COURSE NAME

SOFTWARE ENGINEERING

CSC 3114

(UNDERGRADUATE)

CHAPTER 11

SOFTWARE QUALITY ATTRIBUTES

PROF. DR. KAMRUDDIN NUR
PROFESSOR, CS, AIUB

OVERVIEW OF SOFTWARE QUALITY REQUIREMENTS

- Software success is more than just delivering the right functionality.
- Characteristics of software that fall into this category include how easy it is to use, how quickly it runs, how often it fails, and how it handles unexpected conditions.
- Such characteristics, collectively known as *software quality attributes* or *quality factors*, are part of the system's nonfunctional (also called non-behavioral) requirements.

OVERVIEW OF SOFTWARE QUALITY ATTRIBUTES

- Customers generally don't present their quality expectations explicitly.
- The trick is to **pin down** just what the users are thinking when they say the software must be user-friendly, fast, reliable, or robust.
- From a technical perspective, quality attributes drive significant architectural and design decisions.
- It's far more difficult and costly to re-architect a completed system to achieve essential quality goals than to design for them at the beginning.

AVAILABILITY

- Availability is a measure of the planned *up time* during which the system is actually available for use and fully operational.

$$\text{Availability equals} = \frac{\text{mean time to failure (MTTF) for the system}}{\text{MTTF + mean time to repair (MTTR) the system after a failure is encountered.}}$$

- Availability requirements become more complex and more important for Web sites or global applications with worldwide users.
- Example: *AV-1. The system shall be at least 99.5 percent available on weekdays between 6:00 a.m. and midnight local time, and at least 99.95 percent available on weekdays between 4:00 p.m. and 6:00 p.m. local time.*

PERFORMANCE

- Performance requirements define how well or how rapidly the system must perform specific functions.
  - speed (e.g. database response times)
  - throughput (transactions per second)
  - capacity (concurrent usage loads)
  - timing (hard real-time demands)
- Performance requirements should also address how the system's performance will degrade in an overloaded situation, such as when a 911 emergency telephone system is flooded with calls.
  - *PE-1. Every Web page shall download in 15 seconds or less over a 50 KBps modem connection.*
  - *PE-2. Authorization of an ATM withdrawal request shall not take more than 10 seconds.*

EFFICIENCY

- Efficiency is a measure of how well the system utilizes processor capacity, disk space, memory, or communication bandwidth (Davis 1993).
- Efficiency is related to performance, (response time) another class of nonfunctional requirement
- If a system consumes too much of the available resources, users will encounter degraded performance, a visible indication of inefficiency.
- Poor performance is an irritant to the user who is waiting for a database query to display results. But performance problems can also represent serious risks to safety, such as when a real-time process control system is overloaded.
- Consider **minimum hardware configurations** when defining efficiency, capacity, and performance goals.
- *Example: EF-1. At least 25 percent of the processor capacity and RAM available to the application shall be unused at the planned peak load conditions.*
- Typical users won't state efficiency requirements in such technical terms. The analyst must ask the questions that will surface user expectations regarding issues such as acceptable performance degradation, demand spikes, and anticipated growth.

FLEXIBILITY

- Flexibility measures how easy it is to add new capabilities to the product
- Also known as *extensibility*, *augmentability*, *extendability*, and *expandability*
- If developers anticipate making many enhancements, they can choose design approaches that maximize the software's flexibility. This attribute is essential for products that are developed in an incremental or iterative fashion through a series of successive releases or by evolutionary prototyping.
- *Example: FL-1. A maintenance programmer who has at least six months of experience supporting this product shall be able to make a new copy output available to the product, including code modifications and testing, with no more than one hour of labor.*

INTEGRITY

- Integrity—which encompasses security, deals with blocking unauthorized access to system functions, preventing information loss, ensuring that the software is protected from virus infection, and protecting the privacy and safety of data entered into the system. Integrity is a major issue with Internet software.
- Users of e-commerce systems want their credit card information to be secure
- Integrity requirements have no tolerance for error
- State integrity requirements in unambiguous terms: user identity verification, user privilege levels, access restrictions, or the precise data that must be protected
  - *Example: IN-1. Only users who have Auditor access privileges shall be able to view customer transaction histories.*

INTEROPERABILITY

- Interoperability indicates how easily the system can exchange data or services with other systems
- 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.
- For example, SIM registration with NID, provide one stop service in supermarket
- *Example: IO-1. The Chemical Tracking System shall be able to import any valid chemical structure from the ChemiDraw (version 2.3 or earlier) and Chem-Struct (version 5 or earlier) tools.*

RELIABILITY

- The probability of the software executing without failure for a specific period of time is known as reliability.
- Ways to measure software reliability include the percentage of operations that are completed correctly and the average length of time the system runs before failing.
- Establish quantitative reliability requirements based on how severe the impact would be if a failure occurred and whether the cost of maximizing reliability is justifiable.
- Systems that require high reliability should also be designed for high testability to make it easier to find defects that could compromise reliability.
- Example: *RE-1. No more than five experimental runs out of 1000 can be lost because of software failures*

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, or unexpected operating conditions
- Robust software recovers gracefully from problem situations and is forgiving of user mistakes
- When eliciting robustness requirements, ask users about error conditions the system might encounter and how the system should react
- Sometimes called *fault tolerance*
- *Example: RO-1. If the editor fails before the user saves the file, the editor shall be able to recover all changes made in the file being edited up to one minute prior to the failure the next time the same user starts the program.*

USABILITY

- Usability measures the effort required to prepare input for, operate, and interpret the output of the product
- Also referred to as *ease of use* and *human engineering*, usability addresses many factors that constitute what users often describe as *user-friendliness*
- Usability also encompasses how easy it is for new or infrequent users to learn to use the product. Ease-of-learning goals can be quantified and measured (e.g. language option)
- *Example: US-1. A trained user shall be able to submit a complete request for a chemical selected from a vendor catalog in an average of four and a maximum of six minutes.*

MAINTAINABILITY

- Maintainability indicates how easy it is to correct a defect or modify the software
- Maintainability depends on how easily the software can be understood, changed, and tested
- It is closely related to flexibility and testability
- High maintainability is critical for products that will undergo frequent revision and for products that are being built quickly
- Maintainability can be measured in terms of the average time required to fix a problem and the percentage of fixes that are made correctly
- *Example: MA-1. A maintenance programmer shall be able to modify existing reports to conform to revised chemical-reporting regulations from the federal government with 20 labor hours or less of development effort.*

REUSABILITY

- Reusability indicates the relative effort involved to convert a software component for use in other applications
- Developing reusable software costs considerably more than creating a component that you intend to use in just one application
- Reusable software must be modular, well documented, independent of a specific application and operating environment, and somewhat generic in capability
- Reusability goals are difficult to quantify
- *Example: RU-1. The chemical structure input functions shall be designed to be reusable at the object code level in other applications that use the international standard chemical structure representations.*

TESTABILITY

- Testability refers to the ease with which software components or the integrated product can be tested to look for defects
- Also known as *verifiability*
- Designing for testability is critical if the product has complex algorithms and logic, or if it contains indirect (ambiguous) functionality interrelationships
- Testability 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
- Example: *TE-1. The maximum cyclomatic complexity* of a module shall not exceed 20.*
- **Cyclomatic complexity* is a measure of the number of logic branches in a source code module

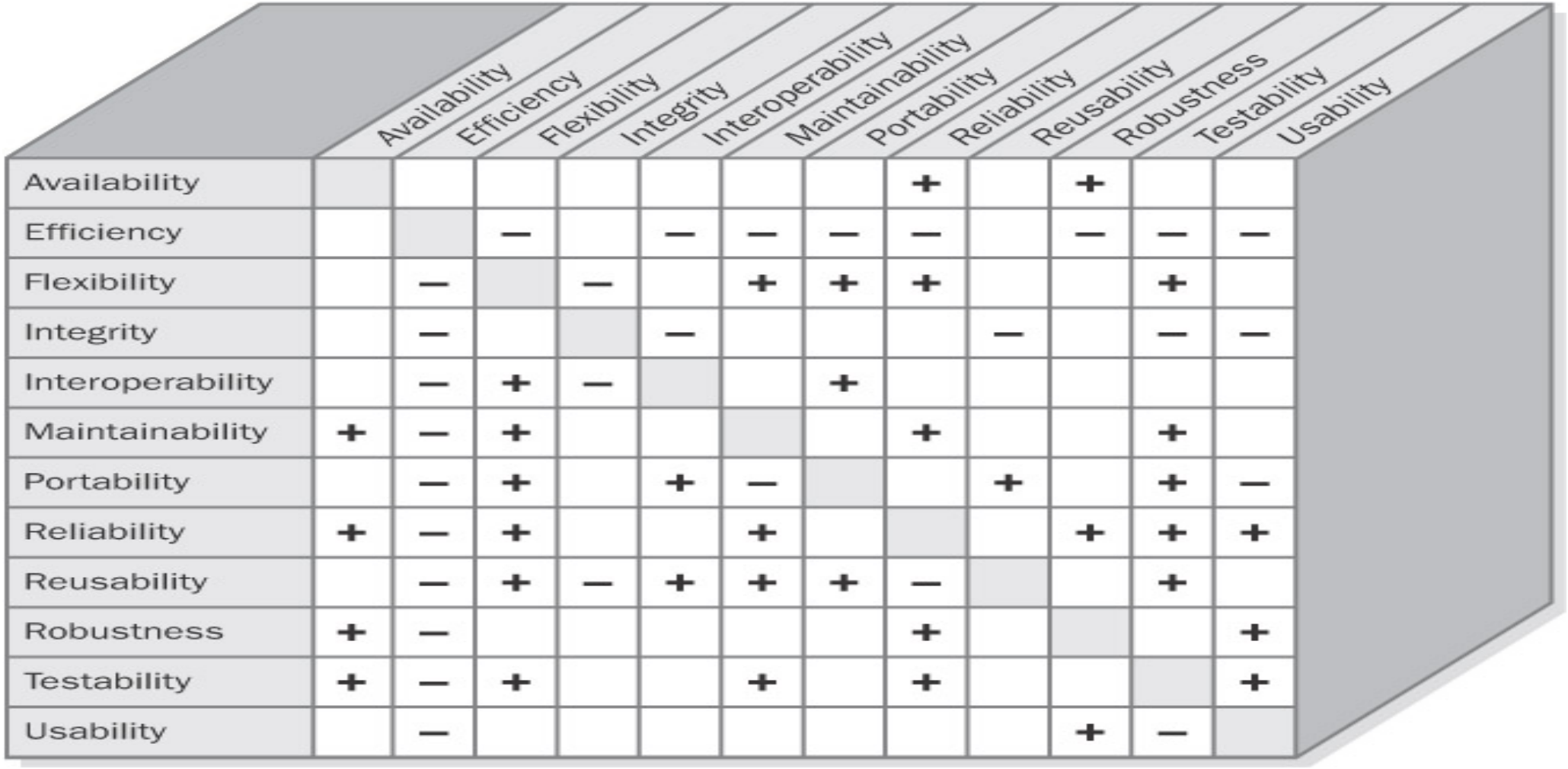
QUALITY ATTRIBUTES

- Different parts of the product need different combinations of quality attributes
- Efficiency might be critical for certain components, while usability is paramount for others
- For different types of systems different types of quality attributes might be critical:
- Embedded systems: efficiency, reliability, safety, installability, serviceability
- Internet and mainframe applications: availability, integrity, maintainability, and scalability
- Desktop systems: interoperability and usability

QUALITY ATTRIBUTES TO WHO?

Important Primarily to Users	Important Primarily to Developers
Availability	Maintainability
Efficiency	Portability (s/w runs on multiple platforms)
Flexibility	Reusability
Integrity	Testability
Interoperability	
Reliability	
Robustness	
Usability	

ATTRIBUTE TRADE-OFFS



	Availability	Efficiency	Flexibility	Integrity	Interoperability	Maintainability	Portability	Reliability	Reusability	Robustness	Testability	Usability
Availability							+		+			
Efficiency			–		–	–	–	–		–	–	–
Flexibility		–		–		+	+	+			+	
Integrity		–			–				–		–	–
Interoperability		–	+	–			+					
Maintainability	+	–	+					+			+	
Portability		–	+		+	–			+		+	–
Reliability	+	–	+			+				+	+	+
Reusability		–	+	–	+	+	+	–			+	
Robustness	+	–						+				+
Testability	+	–	+			+		+				+
Usability		–								+	–	

ATTRIBUTE TRADE-OFFS

- A plus sign in a cell indicates that increasing the attribute in the corresponding row has a positive effect on the attribute in the column
- A minus sign in a cell means that increasing the attribute in that row adversely affects the attribute in the column
- A blank cell indicates that the attribute in the row has little impact on the attribute in the column
- Design approaches that increase a software component's portability also make the software more flexible, easier to connect to other software components, easier to reuse, and easier to test
- Systems that optimize ease of use or that are designed to be flexible, reusable, and interoperable with other software or hardware components often incur a performance penalty

ATTRIBUTE MATRIX

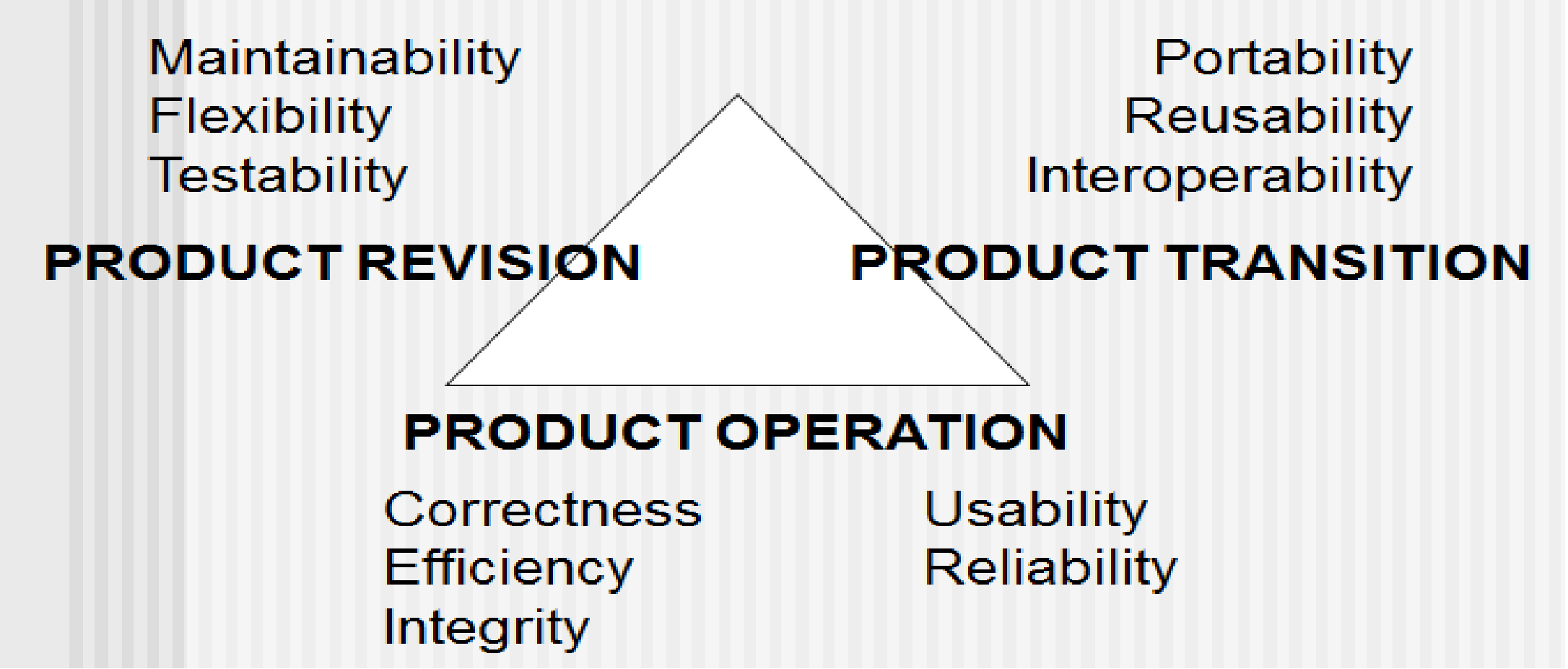
- The matrix isn't **symmetrical** because the effect that increasing attribute A has on attribute B isn't necessarily the same as the effect that increasing B will have on A
- Example shows that designing the system to increase **efficiency** doesn't necessarily have any effect on **integrity**. However, increasing integrity likely will hurt efficiency because the system must go through more layers of user authentications, encryption, virus scanning, and data checkpointing.
- To reach the optimum balance of product characteristics, you must identify, specify, and prioritize the relevant quality attributes during requirements elicitation
- Using the matrix will avoid making commitments to conflicting goals. For example, Don't expect to maximize usability if the software must run on multiple platforms (portability)
- Defining conflicting requirements makes it impossible for the developers to fully satisfy requirements

IMPLEMENTING NON-FUNCTIONAL REQUIREMENTS

- Although quality attributes are nonfunctional requirements, they can lead to derived functional requirements, design guidelines, or other types of technical information that will produce the desired quality characteristics
- *Example: A medical device with strict availability requirements might include a backup battery power supply (architecture) and a functional requirement to visibly or audibly indicate that the product is operating on battery power.*

Quality Attribute Types	**Likely Technical Information Category**
Integrity, interoperability, robustness, usability, safety	Functional requirement
Availability, efficiency, flexibility, performance, reliability	System architecture
Interoperability, usability	Design constraint
Flexibility, maintainability, portability, reliability, reusability, testability, usability	Design guideline
Portability	Implementation constraint

MCCALL'S TRIANGLE OF QUALITY



REFERENCES

- R.S. Pressman & Associates, Inc. (2010). *Software Engineering: A Practitioner's Approach*.
- Kelly, J. C., Sherif, J. S., & Hops, J. (1992). An analysis of defect densities found during software inspections. *Journal of Systems and Software*, *17*(2), 111-117.
- Bhandari, I., Halliday, M. J., Chaar, J., Chillarege, R., Jones, K., Atkinson, J. S., & Yonezawa, M. (1994). In-process improvement through defect data interpretation. *IBM Systems Journal*, *33*(1), 182-214.