BITS Pilani presentation

BITS Pilani
Pilani Campus

Dr. N.Jayakanthan

innovate achieve lead

BITS Pilani

Pilani Campus

SE ZG501
Software Quality Assurance and Testing
Lecture No. 2

QUALITY CULTURE

Tylor [TYL 10] defined **Human Culture as**

“that complex whole which includes knowledge, belief, art, morals, law, custom, and any other capabilities and habits acquired by man as a member of society.”

It is culture that guides the *behaviors, activities, priorities, and decisions* of an individual as well as of an organization.

Wiegers (1996) [WIE 96], in his book “*Creating a Software Engineering Culture,*” illustrates the interaction between the **software engineering culture of an organization, its software engineers, and its projects**

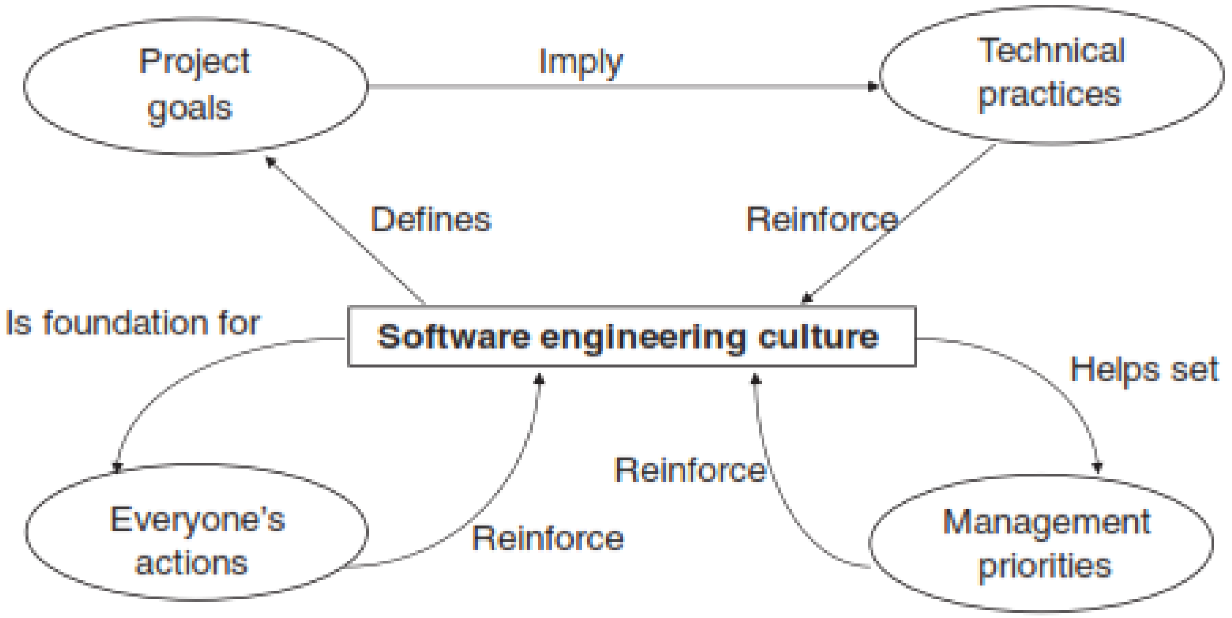


Figure 2.5 Software engineering culture.
Source: Adapted from Wiegers 1996 [WIE 96].

A healthy culture is made up of the following elements:

- The **personal commitment of each developer** to create quality products by systematically applying effective software engineering practices.
- The **commitment to the organization by managers** at all levels to provide an environment in which software quality is a fundamental factor of success and allows each developer to carry out this goal.
- The **commitment of all team members** to constantly improve the processes they use and to always work on improving the products they create.

Figure 2.6 Start coding … I'll go and see what the client wants!
Source: Reproduced with permission of CartoonStock ltd.

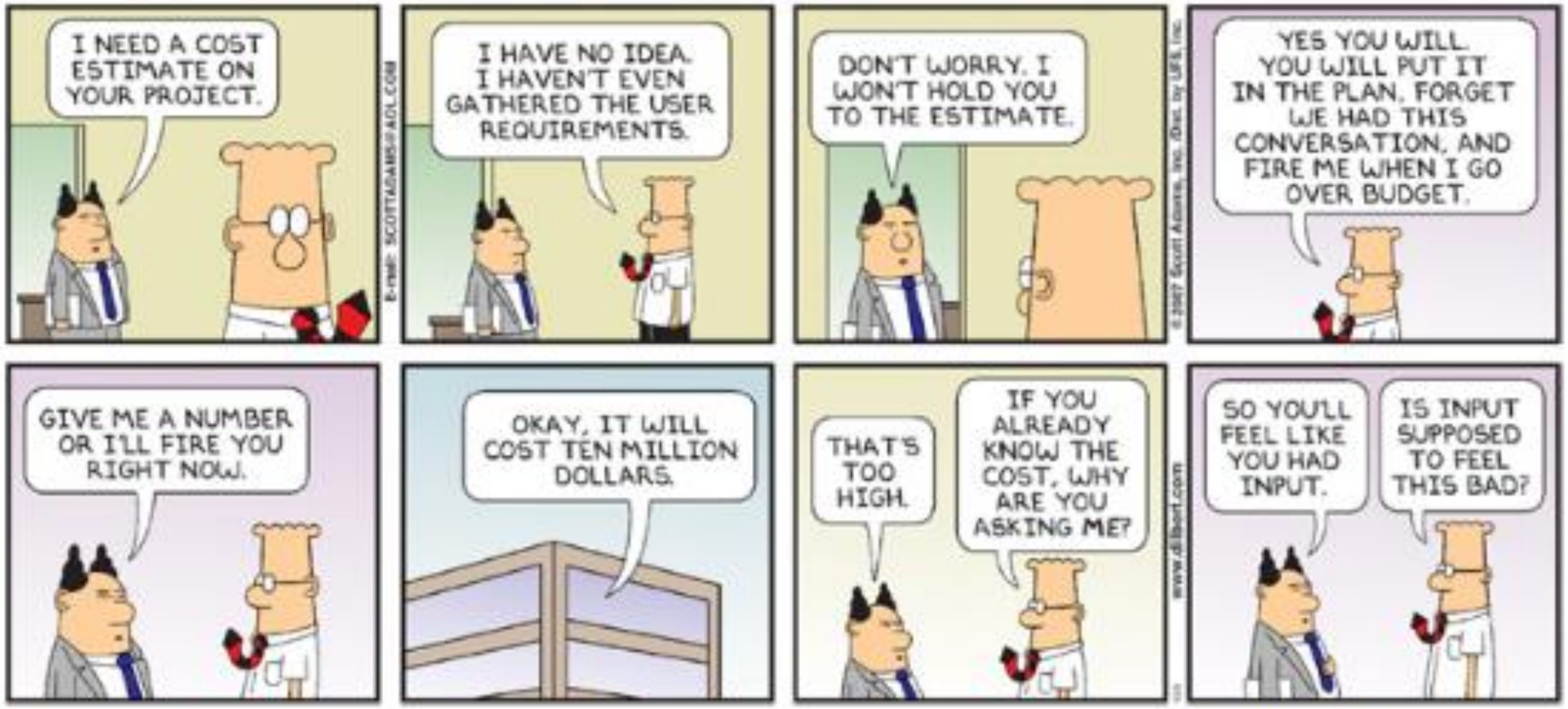

I NEED A COST ESTIMATE ON YOUR PROJECT.
I HAVE NO IDEA. I HAVEN'T EVEN GATHERED THE USER REQUIREMENTS.
DON'T WORRY. I WON'T HOLD YOU TO THE ESTIMATE.
YES YOU WILL. YOU WILL PUT IT IN THE PLAN, FORGET WE HAD THIS CONVERSATION, AND FIRE ME WHEN I GO OVER BUDGET.
GIVE ME A NUMBER OR I'LL FIRE YOU RIGHT NOW.
OKAY, IT WILL COST TEN MILLION DOLLARS.
THAT'S TOO HIGH.
IF YOU ALREADY KNOW THE COST, WHY ARE YOU ASKING ME?
SO YOU'LL FEEL LIKE YOU HAD INPUT.
IS INPUT SUPPOSED TO FEEL THIS BAD?
www.dilbert.com

Figure 2.7 Dilbert is threatened and must provide an estimate on the fly. DILBERT © 2007 Scott Adams. Used By permission of UNIVERSAL UCLICK. All rights reserved.

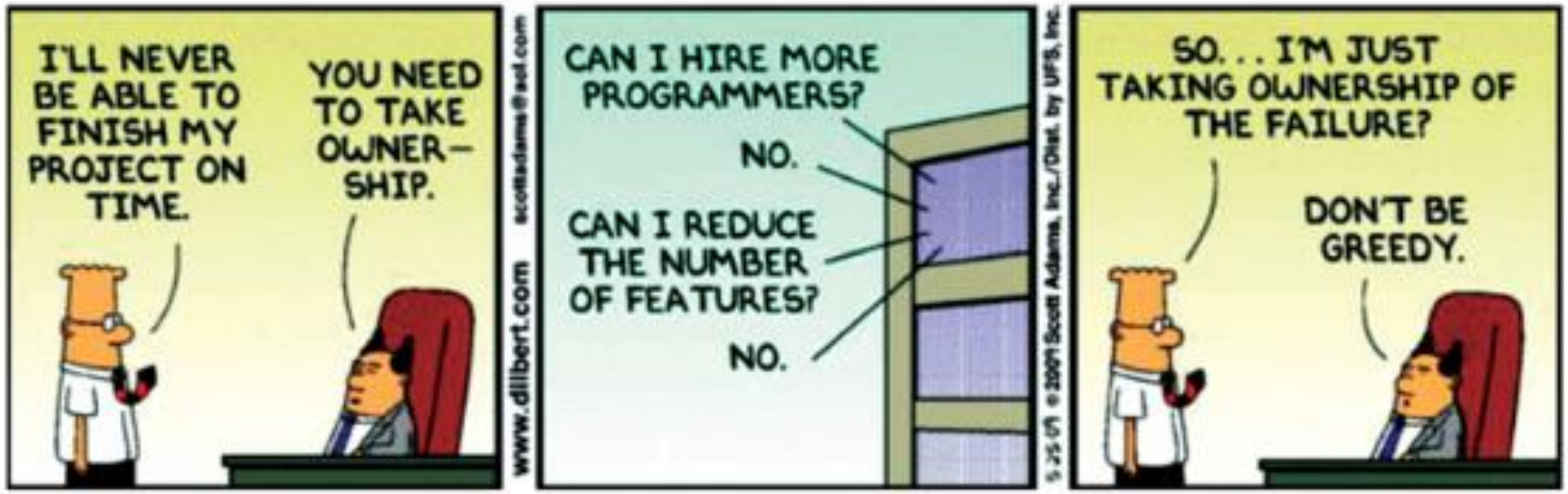

I'LL NEVER BE ABLE TO FINISH MY PROJECT ON TIME.
YOU NEED TO TAKE OWNER-SHIP.
CAN I HIRE MORE PROGRAMMERS?
NO.
CAN I REDUCE THE NUMBER OF FEATURES?
NO.
SO... I'M JUST TAKING OWNERSHIP OF THE FAILURE?
DON'T BE GREEDY.
www.dilbert.com scottadams@aol.com
5-25-09 © 2009 Scott Adams, Inc./Dist. by UFS, Inc.

Figure 2.8 Dilbert tries to negotiate a change in his project. DILBERT © 2009 Scott Adams. Used By permission of UNIVERSAL UCLICK. All rights reserved.

Fourteen principles to follow to develop a culture that fosters quality

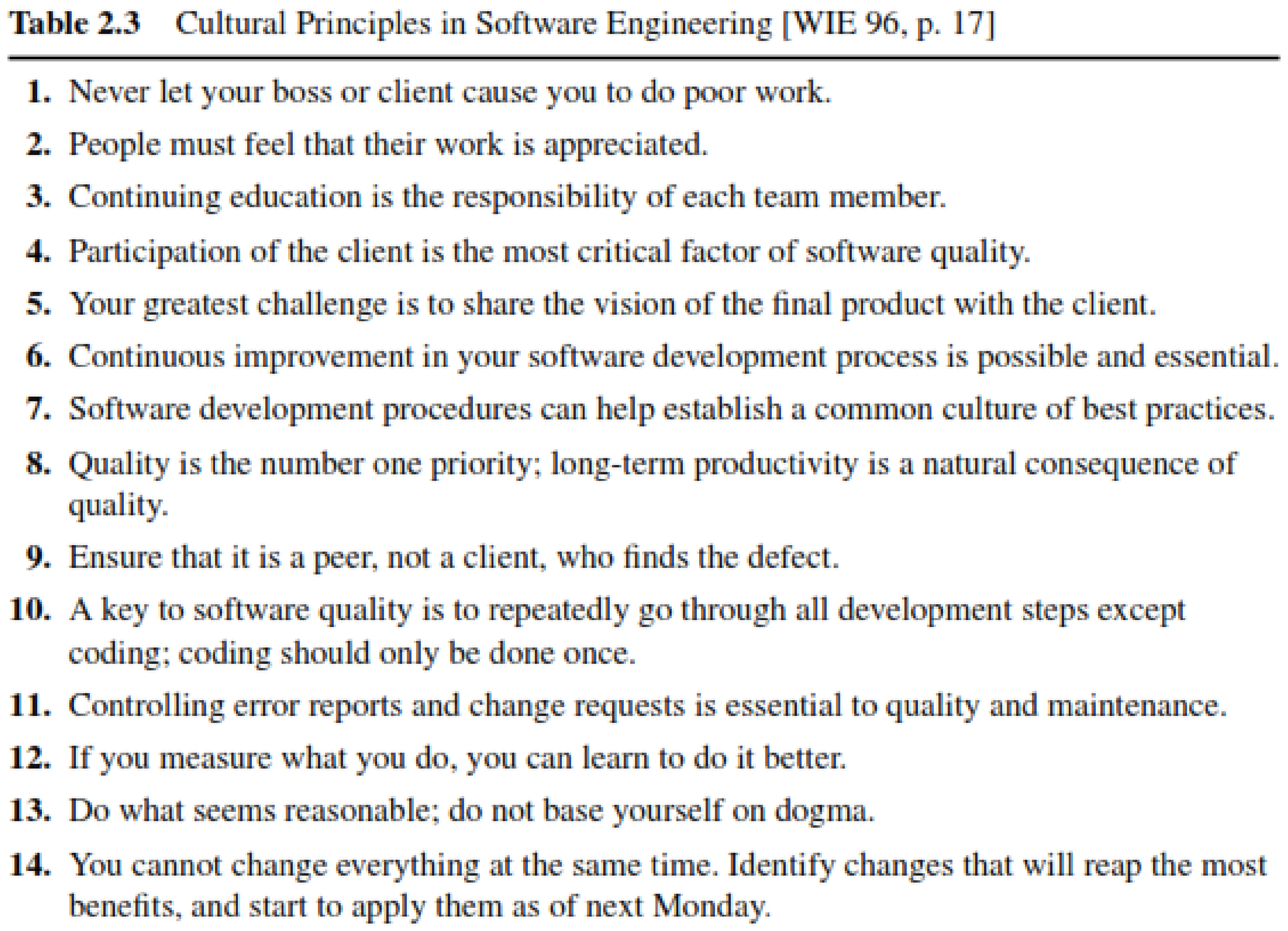


Table 2.3 Cultural Principles in Software Engineering [WIE 96, p. 17]

1. Never let your boss or client cause you to do poor work.
2. People must feel that their work is appreciated.
3. Continuing education is the responsibility of each team member.
4. Participation of the client is the most critical factor of software quality.
5. Your greatest challenge is to share the vision of the final product with the client.
6. Continuous improvement in your software development process is possible and essential.
7. Software development procedures can help establish a common culture of best practices.
8. Quality is the number one priority; long-term productivity is a natural consequence of quality.
9. Ensure that it is a peer, not a client, who finds the defect.
10. A key to software quality is to repeatedly go through all development steps except coding; coding should only be done once.
11. Controlling error reports and change requests is essential to quality and maintenance.
12. If you measure what you do, you can learn to do it better.
13. Do what seems reasonable; do not base yourself on dogma.
14. You cannot change everything at the same time. Identify changes that will reap the most benefits, and start to apply them as of next Monday.

THE SOFTWARE ENGINEERING CODE OF ETHICS

The first draft of the software engineering code of ethics was developed in cooperation with the Institute of Electrical and Electronics Engineers (IEEE) Computer Society and the Association for Computing Machinery (ACM)

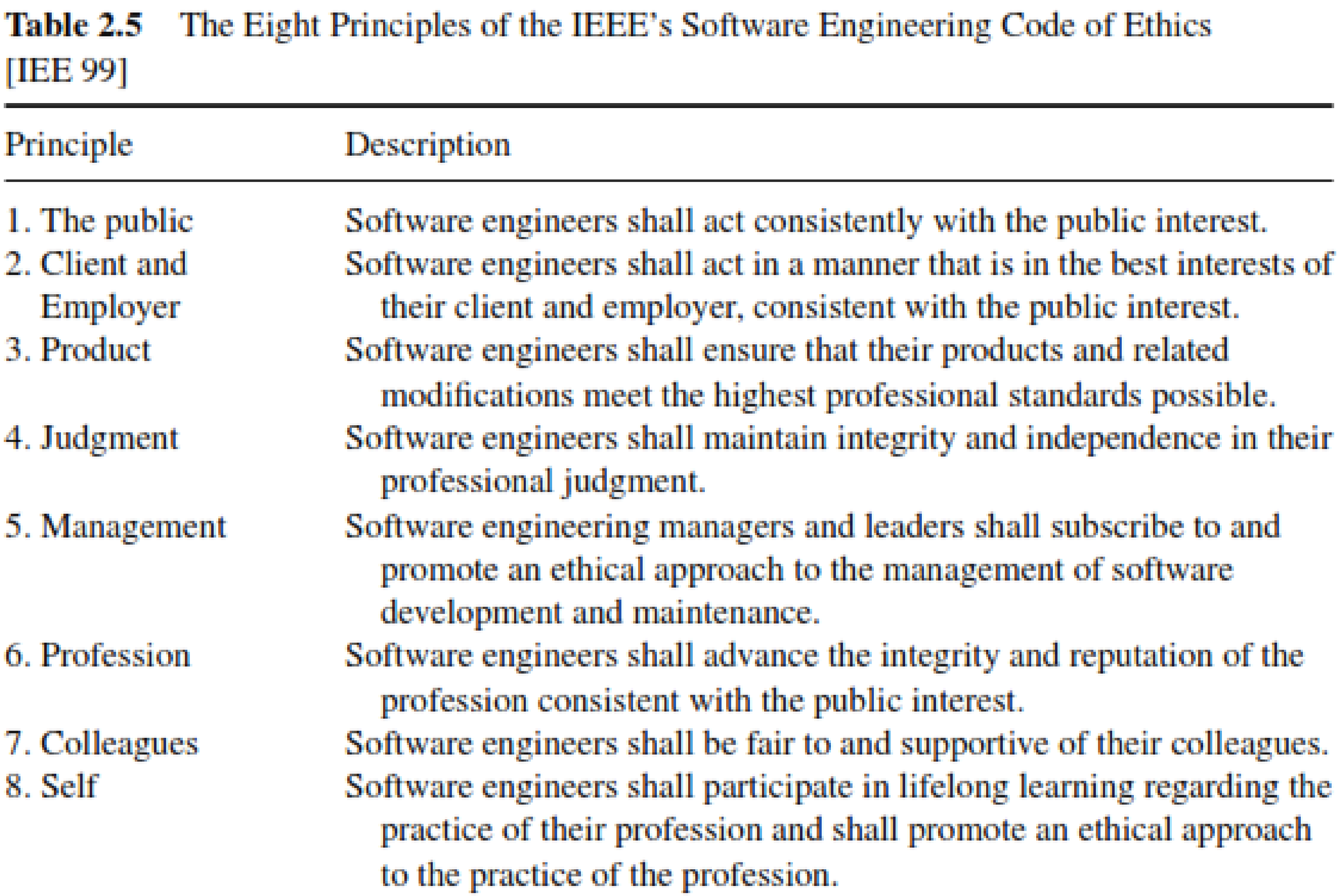


Table 2.5 The Eight Principles of the IEEE's Software Engineering Code of Ethics [IEE 99]

Principle	Description
1. The public	Software engineers shall act consistently with the public interest.
2. Client and Employer	Software engineers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest.
3. Product	Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. Judgment	Software engineers shall maintain integrity and independence in their professional judgment.
5. Management	Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. Profession	Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. Colleagues	Software engineers shall be fair to and supportive of their colleagues.
8. Self	Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Role of SQA in software development life cycle

The Software Development Life Cycle is split into six main phases.

1. Planning
2. Design
3. Implementation
4. Testing
5. Deployment
6. Maintenance

In the planning phase of a software project, Quality assurance is responsible for making sure that the **project meets all quality requirements**, such as scope, budget, timeline, and compliance with standards.

QA can also review user requirements and analyze them to determine if they fit within the scope of the project. This helps ensure that expectations are properly set at the beginning of a project, and that resources are allocated appropriately.

QA is involved in the design phase, they can identify aspects of the design that might cause problems ,while they're still in-progress. This enables the designer or wireframes creator to make changes on the fly.

The role QA plays in the implementation stage:

- Code Reviews
- System Integration Testing
- User Acceptance Testing

The Role of QA in Testing

QA focuses on various aspects, such as *functionality, usability, reliability, performance, and compliance with industry standards*. In order to ensure that the requirements of the application are met before it is released to its users.

The Role of QA in Deployment

In the deployment stage, QA ensures that *all elements of the custom software development process are properly implemented, tested, verified, and deployed*. This ensures the product is released with confidence, free from issues or bugs.

The Role of QA in Maintenance

- Verifying software updates to confirm they are working properly
- Testing changes to make sure they are as expected
- Identifying potential problems with updates
- Following up with customers to ensure they are satisfied with changes and updates
- Documenting all issues related to releases, so that future versions can be improved upon accordingly.
- By investing in QA during maintenance, companies can be sure their products remain reliable and bug-free for customers for the long haul!

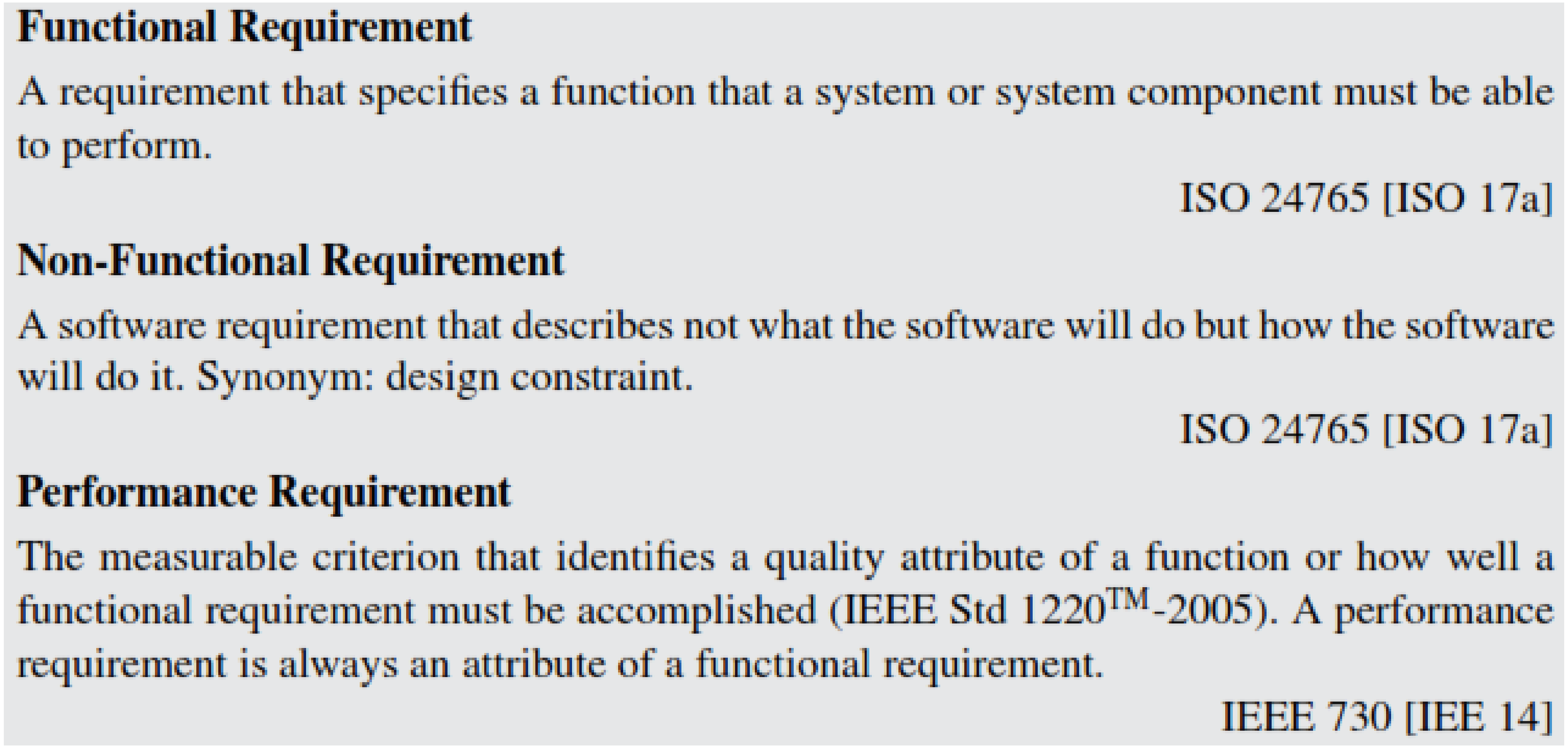
Standardizing SQA: Quality Models and Management

- Concepts conveyed by software quality models.
- Characteristics and sub-characteristics of software quality
- Software quality requirements of a software product
- Software traceability
- Standards for Quality Management
- Frameworks (ITIL, ISO, CMMI)

Requirements

The needs or requirements of these systems are typically documented either in a request for quote (RFQ) or request for proposal (RFP) document, a statement of work (SOW), a software requirements specification (SRS) document or in a system requirements document (SRD).

- Using these documents, the software developer must extract the information needed to define specifications for both the functional requirements and performance or non-functional requirements required by the client.



Functional Requirement

A requirement that specifies a function that a system or system component must be able to perform.

ISO 24765 [ISO 17a]

Non-Functional Requirement

A software requirement that describes not what the software will do but how the software will do it. Synonym: design constraint.

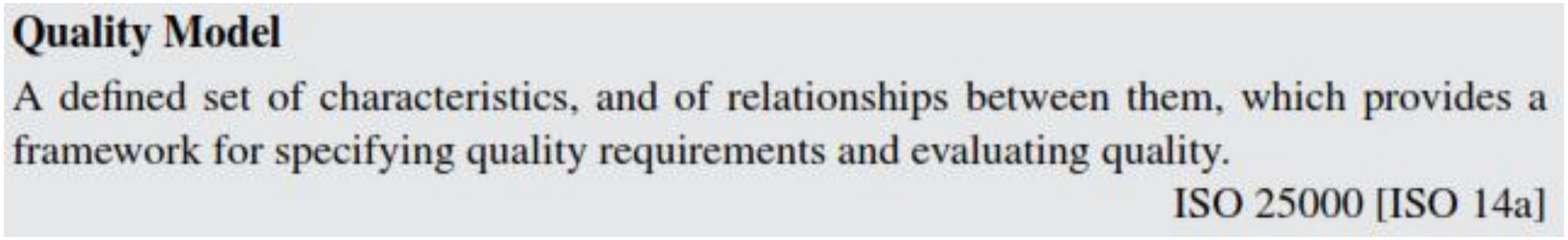
ISO 24765 [ISO 17a]

Performance Requirement

The measurable criterion that identifies a quality attribute of a function or how well a functional requirement must be accomplished (IEEE Std 1220^{TM}-2005). A performance requirement is always an attribute of a functional requirement.

IEEE 730 [IEE 14]

- Software quality assurance (SQA) must be able to support the practical application of these definitions.
- To achieve this, many concepts proposed by software quality models must be mastered.



Quality Model

A defined set of characteristics, and of relationships between them, which provides a framework for specifying quality requirements and evaluating quality.

ISO 25000 [ISO 14a]

Using software quality model client can:

- Define software quality characteristics that can be evaluated.
- Contrast the different perspectives of the quality model that come into play (i.e., internal - process to develop and external-fulfilling requirements perspectives).
- Carefully choose a limited number of quality characteristics that will serve as the non-functional requirements for the software (i.e., quality requirements);
- Set a measure and its objectives for each of the quality requirements.

Evaluation A systematic examination of the extent to which an entity is capable of fulfilling specified requirements.

SOFTWARE QUALITY MODELS

Unfortunately, in software organizations, software quality models are still rarely used.

A number of stakeholders have put forth the hypothesis that these models do not clearly identify all concerns for all of the stakeholders involved and are difficult to use.

Still Formally defining and evaluating the quality of software before it is delivered to the client in a need.

Five quality perspectives described by Garvin

Transcendental approach to quality:

- "Although I can't define quality, I know it when I see it."
- The main problem with this view is that quality is a personal and individual experience.
- it only takes time for all users to see it.

User-based approach: A second approach is that quality software performs as expected from the user's perspective (i.e., fitness for purpose).

Manufacturing-based approach: quality is defined as complying with specifications, is illustrated by many documents on the quality of the development process.

Product-based approach: The product-based quality perspective involves an internal view of the product. The software engineer focuses on the internal properties of the software components, for example, the quality of its architecture.

These internal properties correspond to source code characteristics and require advanced testing techniques.

Value-based approach: focuses on the *elimination of all activities that do not add value*, for example the drafting of certain documents.

In the software domain, the concept of “value” is synonymous with productivity, increased profitability, and competitiveness.

Initial Model Proposed by McCall

It proposes three perspectives for the user and primarily promotes a product-based view of the software product.

- **Operation**: during its use;
- **Revision**: during changes made to it over the years;
- **Transition**: for its conversion to other environments when the time comes to migrate to a new technology.

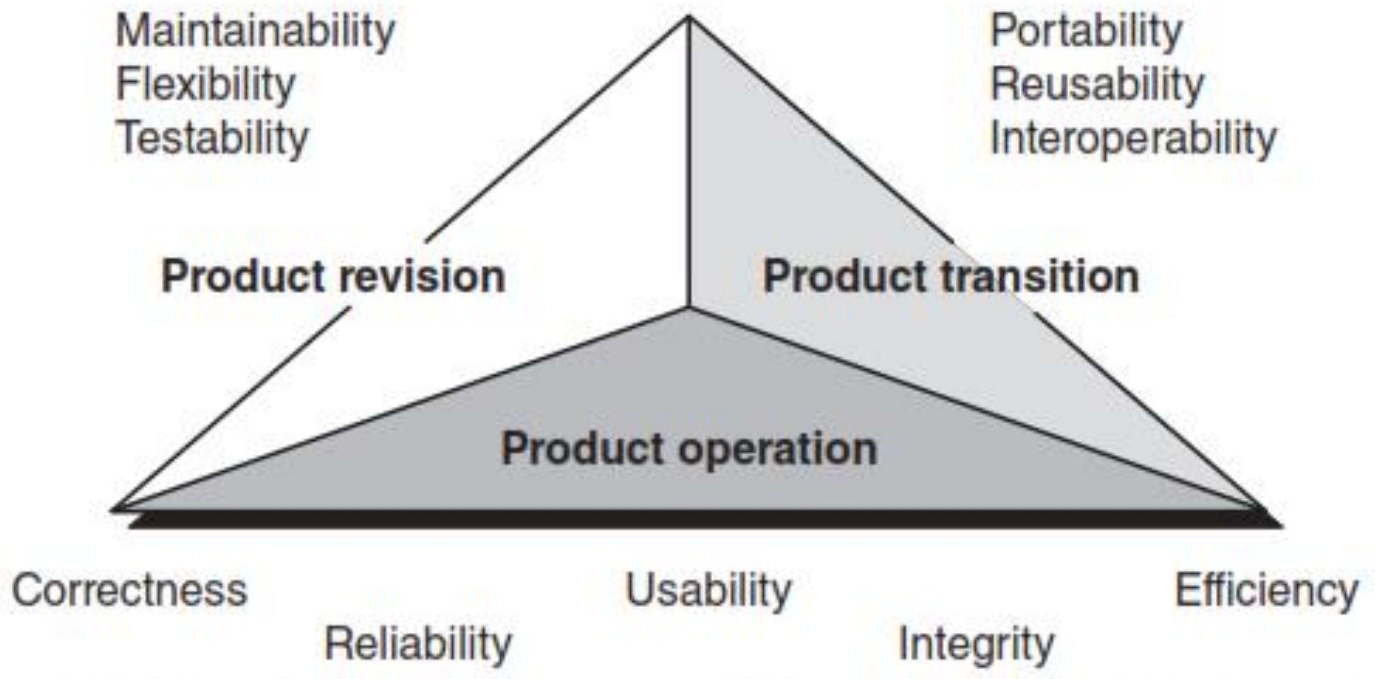


Figure 3.1 The three perspectives and 11 quality factors of McCall et al. (1977) [MCC 77].

- Each perspective is broken down into a number of quality factors.
- The model proposed by McCall and his colleagues lists 11 quality factors.
- Each quality factor can be broken down into several quality criteria (see Figure 3.2).

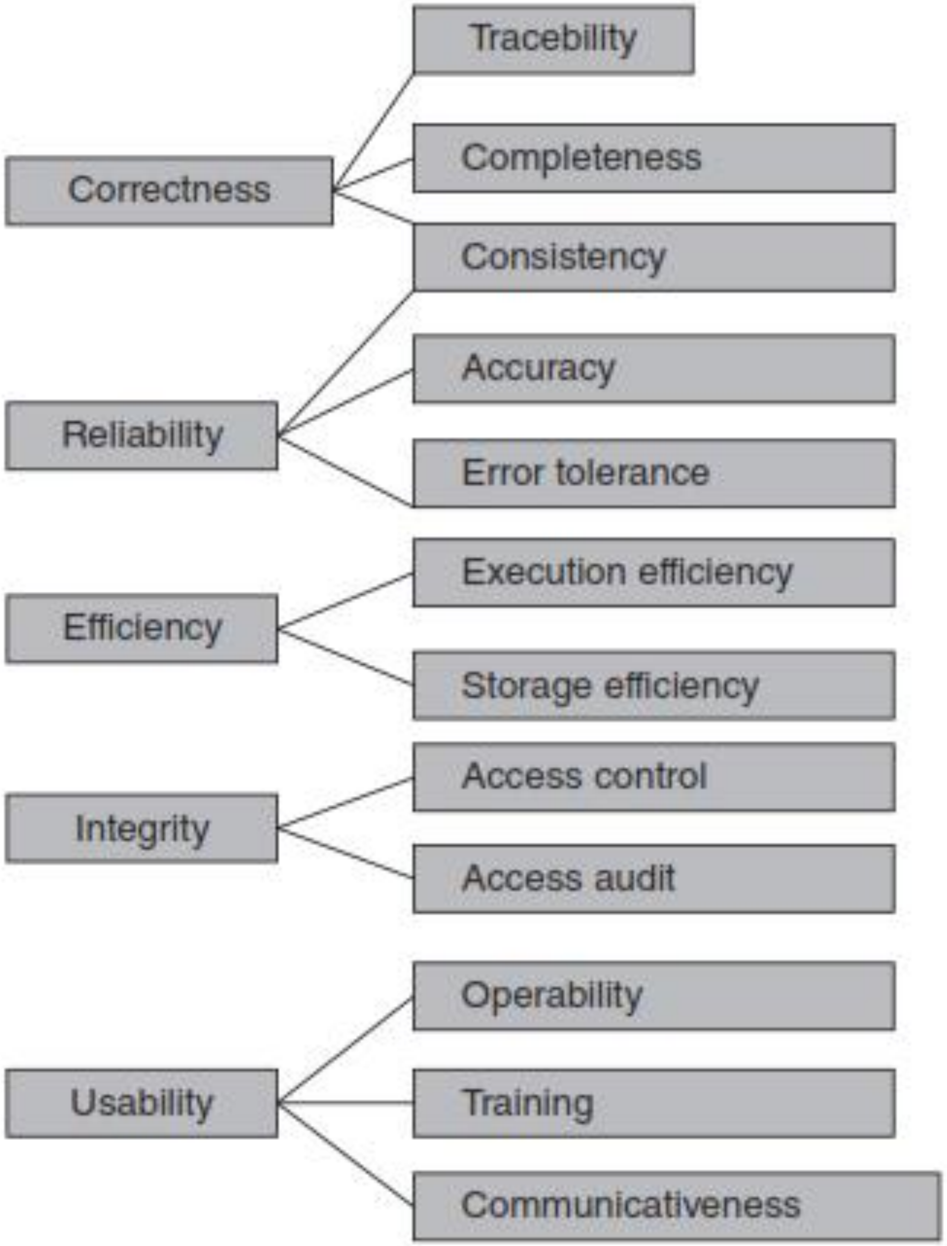


Figure 3.2 Quality factors and criteria from McCall et al. (1977) [MCC 77].

The right side of Figure 3.2 presents the measurable properties (called “quality criteria”), which can be evaluated (through observation of the software) to assess quality.

McCall proposes a subjective evaluation scale of 0 (minimum quality) to 10 (maximum quality).

The McCall quality model was primarily aimed at software product quality (i.e., the internal perspective) and did not easily tie in with the perspective of the user who is not concerned with technical details.

Example: A car owner who is not concerned with the metals or alloys used to make the engine. He expects the car to be well designed so as to minimize frequent and expensive maintenance costs.

The First Standardized Model: IEEE 1061

The IEEE 1061 standard, that is, the *Standard for a Software Quality Metrics Methodology* [IEE 98b], provides a framework for measuring software quality that allows for the establishment and identification of software quality measures based on quality requirements in order to implement, analyze, and validate software processes and products.

This standard claims to adapt to all business models, types of software, and all of the stages of the software life cycle.

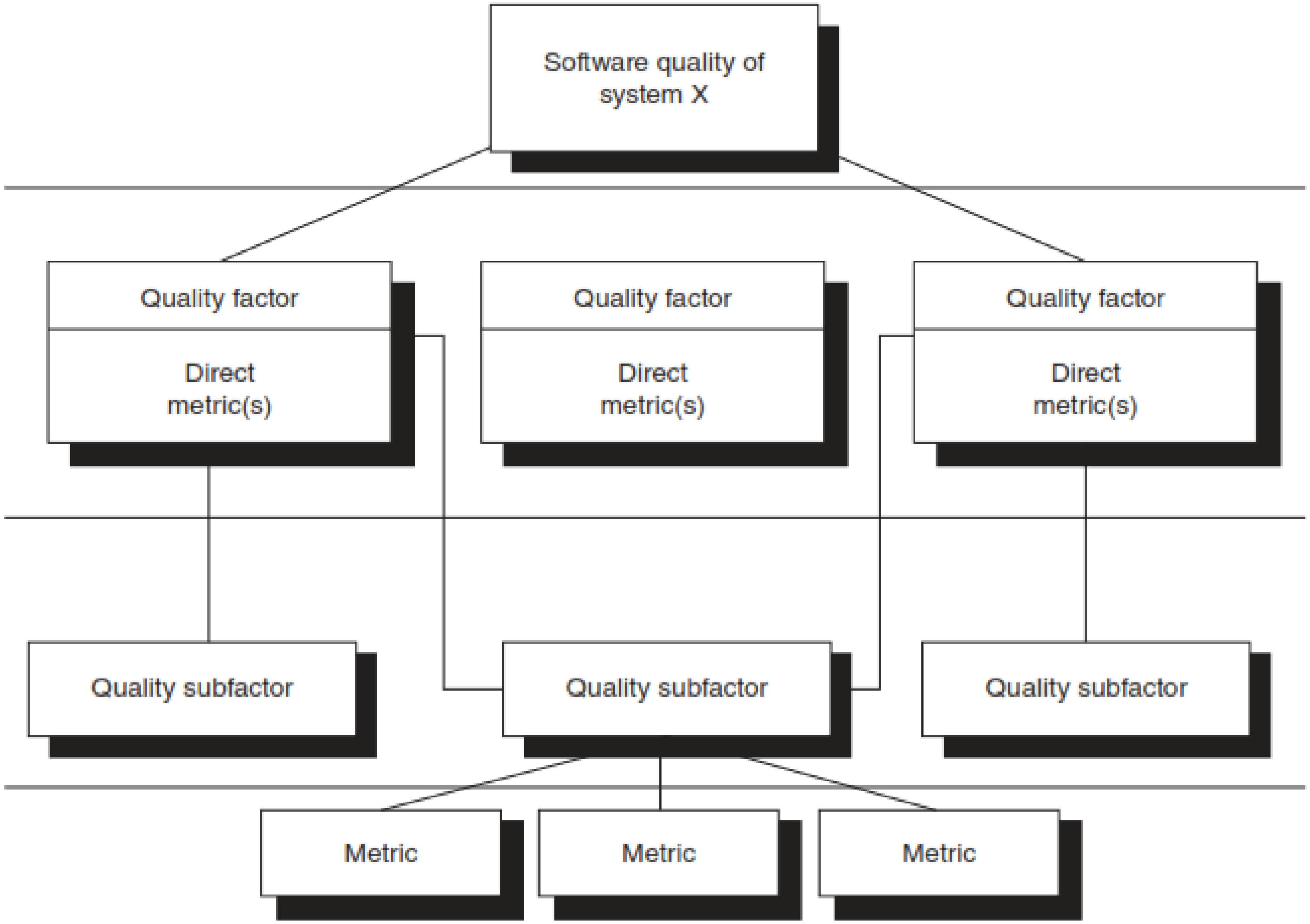
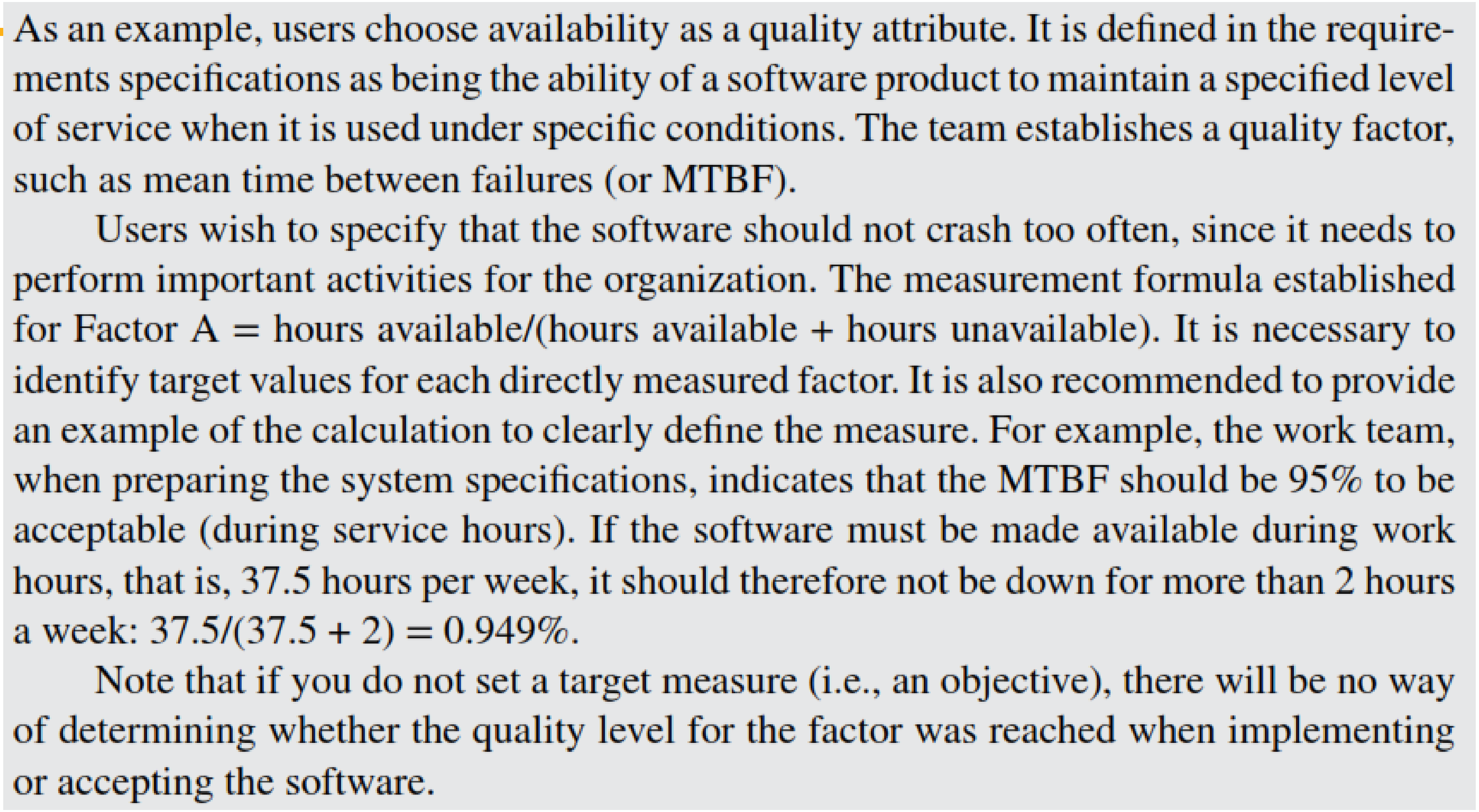


Figure 3.3 Framework for measuring software quality as per the IEEE 1061 [IEE 98b].

- At the top tier, we can see that software quality requires prior specification of a **certain number of quality attributes**, which serve to describe **the final quality desired in the software.**
- The **attributes desired by clients and users** allow for the definition of the software quality requirements.
- Quality factors suggested by this standard are assigned attributes at the next tier.

At the tier below that, and only if necessary, subfactors can then be assigned to each quality factor.

Lastly, measures are associated with each quality factor, allowing for a quantitative evaluation of the quality factor (or subfactor).


As an example, users choose availability as a quality attribute. It is defined in the requirements specifications as being the ability of a software product to maintain a specified level of service when it is used under specific conditions. The team establishes a quality factor, such as mean time between failures (or MTBF).

Users wish to specify that the software should not crash too often, since it needs to perform important activities for the organization. The measurement formula established for Factor A = hours available/(hours available + hours unavailable). It is necessary to identify target values for each directly measured factor. It is also recommended to provide an example of the calculation to clearly define the measure. For example, the work team, when preparing the system specifications, indicates that the MTBF should be 95% to be acceptable (during service hours). If the software must be made available during work hours, that is, 37.5 hours per week, it should therefore not be down for more than 2 hours a week: 37.5/(37.5 + 2) = 0.949%.

Note that if you do not set a target measure (i.e., an objective), there will be no way of determining whether the quality level for the factor was reached when implementing or accepting the software.

This model provides defined steps to use quality measures in the following situations:

Software program acquisition: In order to establish a contractual commitment regarding quality objectives for client-users and verify whether they were met by allowing their measurement when adapting and releasing the software.

Software development: In order to clarify and document quality characteristics on which designers and developers must work in order to respect the customer's quality requirements.

Quality assurance/quality control/audit: In order to enable those outside the development team to evaluate the software quality.

Maintenance: Allow the maintainer to understand the level of quality and service to maintain when making changes or upgrades to the software.

Client/user: Allow users to state quality characteristics and evaluate their presence during acceptance tests (i.e., If the software does not meet the agreed-upon specifications, corrective actions should be taken by the developer)

Steps proposed under the IEEE 1061 [IEE 98b] standard:

- Start By Identifying The List Of Non-functional (Quality) Requirements
- Everybody Involved
- List And Make Sure To Resolve Any Conflicting
- Quantify Each Quality Factor.
- Have Measures And Thresholds Approved.
- Perform A Cost–benefit Study To Identify The Costs Of Implementing The Measures For The Project.
- Implement The Measurement Method
- Analyze The Results
- Validate The Measures

Current Standardized Model: ISO 25000 Set of Standards

The Treasury Board concluded that there were basically two ways to determine the quality of a software product:

(1) assess the quality of the development process,

(2) assess the quality of the final product.

The ISO 25000 [ISO 14a] standard allows for the evaluation of the quality of the final software product.

The ISO 25000's series of standards recommends the following four steps [ISO 14a]:

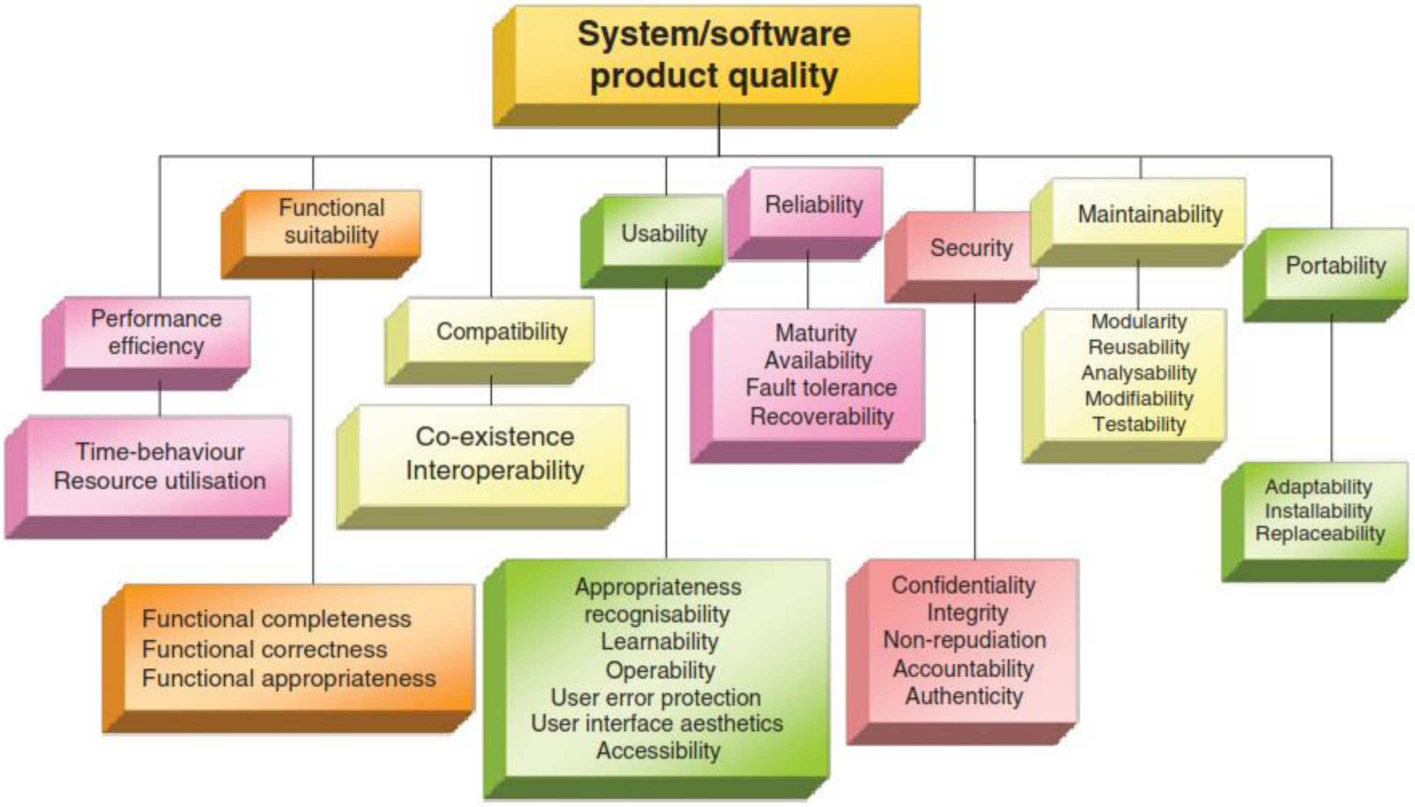
– **Set quality requirements;**

– **Establish a quality model;**

– **Define quality measures;**

– **Conduct evaluations.**

The ISO 25010 standard identifies eight quality attributes for software



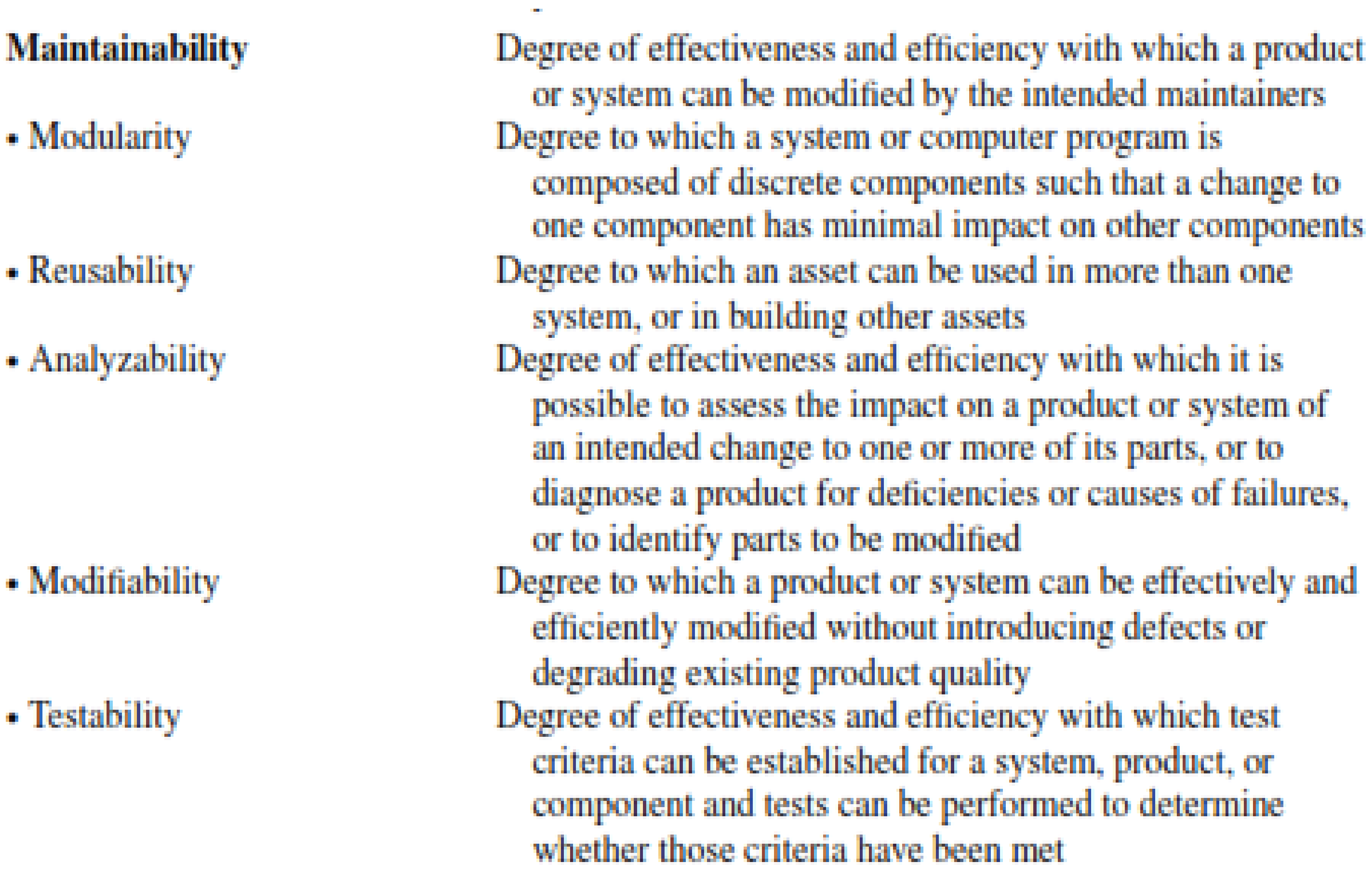
To illustrate how this standard is used, we will describe the characteristic of maintainability, which has **five sub-characteristics**: modularity, reusability, analyzability, modifiability, and testability.

Maintainability is defined as being the level of efficiency and efficacy with which software can be modified. Changes may include software corrections, improvements, or adaptation to changes in the environment, requirements, or functional specifications.

The internal and external points of view, of the maintainability of the software.

External point of view, maintainability attempts to measure the effort required to troubleshoot, analyze, and make changes to specific software.

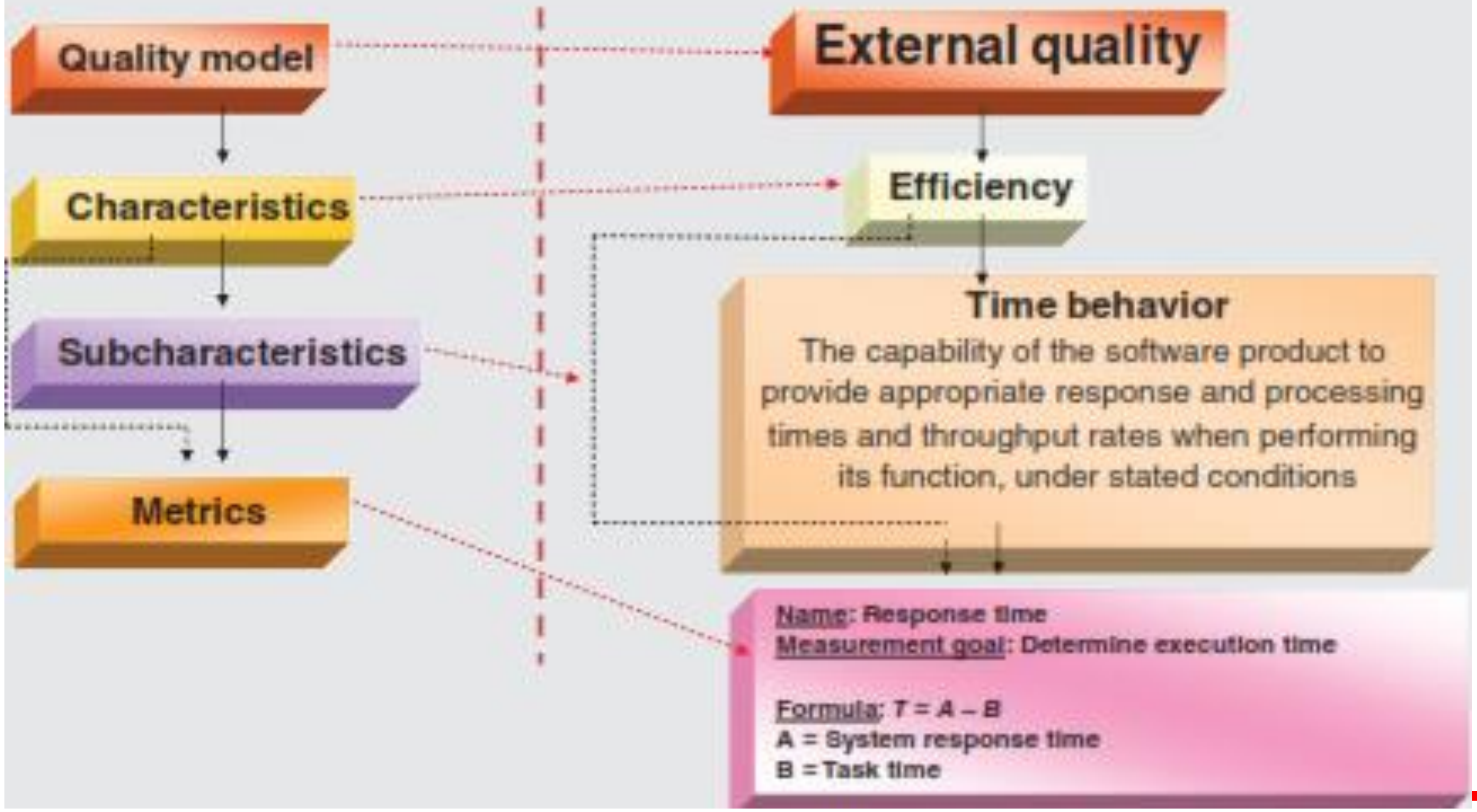
Internal point of view, maintainability usually involves measuring the attributes of the software that influence this change effort.



Maintainability	Degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers
• Modularity	Degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components
• Reusability	Degree to which an asset can be used in more than one system, or in building other assets
• Analyzability	Degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified
• Modifiability	Degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality
• Testability	Degree of effectiveness and efficiency with which test criteria can be established for a system, product, or component and tests can be performed to determine whether those criteria have been met

Evaluating Software Quality with the ISO 25010 Model

The quality model proposed by ISO 25010 is similar to the IEEE 1061 [IEE 98b] model. Choose a quality characteristic to evaluate—efficiency is used in the following example. Then choose one or more quality sub-characteristics to be evaluated (time behavior has been chosen in the following example). The last step is to clearly specify a measurement so that there is no possible misinterpretation of its result. A dotted line indicates that the sub-characteristics can be bypassed if necessary.


Quality model
Characteristics
Subcharacteristics
Metrics
External quality
Efficiency
Time behavior
The capability of the software product to provide appropriate response and processing times and throughput rates when performing its function, under stated conditions
Name: Response time
Measurement goal: Determine execution time
Formula: T = A – B
A = System response time
B = Task time

DEFINITION OF SOFTWARE QUALITY REQUIREMENTS

Process of defining quality requirements for software (i.e., a process that supports the use of a software quality model).

In the classical engineering approach, requirements are considered to be prerequisites to the design and development stages of a product.

The requirements development phase may have been preceded by a feasibility study, or a design analysis phase for the project.

Once the stakeholders have been identified, activities for software specifications can be broken down into:

– gather: collect all wishes, expectations, and needs of the stakeholders;

– prioritize: debate the relative importance of requirements based on, for example, two priorities (essential, desirable);

– analyze: check for consistency and completeness of requirements;

– describe: write the requirements in a way that can be easily understood by users and developers;

– specify: transform the business requirements into software specifications (data sources, values and timing, business rules).

Requirements are generally grouped into three categories:

1) Functional Requirements: These describe the characteristics of a system or processes that the system must execute. This category includes business requirements and functional requirements for the user.
2) Non-Functional (Quality) Requirements: These describe the properties that the system must have, for example, requirements translated into quality characteristics and sub-characteristics such as security, confidentiality, integrity, availability, performance, and accessibility.
3) Constraints: Limitations in development such as infrastructure on which the system must run or the programming language that must be used to implement the system.

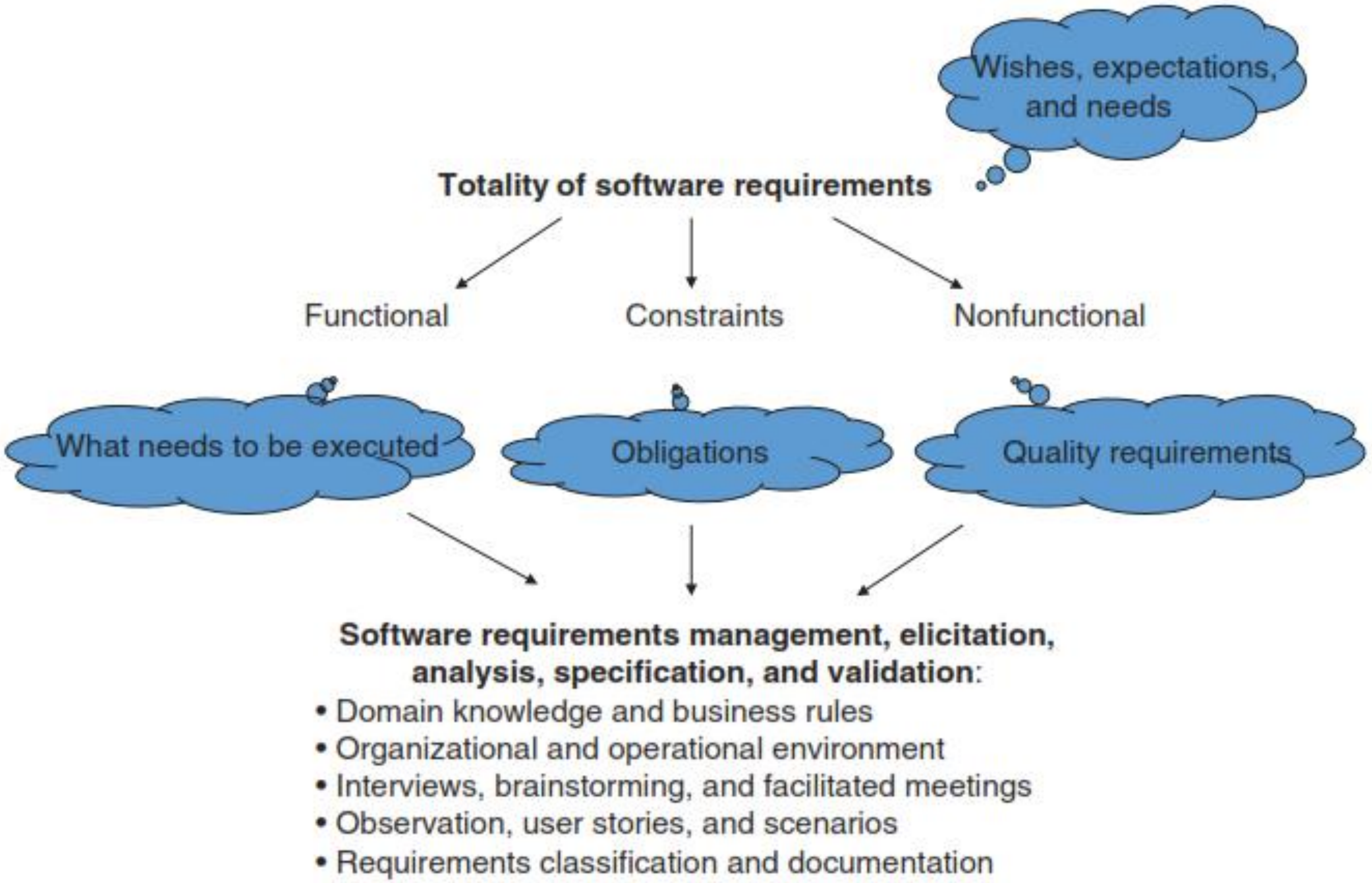

Wishes, expectations, and needs
Totality of software requirements
Functional
Constraints
Nonfunctional
What needs to be executed
Obligations
Quality requirements
Software requirements management, elicitation, analysis, specification, and validation:
• Domain knowledge and business rules
• Organizational and operational environment
• Interviews, brainstorming, and facilitated meetings
• Observation, user stories, and scenarios
• Requirements classification and documentation

Figure 3.5 Context of software requirements elicitation.

Characteristics to measure quality of a requirement

- Necessary: They must be based on necessary elements
- Unambiguous: clear enough to be interpreted in only one way.
- Concise: They must be stated in a language that is precise, brief, and easy to read.
- Coherent: They must not contradict the requirements.
- Complete: They must all be stated fully
- Accessible: They must be realistic regarding their implementation (time ,budget ,resource)
- Verifiable: inspection, analysis, demonstration, or tests.

THANK YOU