SOFTWARE REQUIREMENT ENGINEERING

LECTURE NO: 8

BY: NAZISH NOUMAN

Beyond Functionality

The Intro…

There's more to software success than just delivering the right functionality.

- **Users also have expectations, often unstated, about *how well* the product will work.**
- **Such expectations include *how easy it is to use*, *how quickly it executes, how rarely it fails, how it handles unexpected conditions* and perhaps, *how loud it is.***
- **Such characteristics, collectively known as *quality attributes, quality factors, quality requirements, quality of service requirements*, or the "*–ilities*," constitute a major portion of the system's nonfunctional requirements**

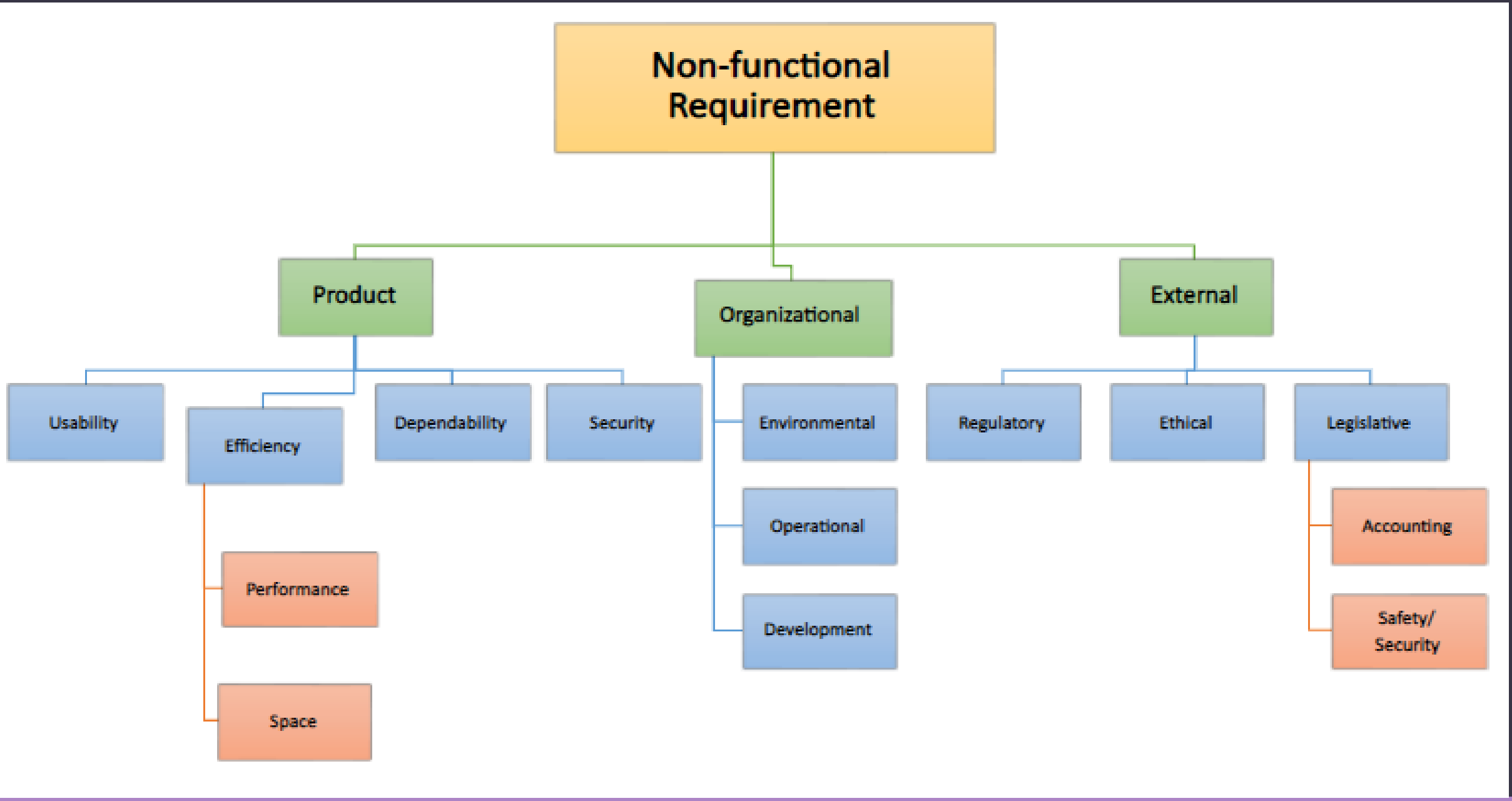
Non- Functional Requirements

They are however quite simple; they are the restrictions or constraints to be placed on the system and how to build it.

Their purpose is to restrict the number of solutions that will meet a set of requirements.

IEEE Definition of NFR ..

"non functional requirement – in software system engineering, a software requirement that describes *not what* the software will do, *but how* the software will do it, for example, software performance requirements, software external interface requirements, design constraints, and software quality attributes. Nonfunctional requirements are difficult to test; therefore, they are usually evaluated subjectively."


Non-functional Requirement
Product
Organizational
External
Usability
Efficiency
Dependability
Security
Environmental
Operational
Development
Regulatory
Ethical
Legislative
Performance
Space
Accounting
Safety/ Security

Non-Functional Requirements: Product Requirements

Product requirements: **These requirements specify how software product performs**

Organizational requirements: **These requirements are derived from the policies and procedures of an organization.**

External requirements: **These requirements include all the requirements that affect the software or its development process externally.**

Software Quality Attributes

One way to classify quality attributes distinguishes those characteristics that are discernible through execution of the software (external quality) from those that are not (internal quality)

External quality factors are primarily important to users, whereas internal qualities are more significant to development and maintenance staff.

Internal quality attributes indirectly contribute to customer satisfaction by making the product easier to enhance, correct, test, and migrate to new platforms.

Software Quality Attributes

TABLE 14-1 Some software quality attributes

External quality	Brief description
Availability	The extent to which the system's services are available when and where they are needed
Installability	How easy it is to correctly install, uninstall, and reinstall the application
Integrity	The extent to which the system protects against data inaccuracy and loss
Interoperability	How easily the system can interconnect and exchange data with other systems or components
Performance	How quickly and predictably the system responds to user inputs or other events
Reliability	How long the system runs before experiencing a failure
Robustness	How well the system responds to unexpected operating conditions
Safety	How well the system protects against injury or damage
Security	How well the system protects against unauthorized access to the application and its data
Usability	How easy it is for people to learn, remember, and use the system
Internal quality	**Brief description**
Efficiency	How efficiently the system uses computer resources
Modifiability	How easy it is to maintain, change, enhance, and restructure the system
Portability	How easily the system can be made to work in other operating environments
Reusability	To what extent components can be used in other systems
Scalability	How easily the system can grow to handle more users, transactions, servers, or other extensions
Verifiability	How readily developers and testers can confirm that the software was implemented correctly

External Quality Attributes

External quality attributes

External quality attributes describe characteristics that are observed when the software is executing.

They profoundly influence the user experience and the user's perception of system quality.

1. Availability

Availability is a measure of the planned up time during which the system's services are available for use and fully operational.

- Formally, availability **equals the ratio of up time to the sum of up time and down time**.
- Availability equals the **mean time between failures (MTBF) for the system divided by the sum of the MTBF and the mean time to repair (MTTR) the system after a failure is encountered.**
- **Scheduled maintenance periods also affect availability.**
- Availability is closely related to **reliability** and is strongly **affected by the maintainability** subcategory of modifiability.

Availability Example

AVL-1. **The system shall be at least 95 percent available on weekdays between 6:00 A.M. and midnight Eastern Time, and at least 99 percent available on weekdays between 3:00 P.M. and 5:00 P.M. Eastern Time.**

AVL-2. **Down time that is excluded from the calculation of availability consists of maintenance scheduled during the hours from 6:00 P.M. Sunday Pacific Time, through 3:00 A.M. Monday Pacific Time**

Availability

The cost of quality

Beware of specifying 100 percent as the expected value of a quality attribute such as reliability or availability. It will be impossible to achieve and expensive to strive for. Life-critical applications such as air traffic control systems do have very stringent—and legitimate—availability demands. One such system had a "five 9s" requirement, meaning that the system must be available 99.999 percent of the time. That is, the system could be down no more than 5 minutes and 15 seconds per year. This one requirement contributed to perhaps 25 percent of the system costs. It virtually doubled the hardware costs because of the redundancy required, and it introduced very complex architectural elements to handle a hot backup and failover strategy for the system.

When eliciting availability requirements, ask questions to explore the following issues

- What portions of the system are most critical for being available
- If scheduled maintenance must be performed periodically, when should it be scheduled? What is the impact on system availability? What are the minimum and maximum durations of the maintenance periods? How are user access attempts to be managed during the maintenance periods?
- What user notifications are necessary if the system becomes unavailable?
- What are the business consequences of the system being unavailable to its users?
- What portions of the system have more stringent availability requirements than others?
- What availability dependencies exist between functionality groups (such as not accepting credit card payment for purchases if the credit-card authorization function is not available)?

2. Installability

Software is not useful until it is installed on the appropriate device or platform.

- Some examples of software installation are:
  - Downloading apps to a phone or tablet; moving software from a PC onto a web server;
  - Updating an operating system;
  - installing a huge commercial system, such as an enterprise resource planning tool;
- Installability describes how easy is it to perform these operations correctly.
- Increasing a system's installability reduces the time, cost, user disruption, error frequency, and skill level needed for an installation operation.

Instability Requirements (Example)

INS-1. **An untrained user shall be able to successfully perform an initial installation of the application in an average of 10 minutes.**

INS-2. **When installing an upgraded version of the application, all customizations in the user's profile shall be retained and converted to the new version's data format if needed.**

INS-3. **Installing this software on a server requires administrator privileges.**

INS-4. **Following successful installation, the installation program shall delete all temporary, backup, obsolete, and unneeded files associated with the application**

Following are examples of some questions to explore when eliciting installability requirements:

- **What installation operations must be performed without disturbing the user's session?**
- **What installation operations will require a restart of the application? Of the computer or device?**
- **What should the application do upon successful, or unsuccessful, installation?**
- **What operations should be performed to confirm the validity of an installation?**
- **Does the user need the capability to install, uninstall, reinstall, or repair just selected portions of the application? If so, which portions?**
- **What other applications need to be shut down before performing the installation?**
- **How should the system handle an incomplete installation, such as one interrupted by a power failure or aborted by the user?**

3. Integrity

Integrity deals with preventing information loss and preserving the correctness of data entered into the system.

- Integrity requirements have no tolerance for error: the data is either in good shape and protected, or it is not.
- Data needs to be protected against threats such as accidental loss or corruption, ostensibly identical data sets that do not match, physical damage to storage media, accidental file erasure, or data overwriting by users.
- **Security sometimes is considered a subset of integrity, because some security requirements are intended to prevent access to data by unauthorized users**

Integrity Example

INT-1. **After performing a file backup, the system shall verify the backup copy against the original and report any discrepancies.**

INT-2. **The system shall protect against the unauthorized addition, deletion, or modification of data.**

INT-3. **The system shall confirm daily that the application executable have not been by the addition of unauthorized code**

Some factors to consider when discussing integrity requirements include

- Ensuring the persistence of changes that are made in the data.
- Ensuring that changes in the data are made either entirely or not at all. This might mean backing out of a data change if a failure is encountered partway through the operation
- Coordinating changes made in multiple data stores, particularly when changes have to be made simultaneously (say, on multiple servers) and at a specifc time (say, at 12:00 A.M. GMT on January 1 in several locations).
- Performing data backups. (At what frequency? Automatically and/or on demand? Of what fles or databases? To what media? With or without compression and verifcation?)
- Restoring data from a backup.
- Archiving of data: what data, when to archive, for how long, with what deletion requirements.
- Protecting data stored or backed up in the cloud from people who aren’t supposed to access it

4. Interoperability

Interoperability indicates how readily the system can exchange data and services with other software systems and how easily it can integrate with external hardware devices.

To assess interoperability, you need to know which other applications the users will employ in conjunction with your product and what data they expect to exchange.

IOP-1. ***The Chemical Tracking System shall be able to import any valid chemical structure from the ChemDraw (version 13.0 or earlier) and MarvinSketch (version 5.0 or earlier) tools.***

Following are some questions you can use when exploring interoperability requirements:

- To what other systems must this one interface? What services or data must they exchange?
- What standard data formats are necessary for data that needs to be exchanged with other systems?
- What specific hardware components must interconnect with the system?
- What messages or codes must the system receive and process from other systems or devices?
- What standard communication protocols are necessary to enable interoperability?
- What externally mandated interoperability requirements must the system satisfy ?

5. Performance

Performance is one of the quality attributes that users often will bring up spontaneously. Performance represents the responsiveness of the system to various user inquiries and actions.

Performance dimension	Example
Response time	Number of seconds to display a webpage
Throughput	Credit card transactions processed per second
Data capacity	Maximum number of records stored in a database
Dynamic capacity	Maximum number of concurrent users of a social media website
Predictability in real-time systems	Hard timing requirements for an airplane's flight-control system
Latency	Time delays in music recording and production software
Behavior in degraded modes or overloaded conditions	A natural disaster leads to a massive number of emergency telephone system calls

Performance…

- All users want their applications to run instantly, but the real performance requirements will be different for a spell-check feature than for a missile's radar guidance system.

- Satisfying performance requirements can be tricky because they depend so much upon external factors such as the speed of the computer being used, network connections, and other hardware components.

- Performance is an external quality attribute because it can be observed only during program execution. It is closely related to the internal quality attribute of *effciency*, which has a big impact on the user-observed performance.

Performance Example

PER-1. **Authorization of an ATM withdrawal request shall take no more than 2.0 seconds.**

PER-2. **The anti-lock braking system speed sensors shall report wheel speeds every 2 milliseconds with a variation not to exceed 0.1 millisecond.**

PER-3. **Webpages shall fully download in an average of 3 seconds or less over a 30 megabits/second Internet connection.**

PER-4. **At least 98 percent of the time, the trading system shall update the transaction status display within 1 second after the completion of each trade.**

6. Reliability

The probability of the software executing without failure for a specific period of time is known as *reliability*

- Reliability problems can occur because of improper inputs, errors in the software code itself, components that are not available when needed, and hardware failures.
- Robustness and availability are closely related to reliability.
- Ways to specify and measure software reliability include the percentage of operations that are completed correctly, the average length of time the system runs before failing (mean time between failures, or MTBF), and the maximum acceptable probability of a failure during a given time period.

Example

REL-1. **The mean time between failures of the card reader component shall be at least 90 days.**

REL-2. **The system defect rate shall be less than 1 failure per 1000 hours of operation.**

REL-3. **No more than 1 per 1000000 transactions shall result in a failure requiring a system restart.**

Following are some questions to ask user representatives when you're eliciting reliability requirements:

- How would you judge whether this system was reliable enough?
- What would be the consequences of experiencing a failure when performing certain operations with the system?
- What would you consider to be a critical failure, as opposed to a nuisance?
- Under what conditions could a failure have severe repercussions on your business operations?
- No one likes to see a system crash, but are there certain parts of the system that absolutely have to be super-reliable?
- If the system goes down, how long could it stay offline before it significantly affects your business operations?

7. Robustness

Robustness is the degree to which a system continues to function properly when confronted with invalid inputs, defects in connected software or hardware components, external attack, or unexpected operating conditions

- Robust software recovers gracefully from problem situations and is forgiving of user mistakes. It recovers from internal failures without adversely affecting the end-user experience.
- Software errors are handled in a way the user perceives as reasonable, not annoying.

Other Attribute Terms of Robustness

Other attribute terms associated with robustness are *fault tolerance are:*

a) ***fault tolerance*** (are user input errors caught and corrected?)

b) ***survivability*** (can the camera experience a drop from a certain height without damage?)

c) ***recoverability*** (can the PC resume proper operation if it loses power in the middle of an operating system update?)

Robustness Example

ROB-1. **If the text editor fails before the user saves the file, it shall recover the contents of the file being edited as of, at most, one minute prior to the failure the next time the same user launches the application.**

ROB-2. **All plot description parameters shall have default values specified, which the Graphics Engine shall use if a parameter's input data is missing or invalid.**

8. Safety

Safety requirements deal with the need to prevent a system from doing any injury to people or damage to property

- Safety requirements might be dictated by *government regulations* or other business rules, and legal or certification issues could be associated with satisfying such requirements.
- Safety requirements frequently are written in the form of *conditions or actions* the system must not allow to occur.

Safety Example

***SAF-1*. The user shall be able to see a list of all ingredients in any menu items, with ingredients highlighted that are known to cause allergic reactions in more than 0.5 percent of the North American population.**

***SAF-2.* If the reactor vessel's temperature is rising faster than 5°C per minute, the Chemical Reactor Control System shall turn off the heat source and signal a warning to the operator.**

***SAF-3.* The system shall terminate any operation within 1 second if the measured tank pressure exceeds 90 percent of the specified maximum pressure.**

When eliciting safety requirements you might need to interview SME who are very familiar with the operating environment or people who have thought a lot about project risks

- Under what conditions could a human be harmed by the use of this product? How can the
system detect those conditions? How should it respond?
- What is the maximum allowed frequency of failures that have the potential to cause injury?
- What failure modes have the potential of causing harm or property damage?
- What operator actions have the potential of inadvertently causing harm or property damage?
- Are there specific modes of operation that pose risks to humans or property?

9. Security

Security deals with blocking unauthorized access to system functions or data, ensuring that the software is protected from malware attacks, and so on.

- Security is a major issue with Internet software
- As with integrity requirements, security requirements have no tolerance for error.

Following are some considerations to examine when eliciting security requirements:

- User authorization or privilege levels (ordinary user, guest user, administrator) and user access controls.
- User identification and authentication (password construction rules, password change frequency, security questions, forgotten logon name or password procedures, biometric identification, account locking after unsuccessful access attempts, unrecognized computer)
- Data privacy (who can create, see, change, copy, print, and delete what information)
- Deliberate data destruction, corruption, or theft
- Protection against viruses, worms, Trojan horses, spyware, rootkits, and other malware
- Firewall and other network security issues
- Encryption of secure data
- Building audit trails of operations performed and access attempts

Security Examples

SEC-1. **The system shall lock a user's account after four consecutive unsuccessful logon attempts within a period of five minutes.**

SEC-2. **The system shall log all attempts to access secure data by users having insufficient privilege levels.**

SEC-3. **A user shall have to change the temporary password assigned by the security officer to a previously unused password immediately following the first successful logon with the temporary password.**

Security Examples

SEC-4. **A door unlock that results from a successful security badge read shall keep the door unlocked for 8.0 seconds, with a tolerance of 0.5 second.**

SEC-5. **The resident antimalware software shall quarantine any incoming Internet traffic that exhibits characteristics of known or suspected virus signatures.**

SEC-6. **The magnetometer shall detect at least 99.9 percent of prohibited objects, with a false positive rate not to exceed 1 percent.**

Following are some questions to explore when eliciting security requirements:

- What sensitive data must be protected from unauthorized access?
- Who is authorized to view sensitive data? Who, specifically, is not authorized?
- Under what business conditions or operational time frames are authorized users allowed to access functionality?
- What checks must be performed to confirm that the user is operating the application in a secure environment?
- How frequently should virus software scan for viruses?
- Is there a specifc user authentication method that must be used?

10. Usability

Usability addresses the myriad factors that constitute what people describe colloquially as user-friendliness, ease of use, and human engineering.

- Analysts and developers shouldn't talk about "friendly" software but rather about software that's designed for effective and unobtrusive usage.
- Usability measures the effort required to prepare input for a system, operate it, and interpret its outputs

Usability encompasses several subdomains beyond the obvious ease of use, including *ease of learning; memorability; error avoidance, handling, and recovery; efficiency of interactions; accessibility; and ergonomics*

Possible design approaches for ease of learning and ease of use

Ease of learning	Ease of use
Verbose prompts	Keyboard shortcuts
Wizards	Rich, customizable menus and toolbars
Visible menu options	Multiple ways to access the same function
Meaningful, plain-language messages	Auto completion of entries
Help screens and tooltips	Auto correction of errors
Similarity to other familiar systems	Macro recording and scripting capabilities
Limited number of options and widgets displayed	Ability to carry over information from a previous transaction
	Command line interface

Usability Examples

USE-1. **A trained user shall be able to submit a request for a chemical from a vendor catalog in an average of three minutes, and in a maximum of five minutes, 95 percent of the time.**

USE-2. **All functions on the File menu shall have shortcut keys defned that use the Control key pressed simultaneously with one other key. Menu commands that also appear in Microsoft Word shall use the same default shortcut keys that Word uses.**

USE-3**. 95 percent of chemists who have never used the Chemical Tracking System before shall be able to place a request for a chemical correctly with no more than 15 minutes of orientation.**

Usability indicators include:

- The average time needed for a specific type of user to complete a particular task correctly.
- How many transactions the user can complete correctly in a given time period.
- What percentage of a set of tasks the user can complete correctly without needing help.
- How many errors the user makes when completing a task.
- How many tries it takes the user to accomplish a particular task, like finding a specific function buried somewhere in the menus.
- The delay or wait time when performing a task.
- The number of interactions (mouse clicks, keystrokes, touch-screen gestures) required to get to a piece of information or to accomplish a task.

Internal Quality Attributes

Internal quality attributes

Internal quality attributes are not directly observable during execution of the software.

- They are properties that a developer or maintainer perceives while looking at the design or code to modify it, reuse it, or move it to another platform.

- Internal attributes can indirectly affect the customer's perception of the product's quality if it later proves difficult to add new functionality or if internal inefficiencies result in performance degradation.

1. Efficiency

Efficiency is a measure of how well the system utilizes *processor capacity, disk space, memory, or communication bandwidth*. If a system consumes too much of the available resources, users will encounter degraded performance.

Efficiency is closely related to the *external quality attribute of performance*

Efficiency Example

EFF-1At least 30 percent of the processor capacity and memory available to the application shall be unused at the planned peak load conditions.

EFF-2. The system shall provide the operator with a warning message when the usage load exceeds 80 percent of the maximum planned capacity.

The BA must ask these type of questions

- What is the maximum number of concurrent users now and anticipated in the future?
- By how much could response times or other performance indicators decrease before users or the business suffer adverse consequences?
- How many operations must the system be able to perform simultaneously under both normal and extreme operating conditions?

2.Modifiability

Modifiability addresses how easily the software designs and code can be understood, changed, and extended.

- It is closely related to **verifiability**.
- If developers anticipate making many enhancements, they can choose design approaches that maximize the software's modifiability.
- High modifiability is critical for systems that will undergo frequent revision, such as those being developed by using an incremental or iterative life cycle.

Some aspects of Modifiability

Modifiability encompasses several other quality attribute terms that relate to different forms of software maintenance

Maintenance type	Modifiability dimensions	Description
Corrective	Maintainability, understandability	Correcting defects
Perfective	Flexibility, extensibility, and augmentability	Enhancing and modifying functionality to meet new business needs and requirements
Adaptive	Maintainability	Modifying the system to function in an altered operating environment without adding new capabilities
Field support	Supportability	Correcting faults, servicing devices, or repairing devices in their operating environment

Modifiability Examples

MOD-1. **A maintenance programmer experienced with the system shall be able to modify existing reports to conform to revised chemical-reporting regulations from the federal government with 10 hours or less of development effort.**

MOD-2. **Function calls shall not be nested more than two levels deep**

SUP-1. **A certified repair technician shall be able to replace the scanner module in no more than 10 minutes.**

SUP-2. **The printer shall display an error message if replacement ink cartridges were not inserted in the proper slots**

3.Portability

The effort needed to migrate software from one operating environment to another is a measure of portability.

Portability has become increasingly important as applications must run in multiple environments, such as Windows, Mac, and Linux; iOS and Android; and PCs, tablets, and phones.

Data portability requirements are also important.

Portability Examples

POR-1. **Modifying the iOS version of the application to run on Android devices shall require changing no more than 10 percent of the source code.**

POR-2. **The user shall be able to port browser bookmarks to and from Firefox, Internet Explorer, Opera, Chrome, and Safari.**

POR-3. **The platform migration tool shall transfer customized user profiles to the new installation with no user action needed.**

When you are exploring portability, questions like the following might be helpful:

- What different platforms will this software need to run on, both now and in the future?
- What portions of the product need to be designed for greater portability than other portions?
- What data fles, program components, or other elements of the system need to be portable?
- By making the software more portable, what other quality attributes might be compromised?

4. Reusability

Reusability indicates the relative effort required to convert a software component for use in other applications

Reusable software must be modular, well documented, independent of a specific application and operating environment, and somewhat generic in capability.

Numerous project artifacts offer the potential for reuse, including requirements, architectures, designs, code, tests, business rules, data models, user class descriptions, stakeholder profiles, and glossary terms

Reusability Examples

REU-1. **The chemical structure input functions shall be reusable at the object code level in other applications.**

REU-2. **At least 30 percent of the application architecture shall be reused from the approved reference architectures.**

REU-3. **The pricing algorithms shall be reusable by future store-management applications.**

Set of questions when you are want to know about reusability requirements

- What existing requirements, models, design components, data, or tests could be reused in this application?
- What functionality available in related applications might meet certain requirements for this application?
- What existing requirements, models, design components, data, or tests could be reused in this application?
- What functionality available in related applications might meet certain requirements for this application?

5. Scalability

Scalability requirements address the ability of the application to grow to accommodate more users, data, servers, geographic locations, transactions, network traffic, searches, and other services without compromising performance or correctness

Scalability is related to **modifiability** and to **robustness**, because one category of robustness has to do with how the system behaves when capacity limits are approached or exceeded.

Scalability..

Scalability has both hardware and software implications.

Scaling up a system could mean acquiring faster computers, adding memory or disk space, adding servers, mirroring databases, or increasing network capacity.

Software approaches might include distributing computations onto multiple processors, compressing data, optimizing algorithms, and other performance-tuning techniques.

Scalability Examples

SCA-1. **The capacity of the emergency telephone system must be able to be increased from 500 calls per day to 2,500 calls per day within 12 hours.**

SCA-2. **The website shall be able to handle a page-view growth rate of 30 percent per quarter for at least two years without user-perceptible performance degradation.**

SCA-3. **The distribution system shall be able to accommodate up to 20 new warehouse centers.**

The following questions could be helpful during BAs discussion with Project Sponsors or SMEs of how much the user base, data volume, or other parameters could grow over time

- What are your estimates for the number of total and concurrent users the system must be able to handle over the next several months, quarters, or years?
- Can you describe how and why data capacity demands of the system might grow in the future?
- What are the minimum acceptable performance criteria that must be satisfied regardless of the number of users?
- What growth plans are available regarding how many servers, data centers, or individual installations the system might be expected to run on?

6. Verifiability

More narrowly referred to as testability, verifiability refers to how well software components or the integrated product can be evaluated to demonstrate whether the system functions as expected.

Designing for verifiability is critical if the product has complex algorithms and logic, or if it contains subtle functionality interrelationships.

Verifiability is also important if the product will be modified often, because it will undergo frequent regression testing to determine whether the changes damaged any existing functionality

Verifiability Examples

VER-1. **The development environment configuration shall be identical to the test configuration environment to avoid irreproducible testing failures.**

VER-2. **A tester shall be able to configure which execution results are logged during testing.**

VER-3**. The developer shall be able to set the computational module to show the interim results of any specified algorithm group for debugging purposes**

Defning verifability requirements can be diffcult. Explore questions like the following:

How can we confrm that specifc calculations are giving the expected results?
■■ Are there any portions of the system that do not yield deterministic outputs, such that it could
be diffcult to determine if they were working correctly?
■■ Is it possible to come up with test data sets that have a high probability of revealing any errors
in the requirements or in their implementation?
■■ What reference reports or other outputs can we use to verify that the system is producing its outputs correctly?

Certain attributes are of particular importance on certain types of projects:

- Embedded systems: performance, efficiency, reliability, robustness, safety, security, usability
- Internet and corporate applications: availability, integrity, interoperability, performance, scalability, security, usability
- Desktop and mobile systems: performance, security, usability
- Gaming companies might want to capture emotional requirements for their software

Implementing quality attribute requirements

Designers and programmers will have to determine the best way to satisfy each quality requirement.

Although these are nonfunctional requirements, they can lead to derived functional requirements, design guidelines, or other types of technical information that will produce the desired product characteristics.

TABLE 14-5 Translating quality attributes into technical specifications

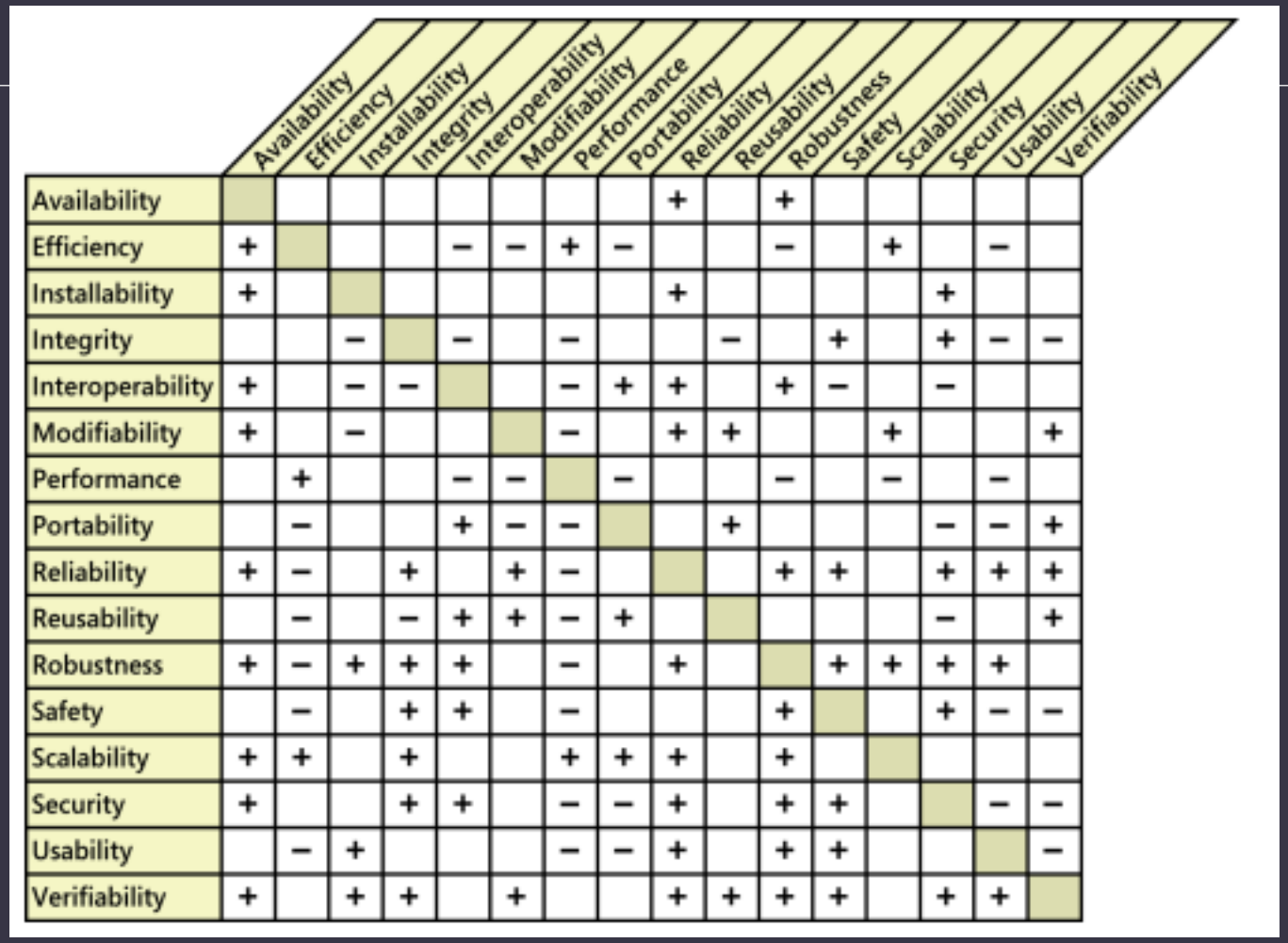
Quality attributes	Likely technical information category
Installability, integrity, interoperability, reliability, robustness, safety, security, usability, verifiability	Functional requirement
Availability, efficiency, modifiability, performance, reliability, scalability	System architecture
Interoperability, security, usability	Design constraint
Efficiency, modifiability, portability, reliability, reusability, scalability, verifiability, usability	Design guideline
Portability	Implementation constraint

For example,

A medical device with stringent availability and reliability requirements might include a backup battery power supply (architecture), along with functional requirements to indicate when the product is operating on battery power, when the battery is getting low, and so forth.

This translation from external or internal quality requirements into corresponding technical information is part of the requirements analysis and high-level design processes.

Quality attribute trade-offs



	Availability	Efficiency	Installability	Integrity	Interoperability	Modifiability	Performance	Portability	Reliability	Reusability	Robustness	Safety	Scalability	Security	Usability	Verifiability
Availability									+		+					
Efficiency	+				−	−	+	−			−		+		−	
Installability	+								+					+		
Integrity			−		−		−			−		+		+	−	−
Interoperability	+		−	−			−	+	+		+	−		−		
Modifiability	+		−				−		+	+			+			+
Performance		+			−	−		−			−		−		−	
Portability		−			+	−	−			+				−	−	+
Reliability	+	−		+		+	−				+	+		+	+	+
Reusability		−		−	+	+	−	+						−		+
Robustness	+	−	+	+	+		−		+			+	+	+	+	
Safety		−		+	+		−				+			+	−	−
Scalability	+	+		+			+	+	+		+					
Security	+			+	+		−	−	+		+	+			−	−
Usability		−	+				−	−	+		+	+				−
Verifiability	+		+	+		+			+	+	+	+		+	+	

Quality attribute trade-offs

- A minus sign in a cell means that increasing the attribute in that row generally adversely affects the attribute in the column.
- An empty cell indicates that the attribute in the row has little effect on the attribute in the column.
- Performance and efficiency have a negative impact on several other attributes. If you write the tightest, fastest code you can, using coding tricks and relying on execution side effects, it's likely to be hard to maintain and enhance. It also could be harder to port to other platforms if you've tuned the code for a specific operating environment.
- Similarly, systems that optimize ease of use or that are designed to be reusable and interoperable with other software or hardware components often incur a performance penalty

Quality attribute trade-offs

- The matrix isn't symmetrical because the effect that increasing attribute A has on attribute B isn't necessarily the same as the effect that increasing B will have on A.

- Figure shows that designing the system to increase performance doesn't necessarily have any effect on security.

- However, increasing security likely will hurt performance because the system must go through more layers of user authentications, encryption, and malware scanning

Constraints

A constraint places restrictions on the design or implementation choices available to the developer.

Constraints can be imposed by external stakeholders, by other systems that interact with the one you're building or maintaining, or by other life cycle activities for your system, such as transition and maintenance.

Sources of constraints include:

- Specific technologies, tools, languages, and databases that must be used or avoided.
- Restrictions because of the product's operating environment or platform, such as the types and versions of web browsers or operating systems that will be used.
- Required development conventions or standards. (For instance, if the customer's organization will be maintaining the software, the organization might specify design notations and coding standards that a subcontractor must follow.)
- Backward compatibility with earlier products and potential forward compatibility, such as knowing which version of the software was used to create a specific data file.

Sources of constraints include:

- Hardware limitations such as timing requirements, memory or processor restrictions, size, weight, materials, or cost.
- Physical restrictions because of the operating environment or because of characteristics or limitations of the users.
- Existing interface conventions to be followed when enhancing an existing product.
- Interfaces to other existing systems, such as data formats and communication protocols.
- Restrictions because of the size of the display, as when running on a tablet or phone.
- Standard data interchange formats used, such as XML, or RosettaNet for e-business

Examples of Constraints..

CON-1. **The user clicks at the top of the project list to change the sort sequence. [specific user interface control imposed as a design constraint on a functional requirement]**

CON-2. **Only open source software available under the GNU General Public License may be used to implement the product. [implementation constraint]**

CON-3. **The application must use Microsoft .NET framework 4.5. [architecture constraint]**

CON-4. **ATMs contain only $20 bills. [physical constraint]**

CON-5. **Online payments may be made only through PayPal. [design constraint]**

CON-6. **All textual data used by the application shall be stored in the form of XML files. [data constraint]**

Handling quality attributes on agile projects in a form of User stories

As a help desk technician, I want the knowledge base to respond to queries within five seconds so the customer doesn't get frustrated and hang up.

As an account owner, I want to prevent unauthorized users from accessing my account so I don't lose any money.