Requirement Engineering

Prof. Dr. Shazia Shoaib

- Non-functional requirements are the general goals.

e.g. Software product should be easy to use. Now this is a goal. It sets a priority for ease of customer. It tells us as a designers that we should focus on these areas which are very dear to the customers.

However, these general goals are very difficult to verify. One can't say that this product is easier to use as compared to the other product. Unless he has certain things to measure for both of these product. Then we can say 'yes', we can quantify these objects and the product is easier than other product.

- General goals should be expressed in quantitively measures. So, we can objectively test them.

e.g. **Goal (unverifiable)**

The system should be easy to use by experienced controllers and should be organized in such a way that user errors are minimized.

Goal (verifiable)

Experienced controller shall be able to use all the system functions after a total of two hours training. After this training the average number of errors made by experienced users shall not exceed two per day.

Explanation

First clearly has no indication of any matrix or any quantifiability with in it. However, in the second requirement which is verifiable requirement this clearly stated that experienced controller shall not required more than 2 hours training and once the training has happened the average no of errors made by the controller shall not exceed 2 per day. So, if product meet that condition or that requirement, then one can say it is very quantifiable requirement and this can be measured and can be coved to the customers that this requirements has been met.

Metrics of Non-functional Requirements

1. Speed

Performance of any product is very important. To measure performance speed is an expect. Speed of execute transactions, performance of functionality etc.

I. **Processed transactions per second.** (how many transaction a product complete in one second. This is for transaction-oriented product)

Transaction oriented: A way of managing data and operations in a system where a group of related tasks or operations are treated as a single unit or transaction.

Give any real time example?

Bill Payment Transaction

A user wants to pay their utility bills through the online banking platform. This involves deducting the bill amount from their checking account and updating the payment records.

Atomicity: The transaction is atomic, meaning either the bill payment is completed successfully, or no changes occur, and the user's account balance remains unchanged.

Consistency: The system ensures that the payment is consistent with the user's account balance and updates the payment records accurately.

Isolation: The transaction is isolated from other transactions, ensuring that it operates independently.

Durability: Once the payment is confirmed, the changes are permanent and will not be lost even in the event of a system failure.

Speed

II. **Response Time.** (after heavy calculation how long to response)

In requirement engineering, response time is a critical speed metric that measures the time taken by a system to respond to a user's input or request.

Any real time example?

E-commerce Website Checkout Process

Customers can browse products, add items to their cart, and proceed to checkout.

- **User Action**: Adding an item to the cart

The user clicks the "Add to Cart" button for a product they wish to purchase.

Response Time: The time taken from clicking the button to the cart displaying the updated item count.

- **User Action:** Proceeding to Checkout ("Checkout process response time should be under 5 seconds")

The user clicks the "Proceed to Checkout" button after adding all desired items to the cart.

Response Time: The time taken from clicking the button to the checkout page being displayed.

- **User Action:** Entering Payment Information

The user fills in their payment details and clicks the "Submit Payment" button.

Response Time: The time taken from clicking the button to the system confirming successful payment

Speed

III. **Screen refresh time.** (computer games, graphic oriented use this more)

Why use screen refresh time?

Screen refresh time, or refresh rate, is an important consideration for various reasons related to user experience, visual quality, health, and the requirements of specific applications.

Example:

- **Gameplay Experience:**

The gamer is playing a fast-paced action video game, like a first-person shooter.

Refresh Rate: The monitor has a refresh rate of 144Hz, meaning the screen refreshes 144 times per second.

User Experience: The gamer experiences smoother, more responsive gameplay because the screen refreshes quickly, reducing motion blur and providing a competitive edge due to faster updates of the on-screen action.

Comparison: If the gamer were to play the same game on a standard 60Hz monitor, the experience might be less smooth, and fast movements could appear slightly blurred due to the lower refresh rate.

2. Size

Size of product is also important expect. Complexity of product increases so size also important to solve this. It is also very important for determining the amount of time it takes to develop software.

It is more important in computing power problems, disks space problems.

I. K byte: to measure size in kilo bytes of executable source or source code (not a full proof way to determine exact size but no on function can determine the size)
II. No. of function point: count of function points, which can then be used for estimating project effort, cost, and other metrics.

3. Ease of use

It is important for all software products. It is easy to use. It was used as user-friendly term, but now use as ease of use.

I. Training time: to use a system (associated to general goal of software and also have impact on the training for the easy use of product for user)

II. No of help frames: how much help build within the product (no need to take help from developers by call or mail) - depend on the size of products

Example: frequently asked questions in the products

4. Reliability

Reliability means how much you depend on a software product?

I. Mean time to failure: Mean time to failure calculate the life-time of the non-repairable or replicable asset of product before it fails. Example: Fan belt

II. Rate of failure: how frequently failure occur (less reliable product)

III. Availability: probability of a product for unavailability (should be small)

Example:

If you want to access the website but not able to connect to the site. Then this site is unavailable for service. You may switch to other service. If availability is high, then that product is reliable for customer.

5. Robustness

Cope up with faults that occurred. The processing of system not get effected by the occurrence of an occurrence of a fault and the system must keep going.

Types of faults:

- Hardware faults (disk fail, device timeout)
- Software faults (bugs, error, defects)
- User faults (entering data in different format)

Properties of Robustness

1. **Time to restart after failure:** how long a product take to response after failure has occurred (if down time is small robustness is greater)
2. **Percentage of events causing failure:** These are not minor errors or mistakes but failures.

Example:

If 100 events occurred and 8 or 9 events causing failures, it is a very high number. So, minimize the percentage to 0 of failure occurrence.

3. **Probability of data corruption on failures:** whenever failure occurs, the data that was used by software product become unreliable. Because product can be restarted but it will be very difficult to generate that data. Need to be sure that data is reliable.

6. Portability

- How well actions performed via one platform are run on another.
- How a system can be launched within one environment or another.

We must be careful using system dependent or target system dependent statements.

In large no. of target depended system, product is not easily able to port in it.

Kinds of requirements

- Functional requirements
- Non-functional requirements

➢Domain requirements

➢Inverse requirements

➢Design and implementation constraints

Domain Requirements

- Requirements that come from the application domain and reflect fundamental characteristics of that application domain
- These can be both the functional or non-functional requirements
- These requirements, sometimes, are not explicitly mentioned
- Domain experts find it difficult to convey domain requirements
- Their absence can cause significant dissatisfaction
- Domain requirements can impose strict constraints on solutions. This is particularly true for scientific and engineering domains
- Domain-specific terminology can also cause confusion

Example

- Banking domain has its own specific constraints, for example, most banks do not allow over-draw on most accounts, however, most banks allow some accounts to be overdrawn

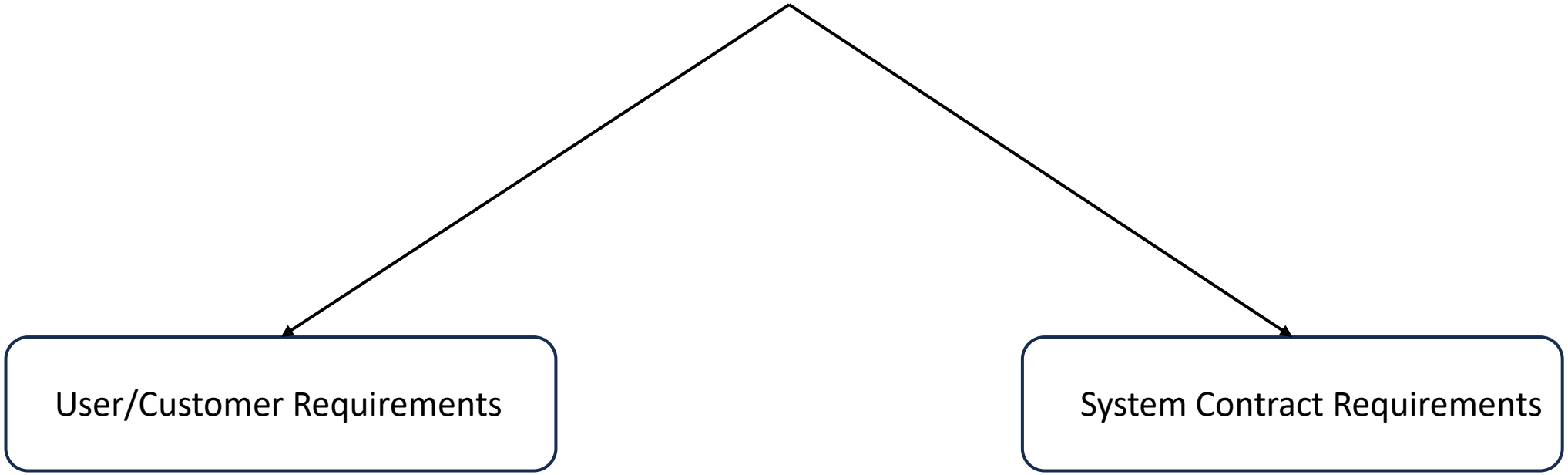
Inverse Requirements

- They explain what the system shall not do. Many people find it convenient to describe their needs in this manner
- These requirements indicate the indecisive nature of customers about certain aspects of a new software product
- Example: The system shall not use red color in the user interface, whenever it is asking for inputs from the end-user

Design and Implementation Constraints

- They are development guidelines within which the designer must work
- These requirements can seriously limit design and implementation options
- Can also have impact on human resources

Another View of Requirements



User/ customer requirements

- Functional or non-functional
- User cannot speak/understand computer language.
- Describe expected services in natural language.
- Requirements and services are understandable by users, because they do not have technical knowledge.
- It eliminate detailed discussion about system requirements.
- Unnecessary details are not added.
- Highlight need if the requirements.

System/ contact requirements

- Provide technical details of system.
- Services and constraint details.
- Should complete and consistent.
- If partially documented, chances of failure.
- Starting step of the system architecture.
- Understand by technical/ developers.

Requirements Problems

- The requirements don’t reflect the real needs of the customer for the system
- Requirements are inconsistent and/or incomplete
- It is expensive to make changes to requirements after they have been agreed upon
- There are misunderstandings between customers, those developing the system requirements, and software engineers developing or maintaining the system

Impact of Wrong Requirements

- When requirements are wrong, systems are late, unreliable and don’t meet customers needs
- Results in enormous loss of time, revenue, market share, and trust of customers