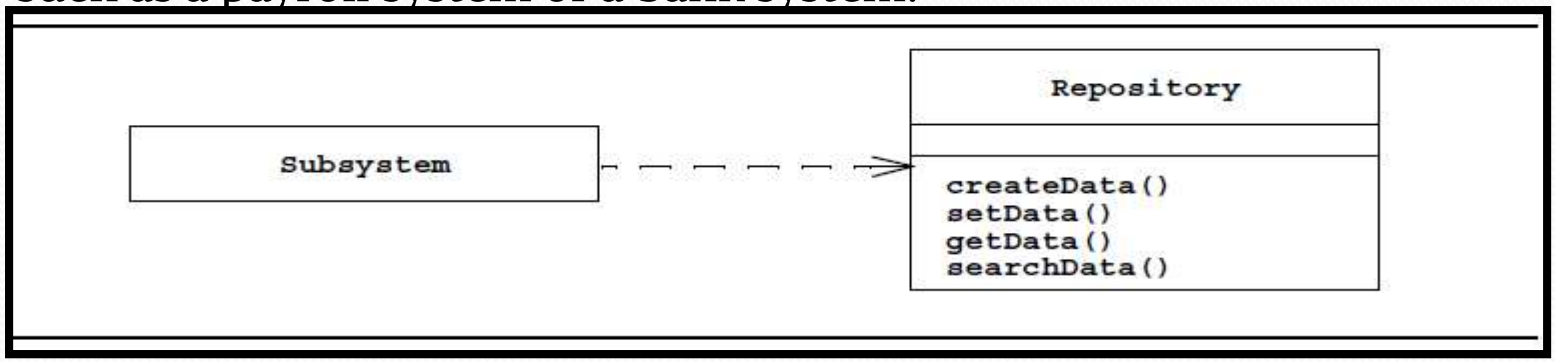
- Functionality – is assessed by evaluating the feature set and capabilities of the program.
  - Functions that are delivered and security of the overall system.
- Usability – assessed by considering human factors, consistency & documentation.
- Reliability – evaluated by
  - measuring the frequency and severity of failure.
  - Accuracy of output results.
  - Ability to recover from failure and predictability of the program.
- Performance - measured by processing speed, response time, resource consumption, efficiency.
- Supportability – combines the ability to extend the program (extensibility), adaptability and serviceability.

Software architecture

- A **software architecture** includes the system decomposition, the global control flow, error-handling policies and inter subsystem communication protocols [Shaw & Garlan, 1996].

Some of architectural style

1. ***Repository architecture:*** An architecture where subsystems access and modify data from a **single data structure** called the central **repository.**
   - Control flow can be dictated either by the central repository
   - The repository architecture is typical for **database management systems**, such as a payroll system or a bank system.



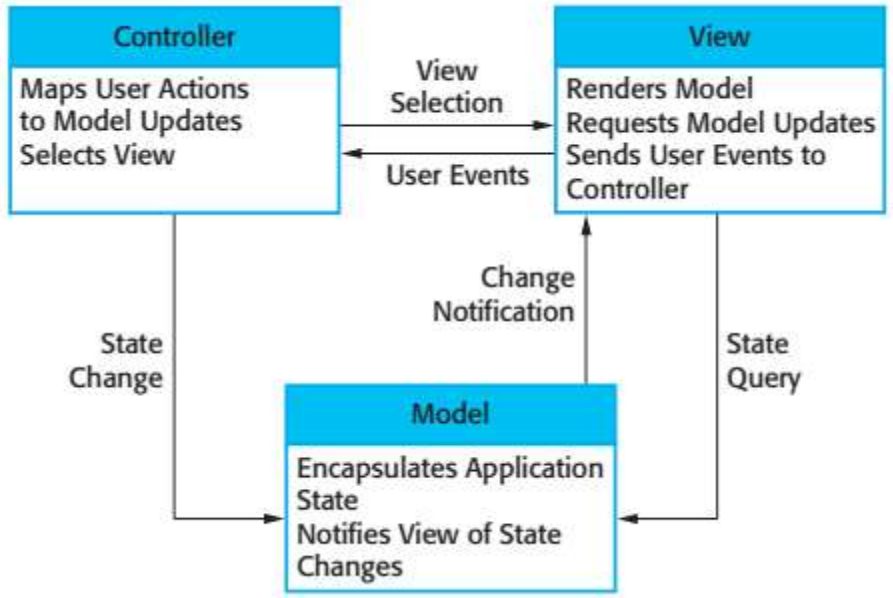
- Fig. Repository architecture (UML class diagram). Every subsystem depends only on a central data structure called the repository. The repository in turn, has no knowledge of the other subsystems.

Cont…

2. *Model/View/Controller:*

- In the Model/View/Controller (MVC) architecture subsystems are classified into three different types:

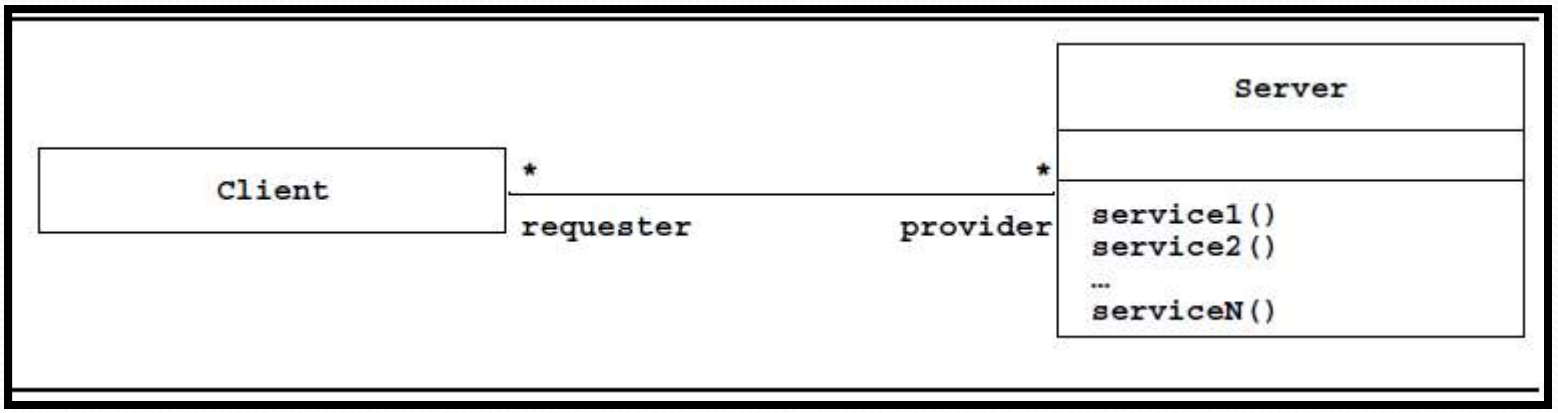
1. **model subsystems** knowledge,
   - It is developed suc… controller subsyste…
   - It maintain the cer…
2. **view subsystems** ar… , and
3. **Controller subsyste…** quence of interactions with t… d send message to the model.



Cont…

3. *Client/server architecture*

- In the client/server architecture a subsystem, the **server**, provides services to instances of other subsystems called the **clients**, which are responsible for interacting with the user.
- The request for a service is usually done via a remote procedure call mechanism or a common object broker.
- The client/server architecture is a generalization of the repository architecture.

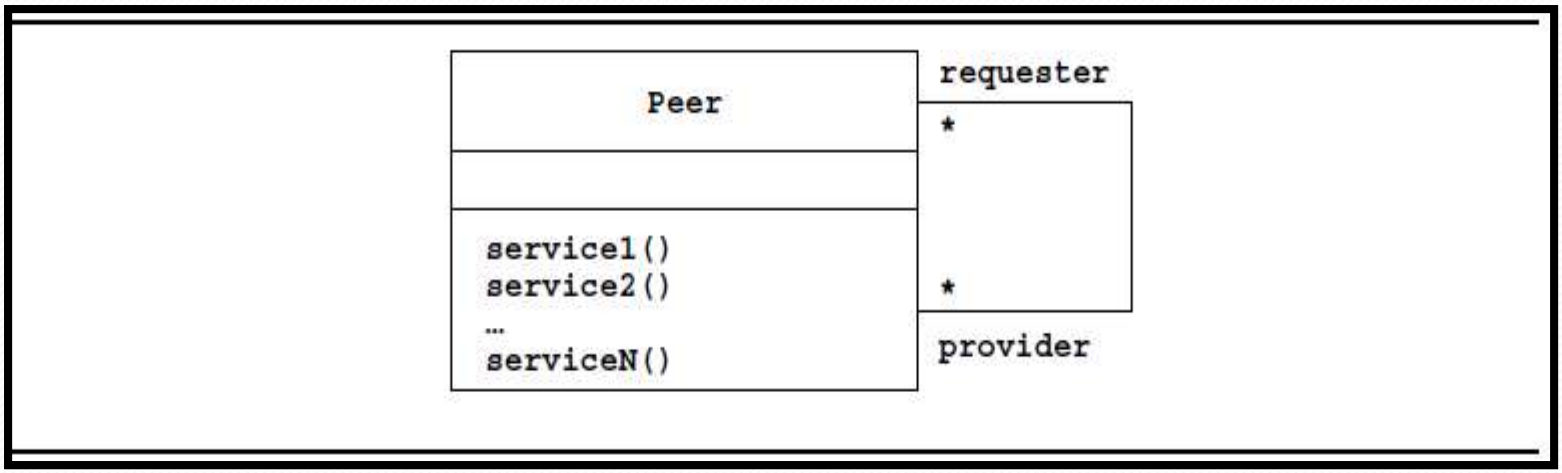


- Fig. Client/server architecture (UML class diagram).

Cont…

4. A peer-to-peer architecture

- It is a generalization of the client/server architecture in which subsystems can act both as client or as servers, in the sense that each subsystem can request and provide services.
- The control flow within each subsystem is independent from the others except for synchronizations on requests.

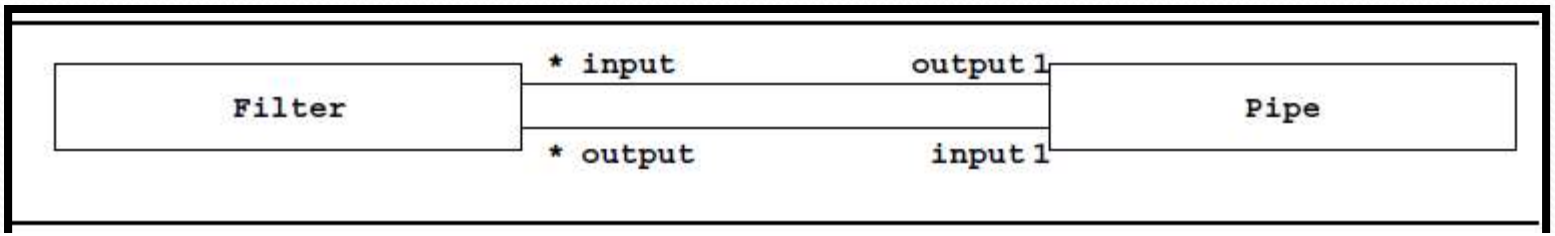


- Fig. Peer-to-peer architecture (UML class diagram).

Cont…

5. *Pipe and filter architecture*

- In this architecture subsystems process data received from a set of inputs and send results to other subsystems via a set of outputs.
- The subsystems are called **filters**, and the associations between the subsystems are called **pipes.**
- Each filter only knows the content and the format of the data received on the input pipes, not the filters that produced them.
- Each filter is executed concurrently and synchronization is done via the pipes.
- Most operating systems and programming languages provide a data stream mechanism; .
- A Filter can have many inputs and outputs. A Pipe connects one of the outputs of a Filter to one of the inputs of another Filter.



6. Service Oriented Architecture

- A business-centric IT architectural approach that supports integrating your business as linked, repeatable business tasks, or services. – IBM
- SOA services are maintained in the organization by a registry which acts as a directory listing. Applications need to look up the services in the registry and invoke the service.

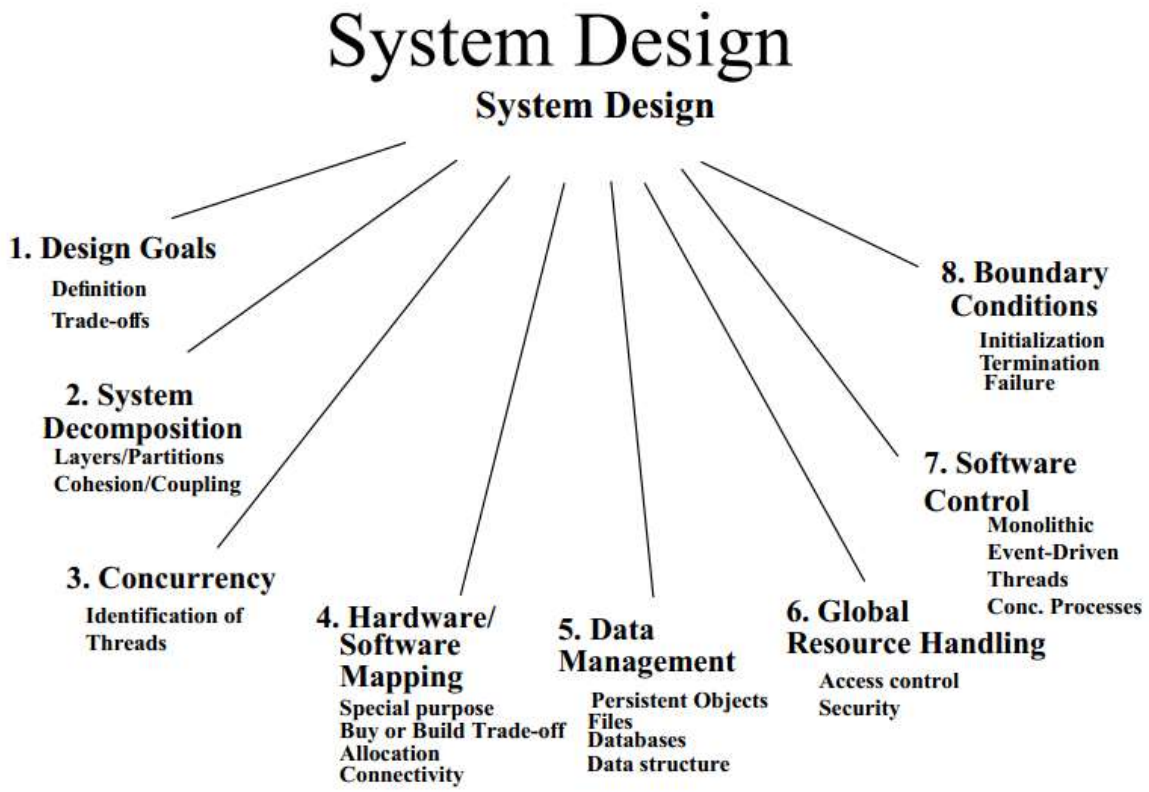
7. Microservice Architecture

- is an architectural development style that allows building applications as a collection of small autonomous services developed for a business domain.
- It is a variant of structural style architecture that helps arrange applications as a loosely coupled service collection.
- The Architecture contains fine-grained services and lightweight protocols.

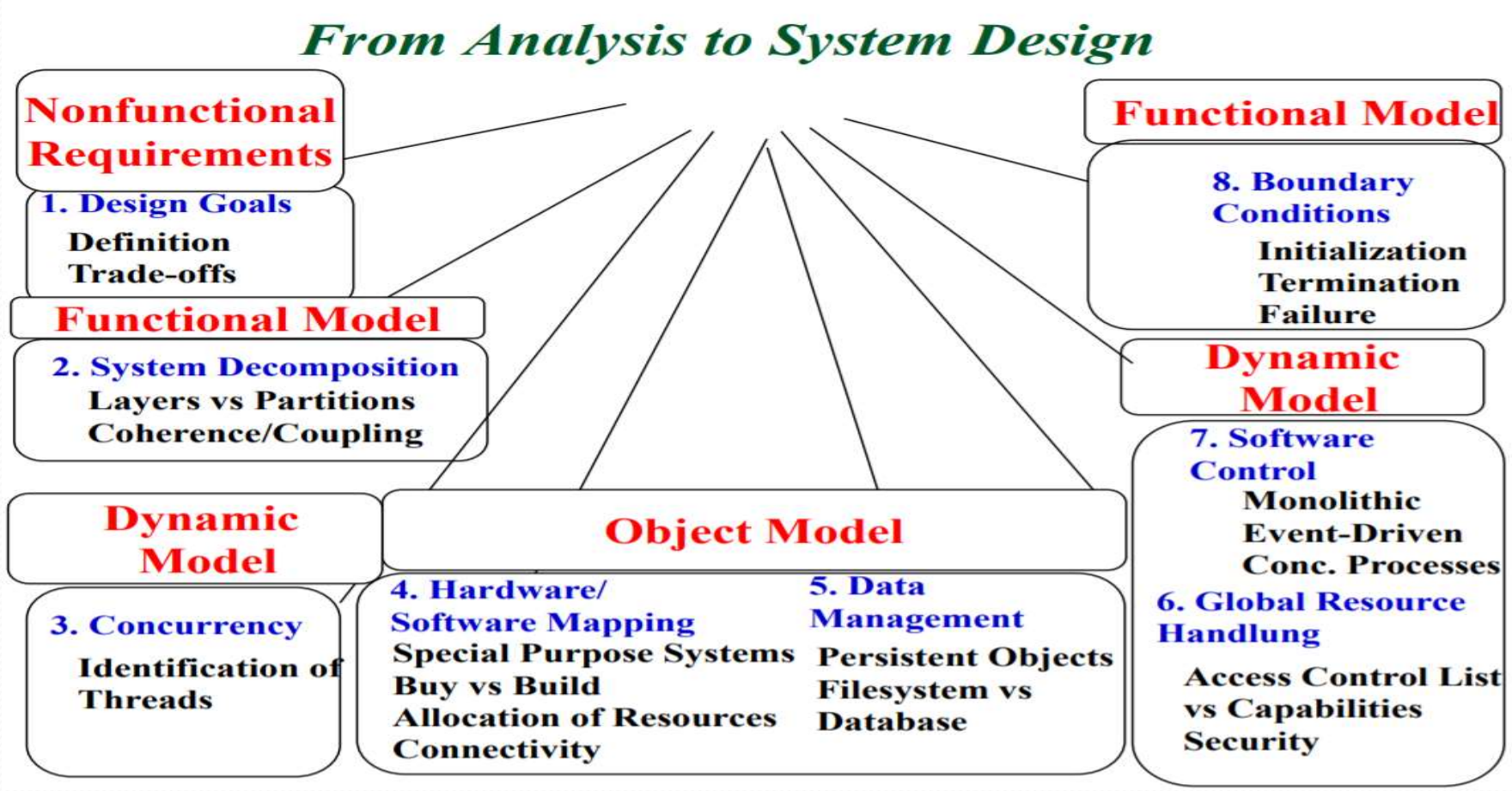
Service Oriented vs Microservice

Parameter	SOA	Microservices
Design type	In SOA, software components are exposed to the outer world for usage in the form of services.	Micro Service is a part of SOA. It is an implementation of SOA.
Dependency	Business units are dependent.	They are independent of each other.
Size of the Software	Software size is larger than any conventional software	The size of the Software is always small in Microservices
Technology Stack	The technology stack is lower compared to Microservice.	Microservice technology stack could be very large
Nature of the application	Monolithic in nature	Full stack in nature
Independent and Focus	SOA applications are built to perform multiple business tasks.	They are built to perform a single business task.
Deployment	The deployment process is time- consuming.	Deployment is straightforward and less time-consuming.
Cost - effectiveness	More cost-effective.	Less cost-effective.
Scalability	Less compared to Microservices.	Highly scalable.

System Design



From Analysis to System Design

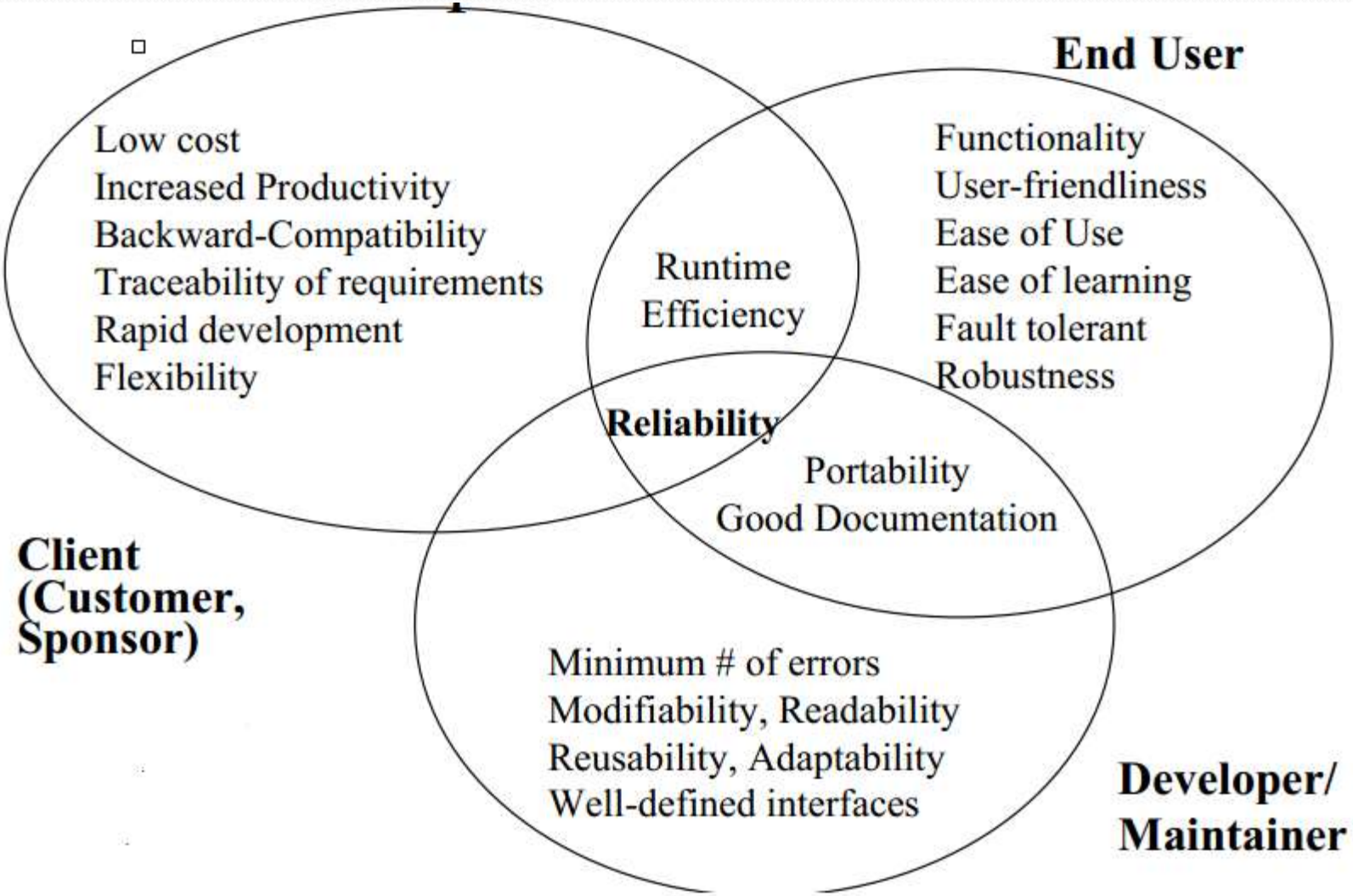


Design Goal

- 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 # of errors
- Readability
- Ease of learning
- Ease of remembering
- Ease of use
- Increased productivity
- Low-cost
- Flexibility

Relationship Between Design Goals



Typical Design Trade-offs

- Functionality vs. Usability
- Cost vs. Robustness
- Efficiency vs. Portability
- Rapid development vs. Functionality
- Cost vs. Reusability
- Backward Compatibility vs. Readability

Typical Design Trade-offs

- Functionality vs. Usability
- Cost vs. Robustness
- Efficiency vs. Portability
- Rapid development vs. Functionality
- Cost vs. Reusability
- Backward Compatibility vs. Readability
- Structure vs. performance
- Centralized vs. distributed
- Sequential vs. concurrent

System Decomposition

- Subsystem (UML: Package)
  - Collection of classes, associations, operations, events and constraints that are interrelated
    - Seed for subsystems: UML Objects and Classes.
- (Subsystem) Service:
  - Group of operations provided by the subsystem
  - Seed for services: Subsystem use cases

Cont.

- Service is specified by Subsystem interface:
  - Specifies interaction and information flow from/to subsystem boundaries, but not inside the subsystem.
  - Should be well-defined and small.
  - Often called API: Application programmer's interface, but this term should used during implementation, not during System Design

Structural Decomposition

- Structural decomposition is concerned with developing a model of the design which shows the dynamic structure i.e. function calls
- This is not necessarily the same as the static composition structure
- The aim of the designer should be to derive design units which are highly cohesive and loosely coupled
- In essence, a data flow diagram is converted to a structure chart

Criteria for subsystem selection:

- Most of the interaction should be within subsystems, rather than across subsystem boundaries (High cohesion).
  - Does one subsystem always call the other for the service?
  - Which of the subsystems call each other for service?
- Primary Question:
  - What kind of service is provided by the subsystems (subsystem interface)?
- Secondary Question:
  - Can the subsystems be hierarchically ordered (layers)?
- What kind of model is good for describing layers and partitions?

Coupling and Cohesion

- Goal: Reduction of complexity while change occurs
- Cohesion measures the dependence among classes

High cohesion: The classes in the subsystem perform similar tasks and are related to each other (via associations)

Low cohesion: Lots of miscellaneous and auxiliary classes, no associations

Cont.

- Coupling measures dependencies between subsystems

High coupling: Changes to one subsystem will have high impact on the other subsystem (change of model, massive recompilation, etc.)

Low coupling: A change in one subsystem does not affect any other subsystem

- Subsystems should have a maximum cohesion and minimum coupling as possible:

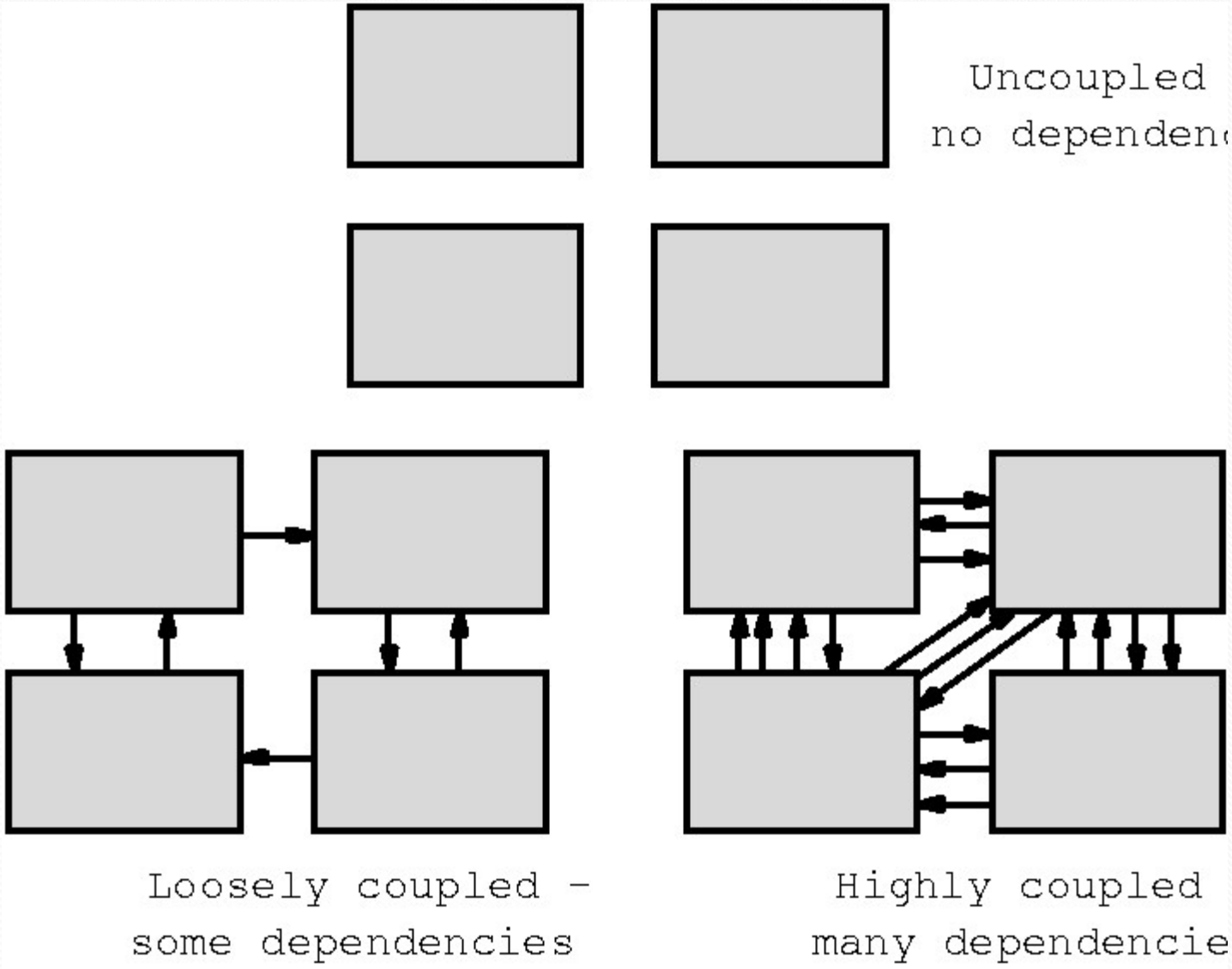
How can we achieve high cohesion?

How can we achieve loose coupling?

Decomposition Guidelines

- The aim of the design process is to identify loosely coupled, highly cohesive functions. Each function should therefore do one thing and one thing only
  - Cohesion – the degree to which a module performs one and only one function.
  - Coupling – the degree to which a module is connected to other modules in the system.

Coupling and Cohesion



Detailed Design

- Concerned with producing a short design specification (minispec) of each function. This should describe the processing, inputs and outputs
- These descriptions should be managed in a data dictionary
- From these descriptions, detailed design descriptions, expressed in a PDL or programming language, can be produced

Object Oriented Design
System design activities: From objects to subsystems
Outline

1. Identify design goals from the nonfunctional requirements
2. Design an initial subsystem decomposition
3. Map subsystems to processors and components
4. Decide storage
5. Define access control policies
6. Select a control flow mechanism
7. Identify boundary conditions

Starting point: Analysis model for a route planning system

- Using **MyTrip**, a driver can plan a trip from a home computer by contacting a trip planning service on the Web (**PlanTrip**). The trip is saved for later retrieval on the server. The trip planning service must support more than one driver.

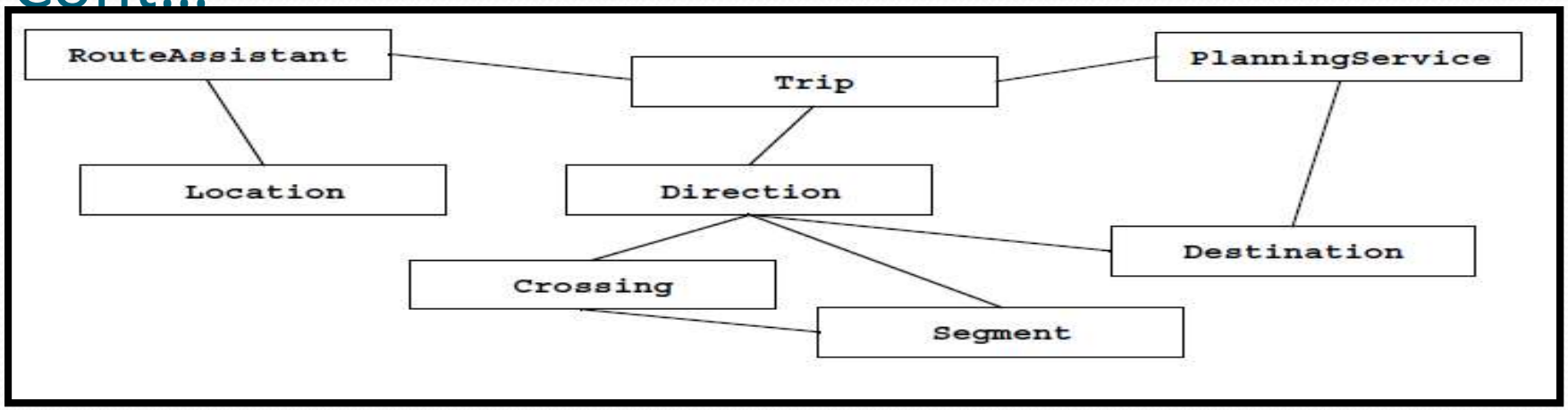
Use case name	`PlanTrip`
Entry condition	1. The `Driver` activates her home computer and logs into the trip planning Web service.
Flow of events	2. Upon successful login, the `Driver` enters constraints for a trip as a sequence of destinations.
	3. Based on a database of maps, the planning service computes the shortest way visiting the destinations in the specified order. The result is a sequence of segments binding a series of crossings and a list of directions.
	4. The `Driver` can revise the trip by adding or removing destinations.
Exit condition	5. The `Driver` saves the planned trip by name in the planning service database for later retrieval.

Cont…

- The driver then goes to the car and starts the trip, while the onboard computer gives directions based on trip information from the planning service and her current position indicated by an onboard GPS system (**ExecuteTrip**)

Use case name	`ExecuteTrip`
Entry condition	1. The `Driver` starts her car and logs into the onboard route assistant.
Flow of events	2. Upon successful login, the `Driver` specifies the planning service and the name of the trip to be executed.
	3. The onboard route assistant obtains the list of destinations, directions, segments, and crossings from the planning service.
	4. Given the current position, the route assistant provides the driver with the next set of directions.
Exit condition	5. The `Driver` arrives to destination and shuts down the route assistant.

Cont…



- **Crossing** :A Crossing is a geographical point were a driver can choose between several Segments.
- **Destination** A Destination represents a location where the driver wishes to go.
- **Direction** Given a Crossing and an adjacent Segment, a Direction describes in natural language terms how to steer the car onto the given Segment.
- **Location** A Location is the position of the car as known by the onboard GPS system or the number of turns of the wheels.
- **PlanningService** A PlanningService is a Web server that can supply a trip, linking a number of destinations in the form of a sequence of crossings and segments.
- **RouteAssistant** A RouteAssistant gives Directions to the driver, given the current Location and upcoming Crossing.
- **Segment** A Segment represents the road between two Crossings.
- **Trip** A Trip is a sequence of Directions between two Destinations.

Cont…

- **Nonfunctional requirements for MyTrip**

1. MyTrip is in contact with the **PlanningService** via a wireless modem. It can be assumed that the wireless modem functions properly at the initial destination.
2. Once the trip has been started, MyTrip should give correct directions even if **modem fails** to maintain a connection with the PlanningService.
3. MyTrip should minimize **connection time** to reduce operation costs.
4. Replanning is possible only if the connection to the PlanningService is possible.
5. The PlanningService can support **at least 50** different drivers and **1000 trips**.

1. Identifying Design Goals

- Is the **first step** of system design, It **identifies the qualities** that our system should focus on; which are mostly inferred from the **nonfunctional requirements** or from **the application domain.**
- **Example: considering the** nonfunctional requirements for MyTrip

Design goals for MyTrip

- **Reliability:** MyTrip should be reliable [generalization of nonfunctional requirement 2].
- **Fault Tolerance:** MyTrip should be fault tolerant to loss of connectivity with the routing service [rephrased nonfunctional requirement 2].
- **Security:** MyTrip should be secure, i.e., not allow other drivers or nonauthorized users to access another driver's trips [deduced from application domain].
- **Modifiability:** MyTrip should be modifiable to use different routing services [**anticipation of change by developers**].

Cont…

- list a number of possible design criteria.
  1. *Performance*
  2. *dependability*
  3. *cost*
  4. *maintenance and*
  5. *end user criteria.*

1. **Performance criteria:** include the speed and space requirements imposed on the system.

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?

Cont..

2. **Dependability criteria:** determine how much effort should be expended in **minimizing system crashes** and their consequences. How often can the system crash? How **available** to the user should the system be? Are there **safety issues** associated with system crashes? Are there **security risks** associated with the system environment?

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.

Cont…

3. **Cost criteria:** include the cost to develop the system, to deploy it, and to administer it.

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.

Cont…

4. **Maintenance criteria:** determine how difficult it is to change the system after deployment. How easily can **new functionality** be added? How easily can existing functions be **revised**? Can the system be **adapted to a different application domain**? How much effort will be required to port the system to a different platform?

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?

Cont…

5. **End user criteria:** include qualities that are desirable from a users' point of view that have not yet been covered under the **performance** and **dependability** 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?

- **N.B.** Developers need to prioritize design goals and trade them off against each other. E.g it is unrealistic to develop software that is **safe, secure, and cheap. Some of the common trade-off are:**
  - Space vs. speed
  - Delivery time vs. functionality
  - Delivery time vs. quality
  - Delivery time vs. staffing

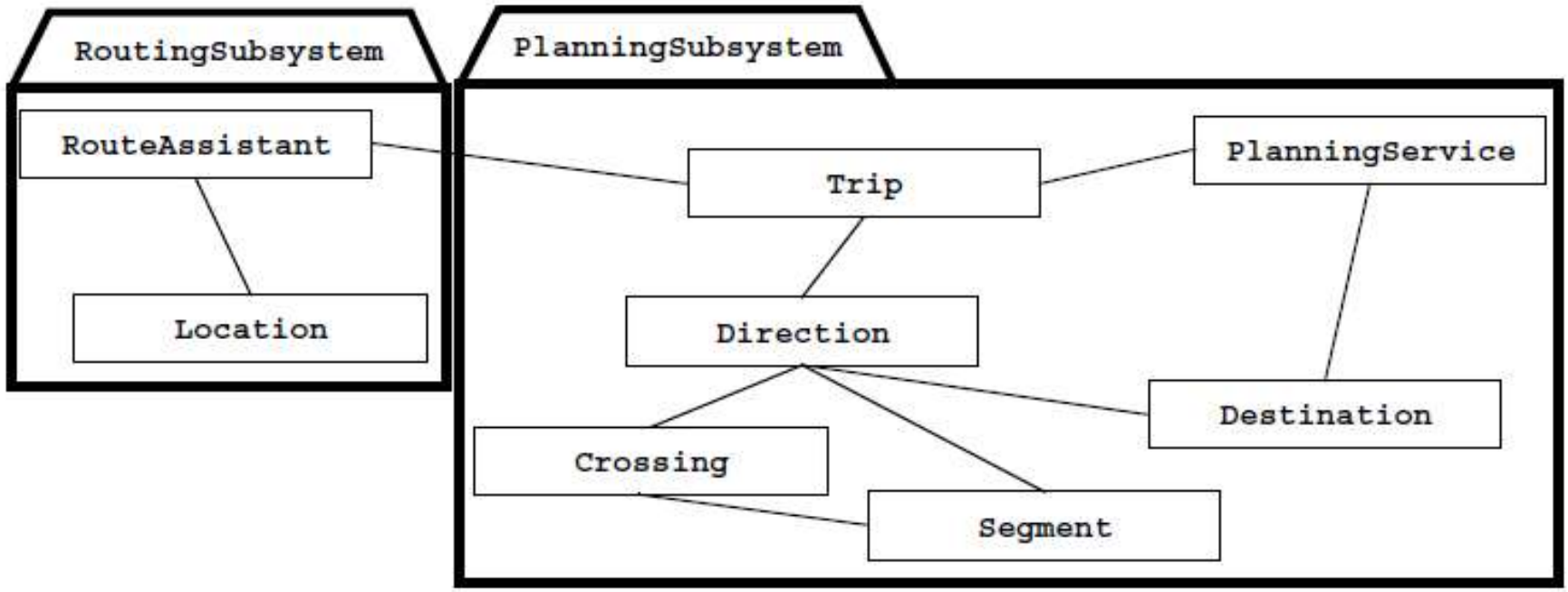
2. Identifying subsystems

- Finding subsystems during system design has many similarities to finding objects during analysis: ***It is a volatile activity driven by heuristics.***

- The initial subsystem decomposition should be derived from the **functional requirements.**

- ***For example :*** in the My-Trip system, we identify two major groups of objects:
  1. PlanTrip use cases
  2. ExecuteTrip use case.
  - ***N.B.*** The Trip, Direction, Crossing, Segment, and Destination classes are shared between both use cases.

Cont…



PlanningSubsystem	The `PlanningSubsystem` is responsible for constructing a `Trip` connecting a sequence of `Destinations`. The `PlanningSubsystem` is also responsible for responding to replan requests from `RoutingSubsystems`.
RoutingSubsystem	The `RoutingSubsystem` is responsible for downloading a `Trip` from the `PlanningService` and executing it by giving `Directions` to the driver based on its `Location`.

Fig. Initial subsystem decomposition for MyTrip (UML class diagram).

Cont…

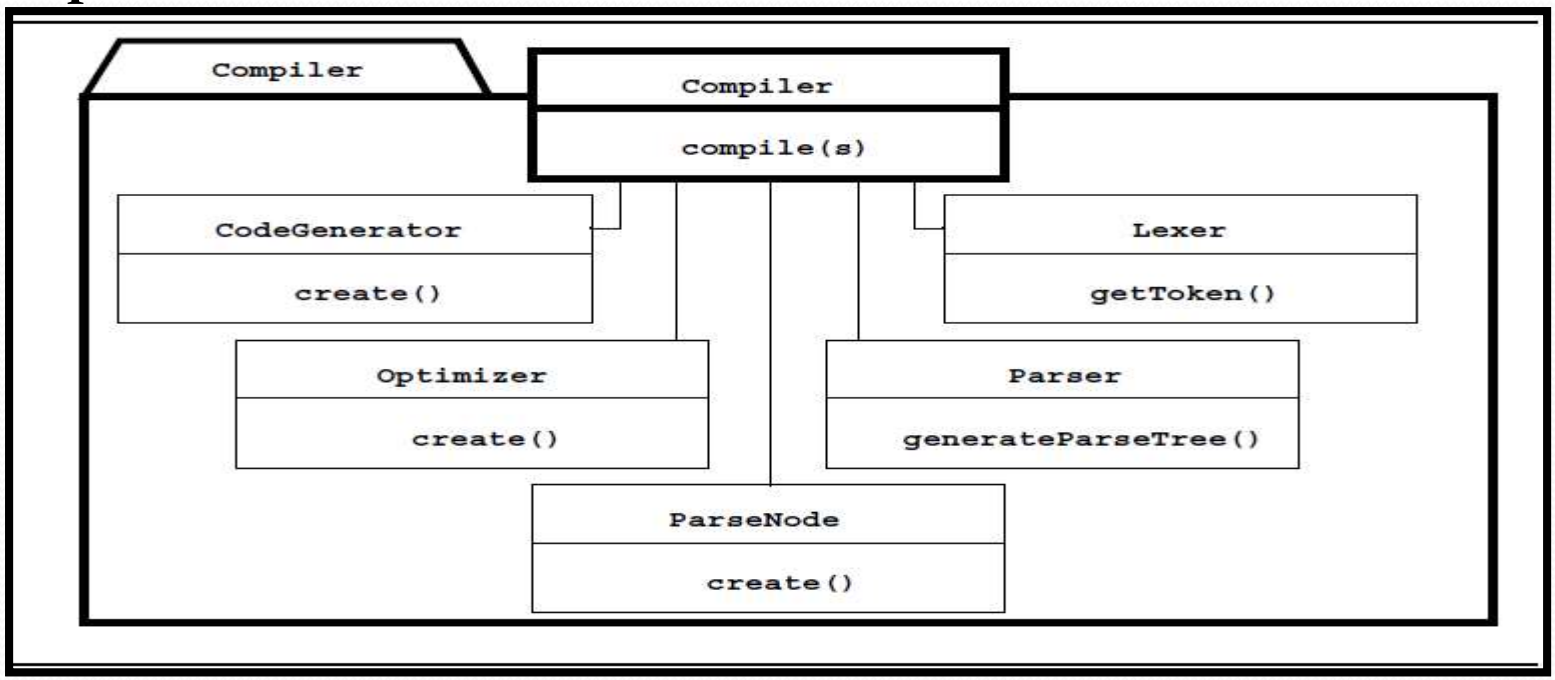
- Another heuristic for subsystem identification is to keep functionally related objects together.

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

Cont..

- ***Encapsulating subsystems***
  - Subsystem decomposition reduces the complexity of the solution domain by minimizing dependencies among classes.
  - The **Facade pattern** [Gamma et al., 1994] allows us to further reduce dependencies between classes by encapsulating a subsystem with a **simple, unified interface**.



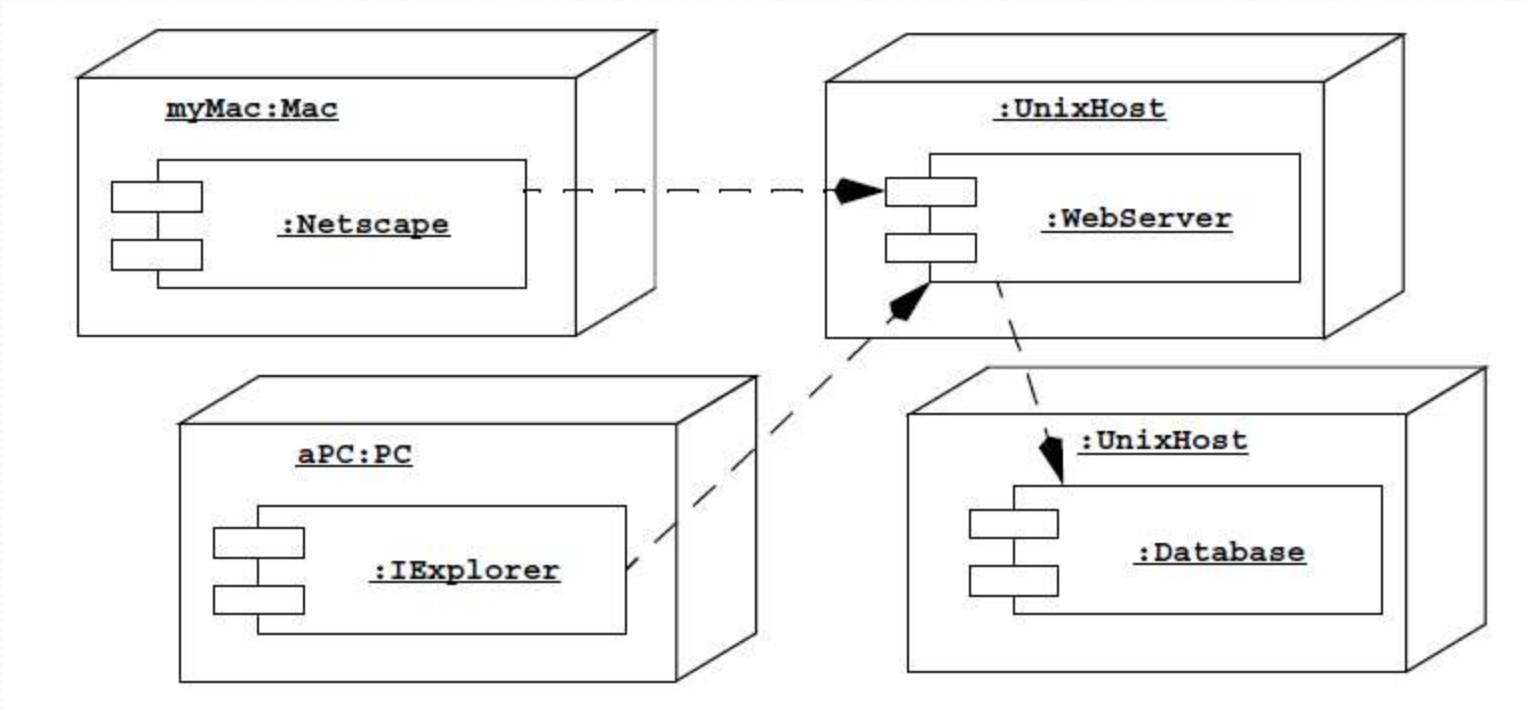
3. Mapping subsystems to processors and components

- ***Selecting a hardware configuration and a platform***
  - Many web based systems need multiple computers and interconnect multiple distributed users.
  - These computers are modeled as **nodes** in UML **deployment diagrams.**
  - Select a hardware configuration also includes selecting a **virtual machine** onto which the system should be built. It includes OS, s/w (DBMS, communication packages).

UML Deployment Diagrams

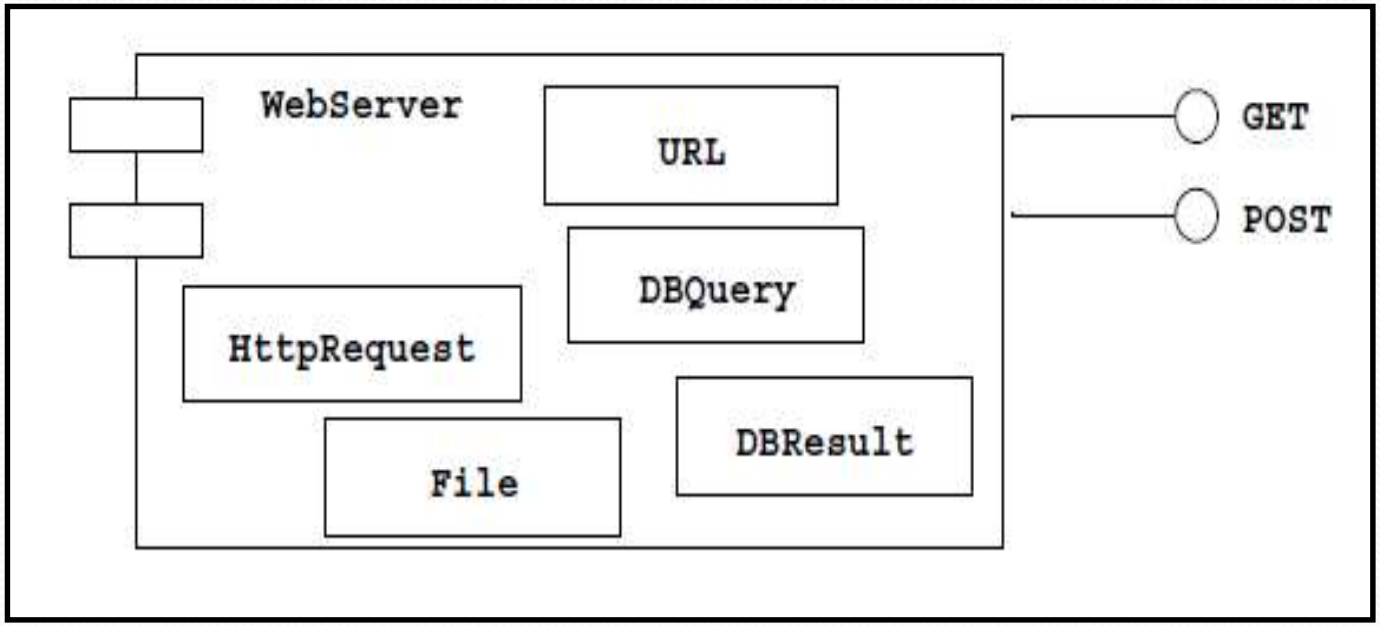
- **UML deployment diagrams:** are used to depict the relationship among **run-time components and hardware** nodes.
  - Components are **self-contained** entities that provide services to other components or actors. E. g. Web server, provides services to Web browsers.
  - In UML deployment diagrams, nodes are represented by **boxes** containing component icons.
  - Dependencies between components are represented by **dashed arrows.**

Cont…



- **Fig.** A UML deployment diagram representing the allocation of components to different nodes and the dependencies among components. Web browsers on PCs and Macs can access a WebServer that provides information from a Database.

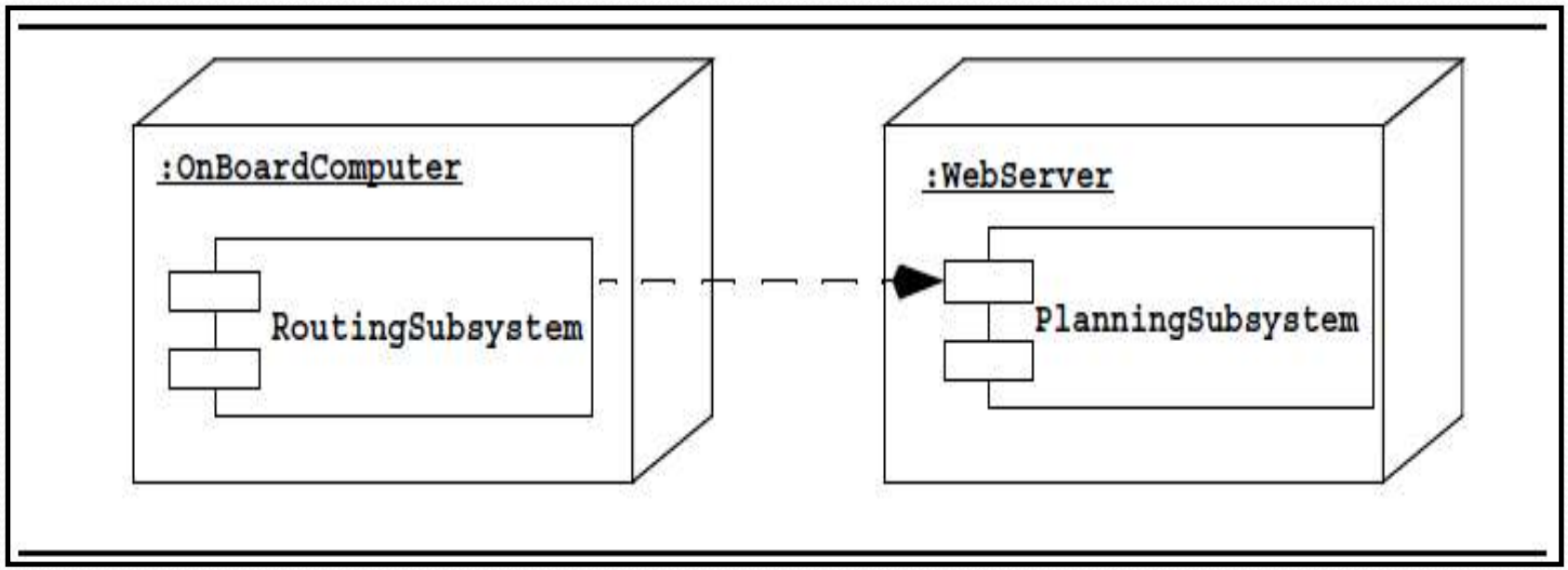
Cont…



- **Fig**. Refined view of the WebServer component (UML deployment diagram). The WebServer component provides two interfaces to browsers: A browser can either request the content of a file referred by a URL (GET) or post the content of a form (POST). The WebServer component contains five classes: URL, HttpRequest, DBQuery, File, and DBResult.

Example In MyTrip

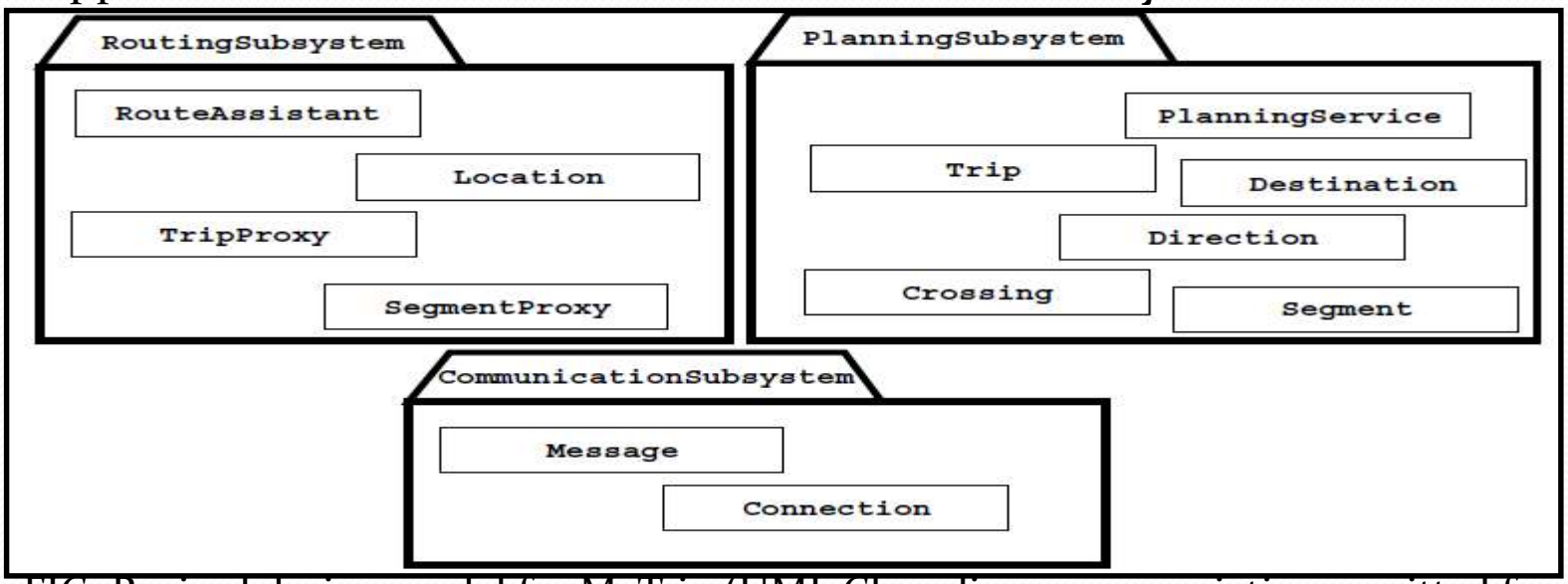
- **Example** In MyTrip, we deduce from the requirements that **PlanningSubsystem** and **RoutingSubsystem** run on two different nodes:
- In the **MyTrip** subsystem We select a **Unix machine** as the virtual machine for the :**WebServer** and the **Web browsers on onboard computer**



- **Fig.** Allocation of MyTrip subsystems to hardware (UML deployment diagram). RoutingSubsystem runs on the OnBoardComputer while PlanningSubsystem runs on a WebServer.

Cont…

- ***Allocating objects and subsystems to nodes***
  - After identification of virtual machines , objects and subsystems are assigned to **nodes**. This often triggers the identification of new objects and subsystems for **transporting data among the nodes**.
  - **Example:** in **MyTrip** system both **RoutingSubsystem** and **PlanningSubsystem** share the objects. thus we create new subsystem to support this communication: **coommunicationSubsystem.**



  - FIG. Revised design model for MyTrip (UML Class diagram, associations omitted for clarity).

4. Defining Persistent Data Stores

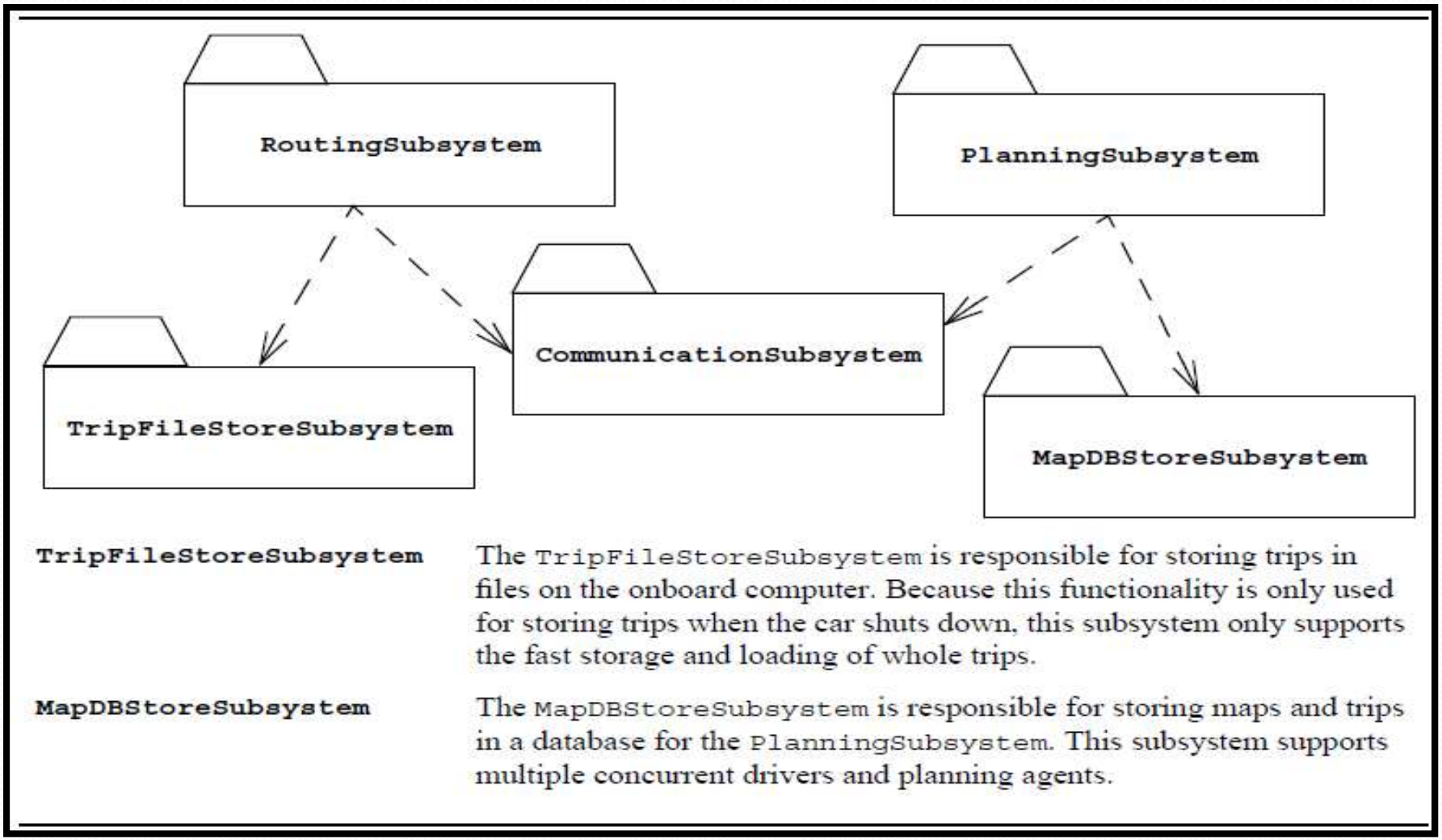
- Where and how data is stored in the system impacts the system decomposition.

- The selection of a specific database management system can also have implications on the overall control strategy and concurrency management.

- we first need to identify which objects need to be persistent. The persistency of objects is directly inferred from the application domain.
  - E.g. In Mytrip - current trip in Rroutngsubsystem-file, location X

    - entire trip in PlanningSubsystem-database

Cont…



TripFileStoreSubsystem	The `TripFileStoreSubsystem` is responsible for storing trips in files on the onboard computer. Because this functionality is only used for storing trips when the car shuts down, this subsystem only supports the fast storage and loading of whole trips.
MapDBStoreSubsystem	The `MapDBStoreSubsystem` is responsible for storing maps and trips in a database for the `PlanningSubsystem`. This subsystem supports multiple concurrent drivers and planning agents.

- Fig. Subsystem decomposition of MyTrip after deciding on the issue of data stores (UML class diagram, packages collapsed for clarity).

5. Defining access control

- In multiuser systems, different actors have access to different functionality and data. We modeled these distinctions by associating different **use cases** to different actors.
- During system design, we model access by examining the object model, by **determining which objects are shared among actors**, and by defining how actors can control access.
- Depending on the **security requirements** on the system, we also define **how actors are authenticated** to the system (i.e., how actors prove to the system who they are) and how selected data in the system should be **encrypted**.

Cont…

CommunicationSubsystem	The `CommunicationSubsystem` is responsible for transporting `Trips` from the `PlanningSubsystem` to the `RoutingSubsystem`. *The CommunicationSubsystem uses the Driver associated with the Trip being transported for selecting a key and encrypting the communication traffic.*
PlanningSubsystem	The `PlanningSubsystem` is responsible for constructing a `Trip` connecting a sequence of `Destinations`. The `PlanningSubsystem` is also responsible for responding to replan requests from `RoutingSubsystem`. *Prior to processing any requests, the PlanningSubsystem authenticates the Driver from the RoutingSubsystem. The authenticated Driver is used to determine which Trips can be sent to the corresponding RoutingSubsystem.*
Driver	*A Driver represents an authenticated user. It is used by the CommunicationSubsystem to remember keys associated with a user and by the PlanningSubsystem to associate Trips with users.*

- Fig. Revisions to the design model stemming from the decision to authenticate Drivers and encrypt communication traffic. The text added to the model is in *italics*.

Cont…

- **Access matrix:** Is a table where the *rows* of the matrix represents the **actors** of the system. The *columns* represent **classes** whose access we control.
- An entry (class, actor) in the access matrix is called an **access right** and lists the **operations** that can be executed on instances of the class or by the actor.

Objects Actors	Corporation	LocalBranch	Account

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

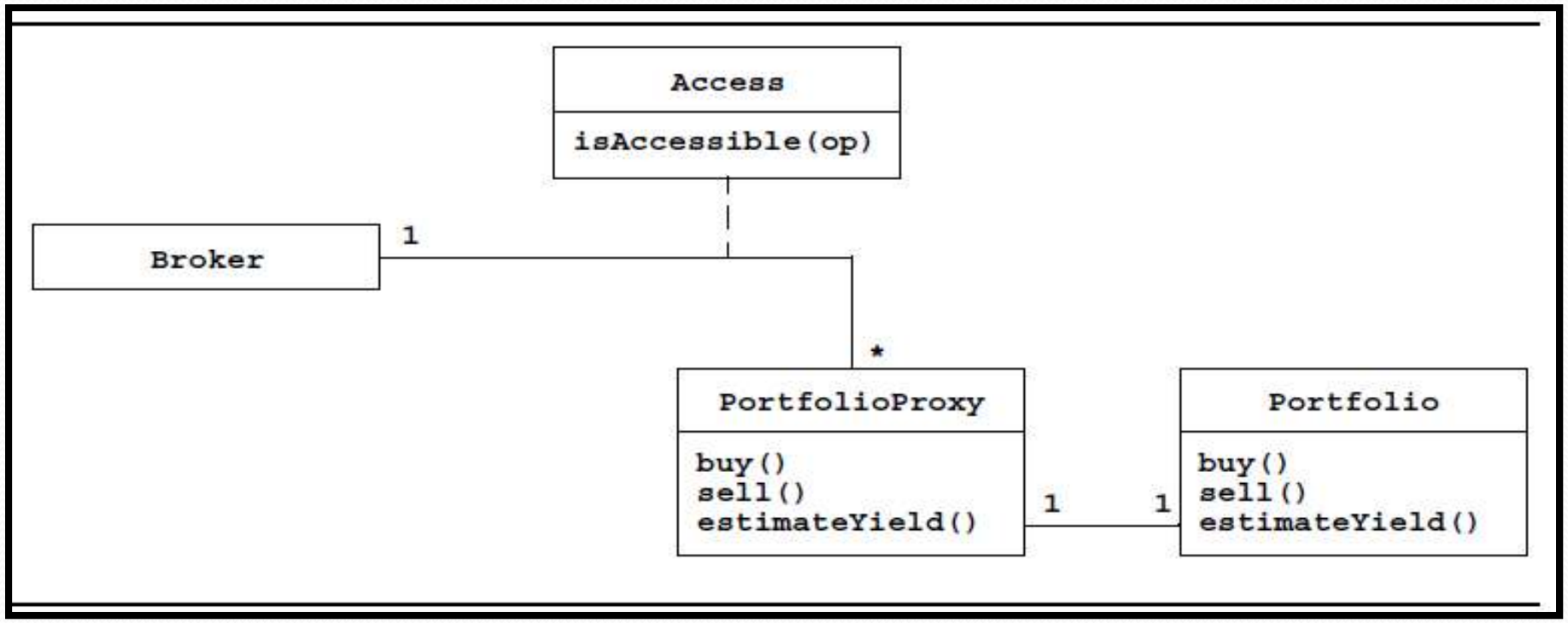
`Manager`		`lookupLocalAccount()`	`postSmallDebit()`
			`postSmallCredit()`
			`postLargeDebit()`
			`postLargeCredit()`
			`examineBalance()`
			`examineHistory()`

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

- Fig. Access matrix for a banking system.

Cont…

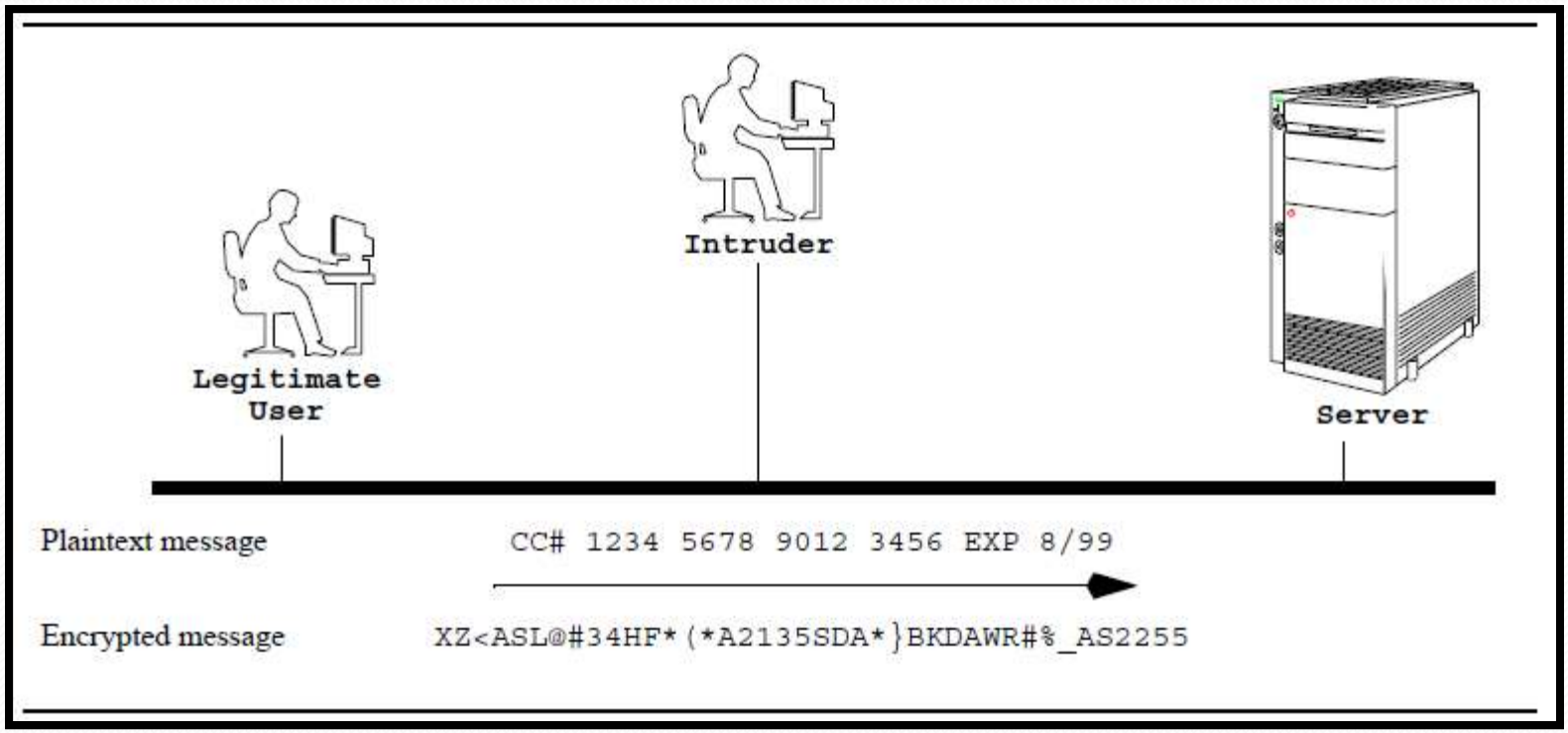
- An access matrix only represents **static access control.** This means that access rights can be modeled as **attributes** of the objects of the system.
- In a model if the access rights is allocated dynamically in the system, this type of access is called **dynamic access control.**



- **Fig.** Dynamic access implemented with a protection Proxy.

Cont…

- In an environment where resources are shared among **multiple users, authentication is usually not sufficient**.
- **Encryption** is used to prevent such unauthorized accesses. Using an encryption algorithm, we can translate a message, called **plaintext,** into a encrypted message, called a **ciphertext,**



6. Designing the global control flow

- **Control flow :** is the sequencing of **actions in a system**.
- In an object-oriented systems, sequencing actions includes deciding **which operations** should be executed and **in which order.**
  - These decisions are based on **external events** generated by an **actor** or on the passage **of time**.
  - Control flow is a design problem.

Three types of control flow

1. **Procedure-driven control:** Operations wait for input whenever they need data from an actor.
   - This kind of control flow is mostly used procedural languages.
   - It introduces difficulties when used with object-oriented languages. As the sequencing of operations is distributed among a large set of objects.

Cont…

2. **Event-driven control:** A main loop waits for an external event.
   - Whenever an event becomes available, it is **dispatched to the appropriate object**, based on information **associated with the event**.
   - This kind of control flow has the advantage of leading to a **simpler structure** and to centralizing all input .

3. **Thread:** are the concurrent variation of procedure-driven control: The system can create an arbitrary number of threads, each responding to a different event.
   - If a thread needs additional data, it waits for input from a specific actor.

7.Identifying boundary conditions

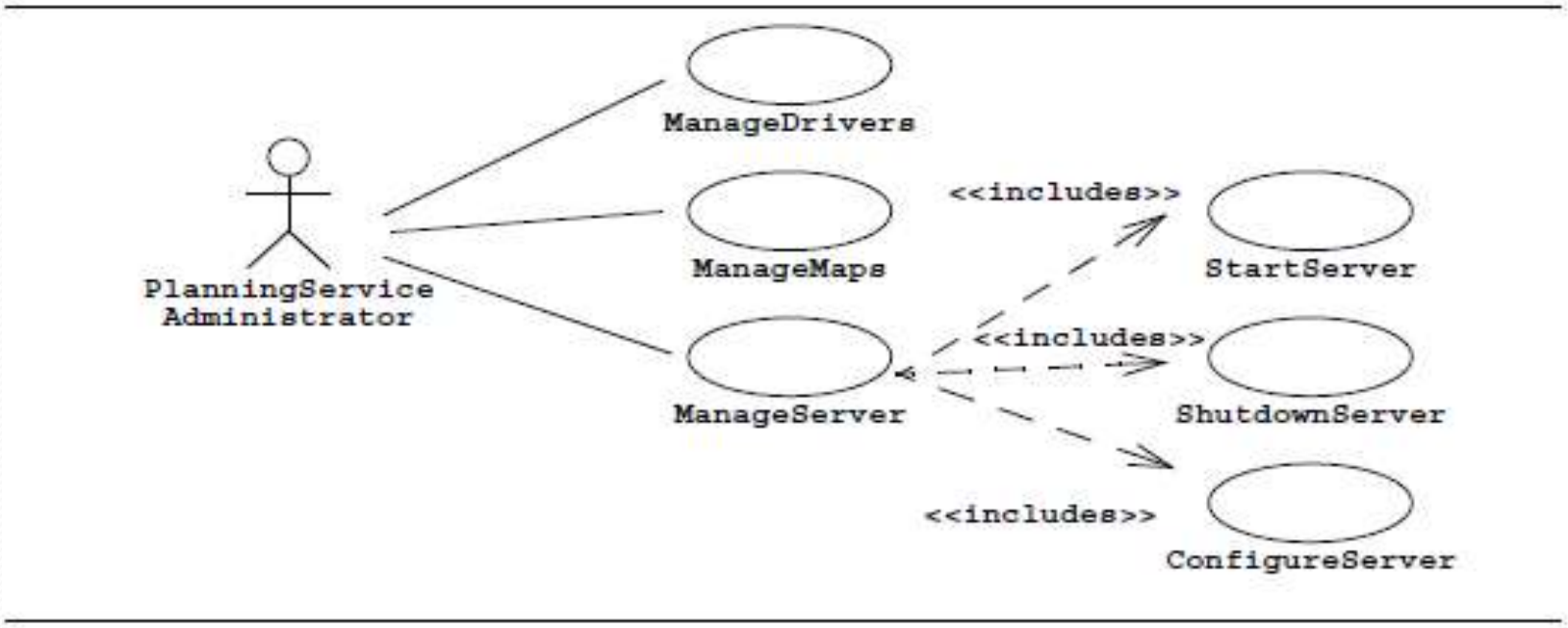
- It is deciding how the system is **started, initialized, and shut down**, and how to deal with **major failures**, such as data corruption, whether they are caused by a **software error** or a **power outage**.

- Considering the MyTrip system our system design doesn't address the following questions:
  - How MyTrip is initialized.
  - How are maps loaded into the PlanningService?
  - How is MyTrip installed in the car?
  - How does MyTrip know which PlanningService to connect to?
  - How are drivers added to the PlanningService?

- We discover a set of **use cases** that has not been specified. We call these the **system administration use cases,** which specify the system during startup and shutdown phases.

Cont…

- Modify the analysis model for MyTrip to include the administration use cases : **ManageDrivers,** to add, remove, and edit drivers; **ManageMaps**, to add, remove, and update maps used to generate trips; and **ManageServer,** to perform routine configuration, start-up, and shutdown. StartServer, part of ManageServer.



- Fig. Administration use cases for MyTrip (UML use case diagram).

Documenting system design

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

Glossary

Thank You!

Q?

Quiz-1 -10%

1. Describe at least two Architectural style
2. Describe at list two subsystem of your course project
3. Describe basic process of system decomposition with your course project example