{

Software Engineering

Eng. Mohammed Hassan

Chapter 1

Introduction

Topics Covered

- **Professional software development**
  - What is meant by software engineering.
- **Software engineering ethics**
  - A brief introduction to ethical issues that affect software engineering.
- **Case studies**
  - An introduction to three examples that are used in later chapters.

Software engineering

- The economies of all industrial nations are dependent on software.
- Software engineering is concerned with theories, methods and tools for professional software development.
- Expenses on software represents a significant part in all developed countries.

Software costs

The costs of software on a PC are often greater than the hardware cost.

Software costs more to maintain than it does to develop. For systems with along life, maintenance costs may be several times development costs.

Software engineering is concerned with cost effective software development.

Software products

- Generic products:
  - Stand alone systems that are marketed and sold to any customer who wishes to buy them.
  - Examples: PC software such as graphics programs, project management tools, programming languages php, c++.
- Customized products
  - Software that is specially made by a specific customer to meet their own needs.
  - Examples: specific company system, factory system.

Software products

- Generic products:
  - The specification of what the software should do is owned by the software developer and decisions on software change are made by the developer.
- Customized products
  - The specification of what the software should do is owned by the customer for the software and they make decisions on software changes that are required.

Software engineering diversity

- There are many different types of software system and there is no universal set of software techniques that is applicable to all of these.

- The software engineering methods and tools used depend on the type of application being developed, the requirements of the customer and the background of the development team.

Application types

1. Stand alone applications.
2. Interactive transaction based applications.
3. Embedded control systems.
4. Batch processing systems.
5. Entertainment systems.
6. Systems for modeling and simulation.
7. Data collection systems.
8. Systems of systems.

Software engineering ethics

- Software engineer must behave in an honest and ethically responsible way if they are to be respected as professionals.

- Ethical behavior is more than simply upholding the law but involves following a set of principles that are ethically correct.

Issues of professional responsibility

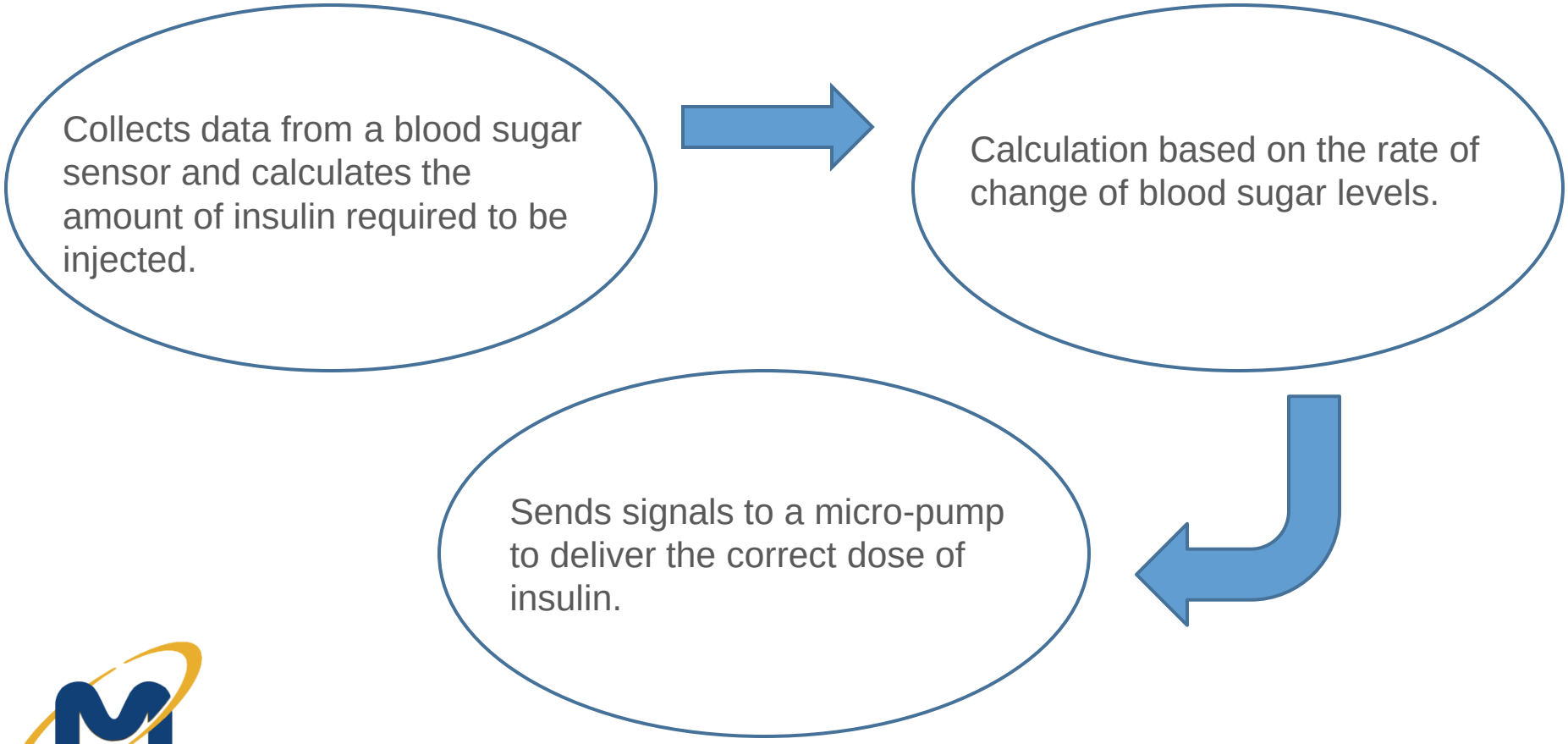
- Confidentiality
  - Engineers should normally respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
- Ability
  - Engineers should not misrepresent their level of ability. They should not knowingly accept work which is out with their ability.

Issues of professional responsibility

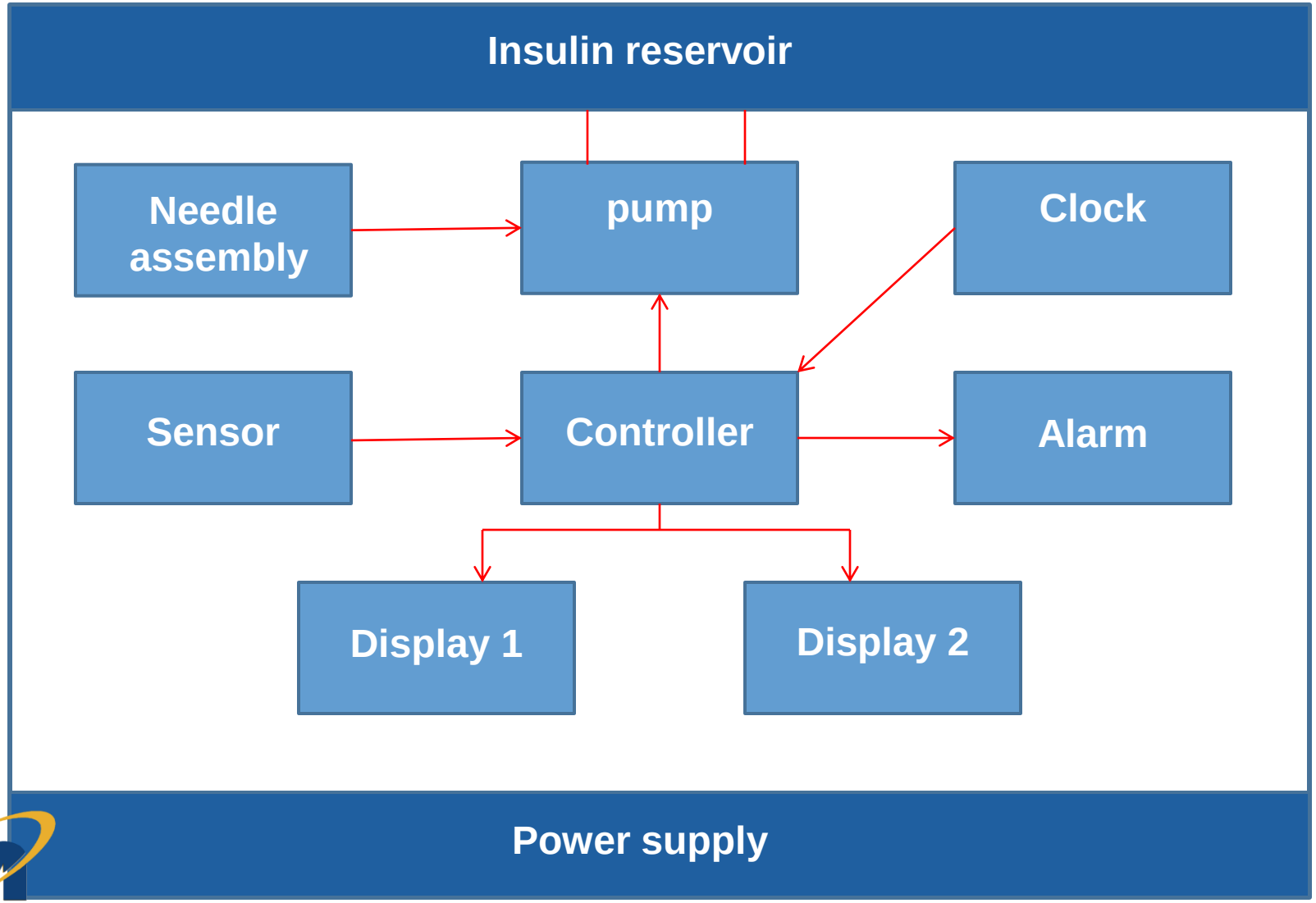
- Intellectual property rights
  - Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected.
- Computer misuse
  - Software engineers should not use their technical skills to misuse other people's computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses).

Case studies

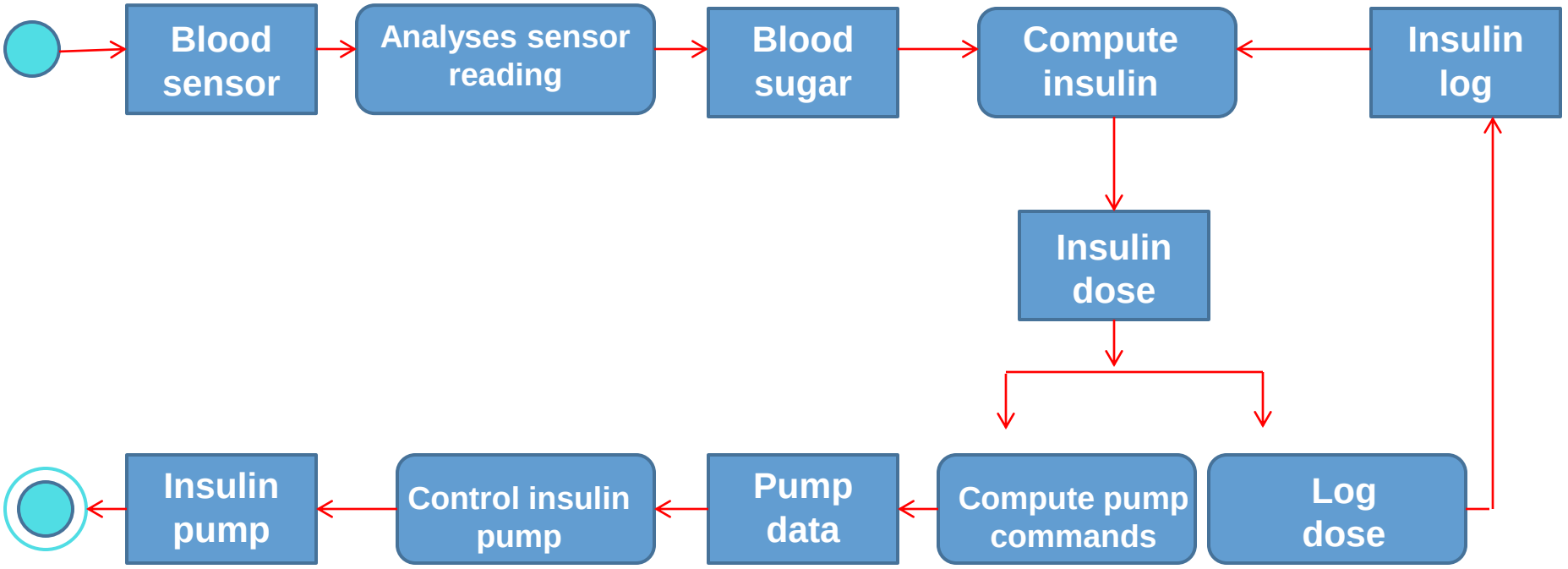
- **Insulin pump control system**



Insulin pump hardware architecture



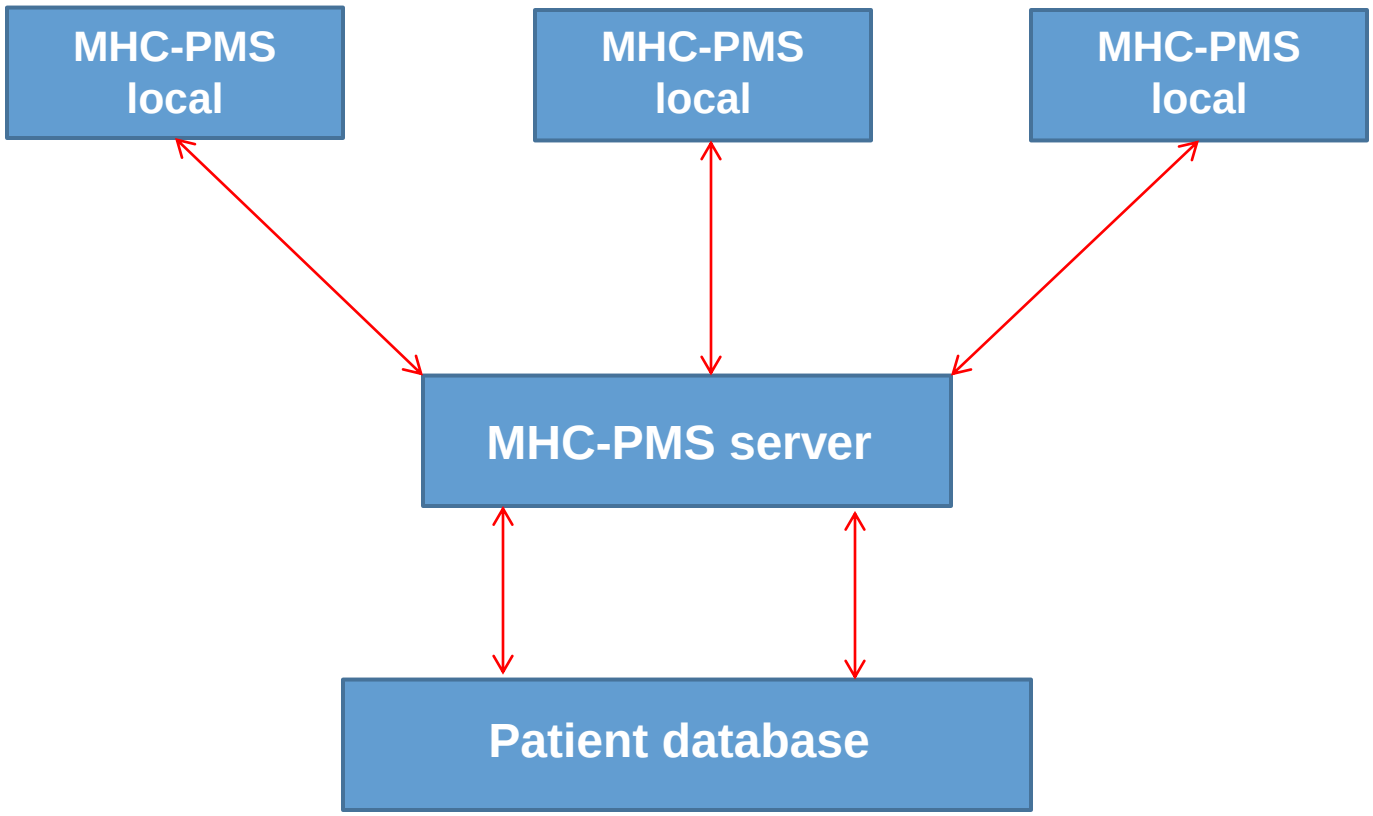
Activity model of the insulin pump



A patient information system for mental health care

- A patient information system to support mental health care is a medical information system that maintains information about patients suffering from mental health problems and the treatments that they have received.
- Most mental health patients do not require dedicated hospital treatment but need to attend specialist clinics regularly where they can meet a doctor who has detailed knowledge of their problems.
- The MHC-PMS(Mental Health Care-Patient Management System) is an information system that is intended for use in clinics.
- When the local systems have secure network access, they use patient information in the database but they can download and use local copies of patient records when they are disconnected.

The organization of the MHC-PMS



MHC-PMS concerns

Privacy

- It is essential that patient information is confidential and is never disclosed to anyone apart from authorized medical staff and the patient themselves.

Safety

- Some mental illnesses cause patients to become danger to other people. The system should warn medical staff about potentially dangerous patients.
- The system must be available when needed.

Chapter 2

Software Processes

Eng. Mohammed Hassan

Topics Covered

- **Software process models**
- **Process activities**
- **Coping with change**

The software process

- **A structured set of activities required to develop a software system.**
- **Many different software processes but all involve:**
  - Documenting external and internal conditions.
  - Analysis.
  - Design.
  - Implementation and write code.
  - Testing.
  - Authentication.
  - Development and maintenance.

Plan-driven and agile (alter) processes

- Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan.

- In agile (alert) processes, Planning is incremental and it is easier to change the process to reflect changing customer requirements.

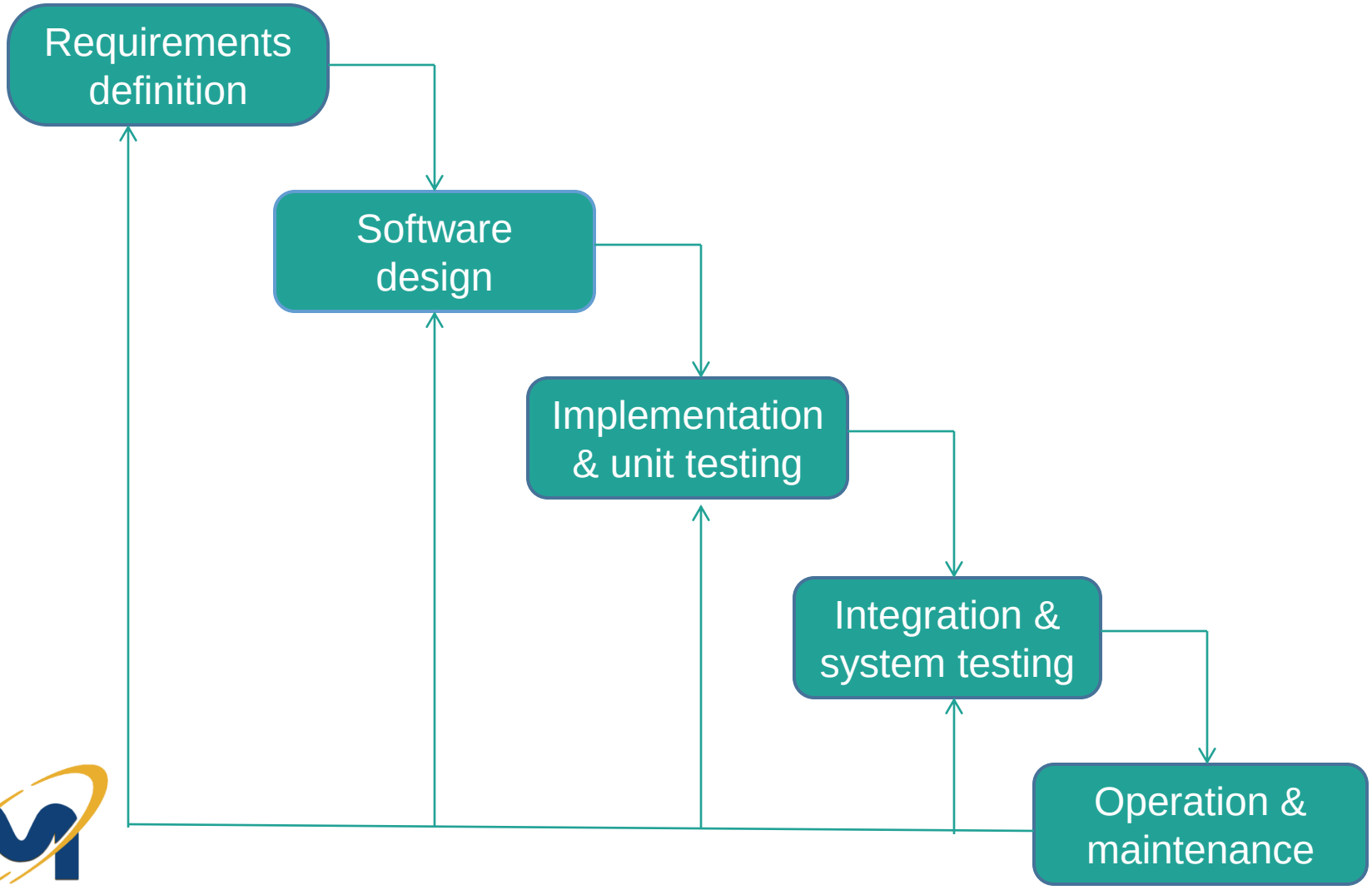
- In practice, most practical processes include elements of both plan driven and agile approaches.
- There are no right or wrong software processes.

Software process models

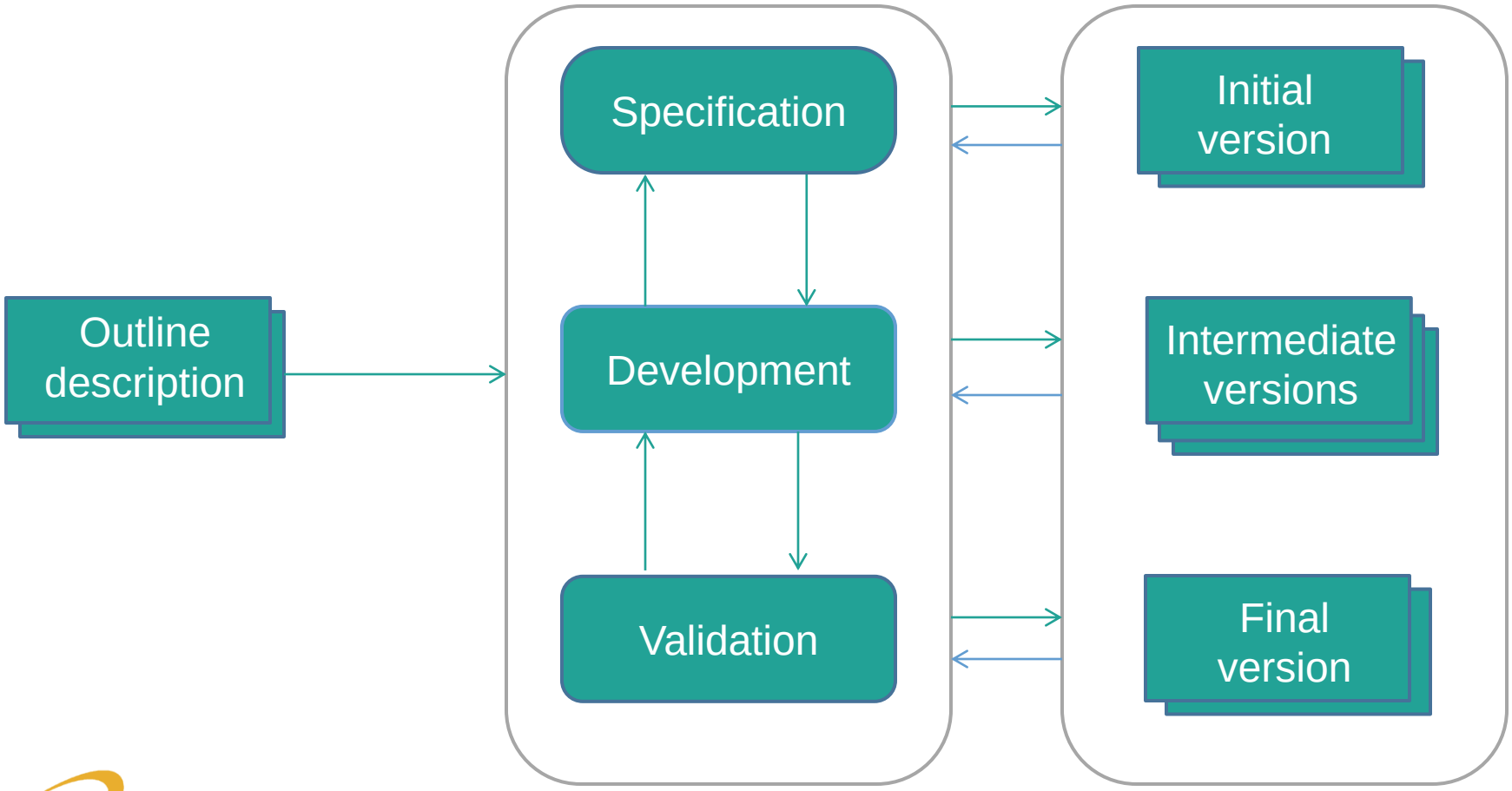
- **The waterfall model**
- **Incremental development**
- **Reuse-oriented software engineer**

In practice, most large systems are developed using a process that incorporates elements from all of these models.

The waterfall model

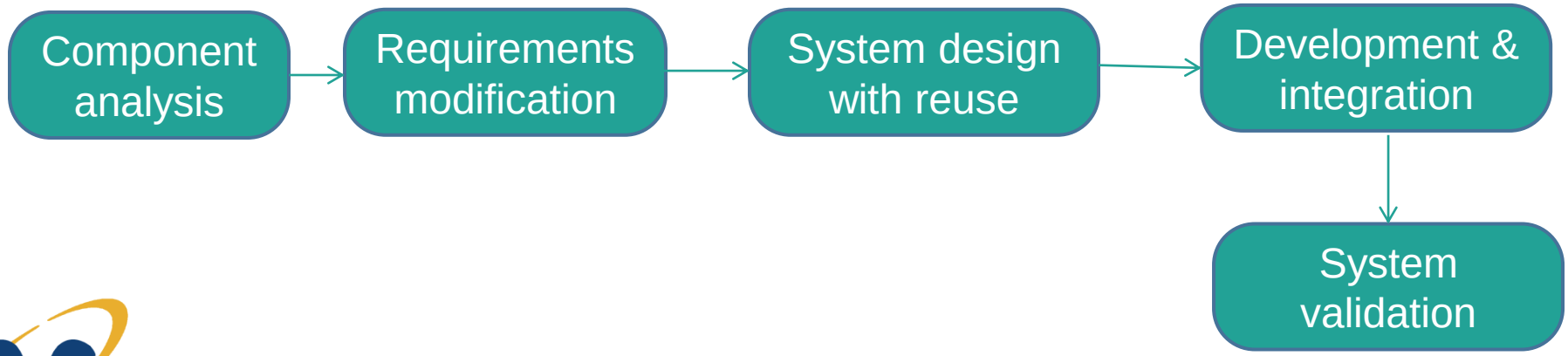


Incremental development



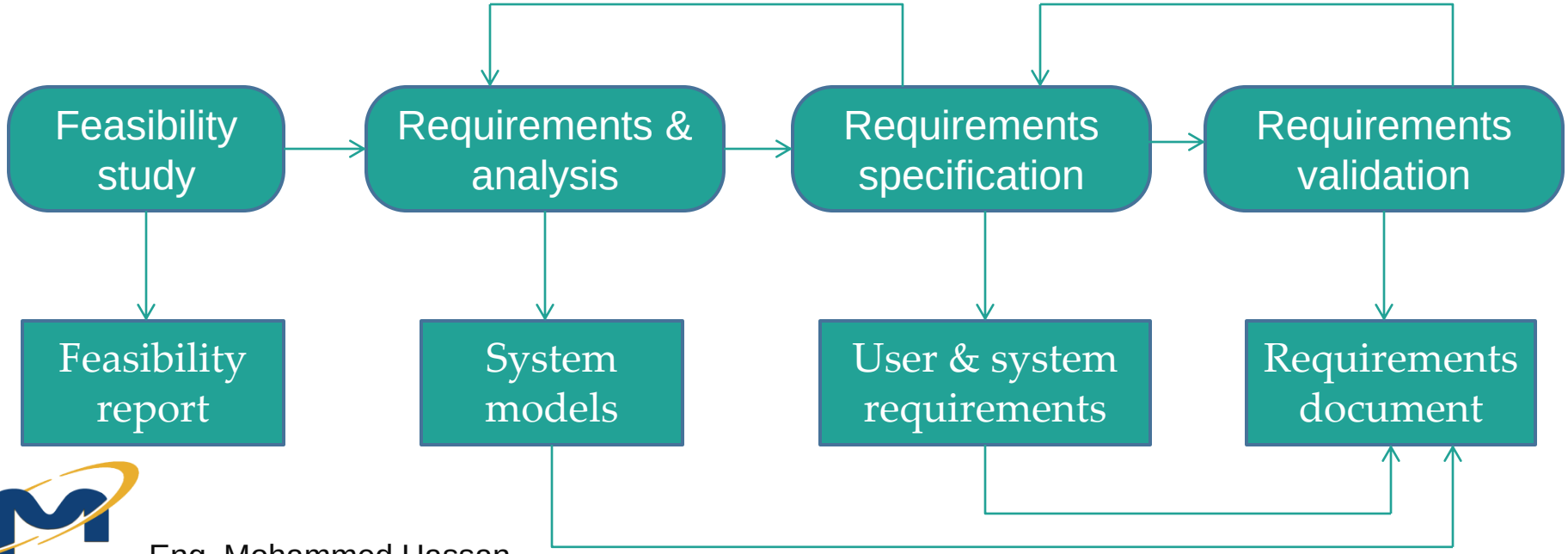
Reuse-oriented software engineering

- **Process stages:**
  - Component analysis.
  - Requirements modification.
  - System design reuse.
  - Development and integration.
- **Reuse is now the standard approach for building many types of business system.**



Software specification

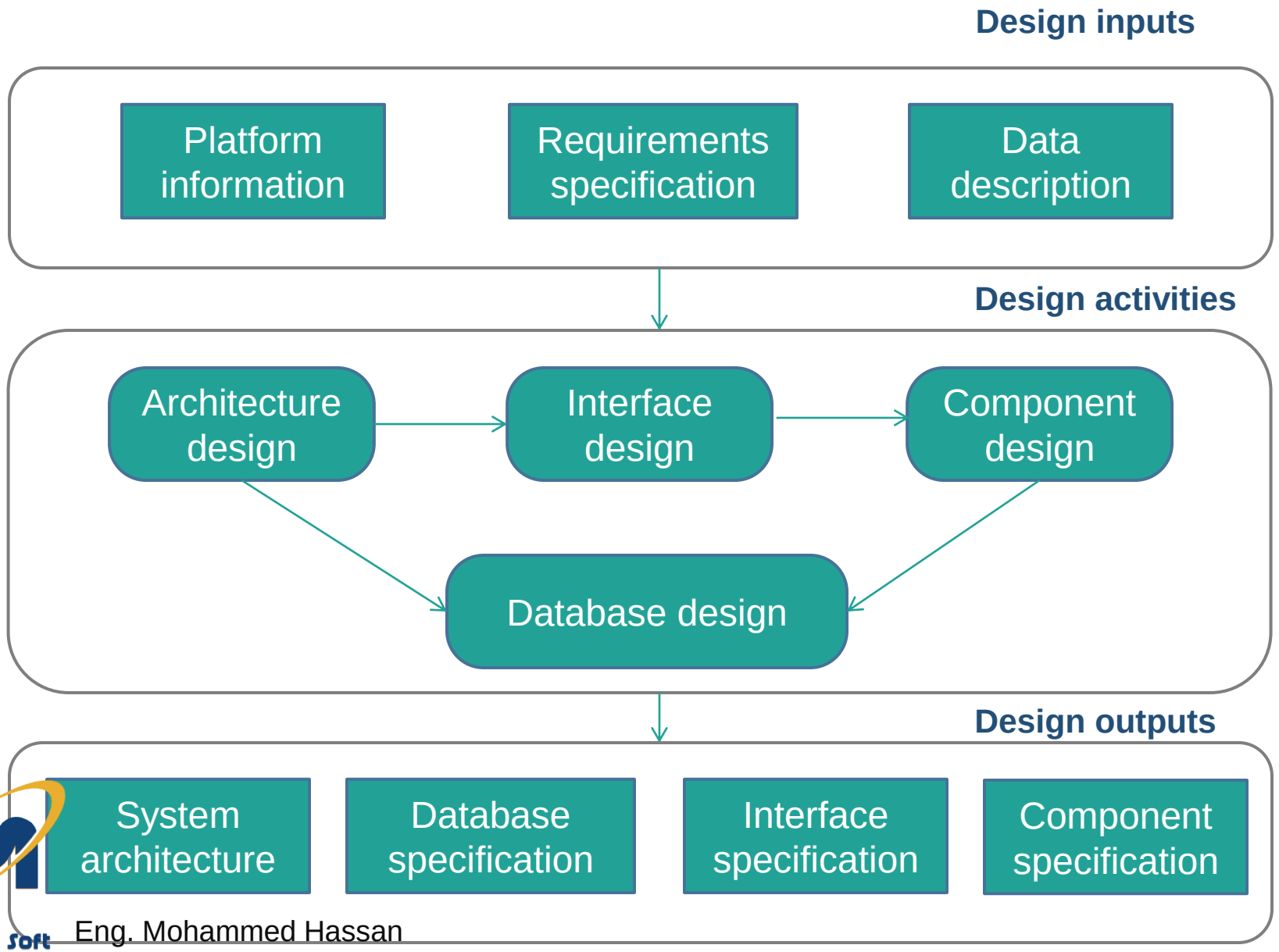
- **Requirements engineering process**
  - Feasibility study.
  - Requirements and analysis.
  - Requirements specification.
  - Requirements validation.



Software design and implementation

- The process of converting the system specification into an executable system.
- Software design
  - Design a software structure that realises specification.
- Implementation
  - Translate this structure into an executable program.
- The activities of design and implementation are closely related and may be interleaved.

A general model of the design process

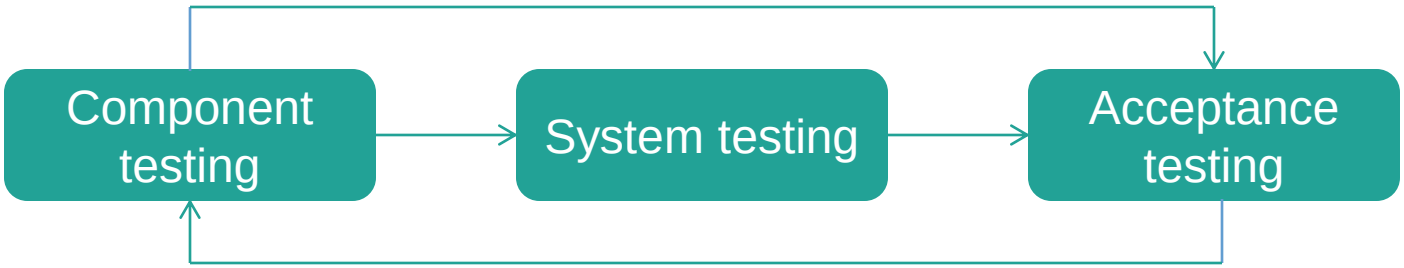


Software validation

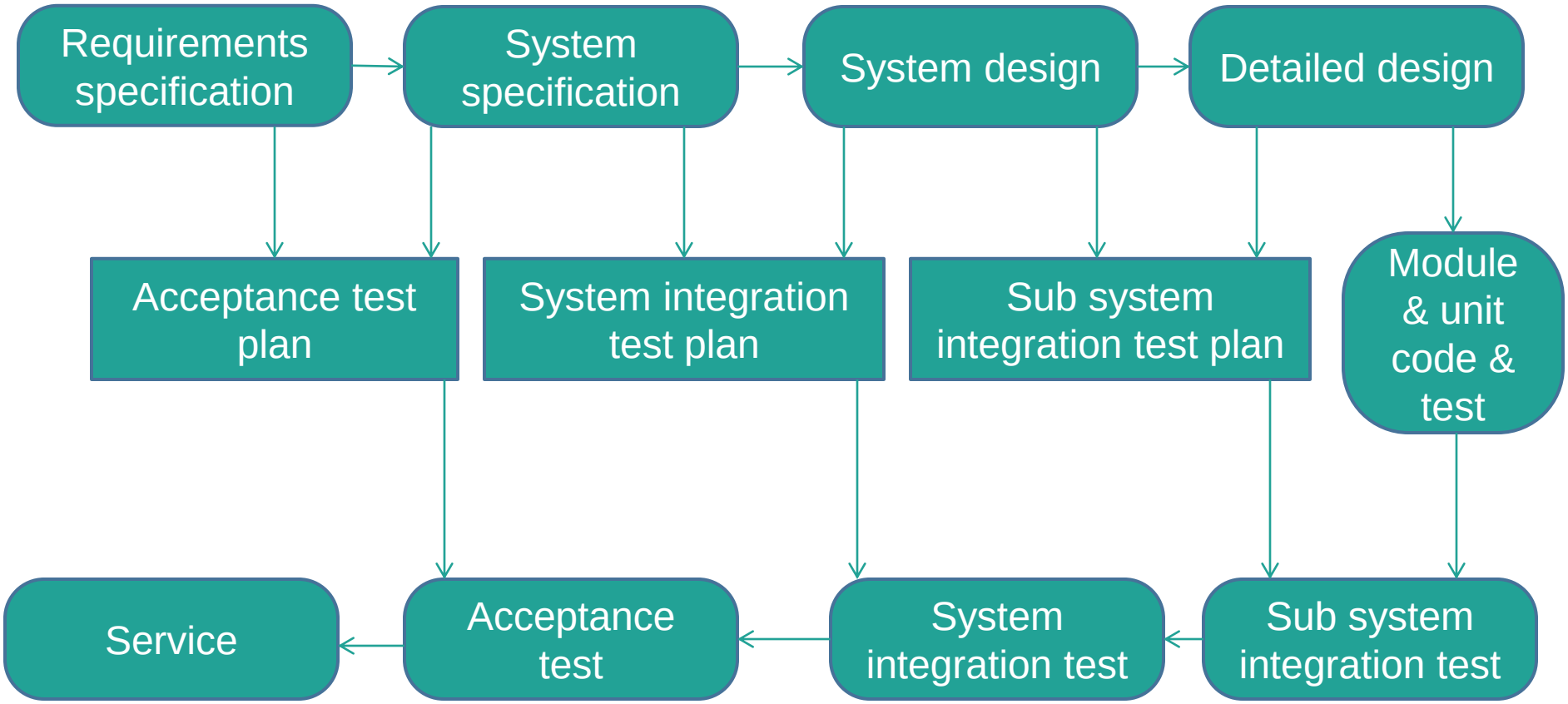
- Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.
- Involves checking and review processes and system testing.
- System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.

Stages of testing

- Development or component testing
- System testing
- Acceptance testing

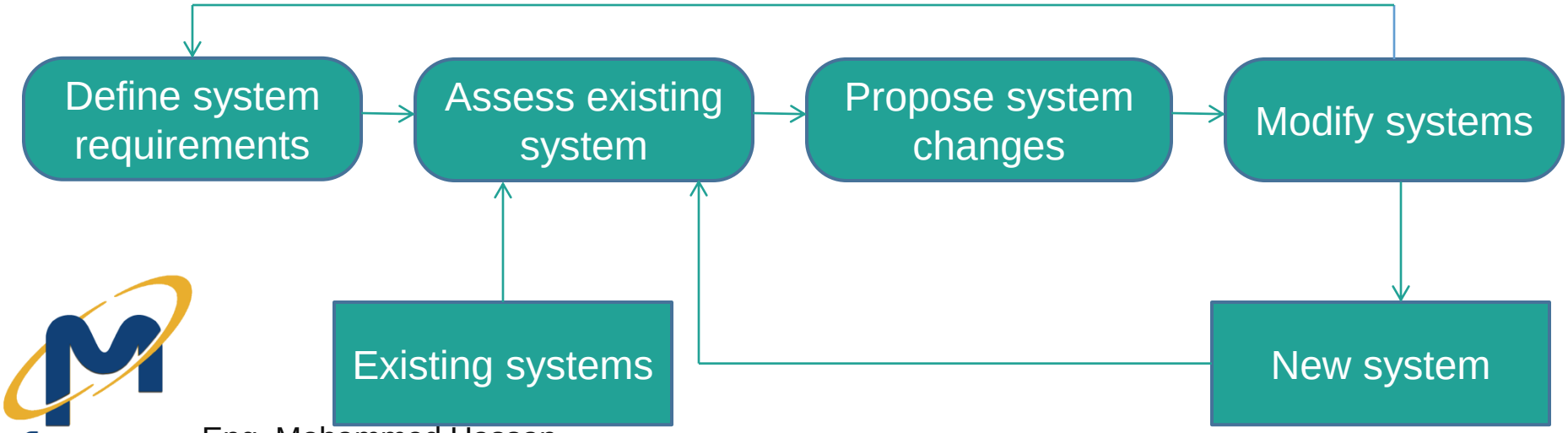


Testing phases



Software evolution

- Software is inherently flexible and can change.
- As requirements change through changing business circumstances, the software that supports the business must also evolve and change.



Chapter 3

Agile software Development

Eng. Mohammed Hassan

Topics Covered

- Agile methods
- Plan driven and agile development
- Extreme programming
- Pair programming

Rapid software development

- Rapid development and delivery is now often the most important requirement for software systems
  - Businesses operate in a fast changing requirement and it is practically impossible to produce a set of stable software requirements.
  - Software has to evolve quickly to reflect changing business needs.
- Rapid software development
  - Specification, design and implementation are inter-leaved.
  - System is developed as a series of versions with stakeholders involved in version evaluation.

Agile method applicability

- Product development where a software company is developing a small or medium sized product for sale.
- Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are not a lot of external rules and regulations that affect the software.
- Because of their focus on small, tightly integrated teams, there are problems in scaling agile methods to large systems.

Problems with agile methods

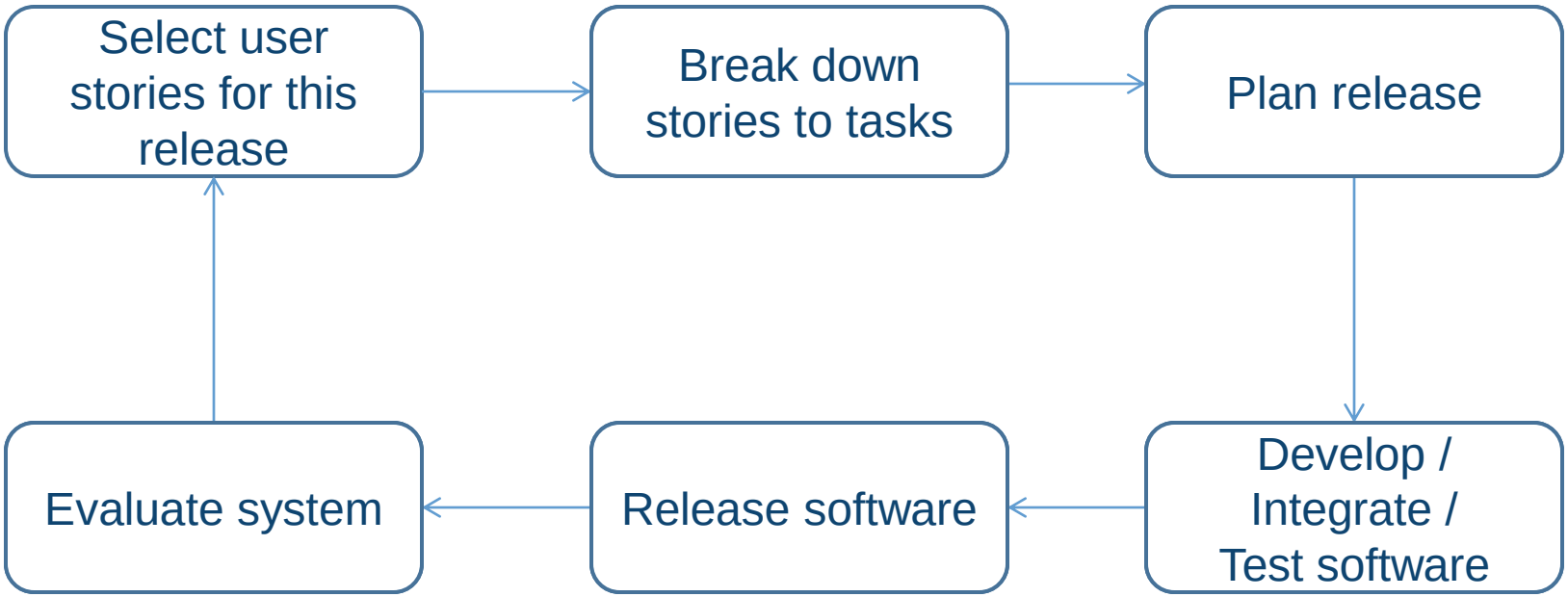
- It can be difficult to keep the interest of customers who are involved in the process.
- Team members may be unsuited to the intense involvement that characterizes agile methods.
- Prioritizing changes can be difficult where there are multiple stakeholders.
- Maintaining simplicity requires extra work.
- Contracts may be a problem as with other approaches to iterative development.

Extreme programming

- Perhaps the best-known and most widely used agile method.

- Extreme (great) programming (XP) takes an ‘extreme’ approach to iterative development.
  - New versions may be built several times per day.
  - Increments are delivered to customers every 2 weeks.
  - All tests must be run for every build and the build is only accepted if tests run successfully.

The extreme programming release cycle



A suggesting medicine scenario

Ahmed is a doctor who wishes to prescribe medication for a patient attending a clinic. The patient record is already displayed on his computer so he clicks on the medication field and can select current. The system displays a list of possible drugs starting with these letters. He chooses the required medication and the system responds by asking him to check that the medication selected is correct. He enters the dose and then confirms the prescription. The system checks that the dose is within the approved range.

Examples of task cards for prescribing medication

Task 1: Dose checking

Dose checking is a safety precaution to check that the doctor has not prescribed a dangerously small or large dose. Using the formulary ID for the generic drug name. Look up the formulary and retrieve the recommended maximum dose. Check the prescribed dose against the minimum and maximum. If outside the rang, issue an error message saying that the dose is too high or too low. If within the range, enable the Confirm button.

Refactoring

- Programming team look for possible software improvements and make these improvements even where is no need for them.
- Changes are easier to make because the code is well structured and clear.
- However, some changes requires architecture refactoring and this is much more expensive.

Examples of refactoring

- Re-organization of a class to remove duplicate code.
- Clean up and renaming attributes and methods to make them easier to understand.
- The replacement of inline code with calls to methods that have been included in a program library.

Testing in XP

- Testing is central to XP and XP has developed an approach where the program is tested after every change has been mode.

- XP testing features:
  - Test-first development
  - User involvement in test development and validation.
  - Automated test harnesses are used to run all component tests each time that a new release is built.

Test first development

- Writing tests before code clarifies the requirements to be implemented.
- Tests are written as programs rather than data so that they can be executed automatically.
- All previous and new tests are run automatically when new functionality is added, thus checking the new functionality has not introduced errors.

Customer involvement

- The role of the customer in the testing process is to help develop acceptance tests for the stories that are to be implemented in the next release of the system.

- The customer who is part of the team writes tests as development proceeds. All new code is therefore validated to ensure that is what the customer needs.

Test automation

- Test automation means that tests are written as executable components before the task is implemented.
  - An automated test framework (e.g. Junit) is a system that makes it easy to write executable tests and submit a set of tests for execution.

- As testing is automated, there is always a set of tests that can be quickly and easily executed.
  - Whenever any functionality is added to the system, the tests can be run and problems that the new code has introduced can be caught immediately.

Pair programming

- In XP, programmers work in pairs, sitting together to develop code.
- This helps develop common ownership of code and spreads knowledge across the team.
- It serves as an informal review process as each line of code is looked at by more than 1 person.

Advantages of pair programming

- It supports the idea of collective ownership and responsibility for the system.
- It helps support refactoring, which is a process of software improvement.

Chapter 4

Requirements Engineering

Topics Covered

- Functional and non-functional requirements
- The software requirements document
- Requirements specification
- Requirements and design

Requirements engineering

- The process of establishing the services that the customer requires from a system and the constrains under which it operates and is developed.

- The requirements themselves are the descriptions of the system services and constrains that are generated during the requirements engineering process.

Types of requirements

User requirements

- Statements in natural language plus diagrams of the services the system provides and its operational constrains written for customers.

System requirements

- A structured document setting out detailed descriptions of the system functions, services and operational constrains.

User and system requirements

User Requirement Definition

1. The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System Requirements Specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
1.2 The system shall automatically generate the report for printing after 17.30 on the last working day of the month.
1.3 A report shall be created for each clinic and shall list the individual drug names. The total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
1.4 If drugs are available in different dose units (e.g. 10mg. 20mg) separate reports shall be created for each dose unit
1.5 Access to all cost reports shall be restricted to authorized users listed on a management access control list.

Functional requirements

- Describe functionality or system services.
- Depend on the type of software, expected users and the type of system where the software is used.
- Functional system requirements should describe the system services in detail.

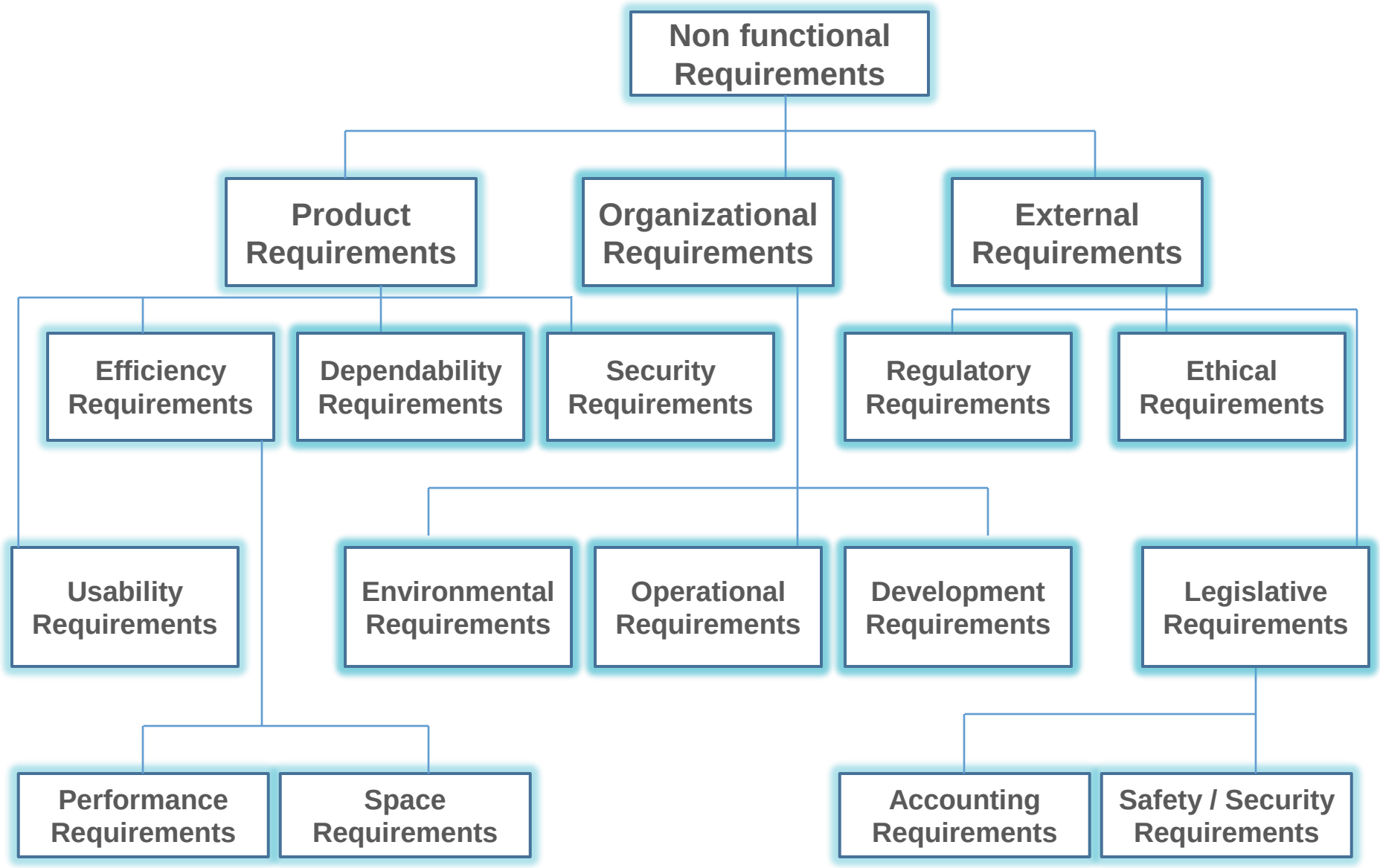
Functional requirements for the MHC-PMS

- A user shall be to search the appointments lists for all clinics.
- The system shall generate each day for each clinic, a list of patients who are expected to attend appointments that day.
- Each staff member using the system shall be uniquely identified by his or her 8 digital employee number.

Non-functional requirements

- These define system properties and constrains e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- Non functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.

Types of nonfunctional requirements



Non-functional classifications

- Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- Requirements which are a consequence of organizational policies and procedures e.g. process standards used, implementation requirements, etc.
- Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, law making requirements, etc.

Examples of nonfunctional requirements in the MHC-PMS

Product requirement

The MHC-PMS shall be available to all clinics during normal working hours. Downtime within normal working hours shall not exceed five seconds in any one day.

Organizational requirement

Users of the MHC-PMS system shall authenticate themselves using their health authority identity card.

External requirement

The system shall implement patient privacy requirements as set out in Hstan-03-2018-priv.

Domain requirements

- **The system's operational domain imposes requirements on the system.**
  - For example, a train control system has to take into account the braking characteristics in different whether conditions.
- **Domain requirements be new functional requirements, constraints on existing requirements or define specific computations**
- **If domain requirements are not satisfied, the system may be unworkable.**

Domain requirements problems

- **Understand ability**
  - Requirements are expressed in the language of the application domain.
  - This is often not understood by software engineers developing the system.

- **Indirectness**
  - Domain specialists understand the area so well that they do not think of making the domain requirements explicit.

Agile methods and requirements

- Many agile methods argue that producing a requirements document is a waste of time as requirements change so quickly.
- The document is therefore always out of date.
- Methods such as XP use incremental requirements engineering and express requirements as ‘user stories’.
- This is practical for business systems but problematic for systems that require a lot of pre delivery analysis (e.g. critical systems) or systems developed by several teams.

Requirements specification

- The process of writing the user and system requirements in a requirements document.
- User requirements have to be understandable by end-users and customers who do not have a technical background.
- System requirements are more detailed requirements and may include more technical information.
- **In principle, requirements should state what the system should do and the design should describe how it does this.**

Natural language specification

- **Requirements are written as natural language sentences supplemented by diagrams and tables.**

- **Means that the requirements can be understood by users and customers.**

Problems with natural language

Lack of clarity

Precision is difficult without making the document difficult to read.

Requirements join us

Several different requirements may be expressed together.

Requirements confusion

Functional and non functional requirements tend to be mixed-up.

Chapter 5

System Modeling

Topics Covered

- Context models
- Interaction models
- Structural models
- Behavior models

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 has now come to mean representing a system using some kind of graphical representation, which is now almost always based on notations in the Unified Modeling Language (UML).

- System modeling helps the analyst to understand the functionality of the system and models are used to communicate with customers.

UML diagram types

Activity diagrams

Which show the activities involved in a process or in data processing.

Use case diagrams

Which show the interactions between a system and its environment.

Sequence diagrams

Which show the interactions between actors and the system and between system components.

Class diagrams

Which show the object classes in the system and the associations between these classes.

State diagrams

Which show how the system reacts to internal and external events.

Use of graphical models

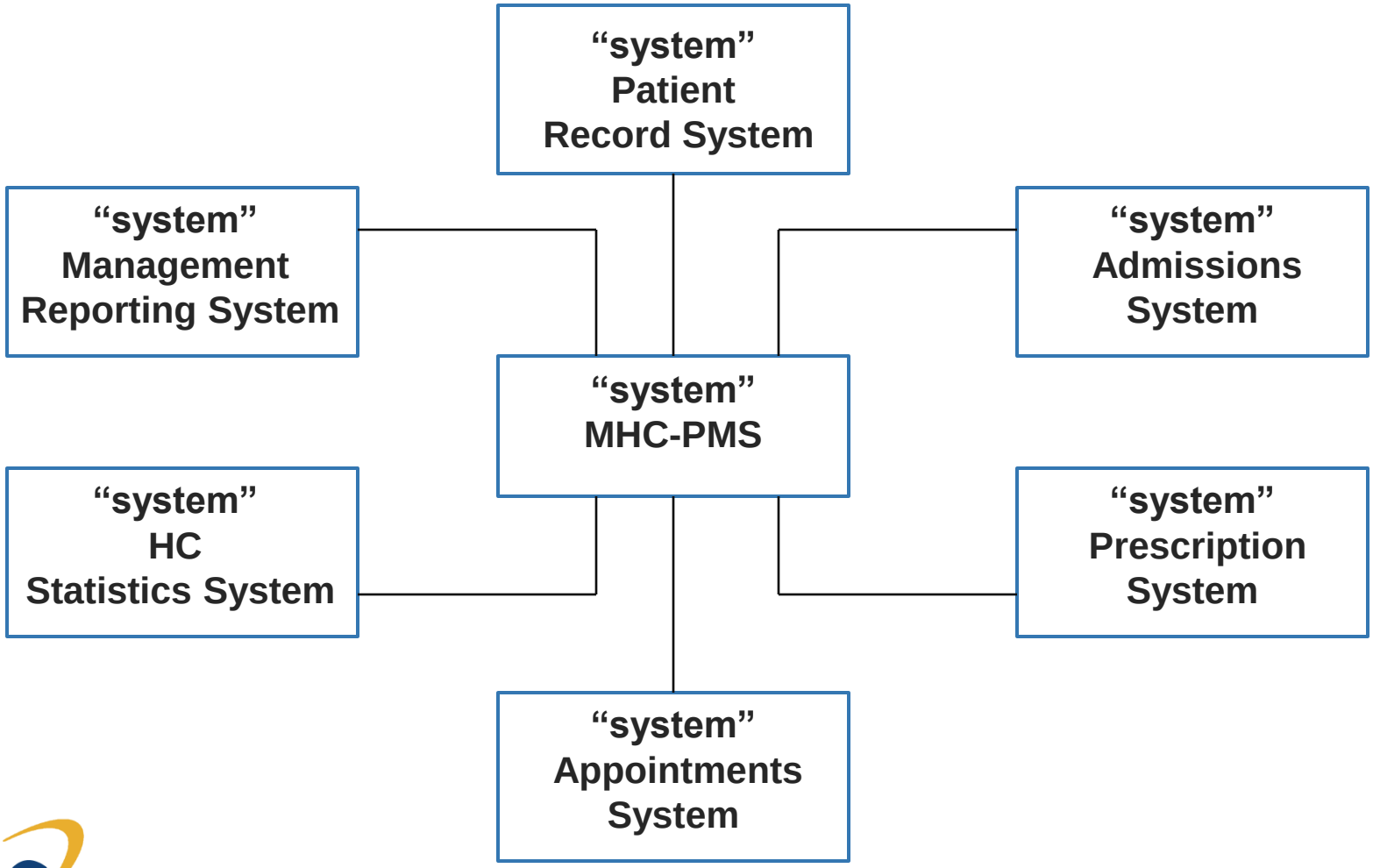
- As a means of facilitating discussion about an existing or proposed system
- As a way of documenting an existing system
- As a detailed system description that can be used to generate a system implementation.

Context (situation) models

- Context models are used to illustrate the operational context of a system they show what lies outside the system boundaries.

- Social and organizational concerns may affect the decision on where to position system boundaries.

The context of the MHC-PMS



Interaction models

- Modeling user interaction is important as it helps to identify user requirements.
- Modeling system-to-system interaction highlights the communication problems that may arise.
- Modeling component interaction helps us understand if a proposed system structure is likely to deliver the required system performance and dependability.
- Use case diagrams and sequence diagrams may be used for interaction modeling.

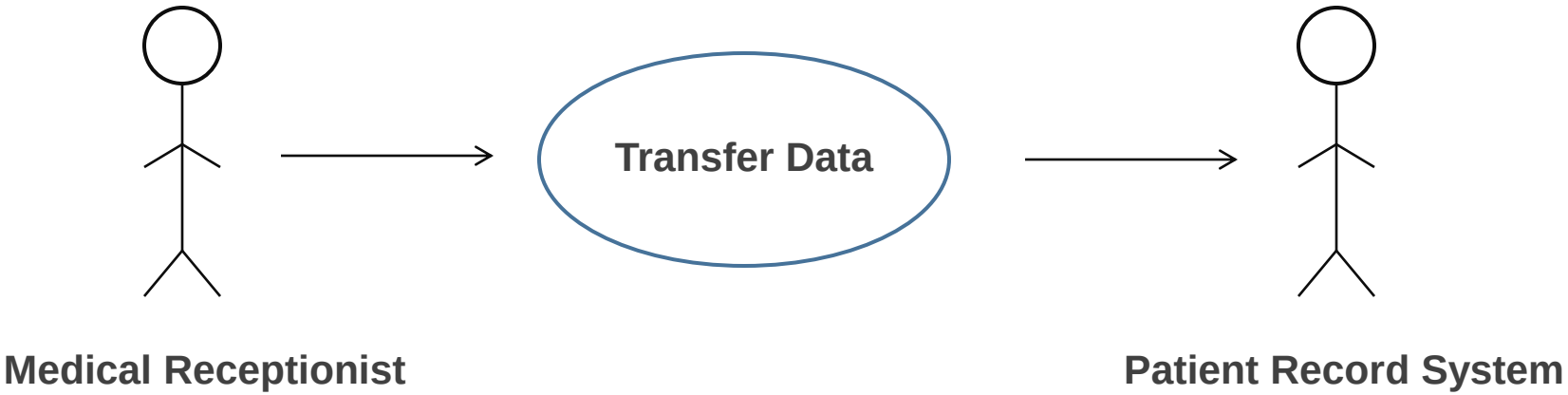
Use case modeling

- Each use case represents a discrete task that involves external interaction with a system.

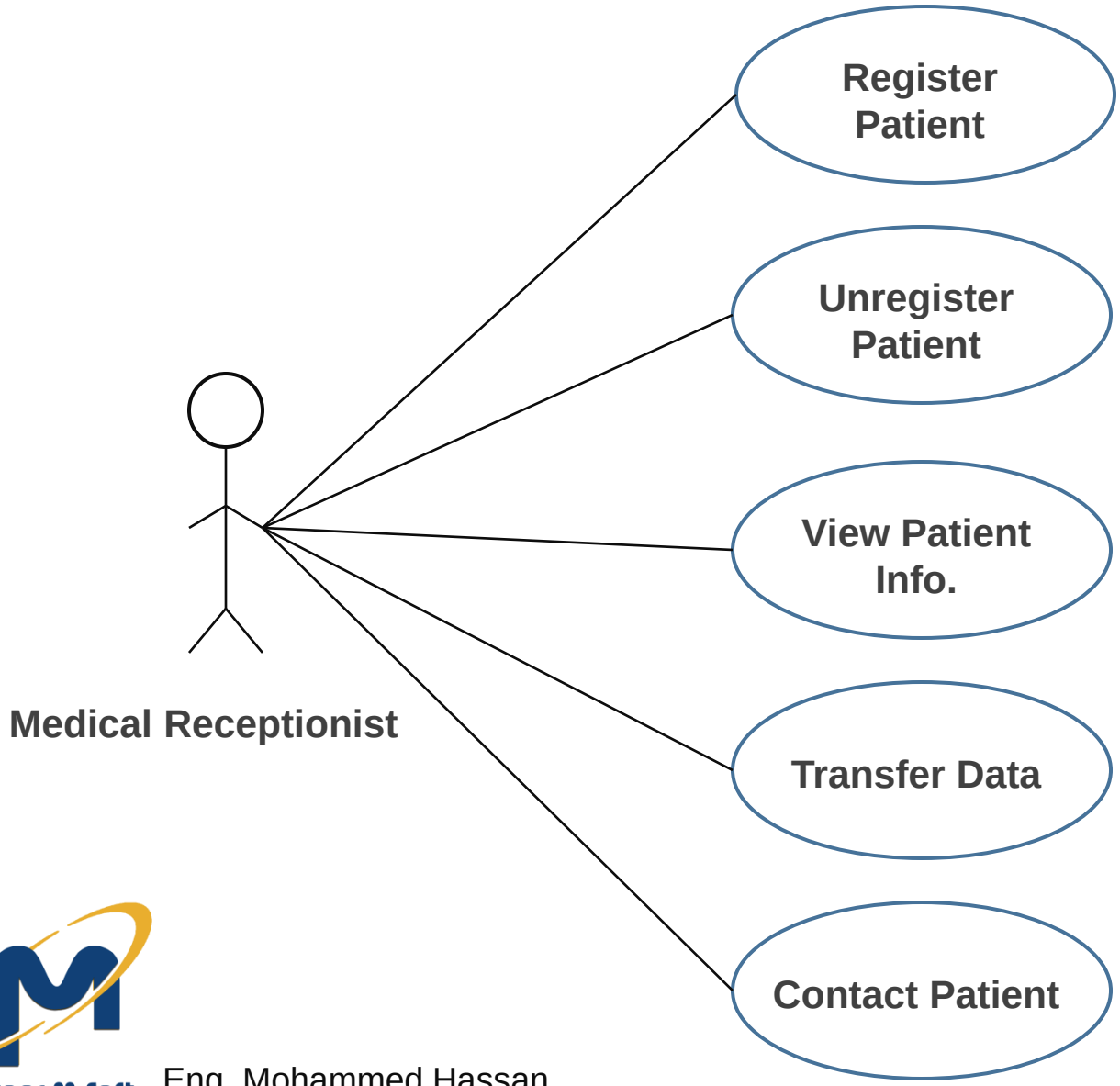
- Actors in a use case may be people or other systems.

Transfer-data use case

- A use case in the MHC-PMS



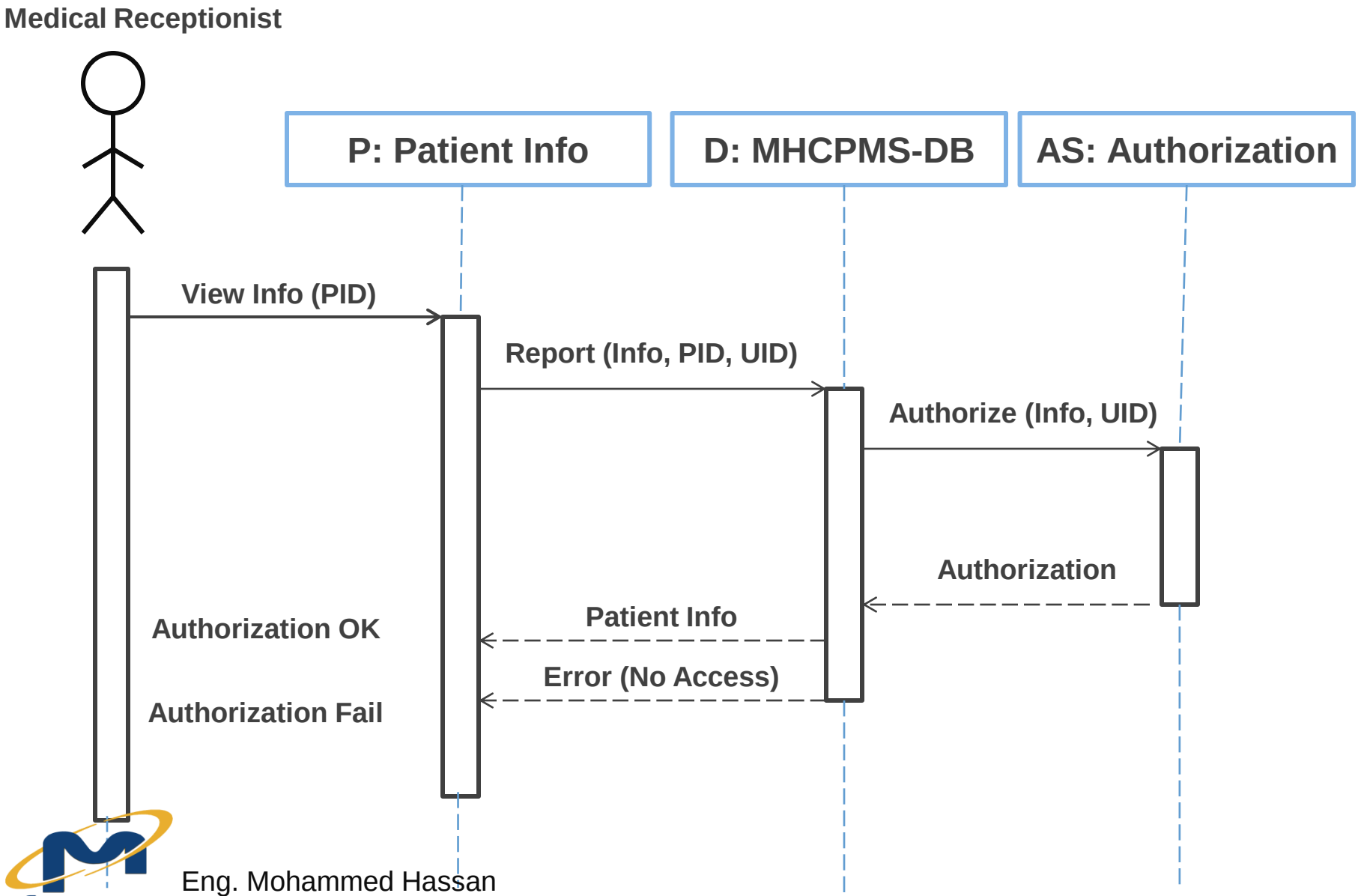
Use cases in the MHC-PMS involving the role ‘Medical Receptionist’



Sequence diagrams

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

Sequence diagram for view patient information



Structural models

- 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 structural 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.

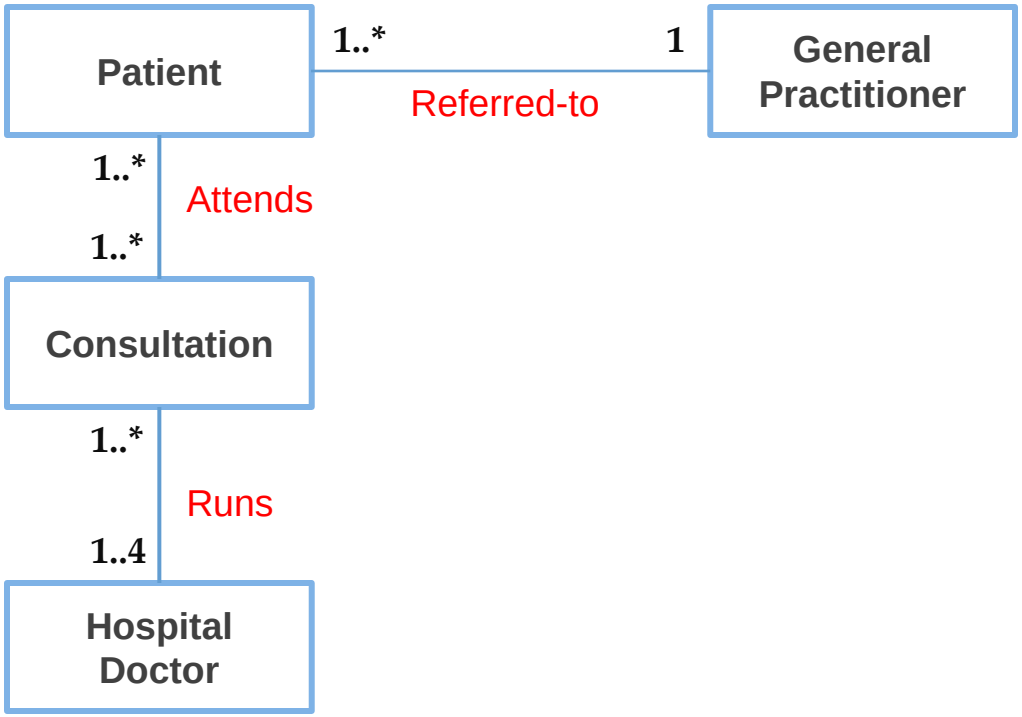
Class diagrams

- Class diagrams are used when developing an object-oriented system model to show the classes in a system, the associations and relationships between these classes.

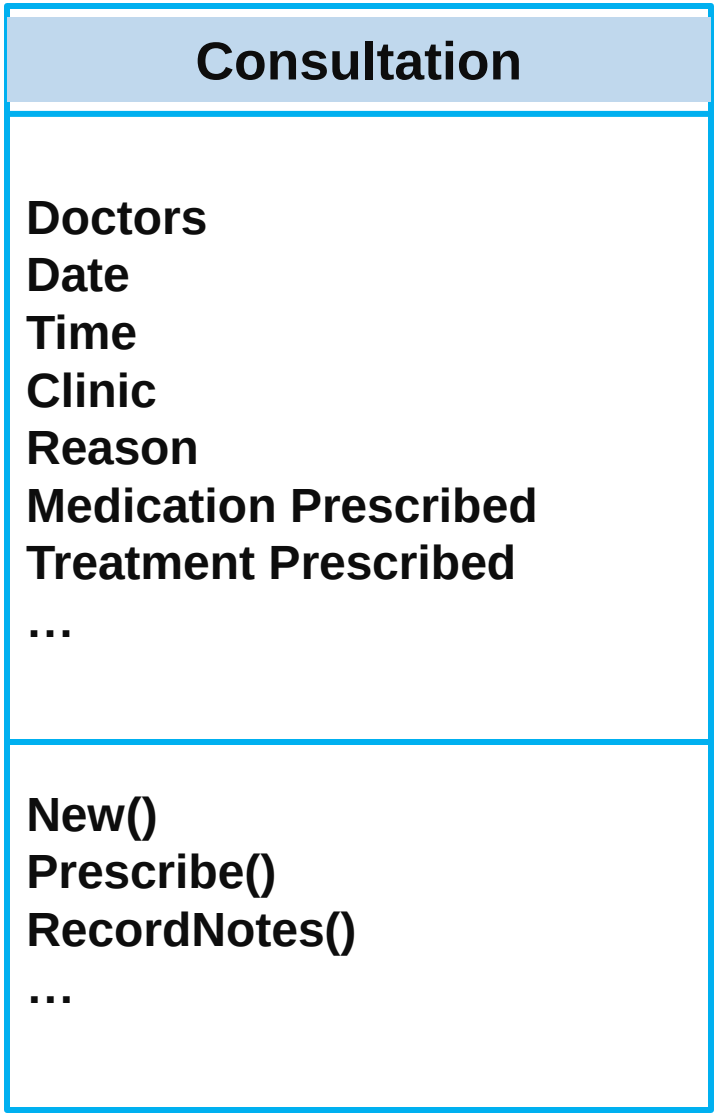
- When you are developing models during the early stages of the software engineering process, objects represent something in the real world, such as a patient, a treatment, doctor, etc.

UML classes and association

Classes and associations in the MHC-PMS



The Consultation class

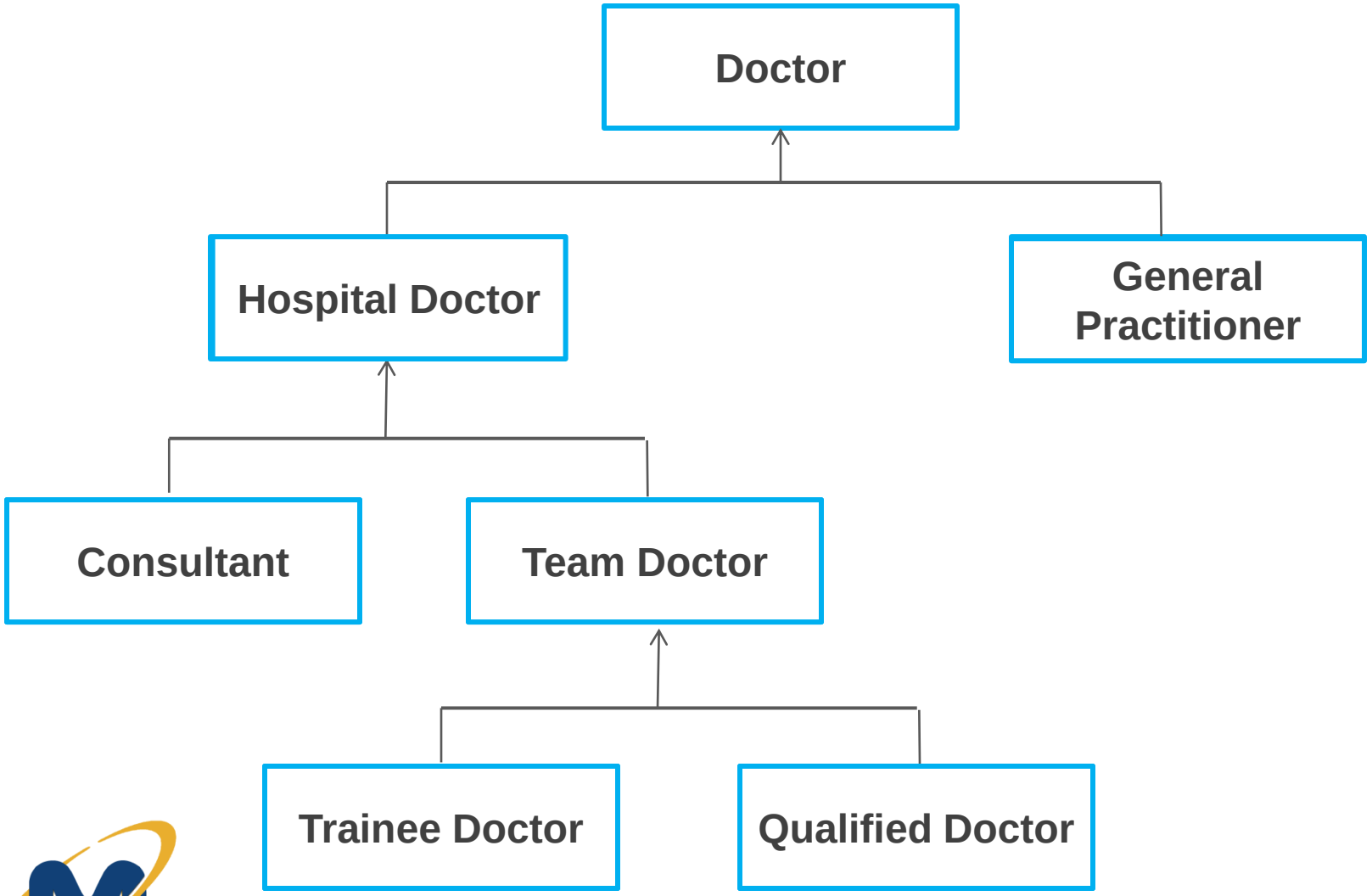


Generalization

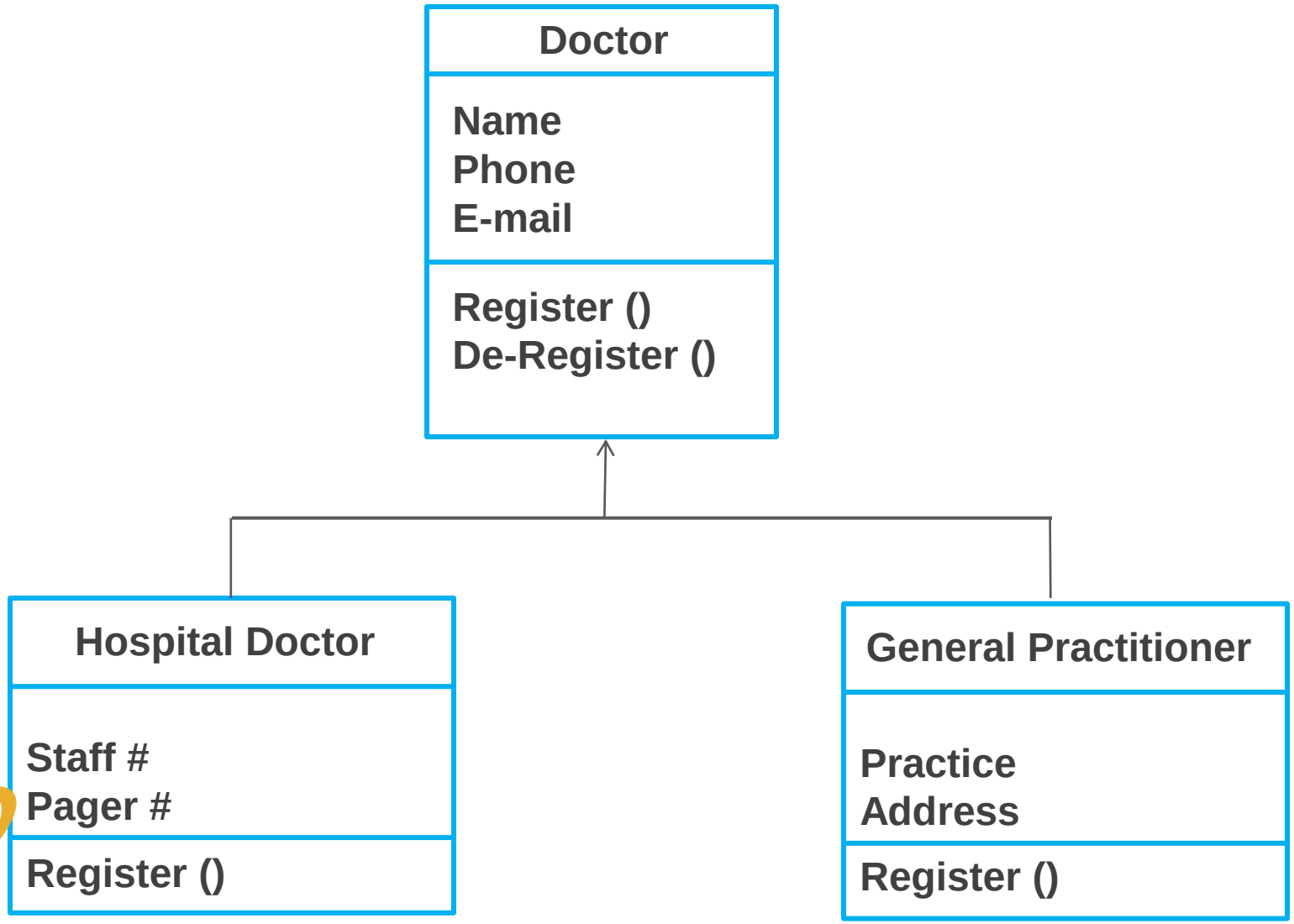
- Generalization is an everyday technique that we use to manage complexity.

- This allow us to infer that different members of these classes have some common characteristics.

Generalization

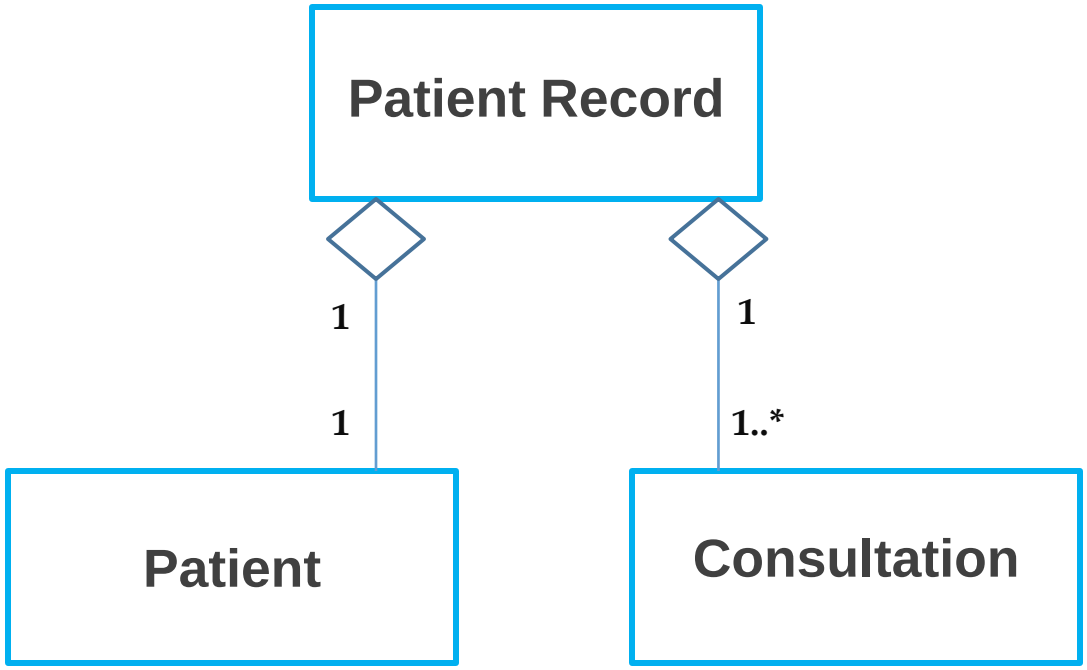


A generalization hierarchy with added detail



Aggregation models

- An aggregation model shows how classes that are composed of other classes.



Behavioral models

- Behavioral models are models of the dynamic behavior of a system as it is executing. They show what happens or what is supposed to happen when a system responds to a motivation from its environment.

- You can think of these stimuli as being of two types:

Data some data arrives that has to be processed by the system.

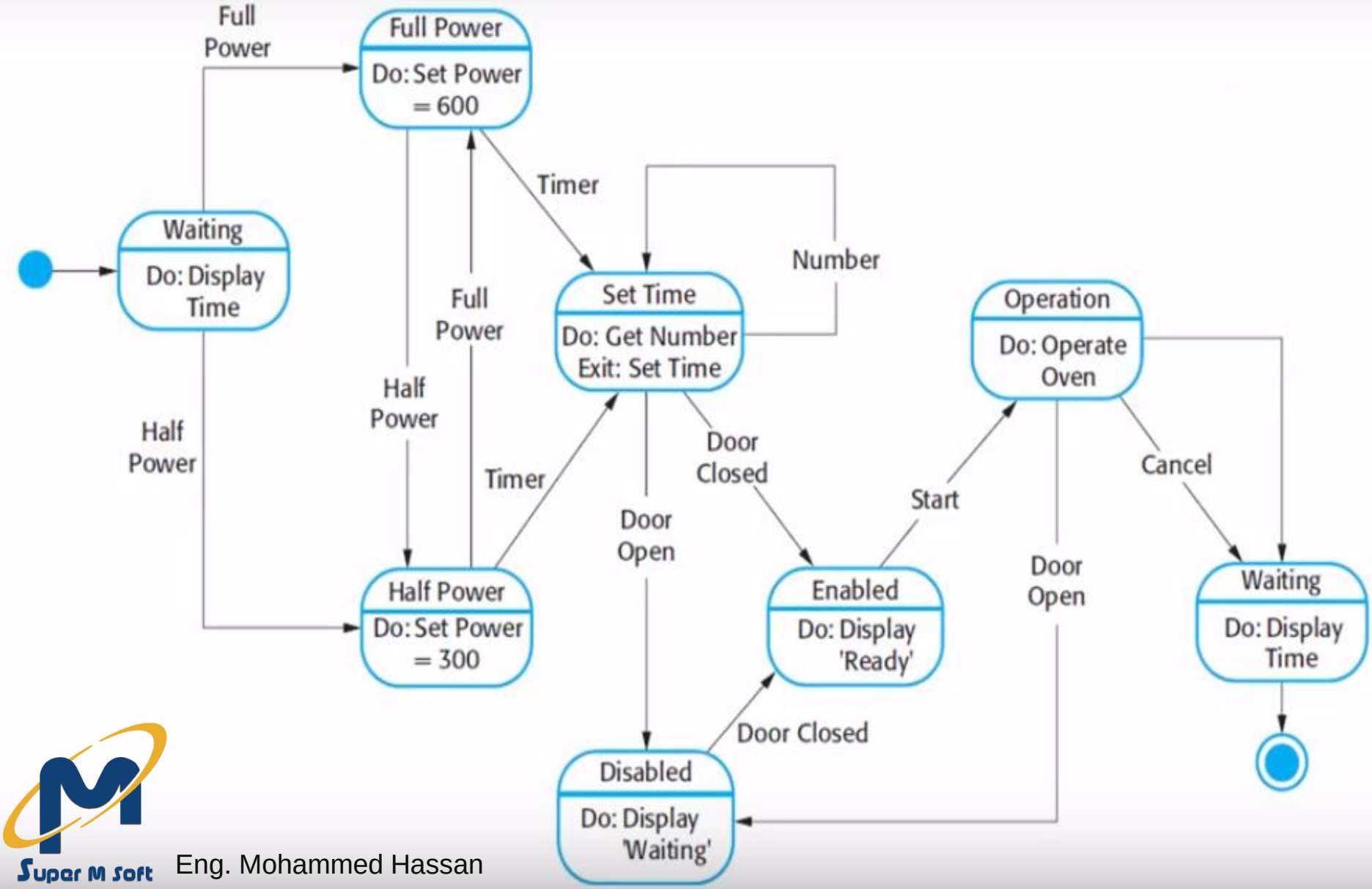
Events some event happens that triggers system processing. Events may have associated data, although this is not always the case.

State machine models

- These model the behavior of the system in response to external and internal events.

- They show the system’s responses to stimuli so are often used for modeling real time systems.

State diagram of a microwave oven



Chapter 6

Design and Implementation

Topics Covered

- Object oriented design using UML
- Design patterns
- Implementation issues
- Open source development

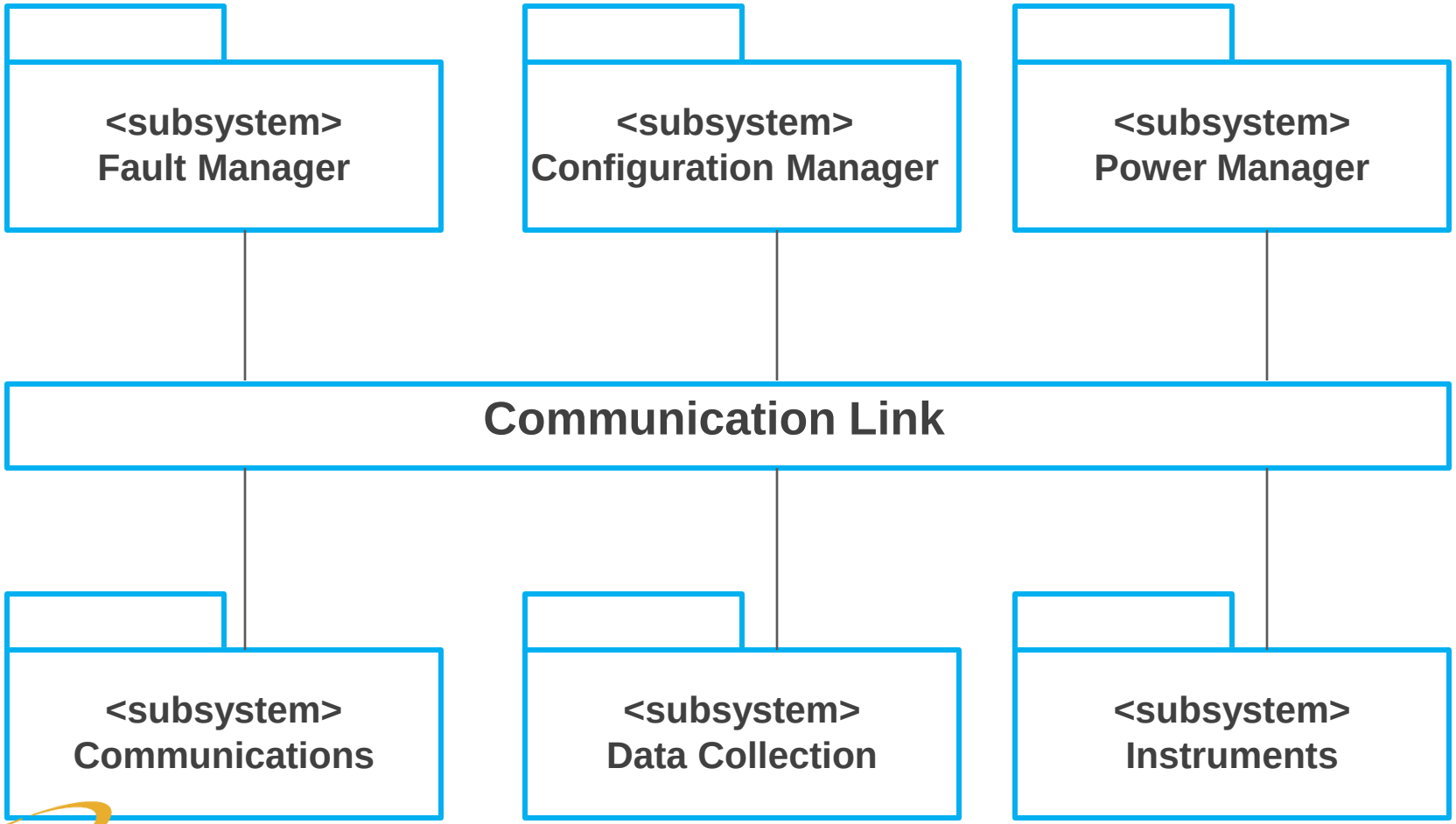
An object-oriented design process

- Structural object-oriented design processes involve developing a number of different system models.
- They require a lot of effort for development and maintenance of these models and for small systems, this may not be cost-effective.
- However, for large systems developed by different groups design models are an important communication mechanism.

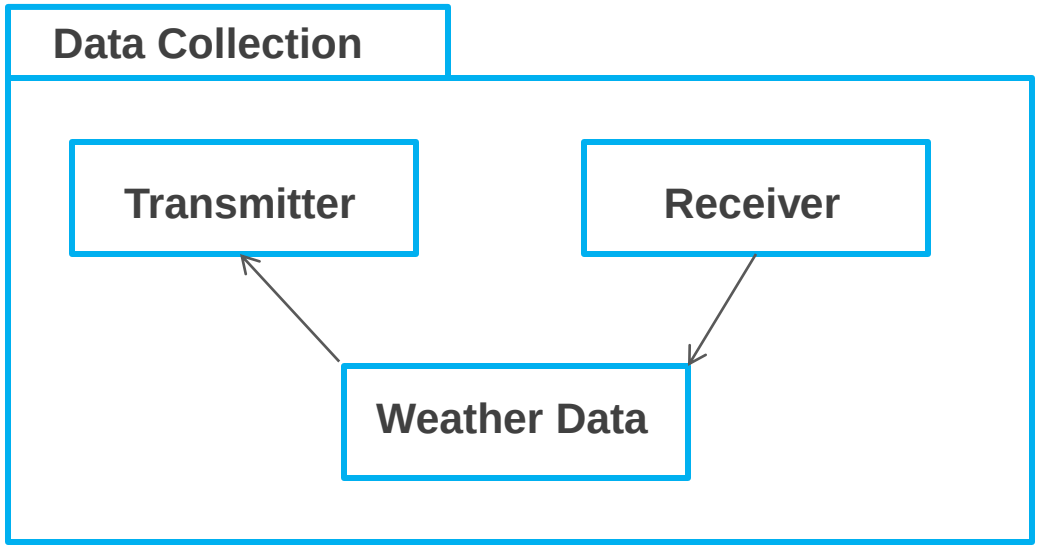
Architectural design

- Once interactions between the system and its environment have been understood, you use this information for designing the system architecture.
- You identify the major components that make up the system and their interactions.
- The weather station is composed of independent subsystems that communicate by broadcasting messages on a common infrastructure.

High-level architecture of the weather station



Architecture of data collection system



Object class identification

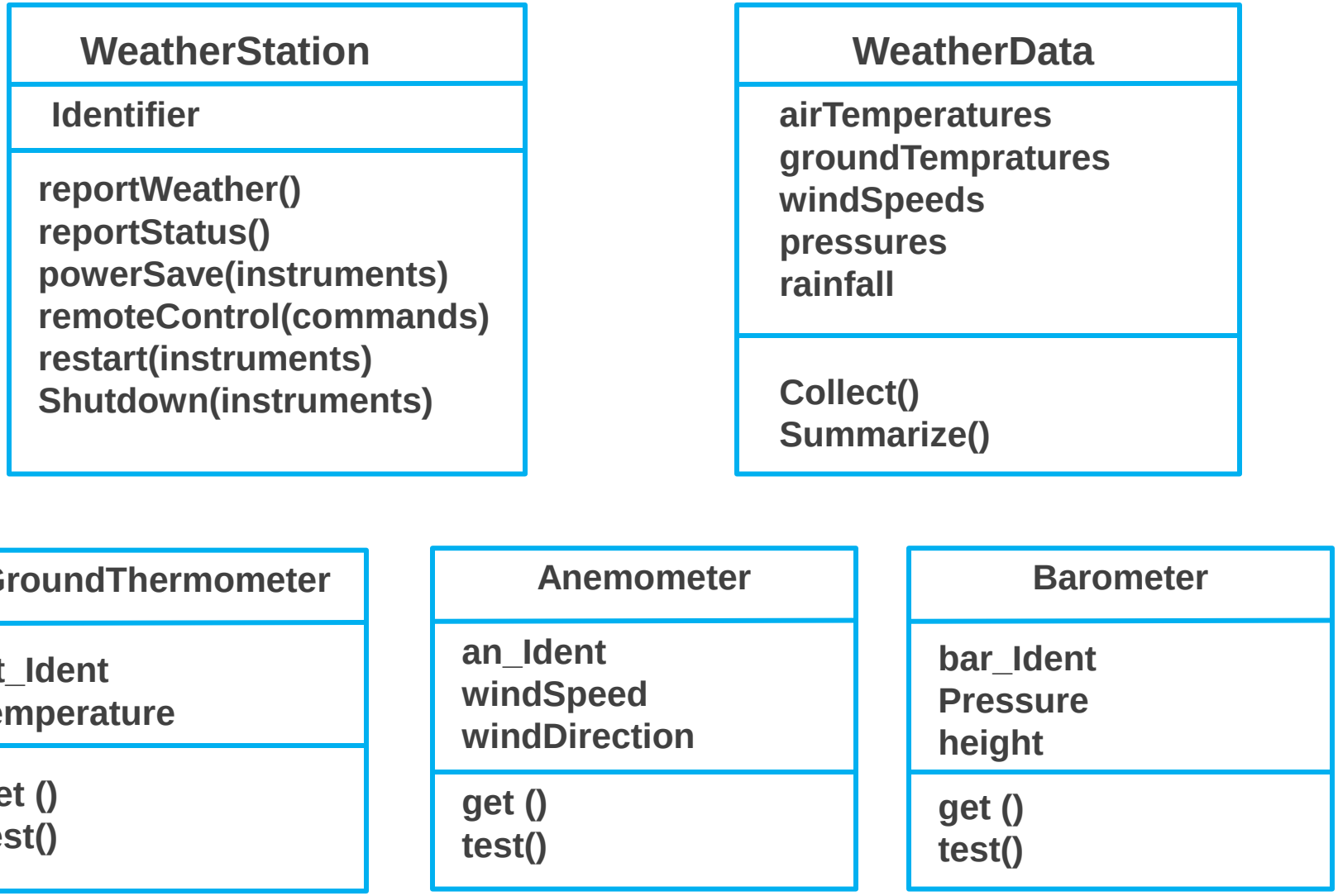
- Identifying object classes is often a difficult part of object oriented design.
- There is no magic informal for object identification. It relies on the skill, experience and domain knowledge of system designers.
- Object identification is an iterative process. You are unlikely to get it right first time.

Whether station description

A weather station is a package of software controlled instruments which collects data, performs some data processing and transmits this data for further processing. The instruments include air and ground thermometers, an anemometer, a wind vane, a barometer and a rain gauge.

When a command is issued to transmit the weather data, the weather station processes and summarises the collected data. The summarised data is transmitted to the mapping computer when a request is received

Whether station description



Design patterns

- A design pattern is a way of reusing abstract knowledge about problem and its solution.
- A pattern is a description of the problem and its solution.
- It should be sufficiently to be reused in different settings.
- Pattern descriptions usually make use of object oriented characteristics such as inheritance and polymorphism.

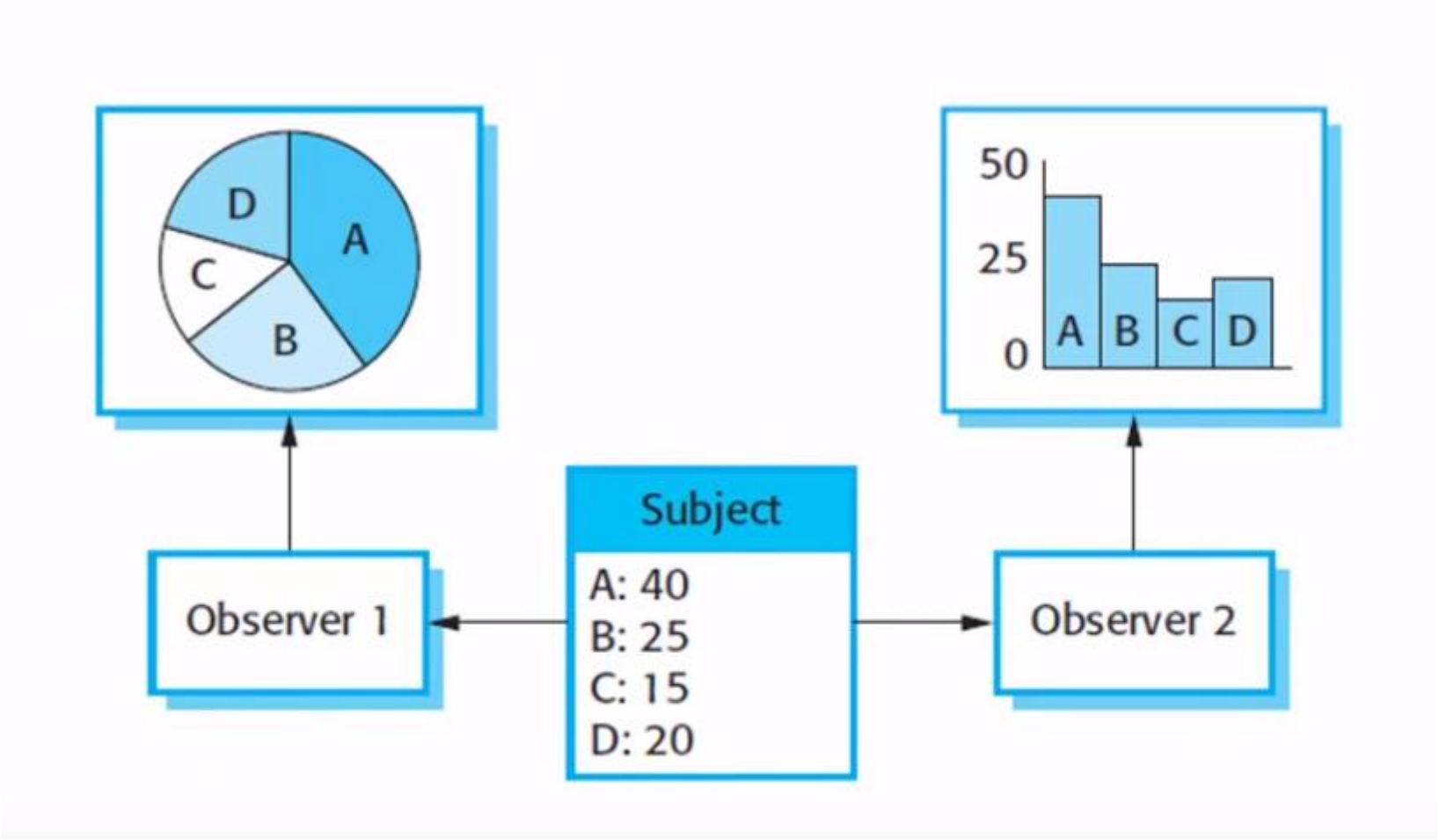
Pattern elements

- Name
- Problem description.
- Solution description
- Consequences

Observer pattern

- Name
  - Observer.
- Description
  - Separates the display of object state from the object itself.
- Problem description.
  - Used when multiple displays of state are needed.
- Solution description
  - See slide with UML description.
- Consequences
  - Optimizations to enhance display performance are impractical.

Multiple displays using the observe pattern



Reuse & Reuse levels

- From the 1960s to the 1990s, most new software was developed from scratch by writing all code in a high level programming language.
- The abstraction levels
  - At this level, you don't reuse software directly but use knowledge of successful abstractions in the design of your software.
- The object level
  - At this level, you directly reuse objects from a library rather than writing the code yourself.
- The component level
  - Components are collections of object classes that you reuse in application systems.
- The system level
  - At this level, you reuse entire application systems.

Reuse costs

- The costs of the time spent in looking for software to reuse and assessing whether or not it meets your needs.
- The costs of adapting and configuring the reusable software components or systems to reflect the requirements of the system that you are developing.
- The costs of integrating reusable software elements with each other and with the new code that you have developed .

Open source systems

- The best known open source product is of course, the Linux operating system which is widely used as a server system and increasingly, as a desktop environment.

- Other important open source products are Java and MySQL database management system.

Chapter 7

Software testing

Eng. Mohammed Hassan

Topics Covered

- Development testing
- Test driven development
- Release testing
- User testing

Program testing

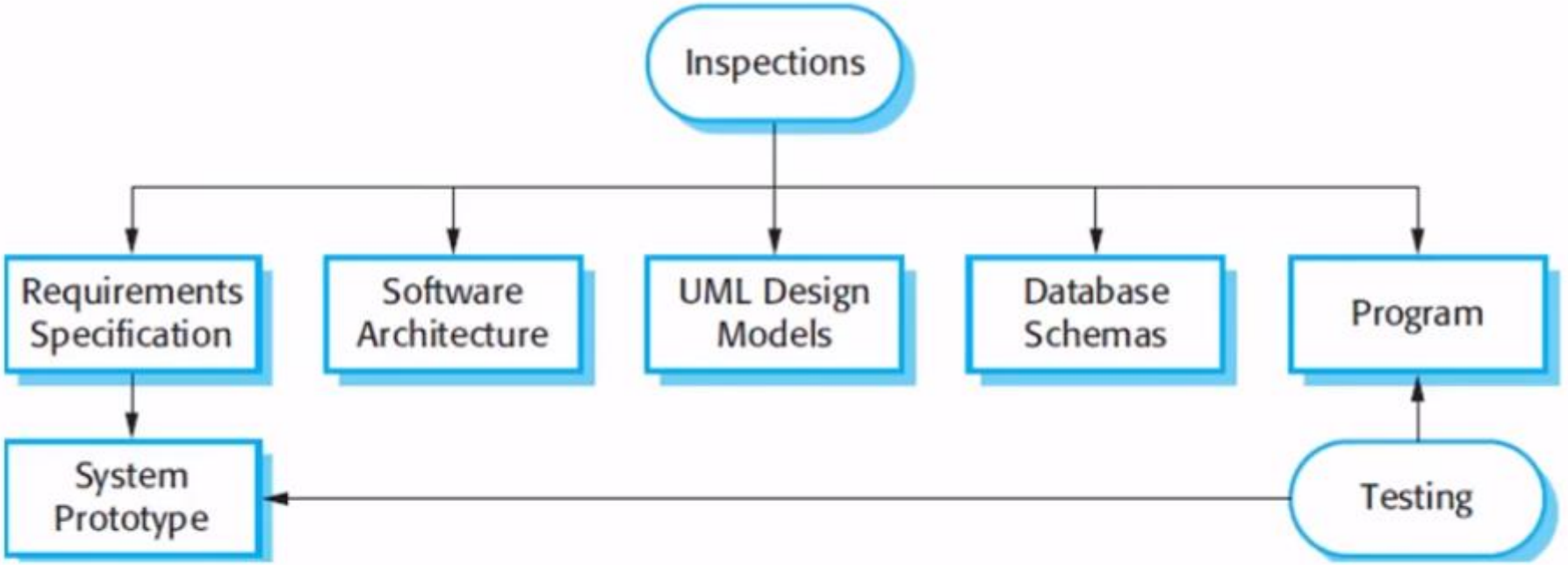
- Testing is intended to show that a program does what is intended to do and to discover program defects before it is put into use.
- When you test software, you execute a program using artificial data.
- You check the results of the test run for errors, anomalies or information about the program’s non-functional attributes.
- Testing is part of a more general verification and validation process.

Inspections and testing

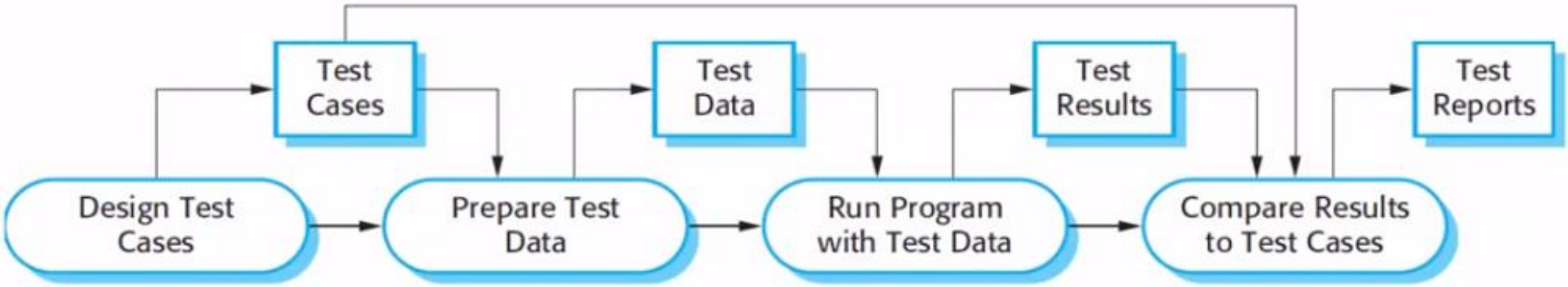
- Software inspections : Concerned with analysis of the static system representation to discover problems (static verification)

- Software testing: Concerned with exercising and observing product behavior (dynamic verification)

Inspections and testing



A model of software testing process



General testing guidelines

- Choose inputs that force the system to generate all error messages .
- Repeat the same input or series of inputs numerous times .
- Force invalid outputs to be generated .
- Force computation results to be too large or too small .

Stages of testing

01 **Development testing,** where the system is tested during development to discover bugs and defects.

02 **Release testing,** where a separate testing team test a complete version of the system before it is released to users.

03 **User testing**, where users or potential users of a system test the system in their own environment.

Development testing

- Development testing includes all testing activities that are carried out by team developing the system.

01 **Unit testing**

02 **Component testing**

03 **System testing**

Unit testing

- Unit testing is the process of testing individual components in isolation

- Units may be :
  - Individual functions or methods within an object.
  - Object classes with several attributes and methods.
  - Composite components with defined interfaces used to access their functionality.

Component testing

- Software components are often composite components that are made up of several interacting objects.
- You access the functionality of these objects through the defined component interface.
- Testing composite components should therefore focus on showing that the component interface behaves according to its specification.

Interface testing