Chapter One

Introduction to Requirements Engineering

What is a requirement?

- The software requirements are description of features and functionalities of the target system.
- Something required, something wanted or needed.
  - Webster's dictionary
- There is a huge difference between ***wanted*** and ***needed*** and it should be kept in mind all the time.
- Requirements convey the expectations of users from the software product.
- The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.
- Requirements form the basis of all software engineering projects.

Cont'd…

- Requirements are defined during the **early stages of a system development**
  - as a specification of what should be implemented or as a constraint of some kind on the system.

- **Requirements may be:**
  - a user-level facility description,
  - a detailed specification of expected system behavior,
  - a general system property,
  - a specific constraint on the system,
  - information on how to carry out some computation,
  - A constraint on the development of the system.

Cont'd…

Example:

- "Write a program that will read in a list of 100 positive integers, sort them in ascending order, display the sorted list and display the average of the numbers"

➢ **Requirements:-**

- ✓ Read in a list of 100 positive integers
- ✓ Sort the list of integers in ascending order
- ✓ Display the average of the numbers

Types of requirements

– ***User requirements***

– ***System requirements***

- Software specifications – provide more (design) detail

Cont'd…

User requirements

- Should describe functional and non-functional requirements so that they are understandable by system users who don't have detailed technical knowledge
- User requirements are defined using natural language, tables, and diagrams

Problems with natural language

- Precision vs. understandability
- Functional vs. non-functional requirements confusion
- Requirements amalgamation

Cont'd…

System requirements

- More detailed specifications of user requirements
- Serve as a basis for designing the system
- May be used as part of the system contract
- System requirements may be expressed using system models.

Cont'd…

What happens if the requirements are wrong?

- The system may be delivered late and cost more than originally expected.
- The customer and end-users are not satisfied with the system.
- The system may be unreliable in use with regular system errors and crashes disrupting normal operation.
- If the system continues in use, the costs of maintaining the system is very high.

Example: A Case.pptx

Requirements Engineering

- Refers to the process of defining, documenting and maintaining requirements in the engineering design process.
  - It is a common role in systems engineering and software 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.

- Requirements specify ***what*** **the system is supposed to do**, but not *how* the system is to accomplish its task.

Cont'd…

- The process to **gather** the software requirements from client, **analyse** and **document** them is known as **requirement engineering**.
- The **goal** of requirement engineering is to develop and maintain refined and descriptive 'System Requirements Specification' document.
- In practice, requirements engineering isn't sequential process, it's an **iterative process** in which activities are interleaved.
  - For example, you iterate first on the user requirements; elicitation, specification, and validation, and repeat the same steps for the system requirements.
- Each software development process goes through the phase of requirements engineering
- The use of the term **'engineering'** implies that systematic and repeatable techniques should be used to ensure that system requirements are complete, consistent, relevant, etc.

Cont'd…

Why is requirements engineering difficult?

- Businesses operate in a rapidly changing environment so their requirements for system support are constantly changing.
- Multiple stakeholders with different goals and priorities are involved in the requirements engineering process.
- System stakeholders do not have clear ideas about the system support that they need.
- Requirements are often influenced by political and organisational factors.

Functional and Non-functional Requirements

Functional Requirements

- Functional requirements define what a system is supposed to *do*.
- Functional requirements are usually in the form of "system shall do <requirement>", an individual action or part of the system.
- They define functions and functionality within and from the software system.
- Functional requirement captures the behavioral aspects / function of the proposed automated system.
- Functional requirements are the back bone of all software products.

Cont'd…

- **Requirements are categorized logically as:**
  - **Must Have** : Software cannot be said operational without them.
  - **Should have** : Enhancing the functionality of software.
  - **Could have** : Software can still properly function with these requirements.
  - **Wish list** : These requirements do not map to any objectives of software.

- While developing software, 'Must have' must be implemented, 'Should have' is a matter of debate with stakeholders, whereas 'could have' and 'wish list' can be kept for software updates.

Cont'd…

❖ **Examples of functional requirement:**

- **Requirement #1:**
  - ✓ The system shall solve a quadratic equation using the following formula.
  - ✓ $X=(- b + sqrt (b^2 - 4 * a * c))/2 * a$
- **Requirement #2:**
  - – The user shall be able to search either the entire database of patients or select a subset from it (admitted patients or patients with asthma, etc)
- **Requirement #3:**
  - – The system shall provide appropriate viewers for the user to read documents in the document store.

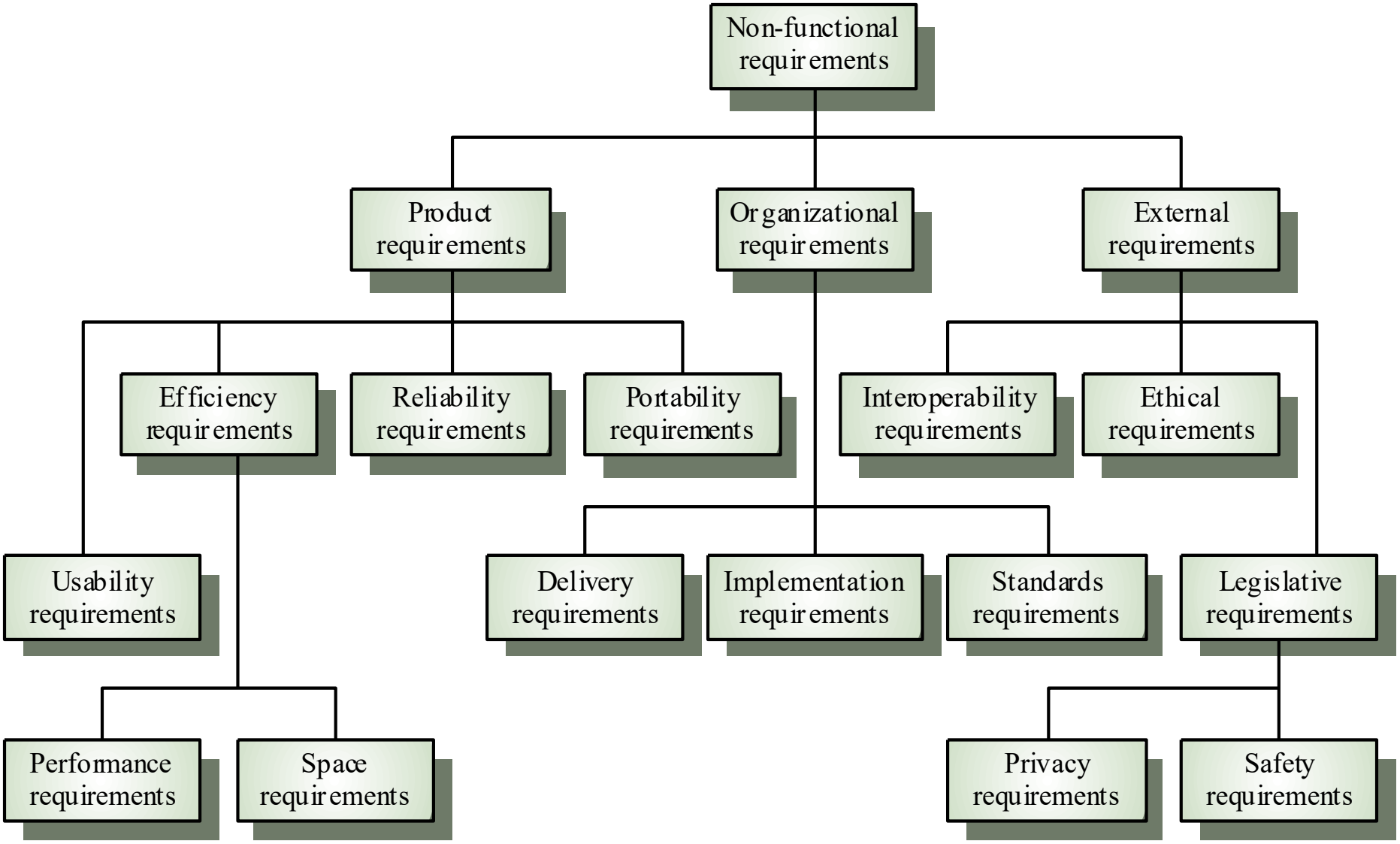
Non-Functional Requirements

- Non-functional requirements define **how a system is supposed to *be*.**
- Non-functional requirements are in the form of "system shall be <requirement>", an overall property of the system as a whole or of a particular aspect and not a specific function.
- Non-functional requirements are often called "quality attributes" of a system.
- Other terms for non-functional requirements are "qualities", "quality goals", "quality of service requirements", "constraints" and "non-behavioral requirements".
- Informally these are sometimes called the "ilities", from attributes like stability and portability.

Cont'd…

- Qualities that is non-functional requirements can be divided into two main categories:
  - **Execution qualities**, such as safety, security and usability, which are observable during operation (at run time).
  - **Evolution qualities**, such as testability, maintainability, extensibility and scalability, which are embodied in the static structure of the system.

Types of Non-Functional Requirements (NFRs)



Product-oriented attributes

- ✓ **Performance :** (a) response time, (b) throughput (number of operations performed per second)
- ✓ **Usability:** effort required to learn, use, provide input and interpret results of a program
- ✓ **Efficiency:** minimal use of resources (memory, processor, disk, network…)
- ✓ **Reliability:** of computations, precision
- ✓ **Security:** resistance to unauthorized attempts
- ✓ **Robustness:** in the presence of faults, stress, invalid inputs…
- ✓ **Adaptability:** to other environments or problems
- ✓ **Scalability:** for large number of users or quantities of data
- ✓ **Cost:** total cost of ownership (TCO) for acquisition, installation.
- – **Portability:** does it work for several platforms
- – **Modifiability:** addition of new functionalities
- – **Reusability:** of components, code, designs, and even requirements in other systems

Process-oriented attributes

- **Maintainability:** changes to functionalities, repairs
- **Readability:** of code, documents
- **Testability:** ease of testing and error reporting
- **Understandability:** of design, architecture, code
- **Integrability:** ability to integrate components
- **Complexity:** degree of dependency and interaction between components

Performance Measures

- **Lots of measures**
  - Response time, number of events processed/denied in some interval of time, throughput, capacity, usage ratio, loss of information, latency...
  - Usually with probability and confidence interval.

Cont'd…

- **Examples of performance requirements**
  - The system shall be able to process 100 payment transactions per second in peak load.
  - In standard workload, the CPU usage shall be less than 50%, leaving 50% for background jobs.
  - Production of a simple report shall take less than 20 seconds for 95% of the cases.
  - Scrolling one page up or down in a 200 page document shall take at most 1 second.

Reliability Measures

- **Measure degree to which the system performs as required**
  - Includes resistance to failure
  - Ability to perform a required function under stated conditions for a specified period of time
  - Very important for critical systems
- **Can be measured using**
  - Probability that system will perform its required function for a specified interval under stated conditions
  - Mean-time to failure
  - Defect rate
  - Degree of precision for computations

Cont'd…

- **Examples of Reliability Measures**
  - The system defect rate shall be less than 1 failure per 1000 hours of operation.
  - No more than 1 per 1000000 transactions shall result in a failure requiring a system restart.

Availability Measures

- Definition: Percentage of time that the system is up and running correctly
- Can be calculated based on Mean-Time Between Failure (MTBF) and Mean-Time to Repair (MTTR)
  - **MTBF** : Length of time between failures
  - **MTTR** : Length of time needed to resume operation after a failure
  - Availability = MTBF/(MTBF+MTTR)

Cont'd…

- **Examples**
  - The system shall meet or exceed 99.99% uptime.
  - The system shall not be unavailable more than 1 hour per 1000 hours of operation.
  - Less than 20 seconds shall be needed to restart the system after a failure 95% of the time. (This is a MTTR requirement)

Availability	**Downtime**
90%	36.5 days/year
99%	3.65 days/year
99.9%	8.76 hours/year
99.99%	52 minutes/year
99.999%	5 minutes/year
99.9999%	31 seconds/year

Security Measures

- There are at least two measures:
  - The ability to resist unauthorized attempts at usage
  - Continue providing service to legitimate users while under denial of service attack (resistance to DoS attacks)
- **Measurement methods:**
  - Success rate in authentication
  - Resistance to known attacks
  - Time/efforts/resources needed to find a key
  - Probability/time/resources to detect an attack
  - Percentage of useful services still available during an attack
  - Percentage of successful attacks
  - Lifespan of a password, of a session
  - Encryption level

Cont'd…

- **Security may lead to architectural requirements**
  - Authentication, authorization, audit
  - Detection mechanisms
  - Firewall, encrypted communication channels
- **Examples of Security requirements**
  - The application shall identify all of its client applications before allowing them to use its capabilities.
  - The application shall ensure that the name of the employee in the official human resource and payroll databases exactly matches the name printed on the employee's social security card.
  - At least 99% of intrusions shall be detected within 10 seconds.

Usability Measures

- In general, concerns ease of use and of training end users.

❖ The following more specific measures can be identified:

- **Learnability**
  - Proportion of functionalities or tasks mastered after a given training time.
- **Efficiency**
  - Acceptable response time
  - Number of tasks performed or problems resolved in a given time
  - Number of mouse clicks needed to get to information or functionality
- **Memorability**
  - Number (or ratio) of learned tasks that can still be performed after not using the system for a given time period
- **Error avoidance**
  - Number of error per time period and user class
  - Number of calls to user support

Cont'd…

- **Error handling**
  - Mean time to recover from an error and be able to continue the task
- **User satisfaction**
  - Satisfaction ratio per user class
  - Usage ratio

❖ **Examples**

- Four out of five users shall be able to book a guest within 5 minutes after a 2-hour introduction to the system.
- Novice users shall perform tasks X and Y in 15 minutes.
- Experienced users shall perform tasks X and Y in 2 minutes.
- At least 80% of customers asked after a 3 months usage period shall rate their satisfaction with the system at 7 and more on a scale of 1 to 10.

Maintainability Measures

- **Measures ability to make changes quickly and cost effectively**
  - Extension with new functionality
  - Deleting unwanted capabilities
  - Adaptation to new operating environments (portability)
  - Restructuring (rationalizing, modularizing, optimizing, creating reusable components)
- **Can be measured in terms of**
  - Coupling/cohesion metrics, cyclomatic complexity(measure of the number of linearly independent paths through a program's source code)
  - Mean time to fix a defect, mean time to add new functionality
  - Quality/quantity of documentation

Cont'd…

- **Measurement tools**
  - code analysis tools such as IBM Structural Analysis for Java (http://www.alphaworks.ibm.com/tech/sa4j)
- **Examples of requirements**
  - Every program module must be assessed for maintainability according to procedure xx. 70% must obtain "highly maintainable" and none "poor".
  - The cyclomatic complexity of code must not exceed 7.
  - No method in any object may exceed 200 lines of code.
  - Installation of a new version shall leave all database contents and all personal settings unchanged.
  - The product shall provide facilities for tracing any database field to places where it is used.

Testability Measures

- Measures the ability to detect, isolate, and fix defects
  - Time to run tests
  - Time to setup testing environment (development and execution)
  - Probability of visible failure in presence of a defect
  - Test coverage (requirements coverage, code coverage…)
- ➢ May lead to architectural requirements
  - Mechanisms for monitoring
  - Access points and additional control
- Examples
  - The delivered system shall include unit tests that ensure 100% branch coverage.
  - Development must use regression tests allowing for full retesting in 12 hours.

Portability Measures

- Measure ability of the system to run under different computing environments
  - Hardware, software, OS, languages, versions, combination of these
- **Can be measured as**
  - Number of targeted platforms (hardware, OS…)
  - Proportion of platform specific components or functionality
  - Mean time to port to a different platform
- **Examples**
  - No more than 5% of the system implementation shall be specific to the operating system.
  - The meantime needed to replace the current Relational Database System with another Relational Database System shall not exceed 2 hours.
  - No data loss should arise.

Integrability and Reusability Measures

- **Integrability**
  - Measures ability to make separated components work together
  - Can be expressed as
    - Mean time to integrate with a new interfacing system
- **Reusability**
  - Measures ability that existing components can be reused in new applications
  - Can be expressed as
    - Percentage of reused requirements, design elements, code, tests…
    - Coupling of components
    - Degree of use of frameworks

Robustness Measures

- Measure ability to cope with the unexpected situations.
  - Percentage of failures on invalid inputs
  - Degree of service degradation
    - Minimum performance under extreme loads
    - Active services in presence of faults
    - Length of time for which system is required to manage stress conditions
- **Examples**
  - The estimated loss of data in case of a disk crash shall be less than 0.01%.
  - The system shall be able to handle up to 10000 concurrent users when satisfying all their requirements and up to 25000 concurrent users with browsing capabilities.

Domain-specific Measures

- The most appropriate quality measures may vary from one application domain to another,

❖ E.g.

- **Performance**
  - ➢ **Web-based system:**
    - Number of requests processed per second
  - ➢ **Video games:**
    - Number of 3D images per second
- **Accessibility**
  - – Web-based system:
    - Compliance with standards for the blind
  - – Video games:
    - Compliance with age/content ratings systems (e.g., no violence)

Thank You!

?