

Software Engineering

Prepared by Dr. Yasser Abdelhamid

IT 382
Software Engineering

Lecture 7



LECTURE OUTLINE

- ◉ Requirements engineering
- ◉ User requirements
- ◉ System requirements
- ◉ Functional requirements
- ◉ Non-functional requirements
- ◉ Domain requirements
- ◉ The structure of a requirements document
- ◉ Requirements elicitation
- ◉ Requirements analysis
- ◉ Requirements validation
- ◉ Requirements management

REQUIREMENTS ENGINEERING

- The process of establishing **the services** that the customer requires from a system and **the constraints** under which it operates and is developed.
- **The requirements** themselves are the **descriptions** of the system **services and constraints** that are generated during the requirements engineering process.

WHAT IS A REQUIREMENT?

- Requirements may range from a **high-level abstract** statement of a service or of a system constraint **to a detailed** mathematical functional specification.
 - it must define its needs in a sufficiently **abstract** way that a solution is not pre-defined. As the requirements must be written so that several contractors can bid for the contract.
 - Once a contract has been awarded, the contractor must write a system definition for the client **in more detail** so that the client understands and can validate what the software will do.
 - **Both of these documents** may be called the requirements document for the system.”

TYPES OF REQUIREMENT

- Abstract → **User requirements**
 - Statements in natural language plus diagrams of the services the system provides and its operational constraints. **Written for customers.**
- Detailed → **System requirements**
 - A structured document setting out **detailed descriptions of the system's functions, services and operational constraints.** Defines what should be implemented so may be part of a contract between client and contractor.

EXAMPLE: 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., 10 mg, 20 mg) 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 AND NON-FUNCTIONAL REQUIREMENTS

- Functional requirements

- Statements of services the system should provide, **how the system should react to particular inputs and how the system should behave in particular situations.**
- May state what the system should not do.

- Non-functional requirements

- Constraints on the services or functions offered by the system such as **timing** constraints, constraints on the **development process, standards**, etc.
- Often **apply to the system as a whole** rather than individual features or services.

- Domain requirements

- Constraints on the system from the domain of operation

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 user requirements** may be high-level statements of what the system should do.
- **Functional system requirements** should describe the system services in detail.

EXAMPLE: FUNCTIONAL REQUIREMENTS FOR THE MHC-PMS

- A user shall be able 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-digit employee number**.

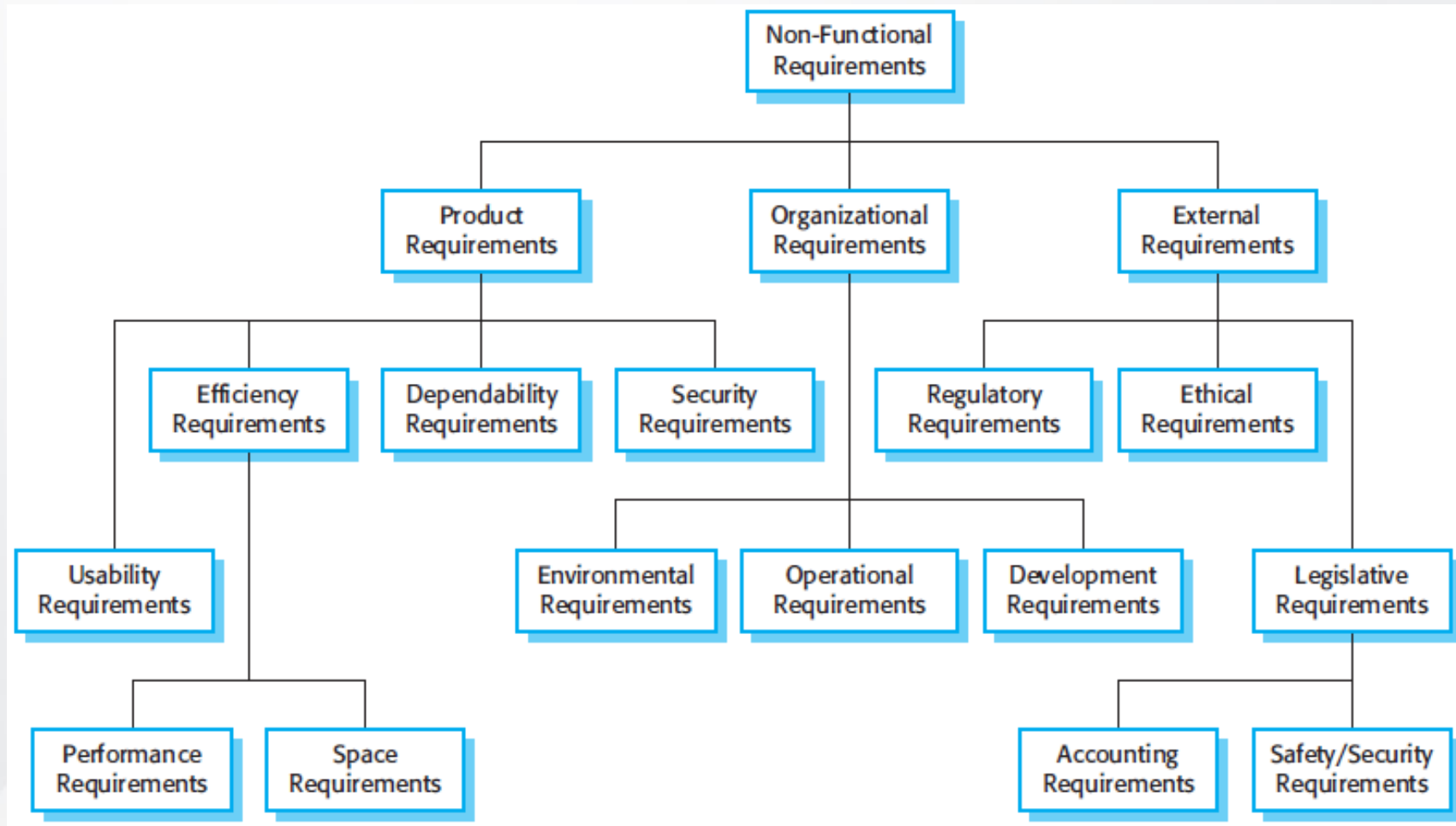
REQUIREMENTS COMPLETENESS AND CONSISTENCY

- Complete
 - They should include descriptions of all facilities required.
- Consistent
 - There should be no conflicts or contradictions in the descriptions of the system facilities.
- In practice, it is impossible to produce a complete and consistent requirements document.

NON-FUNCTIONAL REQUIREMENTS

- These define **system properties and constraints** e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- Process requirements may also be specified mandating a **particular IDE, programming language or development method**.
- Non-functional requirements may be **more critical than functional requirements**. If these are not met, the system may be useless.

TYPES OF NONFUNCTIONAL REQUIREMENT



EXAMPLES OF NONFUNCTIONAL REQUIREMENTS IN THE MHC-PMS

Product requirement

The MHC-PMS shall be available to all clinics during normal working hours (Mon–Fri, 0830–17.30). **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 provisions** as set out in HStan-03-2006-priv.

GOALS AND REQUIREMENTS

- Non-functional requirements may be **very difficult to state** precisely and **imprecise requirements may be difficult to verify**.
- Goal
 - A general intention of the user such as ease of use.
- Verifiable non-functional requirement
 - **A statement using some measure** that can be objectively tested.
- Goals are helpful to developers as they **convey the intentions** of the system users.

EXAMPLE: USABILITY REQUIREMENTS

- The system should be easy to use by medical staff and should be organized in such a way that user **errors are minimized**. (Goal)
- Medical staff shall be able to use all the system functions after four hours of training. After this training, the average number of errors made by experienced users shall not exceed two per hour of system use. (**Testable non-functional requirement**)

METRICS FOR SPECIFYING NONFUNCTIONAL REQUIREMENTS

Property	Measure
Speed	- Processed transactions/second - User/event response time - Screen refresh time
Size	- Mbytes - Number of ROM chips
Ease of use	- Training time - Number of help frames
Reliability	- Mean time to failure - Probability of unavailability - Rate of failure occurrence - Availability
Robustness	- Time to restart after failure - Percentage of events causing failure - Probability of data corruption on failure
Portability	- Percentage of target dependent statements - Number of target systems

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 weather conditions**.
- Domain requirements **may impose new functional requirements**, constraints on existing requirements or define specific computations.
- If domain requirements are not satisfied, the system may be unworkable.

EXAMPLE: TRAIN PROTECTION SYSTEM

- This is a domain requirement for a train protection system:
- The deceleration of the train shall be computed as:
 - $D_{train} = D_{control} + D_{gradient}$
 - where $D_{gradient}$ is $9.81ms^2 * compensated\ gradient / \alpha$ and where the values of $9.81ms^2 / \alpha$ are known for different types of train.
- It is difficult for a non-specialist to understand the implications of this and how it interacts with other requirements.

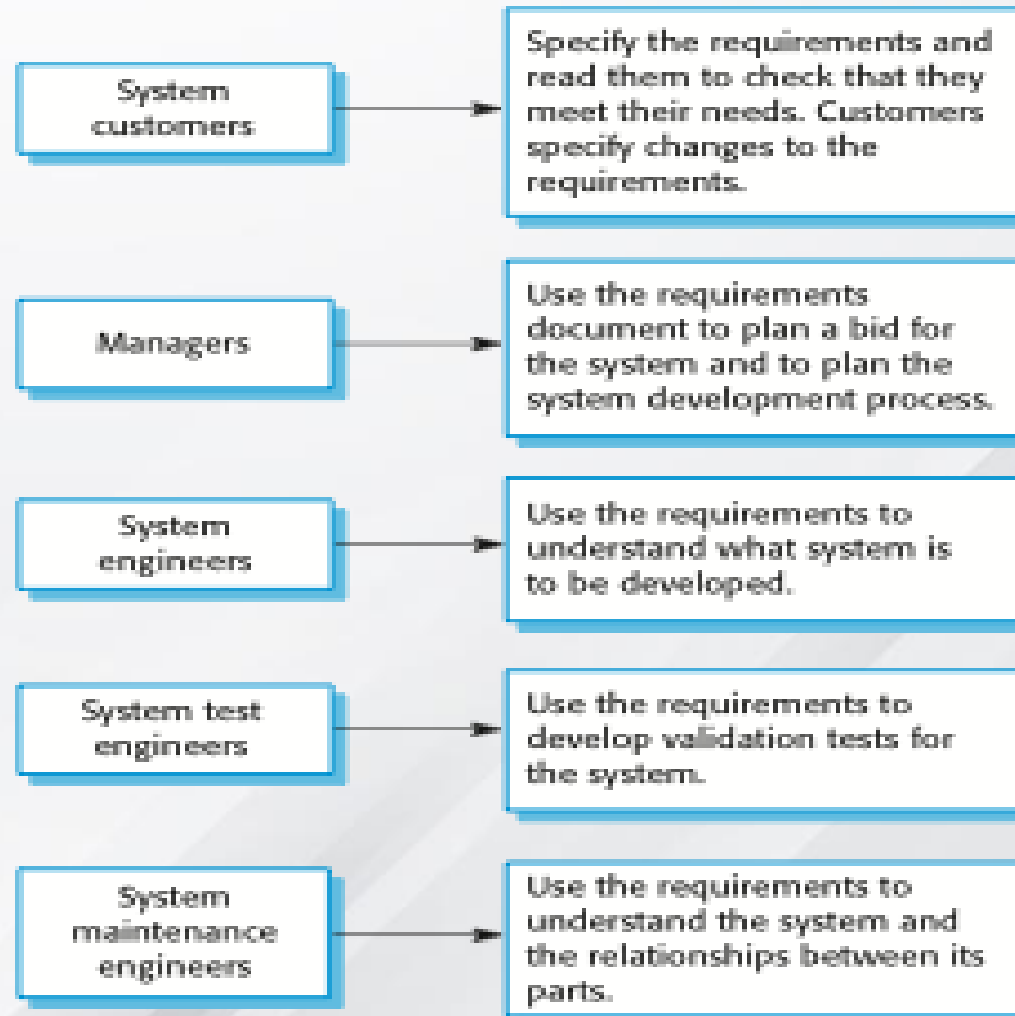
DOMAIN REQUIREMENTS PROBLEMS

- Understandability
 - Requirements are expressed **in the language of the application domain**; This is **often not understood by software engineers** developing the system.
- Implicitness
 - Domain specialists understand the area so well that **they do not think of making the domain requirements explicit.**

THE SOFTWARE REQUIREMENTS DOCUMENT

- The software requirements document is **the official statement** of what is required of the system developers.
- Should include both a definition of **user requirements** and a specification of the **system requirements**.
- **It is NOT a design document.** As far as possible, it should set of **WHAT** the system should do rather than **HOW** it should do it.

USERS OF A REQUIREMENTS DOCUMENT



THE STRUCTURE OF A REQUIREMENTS DOCUMENT

Chapter	Description
Preface	This should define the expected readership of the document and describe its version history , including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system . It should briefly describe the system's functions and explain how it will work with other systems . It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	Here, you describe the services provided for the user . The nonfunctional system requirements should also be described in this section. This description may use natural language, diagrams , or other notations that are understandable to customers. Product and process standards that must be followed should be specified .
System architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules . Architectural components that are reused should be highlighted.

THE STRUCTURE OF A REQUIREMENTS DOCUMENT (CONT.)

Chapter	Description
System requirements specification	This should describe the functional and nonfunctional requirements in more detail . If necessary, further detail may also be added to the nonfunctional requirements. Interfaces to other systems may be defined.
System models	This might include graphical system models showing the relationships between the system components and the system and its environment. Examples of possible models are object models , data-flow models , or semantic data models .
System evolution	This should describe the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs , and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.
Appendices	These should provide detailed , specific information that is related to the application being developed; for example, hardware and database descriptions . Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams , an index of functions , and so on.

NATURAL LANGUAGE SPECIFICATION

- Requirements are written as natural language sentences supplemented by diagrams and tables.
- Used for writing requirements because it is expressive, intuitive and universal. This 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 confusion
 - Functional and non-functional requirements tend to be **mixed-up**.
- Requirements amalgamation
 - **Several** different **requirements** may be expressed **together**.

EXAMPLE REQUIREMENTS FOR THE INSULIN PUMP SOFTWARE SYSTEM

- 3.2 The system **shall measure the blood sugar** and deliver insulin, if required, every 10 minutes. *(Changes in blood sugar are relatively slow so **more frequent** measurement is unnecessary; **less frequent** measurement could lead to unnecessarily high sugar levels.)*
- 3.6 The system **shall run a self-test routine every minute** with the conditions to be tested and the associated actions defined in Table 1. *(A self-test routine can **discover hardware and software problems** and alert the user to the fact the normal operation may be impossible.)*

STRUCTURED SPECIFICATIONS

- An approach to writing requirements where the freedom of the requirements writer is limited, and requirements are written in a **standard way**.
- This works well for some types of requirements e.g. requirements for **embedded control system** but is sometimes **too rigid for writing business system** requirements.

FORM-BASED SPECIFICATIONS

- Definition of the **function** or entity.
- Description of **inputs** and where they come from.
- Description of **outputs** and where they go to.
- Information about the **information needed for the computation** and other entities used.
- Description of the **action to be taken**.
- **Pre and post conditions** (if appropriate).
- The **side effects** (if any) of the function.

TABULAR SPECIFICATION

- Used to **supplement natural language**.
- Particularly useful when you have to define a **number of possible alternative courses of action**.
- For example, the insulin pump systems bases its computations on the rate of change of blood sugar level and the tabular specification explains how to calculate the insulin requirement **for different scenarios**.

TABULAR SPECIFICATION OF COMPUTATION FOR AN INSULIN PUMP

Condition	Action
Sugar level falling ($r_2 < r_1$) (falling)	CompDose = 0
Sugar level stable ($r_2 = r_1$) (stable)	CompDose = 0
Sugar level increasing and rate of increase decreasing ($(r_2 - r_1) < (r_1 - r_0)$)	CompDose = 0
Sugar level increasing and rate of increase stable or increasing ($(r_2 - r_1) \geq (r_1 - r_0)$)	CompDose = round $((r_2 - r_1)/4)$ If rounded result = 0 then CompDose = MinimumDose

REQUIREMENTS ENGINEERING PROCESSES

- There are a number of **generic activities** in requirements engineering
 - Requirements **elicitation**;
 - Requirements **analysis**;
 - Requirements **validation**;
 - Requirements **management**.
- In practice, RE is an **iterative** activity in which these processes are **interleaved**.

REQUIREMENTS ELICITATION AND ANALYSIS

- Software engineers work with a range of system stakeholders to find out about the application domain, the services that the system should provide, the required system performance, hardware constraints, other systems, etc.
- Stages include:
 - Requirements **discovery**,
 - Requirements **classification and organization**,
 - Requirements **prioritization and negotiation**,
 - Requirements **specification**.

PROCESS ACTIVITIES

- **Requirements discovery**
 - Interacting with **stakeholders** to discover their requirements. **Domain** requirements are also discovered at this stage.
- **Requirements classification and organisation**
 - Groups related requirements and organises them into coherent **clusters**.
- **Prioritisation and negotiation**
 - Prioritising requirements and **resolving requirements conflicts**.
- **Requirements specification**
 - Requirements are **documented** and input into the next round of the spiral.

REQUIREMENTS DISCOVERY

- The process of gathering information about the required and existing systems.
- Interaction is with system stakeholders from managers to external regulators.

INTERVIEWING

- **Formal or informal** interviews with stakeholders are part of most RE processes.
- Types of interview
 - **Closed interviews** based on pre-determined list of questions
 - **Open interviews** where various issues are explored with stakeholders.

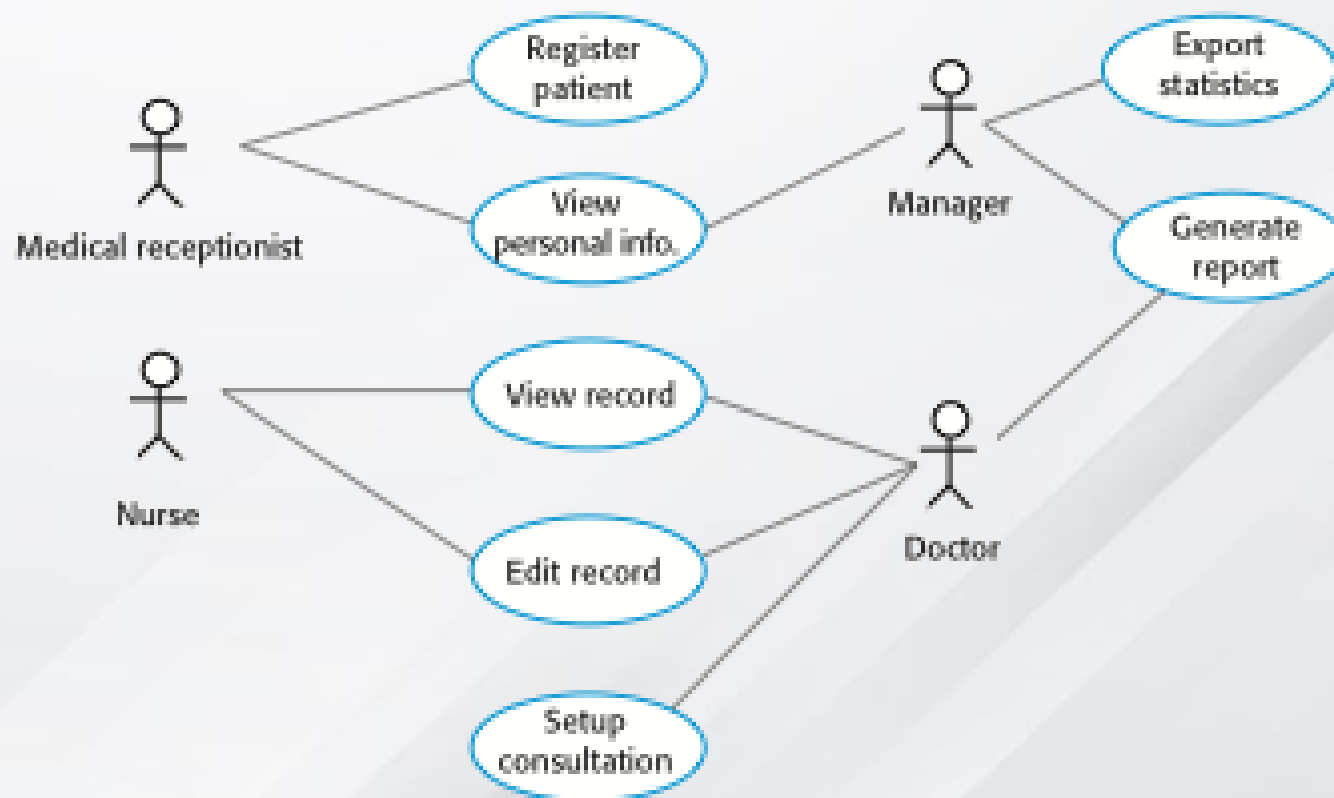
SCENARIOS

- **Scenarios are real-life examples of how a system can be used.**
- **They should include**
 - A description of the **starting situation**;
 - A description of the **normal flow of events**;
 - A description of **what can go wrong**;
 - Information about **other concurrent activities**;
 - A description of the **state when the scenario finishes**.

USE CASES

- Use-cases are a **scenario-based** technique in the UML which identify the actors in an interaction, and which **describe the interaction** itself.
- A set of use cases should describe all possible interactions with the system.
- High-level **graphical model** supplemented by more detailed tabular description.
- **Sequence diagrams** may be used to add detail to use-cases by showing the sequence of event processing in the system.

USE CASES FOR THE MHC-PMS



ETHNOGRAPHY

- A social scientist spends a considerable time observing and analysing how people actually work.
- People do not have to explain or articulate their work.
- Social and organisational factors of importance may be observed.
- Ethnographic studies have shown that **work is usually richer and more complex than suggested by simple system models.**

REQUIREMENTS VALIDATION

- Concerned with demonstrating that the requirements define the system that the customer **really wants**.
- **Requirements error costs are high** so validation is very important
 - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

REQUIREMENTS VALIDATION TECHNIQUES

- Requirements reviews
 - Systematic **manual analysis** of the requirements.
- Prototyping
 - Using an executable model of the system to check requirements. Covered in Chapter 2.
- Test-case generation
 - Developing tests for requirements to **check testability**.

REQUIREMENTS MANAGEMENT

- Requirements management is the process of **managing changing requirements** during the requirements engineering process and system development.
- **New requirements** emerge **as a system is being developed** and after it has gone into use.
- You need to keep track of individual requirements and **maintain links between dependent requirements** so that you can **assess the impact** of requirements changes. You need to establish a **formal process for making change proposals** and linking these to system requirements.

SUMMARY

- Understand the concepts of user and system requirements and why these requirements should be written in different ways;
- Understand the differences between functional and nonfunctional software requirements;
- Understand how requirements may be organized in a software requirements document;
- Understand the principal requirements engineering activities of elicitation, analysis and validation, and the relationships between these activities;
- Understand why requirements management is necessary and how it supports other requirements engineering activities.



Thank You

