BITS Pilani presentation

BITS Pilani
Pilani Campus

Dr. Nagesh BS

innovate achieve lead

BITS Pilani
Pilani Campus

SE ZG501
Software Quality Assurance and Testing
Lecture No. 2

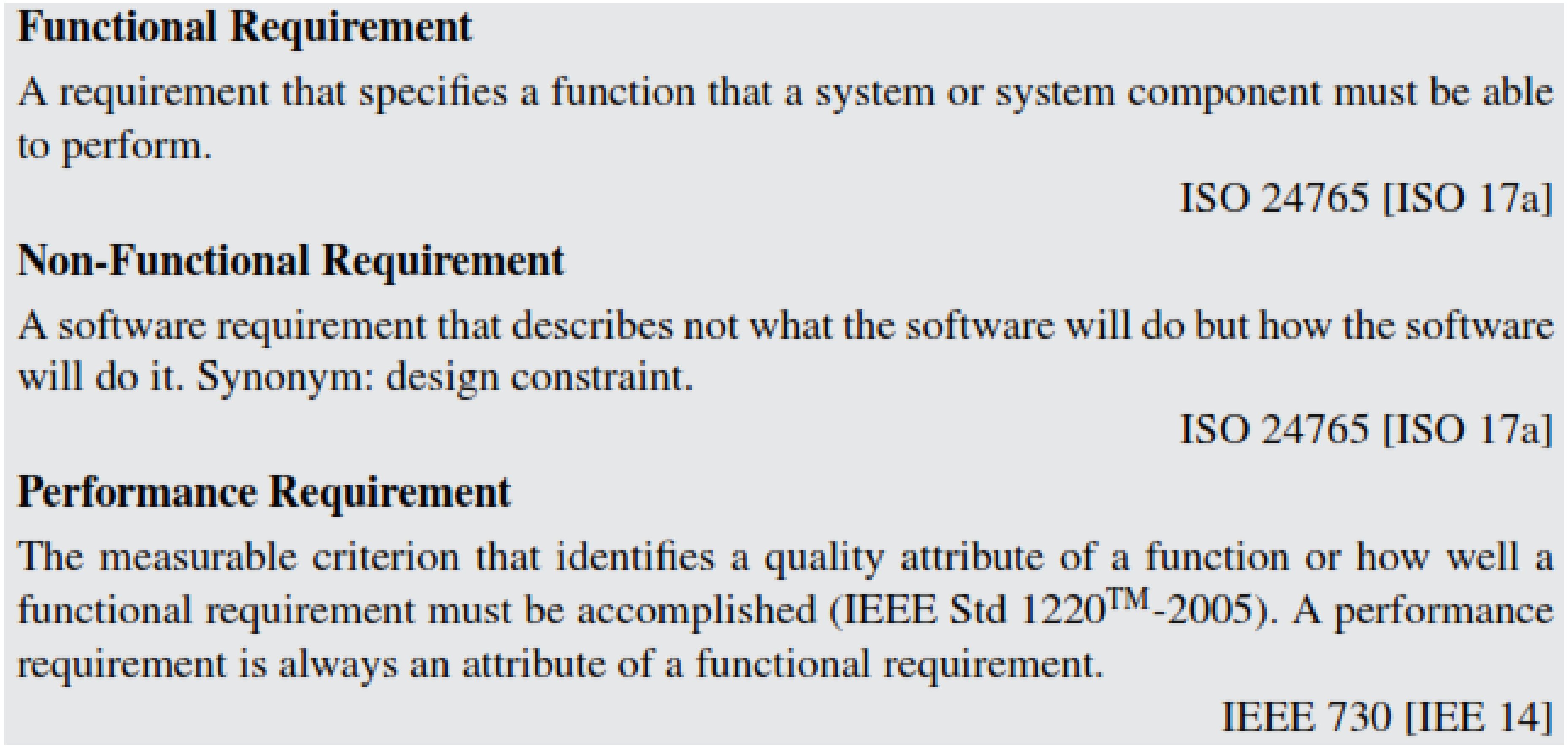
Standardizing SQA: Quality Models and Management

- 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.

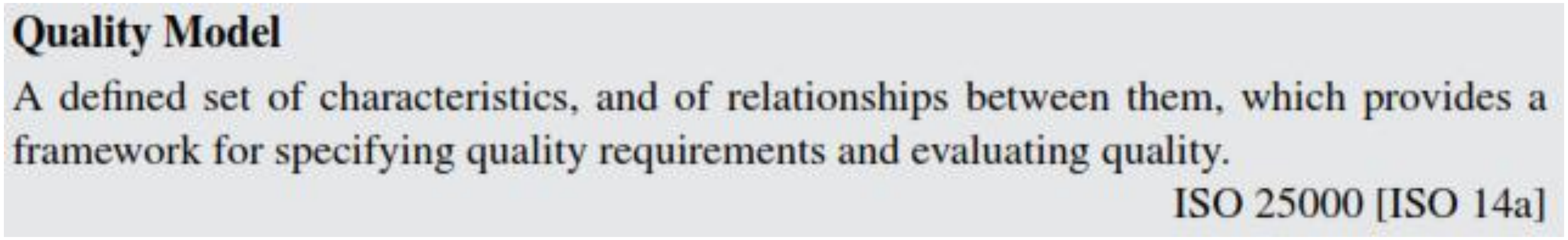
ISO 24765 [ISO 17a]

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.

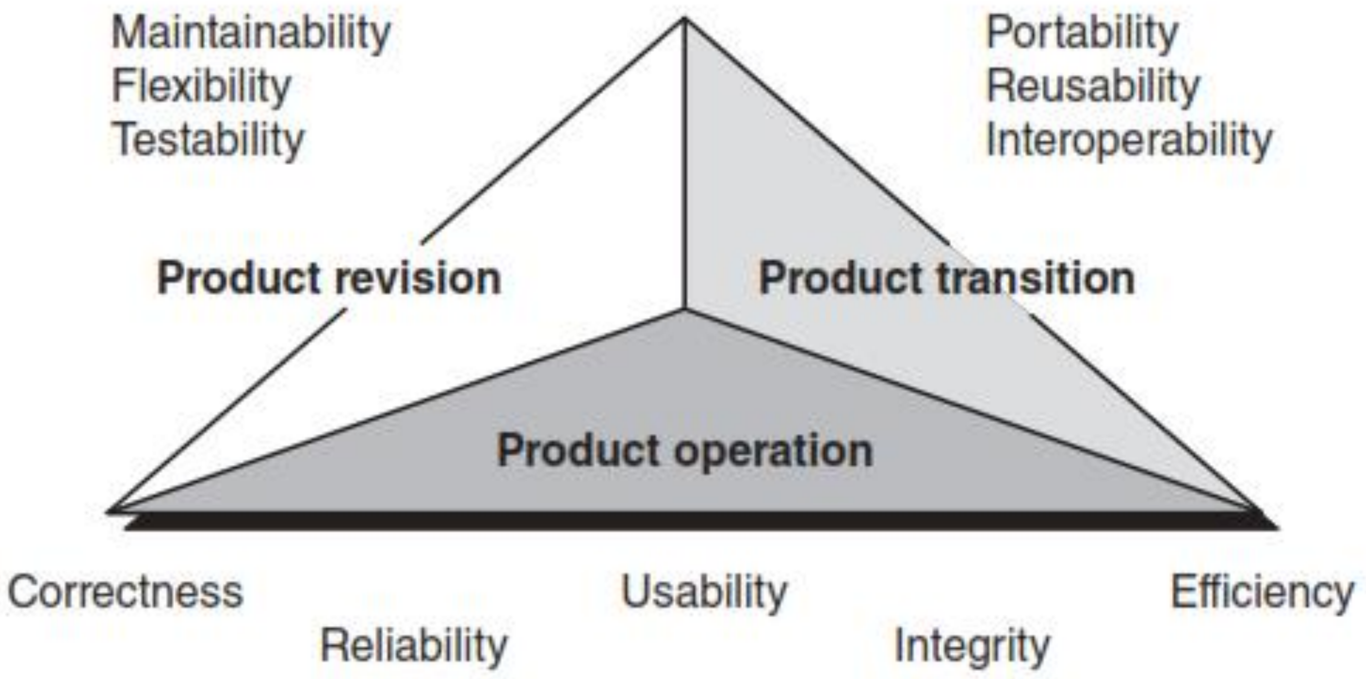


Figure 3.1 The three perspectives and 11 quality factors of McCall et al. (1977) [MCC 77].

- 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).

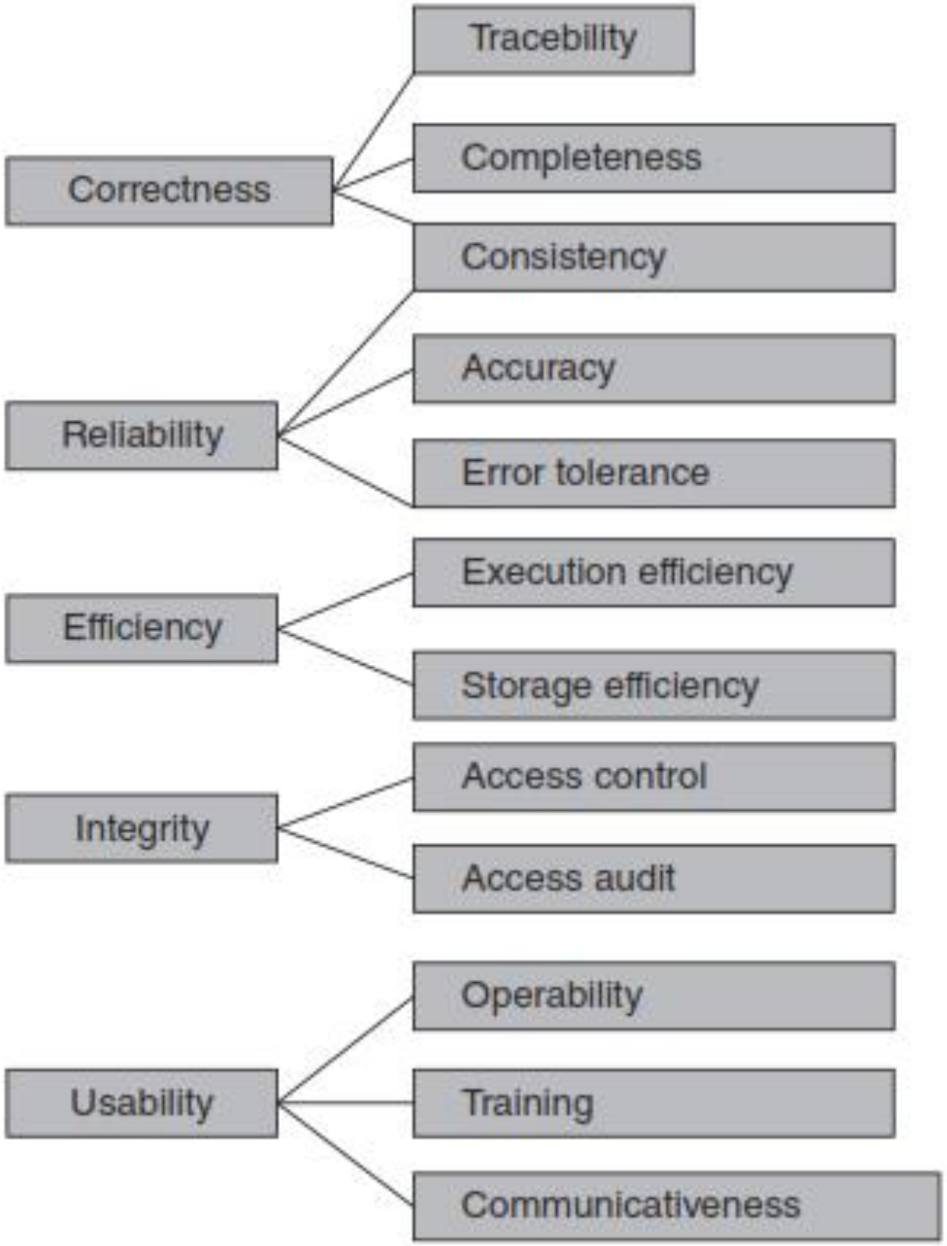


Figure 3.2 Quality factors and criteria from McCall et al. (1977) [MCC 77].

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.

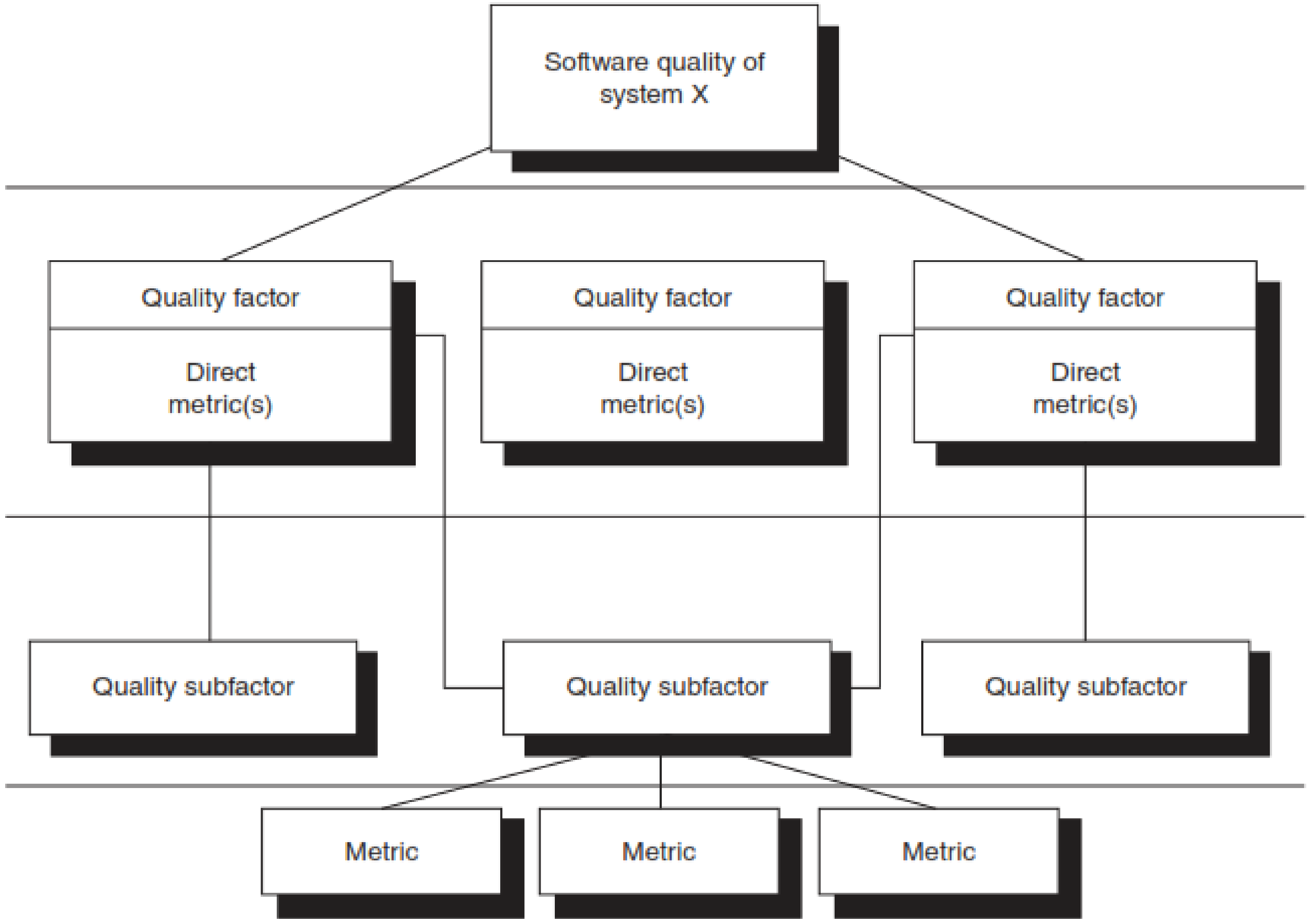


Figure 3.3 Framework for measuring software quality as per the IEEE 1061 [IEE 98b].

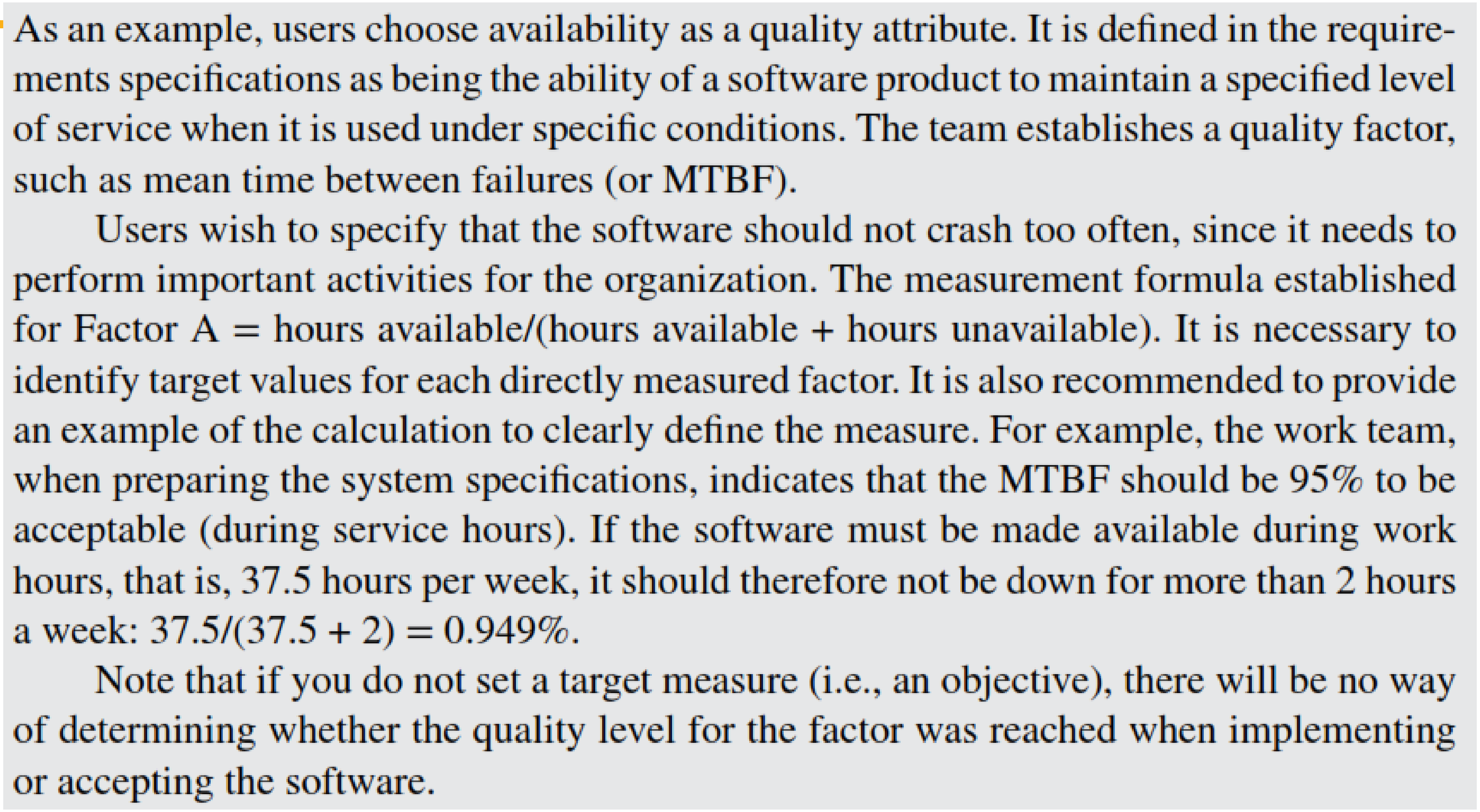
At the top tier, we can see that software quality requires prior specification of a certain number of quality attributes, which serve to describe the final quality desired in the software.

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.

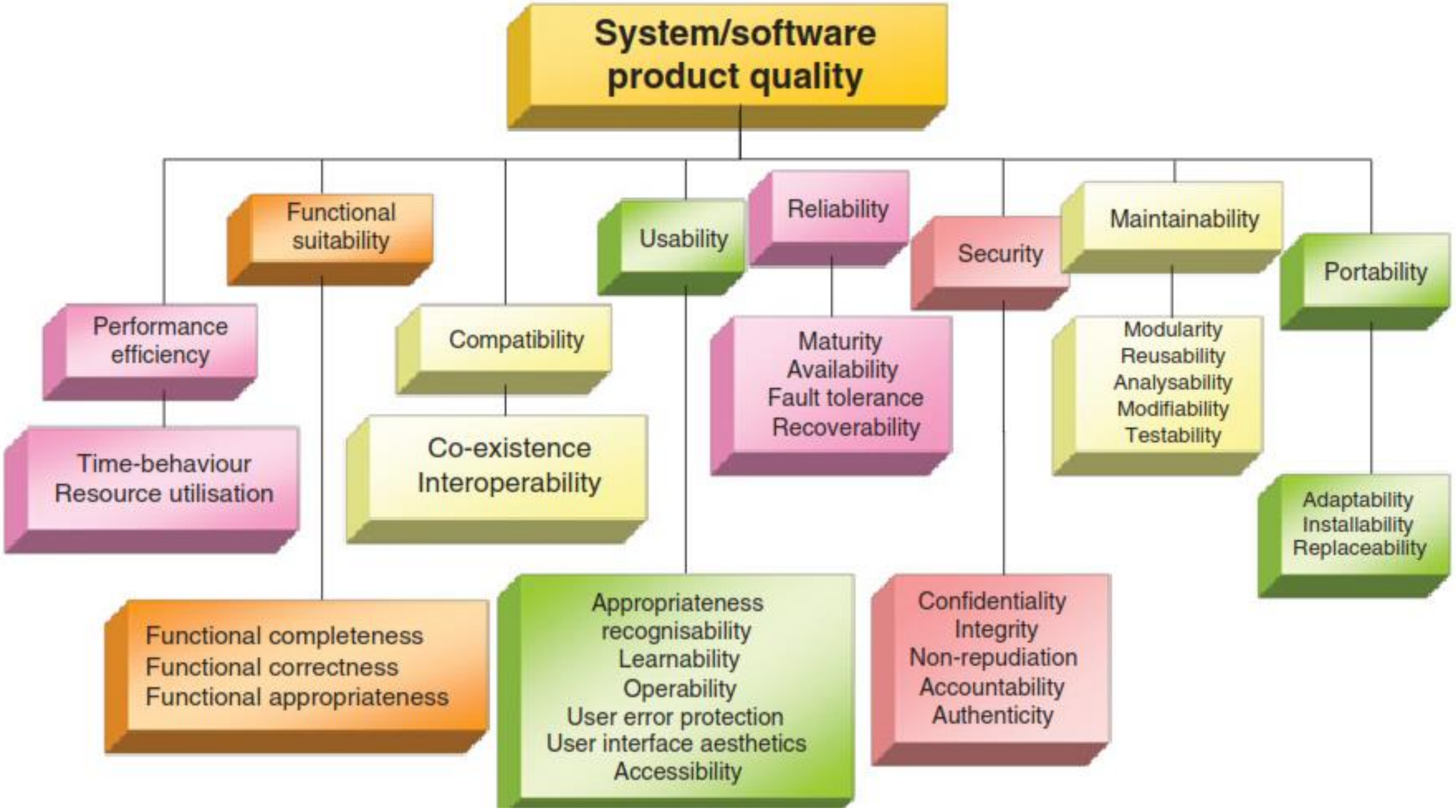
Quality assurance/quality control/audit: In order to enable those outside the development team to evaluate the software quality.

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 specifications agreed upon 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


System/software product quality
Functional suitability
Performance efficiency
Compatibility
Usability
Reliability
Security
Maintainability
Portability
Time-behaviour
Resource utilisation
Co-existence
Interoperability
Maturity
Availability
Fault tolerance
Recoverability
Modularity
Reusability
Analysability
Modifiability
Testability
Adaptability
Installability
Replaceability
Functional completeness
Functional correctness
Functional appropriateness
Appropriateness recognisability
Learnability
Operability
User error protection
User interface aesthetics
Accessibility
Confidentiality
Integrity
Non-repudiation
Accountability
Authenticity

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]:

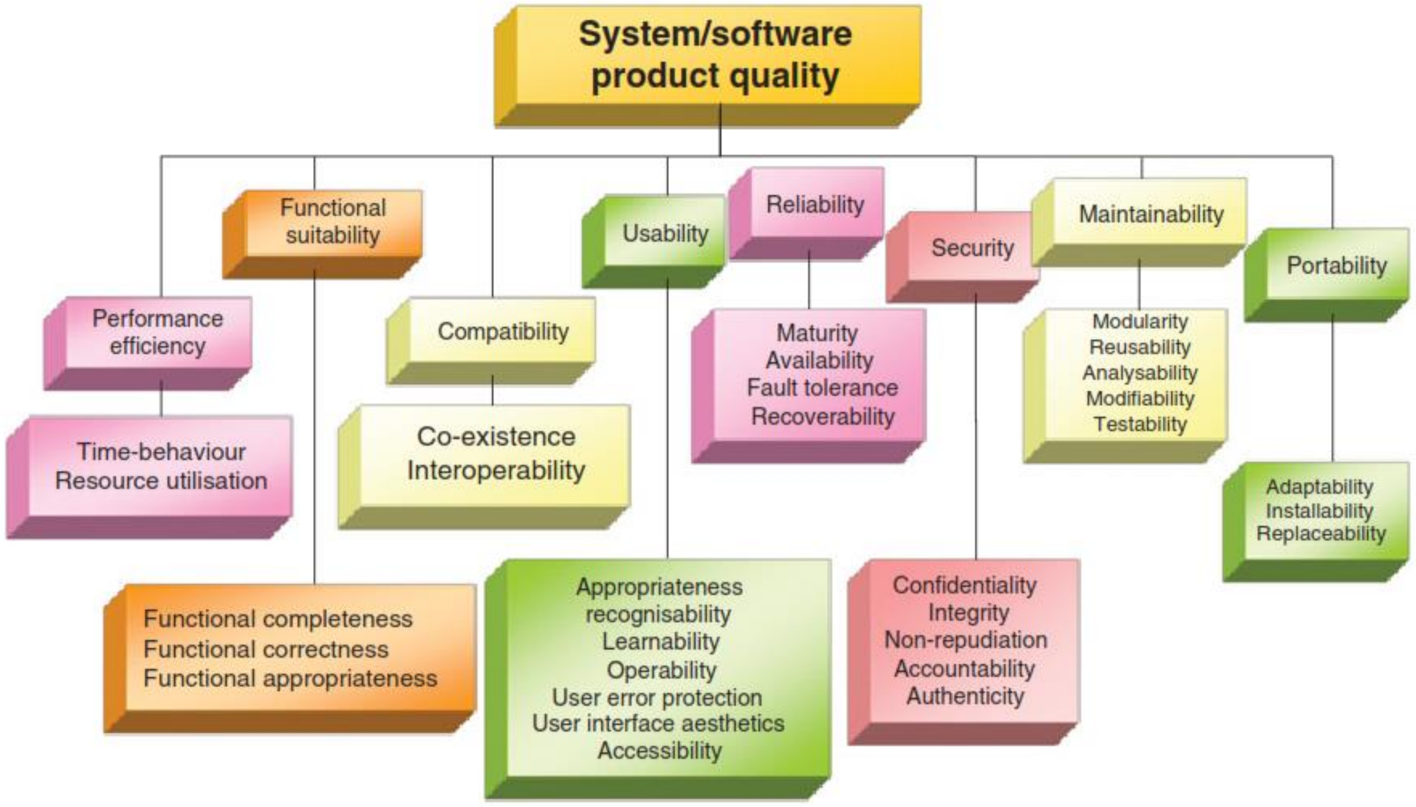
– **Set quality requirements;**

– **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.

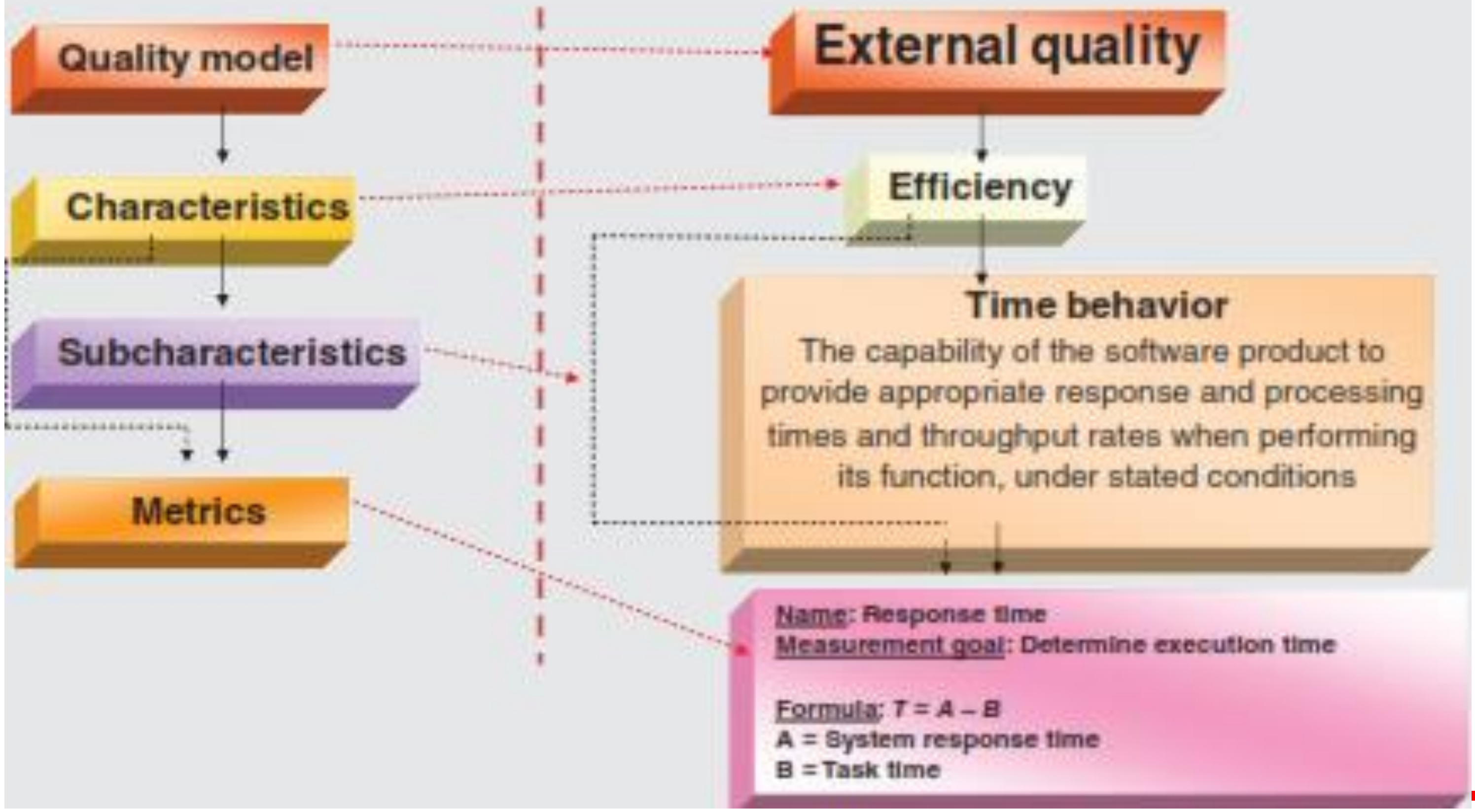
External point of view, maintainability attempts to measure the effort required to troubleshoot, analyze, and make changes to specific software.

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);

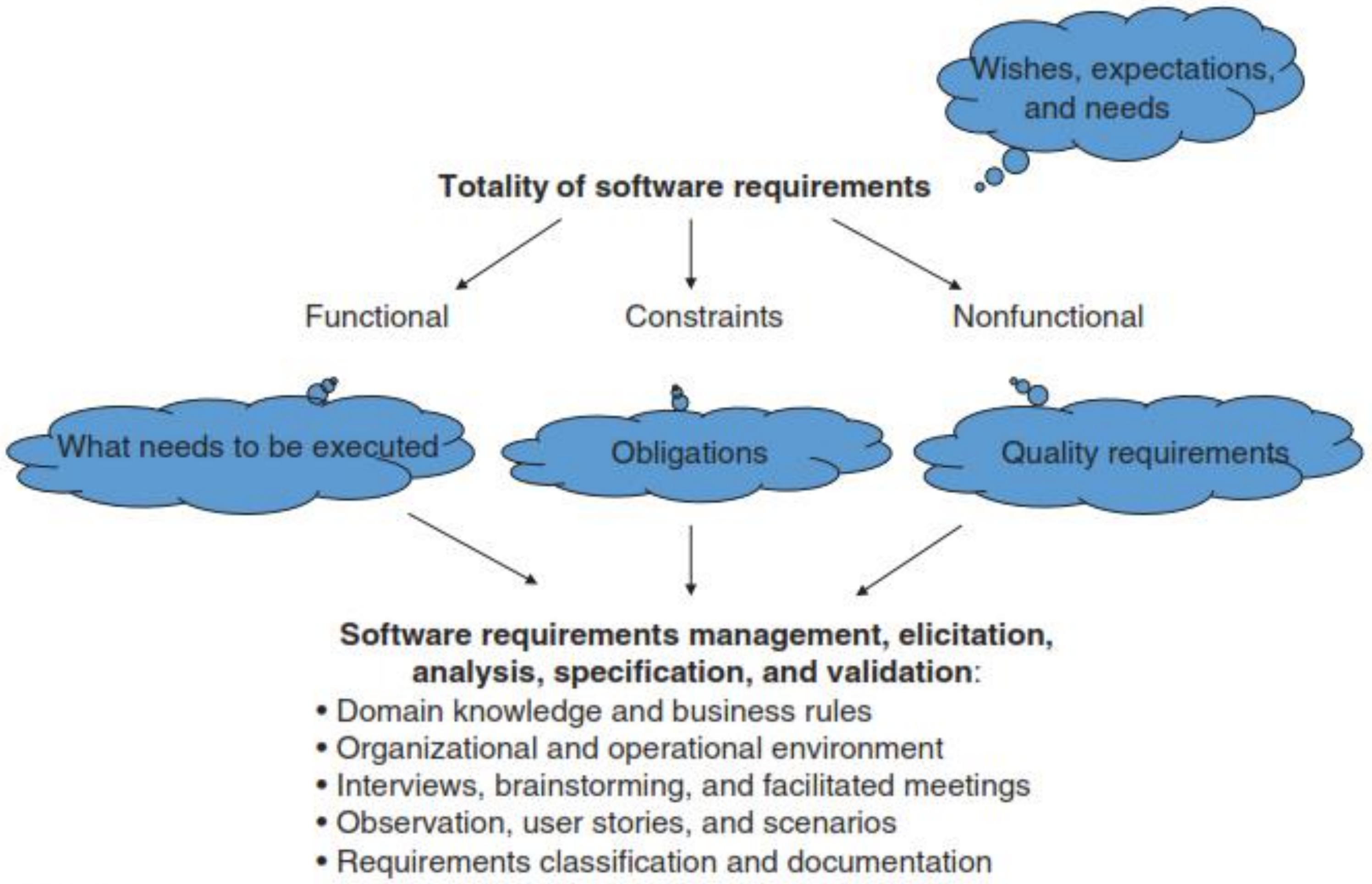
– analyze: check for consistency and completeness of requirements;

– 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.

Specifying Quality Requirements: The Process

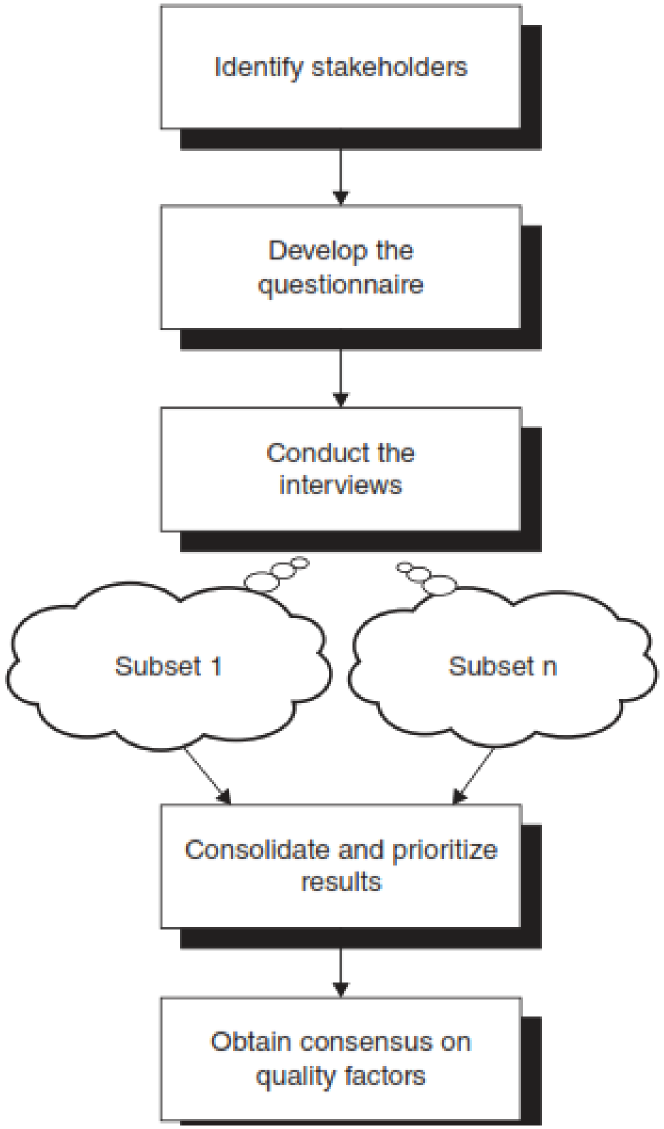

Identify stakeholders
Develop the questionnaire
Conduct the interviews
Subset 1
Subset n
Consolidate and prioritize results
Obtain consensus on quality factors

Figure 3.6 Steps suggested for defining

- Stakeholders are any person or organization that has a legitimate interest in the quality of the software.

  These needs and hopes may change during the system life cycle and must be checked when there is any change.

- Developing the questionnaire that presents the external quality characteristics in terms that are easy to understand for managers.

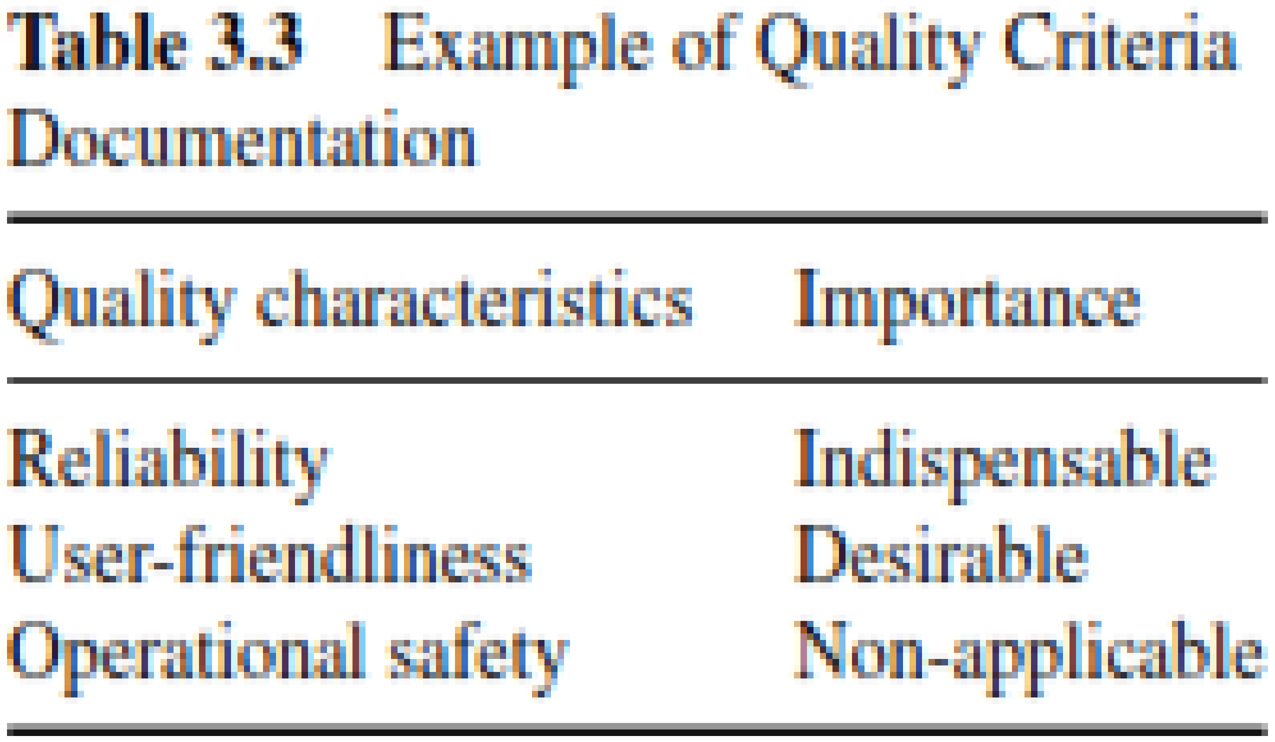
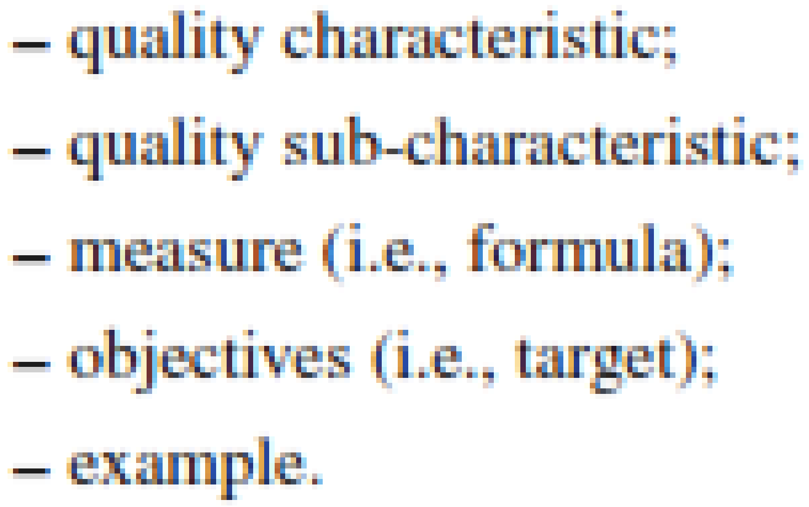


Table 3.3 Example of Quality Criteria Documentation

Quality characteristics	Importance
Reliability	Indispensable
User-friendliness	Desirable
Operational safety	Non-applicable

Next, for each characteristic, the quality measure must be described in detail and include the following information



– quality characteristic;

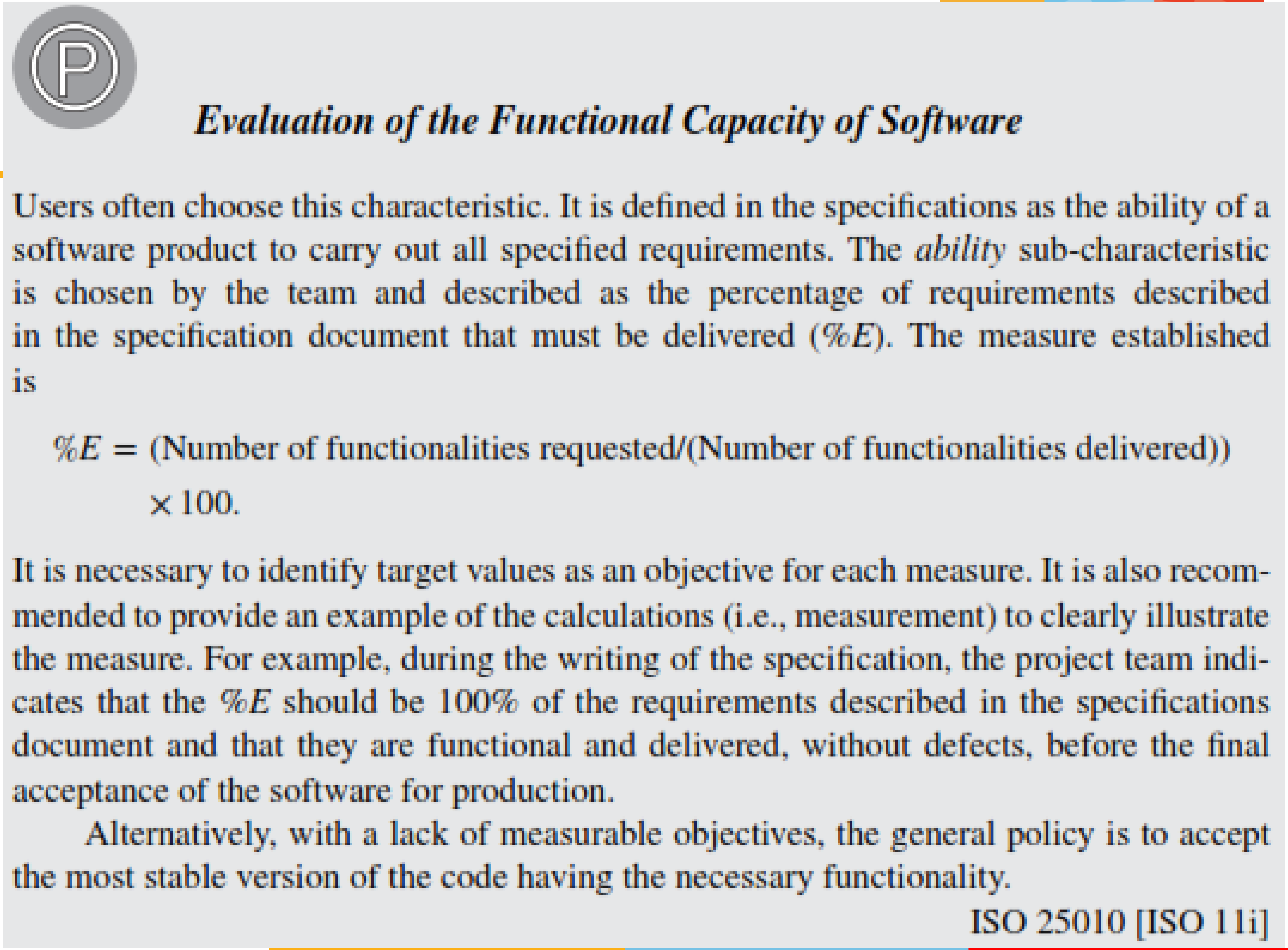
– quality sub-characteristic;

– measure (i.e., formula);

– objectives (i.e., target);

– example.

The last step in defining quality requirements involves having these requirements authorized through consensus.



Evaluation of the Functional Capacity of Software

Users often choose this characteristic. It is defined in the specifications as the ability of a software product to carry out all specified requirements. The *ability* sub-characteristic is chosen by the team and described as the percentage of requirements described in the specification document that must be delivered ($\%E$). The measure established is

$$\%E = (\text{Number of functionalities requested}/(\text{Number of functionalities delivered})) \times 100.$$

It is necessary to identify target values as an objective for each measure. It is also recommended to provide an example of the calculations (i.e., measurement) to clearly illustrate the measure. For example, during the writing of the specification, the project team indicates that the $\%E$ should be 100% of the requirements described in the specifications document and that they are functional and delivered, without defects, before the final acceptance of the software for production.

Alternatively, with a lack of measurable objectives, the general policy is to accept the most stable version of the code having the necessary functionality.

ISO 25010 [ISO 11i]

REQUIREMENT TRACEABILITY DURING THE SOFTWARE LIFE CYCLE

Throughout the life cycle, client needs are documented and developed in different documents, such as specifications, architecture, code, and user manuals.

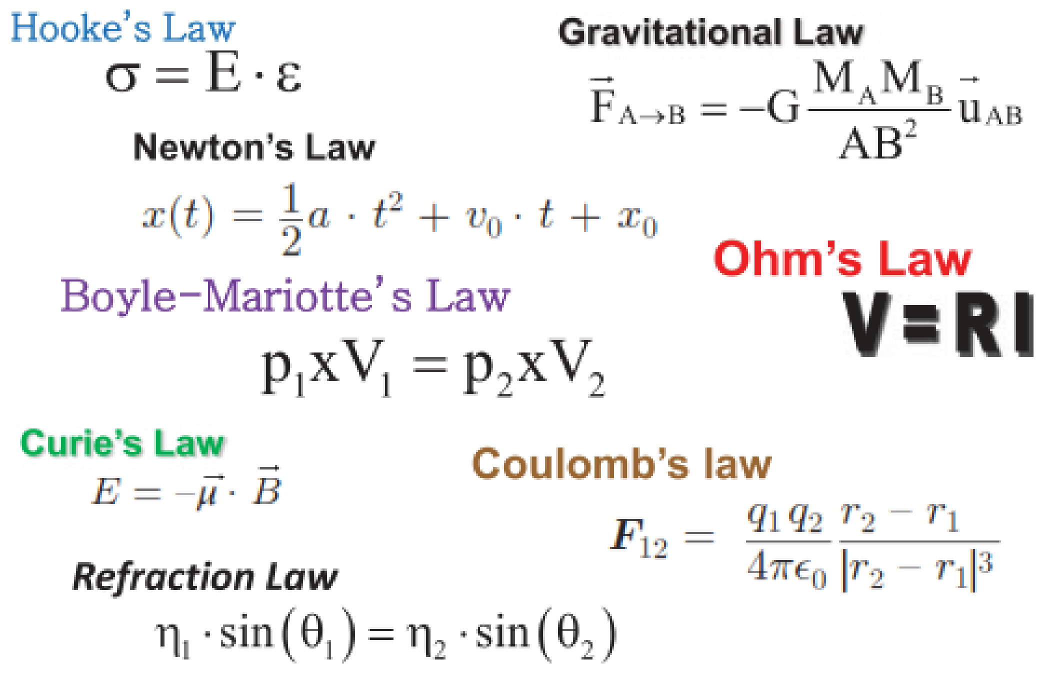
Throughout the life cycle of a system, many changes regarding client needs should be expected.

Every time a need changes, it must be ensured that all documents are updated.

Traceability is a technique that helps us follow the development of needs as well as their changes.

Software Engineering Standards

Other engineering domains such as mechanical, chemical, electrical, or physics engineering are based on the laws of nature as discovered by scientists.



Hooke's Law

$$\sigma = E \cdot \varepsilon$$

Gravitational Law

$$\vec{F}_{A\to B} = -G\frac{M_A M_B}{AB^2}\vec{u}_{AB}$$

Newton's Law

$$x(t) = \frac{1}{2}a \cdot t^2 + v_0 \cdot t + x_0$$

Ohm's Law

$$V=RI$$

Boyle-Mariotte's Law

$$p_1 x V_1 = p_2 x V_2$$

Curie's Law

$$E = -\vec{\mu} \cdot \vec{B}$$

Coulomb's law

$$\boldsymbol{F}_{12} = \frac{q_1 q_2}{4\pi\epsilon_0}\frac{r_2 - r_1}{|r_2 - r_1|^3}$$

Refraction Law

$$\eta_1 \cdot \sin(\theta_1) = \eta_2 \cdot \sin(\theta_2)$$

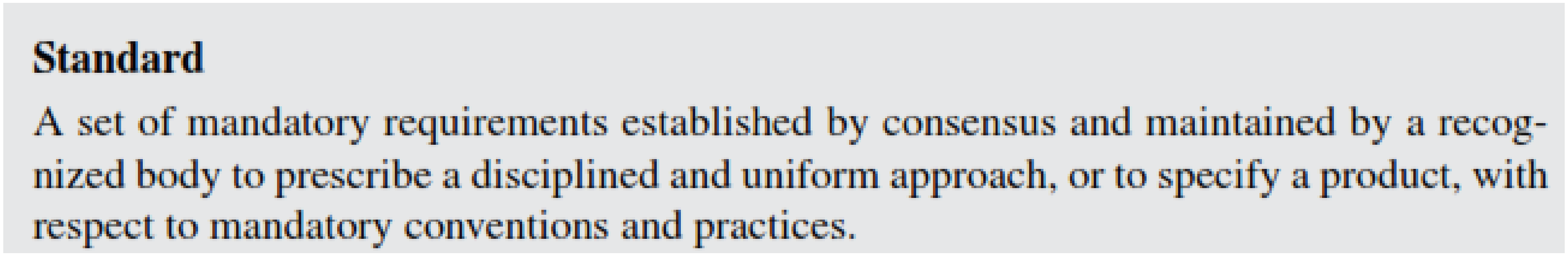
Figure 4.1 A few laws of nature used by some engineering disciplines.

Unfortunately, software engineering, unlike other engineering disciplines, is not based on the laws of nature.

Software engineering, like other disciplines, is based on the use of well-defined practices for ensuring the quality of its products.

In software engineering, there are several standards, which are actually guides for management practices.

A rigorous process is the framework for the way standards are developed and approved including, among others, international ISO standards and standards from professional organizations such as IEEE.



Standard

A set of mandatory requirements established by consensus and maintained by a recognized body to prescribe a disciplined and uniform approach, or to specify a product, with respect to mandatory conventions and practices.

The four principles for the development of ISO standards are:

- **ISO standards meet a market need.**
- **ISO standards are based on worldwide expertise.**
- **ISO standards are the result of a multi-stakeholder process.**
- **ISO standards are based on consensus.**

The ISO standards are developed by consensus

- That all parties were able to express their views;
- The best effort has been made to take into account all opinions and solve all problems (i.e., all the submissions in a vote of the draft of a standard).

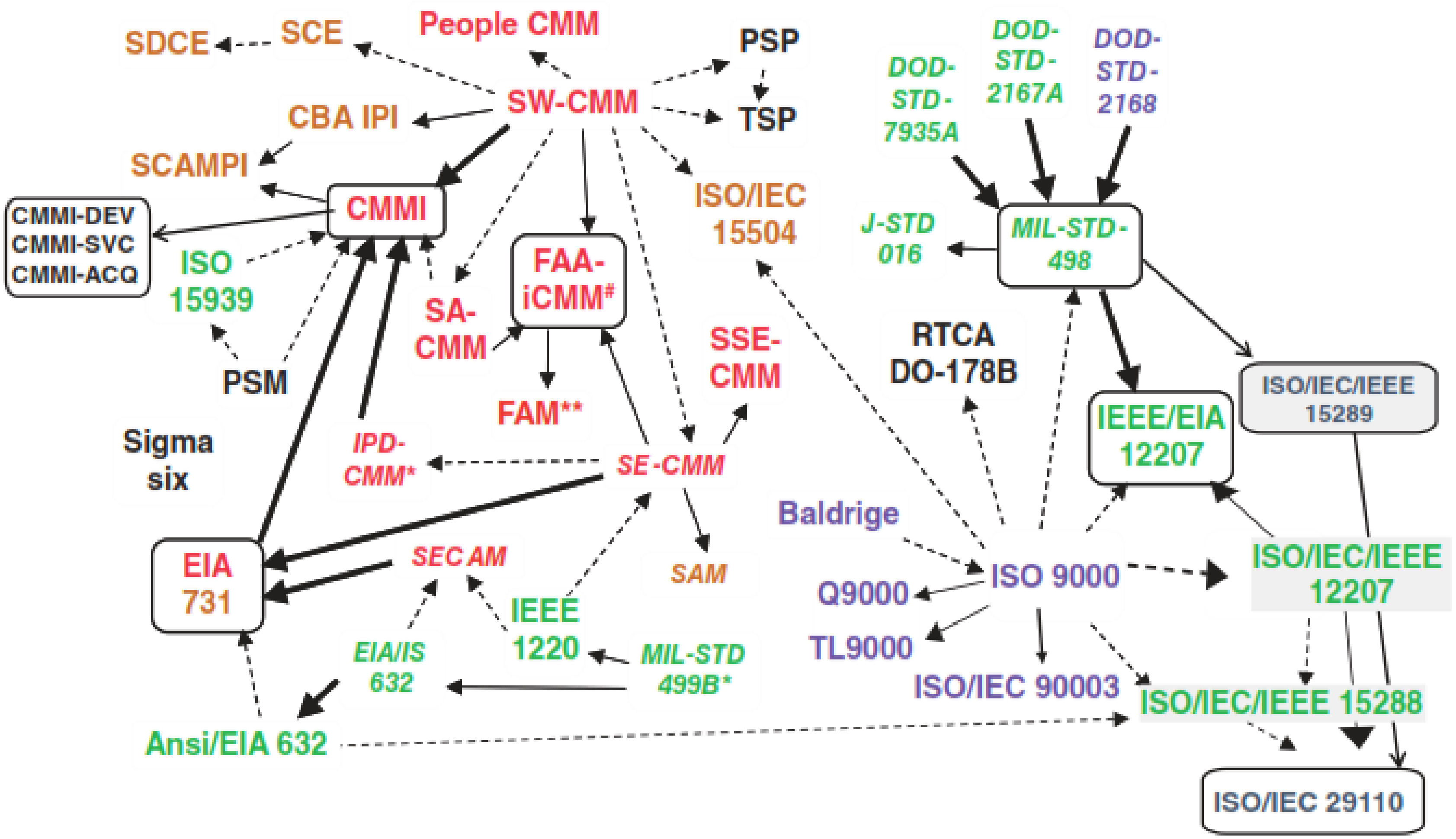


Figure 4.2 The development of standards and models.

- American Department of Defence (DoD) created the “DoD-STD1679A” military standard
- IEEE, the International Organization for Standardization (ISO) European Space Agency (ESA) developed standards
- *“Capability Maturity Model”* (CMM®): developed at the request of the American DoD, by the Software Engineering Institute (SEI) in order to provide a road map of engineering practices to improve the performance of the development, maintenance and service provisioning processes.

Figure 4.3 illustrates the evolution of standards that are maintained and published under the responsibility of the appointed subcommittee for standardized processes, tools, and supporting technologies for software engineering and systems:

The Continuous Evolution of Standards

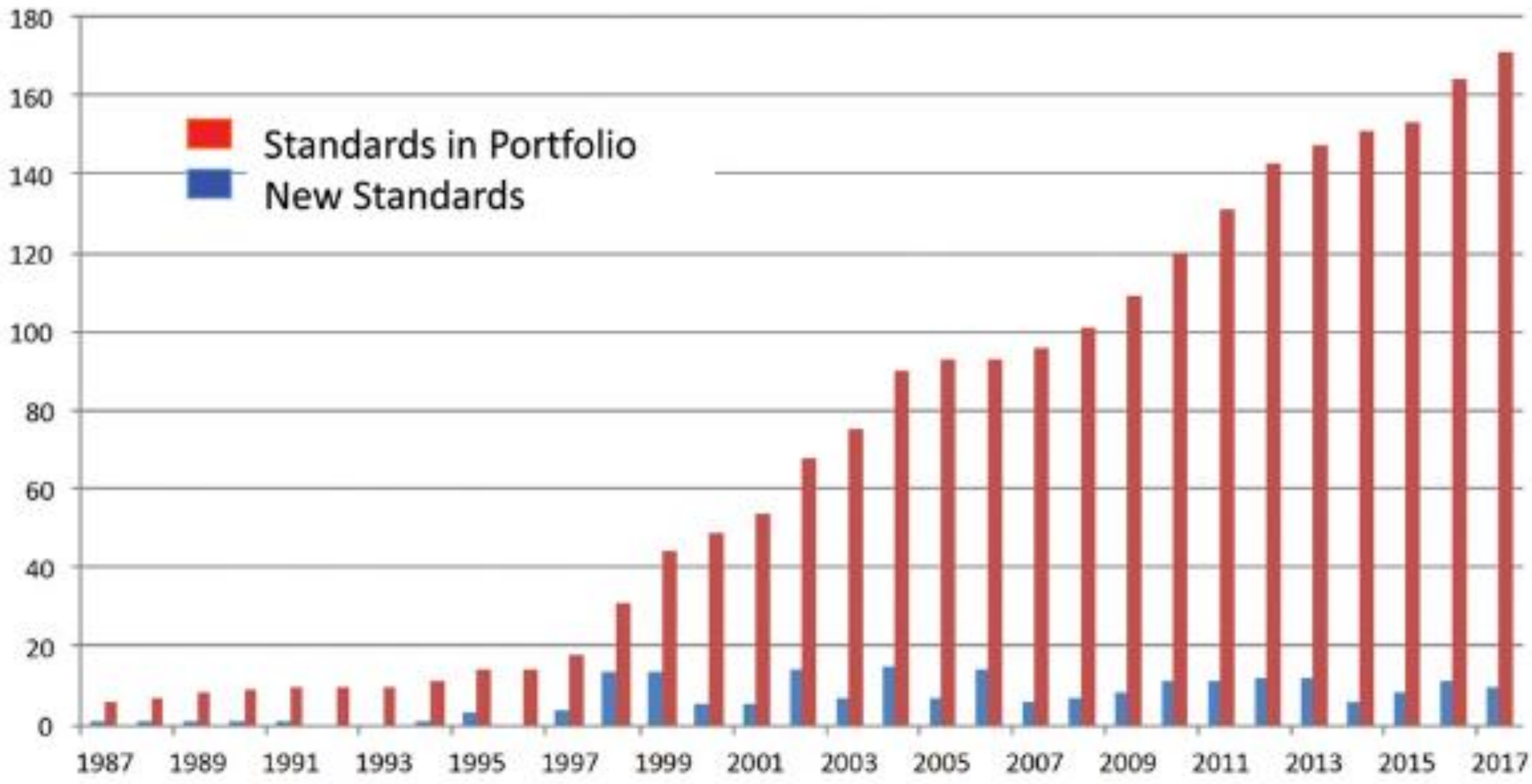


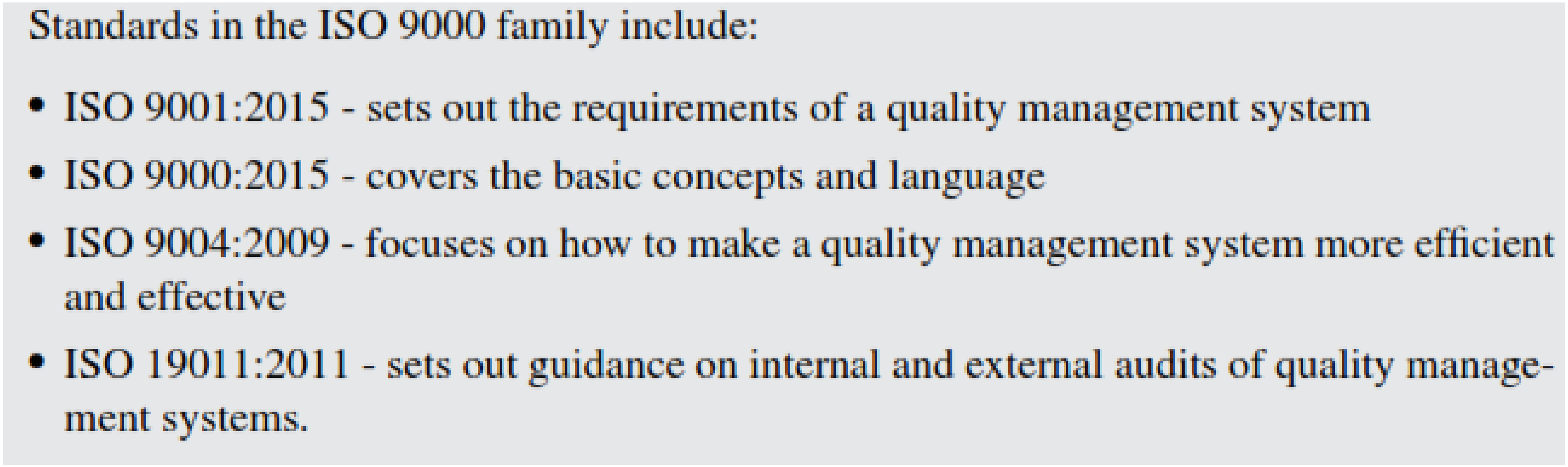
Figure 4.3 The evolution of standards SC7 [SUR 17].

MAIN STANDARDS FOR QUALITY MANAGEMENT

Standards related to the management of software quality:

ISO 9000 [ISO 15b] and ISO 9001 [ISO 15].

The application guide for software, The ISO/IEC 90003 standard.


Standards in the ISO 9000 family include:

- ISO 9001:2015 - sets out the requirements of a quality management system
- ISO 9000:2015 - covers the basic concepts and language
- ISO 9004:2009 - focuses on how to make a quality management system more efficient and effective
- ISO 19011:2011 - sets out guidance on internal and external audits of quality management systems.

- The ISO 9001 standard provides the basic concepts, principles and vocabulary of quality management systems (QMS) and is the basis for other standards for QMSs [ISO 15].
- The “Quality Management Principles” (QMP) are a set of values, rules, standards, and fundamental convictions regarded as fair and that could be the basis for quality management.

The seven QMP of the ISO 9001

- – Principle 1: Customer focus
- – Principle 2: Leadership
- – Principle 3: Involvement of people
- – Principle 4: Process approach
- – Principle 5: System approach to management
- – Principle 6: Factual approach to decision making
- – Principle 7: Mutually beneficial supplier relationships

ISO 9001 uses the process approach, the Plan-Do-Check-Act (PDCA) approach, and a risk-based thinking approach [ISO 15].

- The process approach allows an organization to plan its processes and their interactions.
- The PDCA cycle allows an organization to ensure that its processes are adequately resourced and appropriately managed and that opportunities for improvement are identified and implemented.
- The risk-based thinking approach allows an organization to determine the factors that may cause deviation from its processes and its QMS in relation to expected results, to implement preventive measures in order to limit negative effects and exploit opportunities when they arise.

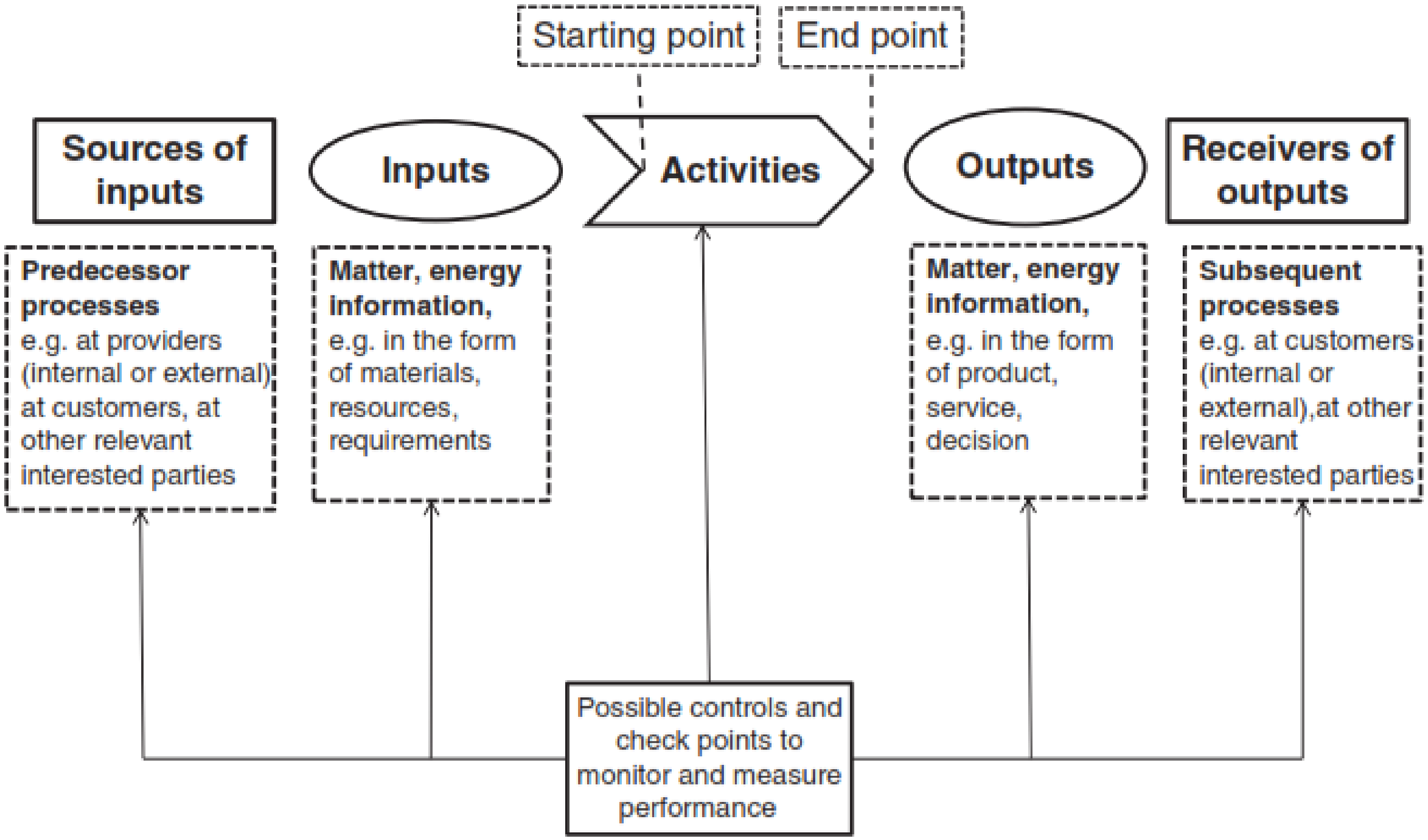

Starting point
End point
Sources of inputs
Inputs
Activities
Outputs
Receivers of outputs
Predecessor processes e.g. at providers (internal or external) at customers, at other relevant interested parties
Matter, energy information, e.g. in the form of materials, resources, requirements
Matter, energy information, e.g. in the form of product, service, decision
Subsequent processes e.g. at customers (internal or external),at other relevant interested parties
Possible controls and check points to monitor and measure performance

Figure 4.4 Elements of a process [ISO 15].

ISO 9001 describes the elements of the PDCA cycle as follows :

– Plan: establish the objectives of the system, processes and resources to deliver results in accordance with customer requirements and policies of the organization, identify and address risks and opportunities;

– Do: implement what has been planned;

– Check: monitor and measure (if applicable) processes and the products and services obtained against policies, objectives, requirements and planned activities, and report the results;

– Act: take actions to improve performance, as needed.

ISO/IEC 90003 Standard

International Electrotechnical Commission.

- provides guidelines for the application of the ISO 9001 standard to computer software.
- It provides organizations with instructions for acquiring, supplying, developing, using and maintaining software.
- Explains what a software audit is for the organization wishing to set up a QMS as well as for the QMS auditor.

ISO/IEC/IEEE 12207 STANDARD

Establishes a common framework for software life cycle processes.

It applies to the acquisition of systems and software products and services, supply, development, operation, maintenance, and disposal of software products and the development of the software part of a system whether performed internally or externally to an organization

ISO 12207 [ISO 17] defines four sets of processes as shown in Figure 4.5:

- **Two agreement processes between a customer and a supplier;**
- **Six organizational project-enabling processes;**
- **Eight processes for technology management;**
- **Fourteen technical processes.**

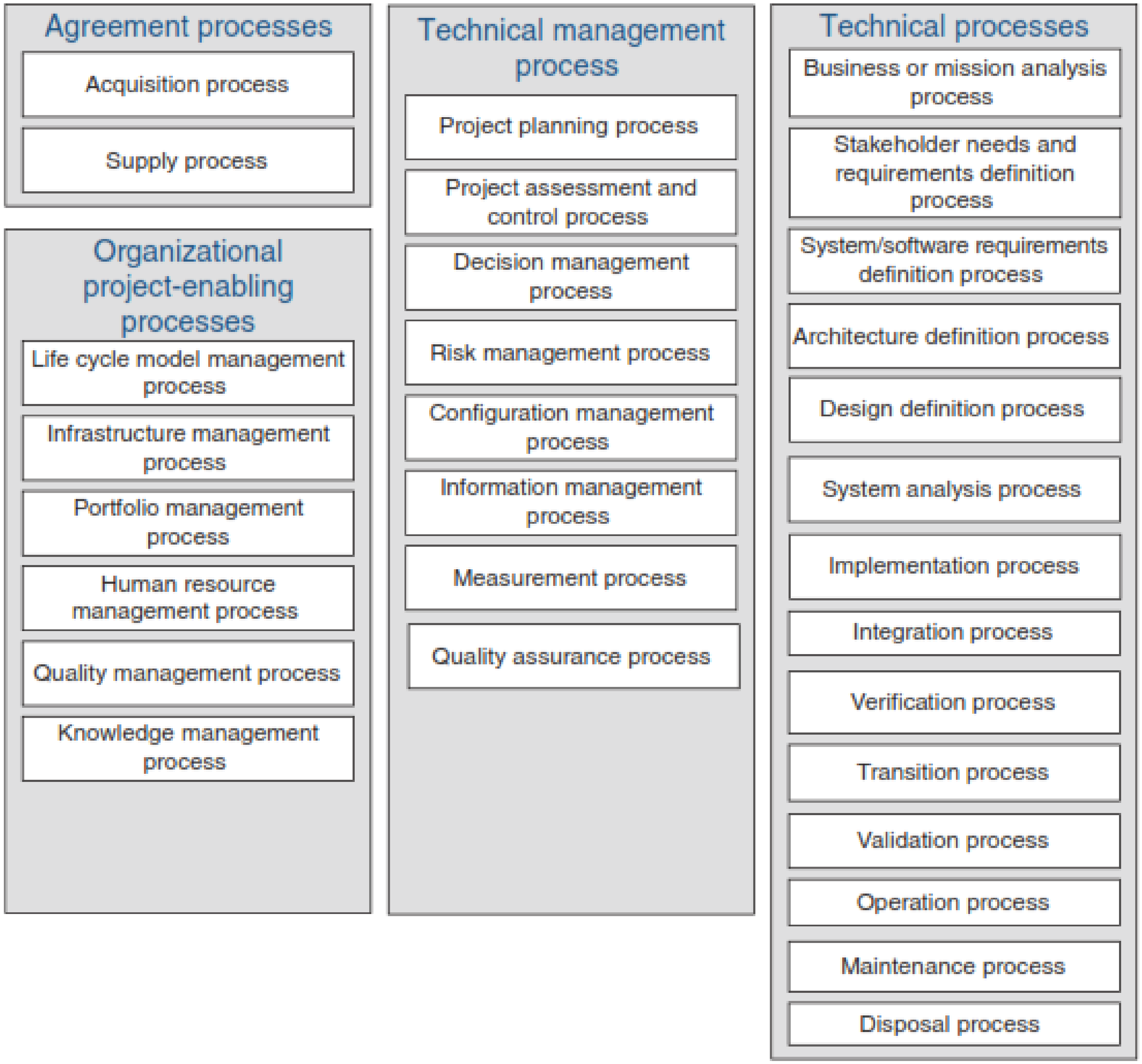

Agreement processes
Acquisition process
Supply process
Organizational project-enabling processes
Life cycle model management process
Infrastructure management process
Portfolio management process
Human resource management process
Quality management process
Knowledge management process
Technical management process
Project planning process
Project assessment and control process
Decision management process
Risk management process
Configuration management process
Information management process
Measurement process
Quality assurance process
Technical processes
Business or mission analysis process
Stakeholder needs and requirements definition process
System/software requirements definition process
Architecture definition process
Design definition process
System analysis process
Implementation process
Integration process
Verification process
Transition process
Validation process
Operation process
Maintenance process
Disposal process

Figure 4.5 The four life cycle process groups of ISO 12207 [ISO 17].

The ISO 12207 standard can be used in one or more of the following modes

By an organization: to help establish an environment of desired processes. These processes can be supported by an infrastructure of methods, procedures, techniques, tools, and trained personnel.

By a project: to help select, structure and employ the elements of an established set of life cycle processes to provide products and services.

By an acquirer and a supplier: to help develop an agreement concerning processes and activities.

By organizations and assessors: to serve as a process reference model for use in the performance of process assessments that may be used to support organizational process improvement.

IEEE 730 STANDARD FOR SQA PROCESSES

QA according to IEEE is a set of proactive measures to ensure the quality of the software product.

The IEEE 730 provides guidance for the SQA activities of products or of services.

The SQA process of the IEEE 730 is grouped into three activities: the implementation of the SQA process, product assurance, and process assurance.

Activities consist of a set of tasks.

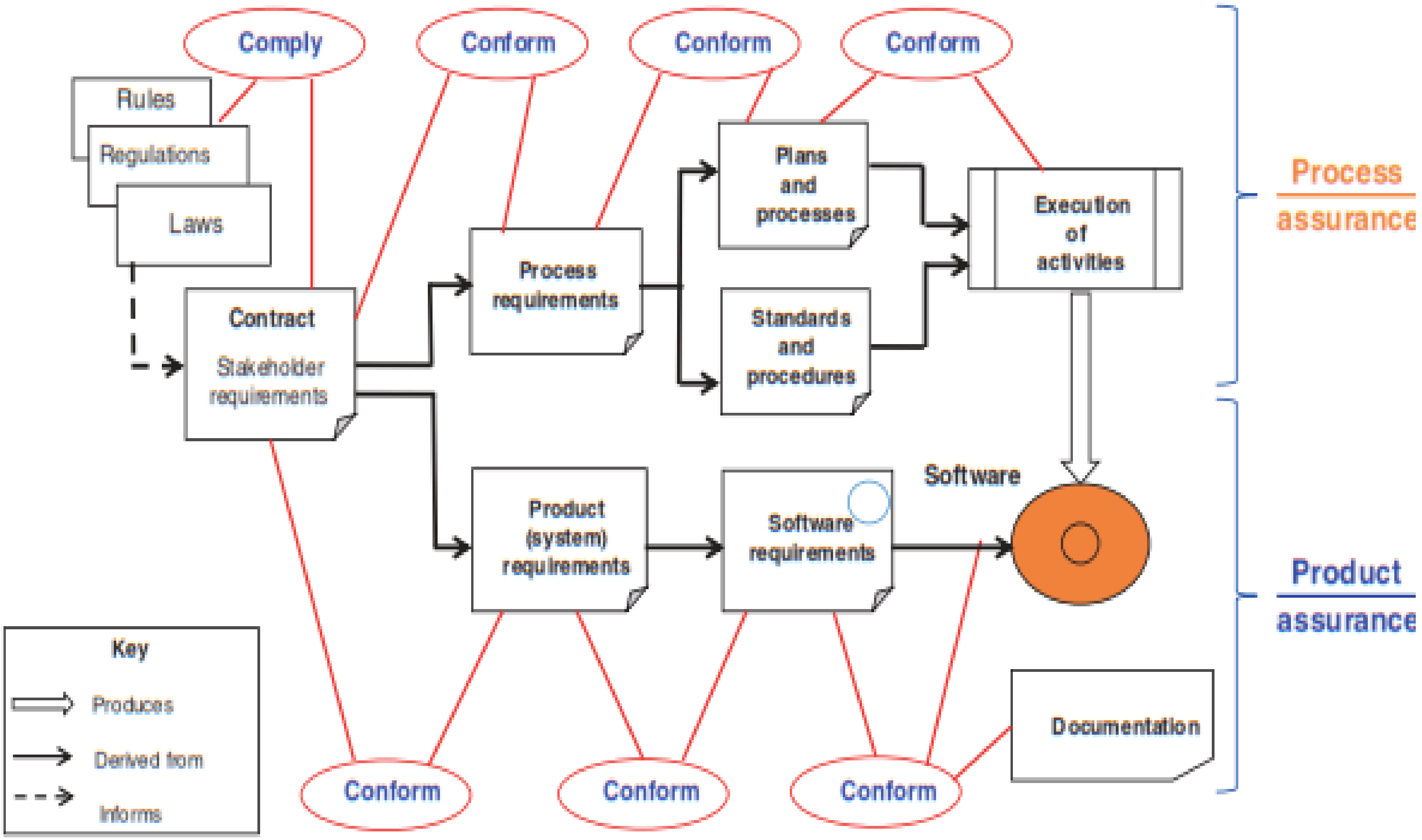


Figure 4.6 The links between requirements and the artifacts of a project [IEE 14].

IEEE 730 [IEE 14] describes what must be done by a project;

- it assumes that the organization has already implemented SQA processes before the start of a project.
- The standard includes a clause that describes what is meant by compliance.

Product Assurance Activities

– evaluate plans for compliance to contracts, standards, and regulations;

– evaluate product for compliance to established requirements;

– evaluate product for acceptability;

– evaluate the compliance of product support;

– measure products.

Process Assurance Activities

– evaluate compliance of the processes and plans;

– evaluate environments for compliance;

– evaluate subcontractor processes for compliance;

– measure processes;

– assess the skill and knowledge of personnel.

Capability Maturity Models (CMM®).

- Tool used to improve and refine software development processes.
- Framework that is used to analyze the approach and techniques followed by any organization to develop software products.
- It also provides guidelines to enhance further the maturity of the process used to develop those software products.

The Capability Maturity Model Integration (CMMI)

- An advanced framework designed to improve and integrate processes across various disciplines such as software engineering, systems engineering, and people management.
- Helps organizations fulfill customer needs, create value for investors, and improve product quality and market growth.

The CMMI model was developed as two versions:

Initial staged version and continuous version, which is the first

CMM model for systems engineering

CMMI-DEV The objective of this model is to encourage organizations to check and continuously improve their development project process and evaluate their level of maturity on a five-level scale as proposed by the staged CMMI model.

The **CMMI for Development (CMMI-DEV)** covers a broader area than its predecessor by adding other practices, such as systems engineering, and the development of integrated processes and products.

The objective of this model is to encourage organizations to check and continuously improve their development project process and evaluate their level of maturity on a five-level scale

Two other CMMI models were developed based on architecture, CMMI for Services (CMMI-SVC) [SEI 10b] and the CMMI for Acquisition (CMMIACQ)

[SEI 10c].

The CMMI-SVC model provides guidelines for organizations that provide services either internally or externally.

The CMMI-ACQ model provides guidelines for organizations that purchase products or services.

All three CMMI models use 16 common process areas.

- For each level of maturity, a set of process areas are defined.
- Each area encompasses a set of requirements that must be met.
- These requirements define which elements must be produced rather than *how they are produced*

Thereby allowing the organization implementing the process to choose its own life cycle model, its design methodologies, its development tools, its programming languages, and its documentation standard.

This approach enables a wide range of companies to implement this model while having processes that are compatible with other standards.

Maturity levels and process areas for each maturity level in the CMMI-DEV model.

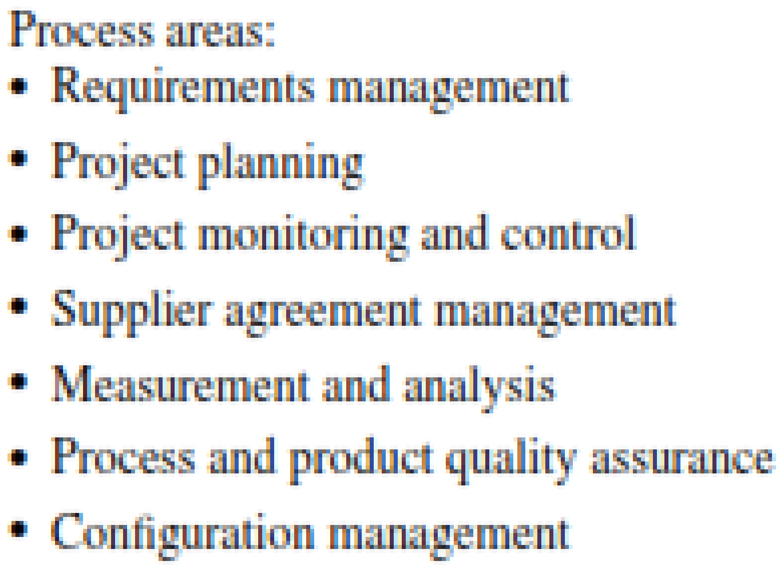
Maturity Level 1: Initial

Processes are usually ad hoc and chaotic.

Maturity level 1 organizations are characterized by a tendency to overcommit, abandon their processes in a time of crisis, and be unable to repeat their successes.

Maturity Level 2: Managed

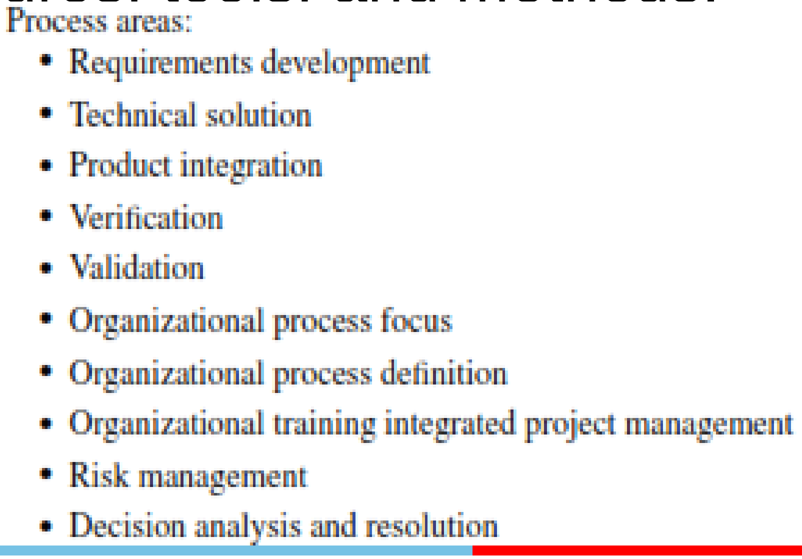
When these practices are in place, projects are performed and managed according to their documented plans.


Process areas:

- Requirements management
- Project planning
- Project monitoring and control
- Supplier agreement management
- Measurement and analysis
- Process and product quality assurance
- Configuration management

Maturity Level 3: Defined

Processes are well characterized and understood, and are described in standards, procedures. tools. and methods.


Process areas:

- Requirements development
- Technical solution
- Product integration
- Verification
- Validation
- Organizational process focus
- Organizational process definition
- Organizational training integrated project management
- Risk management
- Decision analysis and resolution

Maturity Level 4: Quantitatively managed

The organization and projects establish quantitative objectives for quality and process performance and use them as criteria in managing projects.

Process areas:

- Organizational process performance
- Quantitative project management

Maturity Level 5: Optimizing

An organization continually improves its processes based on a quantitative understanding of its business objectives and performance needs.

Process areas

- Organizational performance management
- Causal analysis and resolution

CMMI model structure.

Each process area has generic and specific goals, practices, and sub-practices.

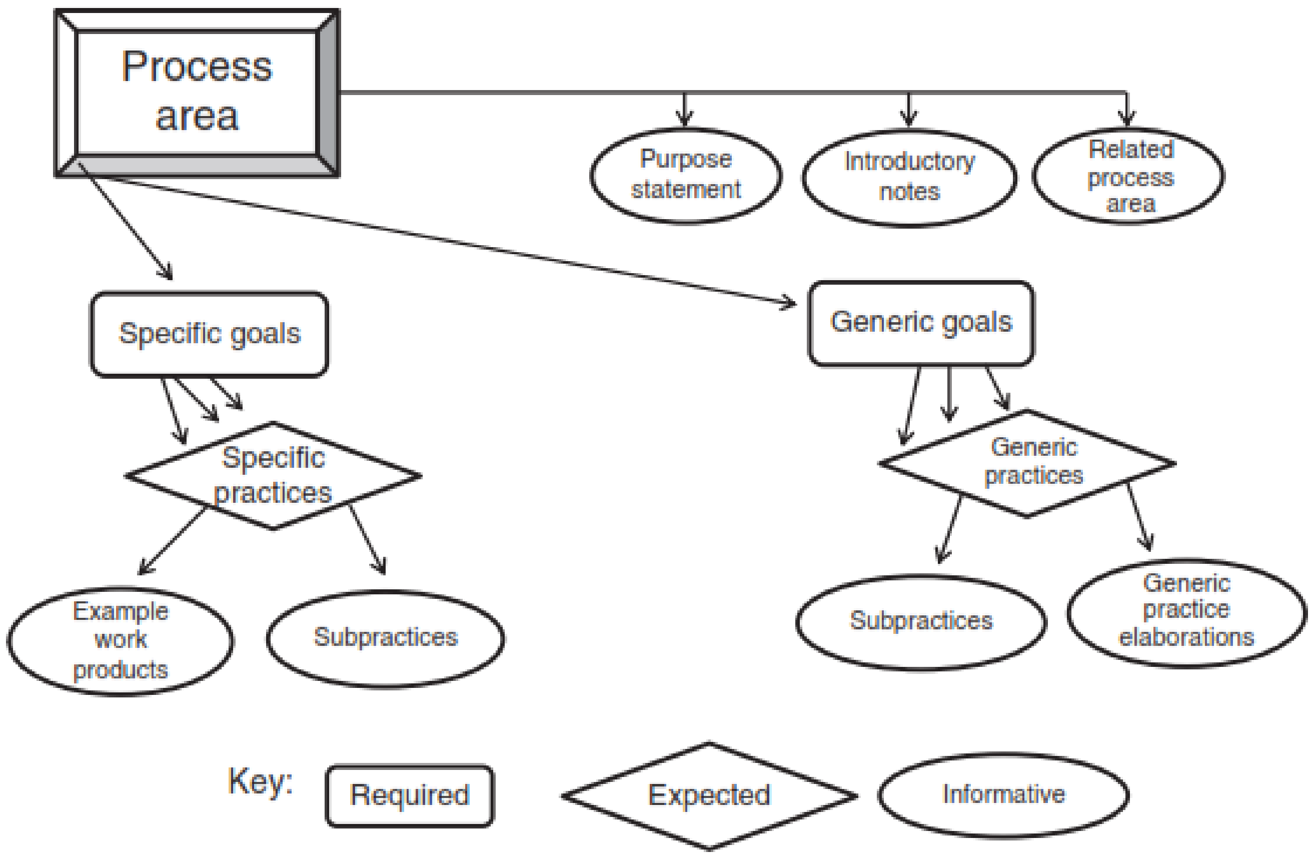
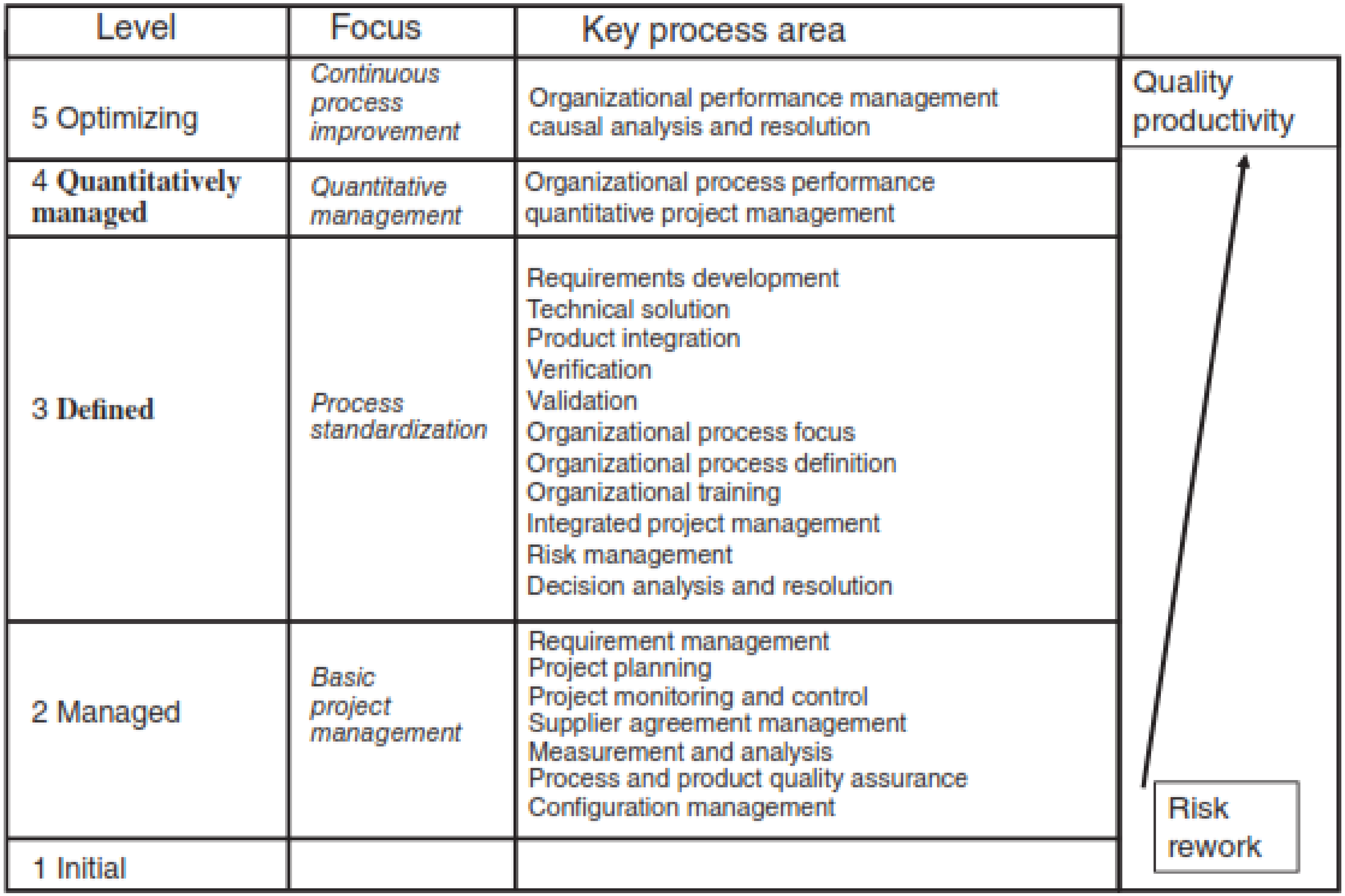


Figure 4.8 Structure of the staged representation of the CMMI [SEI 10a].



Level	Focus	Key process area
5 Optimizing	*Continuous process improvement*	Organizational performance management causal analysis and resolution
4 Quantitatively managed	*Quantitative management*	Organizational process performance quantitative project management
3 Defined	*Process standardization*	Requirements development Technical solution Product integration Verification Validation Organizational process focus Organizational process definition Organizational training Integrated project management Risk management Decision analysis and resolution
2 Managed	*Basic project management*	Requirement management Project planning Project monitoring and control Supplier agreement management Measurement and analysis Process and product quality assurance Configuration management
1 Initial		

Figure 4.9 The staged representation of the CMMI® for Development model.

ITIL Framework

The ITIL framework was created in Great Britain based on good management practices for computer services.

It consists of a set of five books providing advice and recommendations in order to offer quality service to IT service users.

IT services are typically responsible for ensuring that the infrastructures are effective and running (backup copies, recovery, computer administration, telecommunications, and production data)

– strategy;
– design;
– transition;
– operation;
– continuous improvement.

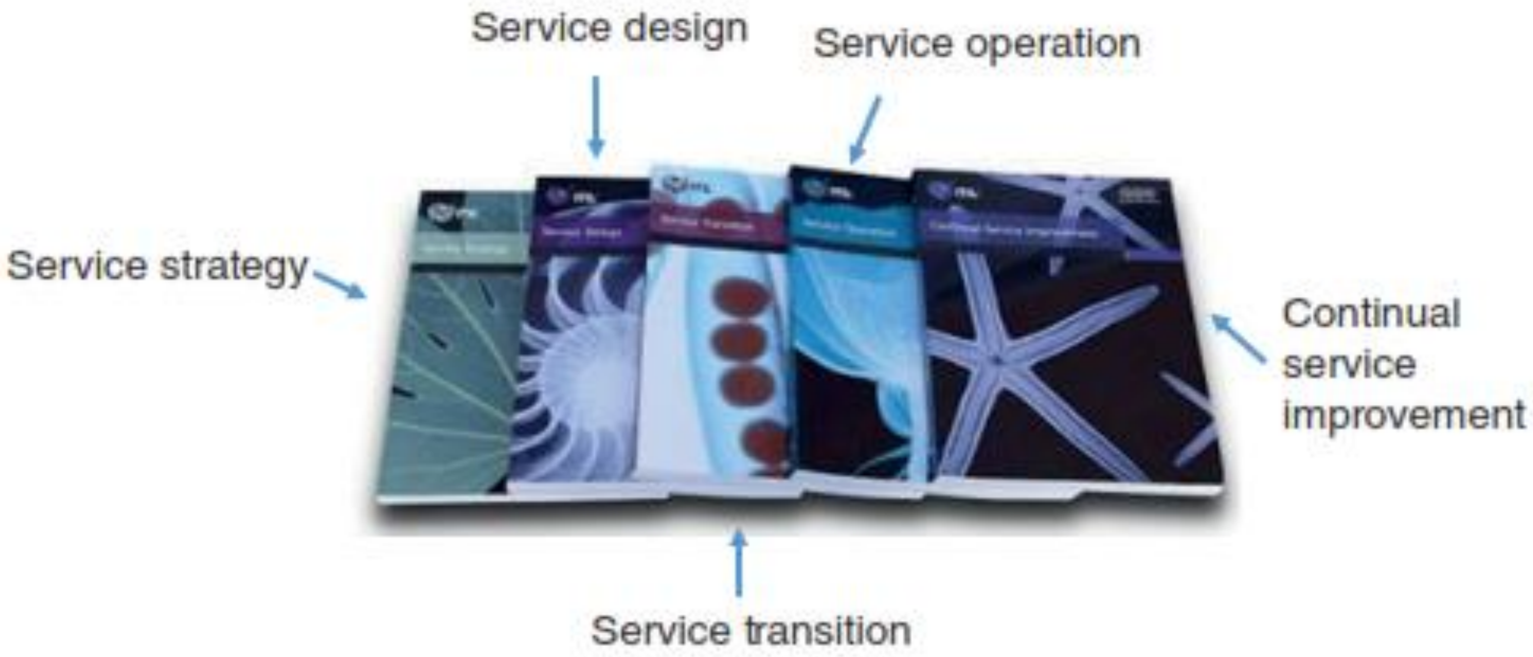


Figure 4.11 The main ITIL guides.

The ITIL framework offers guidance and best practices for managing the five stages of the IT service lifecycle: service strategy, service design, service transition, service operation and continual service improvement.

Support processes described in the ITIL are focused on daily operations. Their main goals are to resolve the problems when they arise or to prevent them from happening when there is a change in the computer environment or in the way the organization does things.

support center function

- – Incident management
- – Problem management
- – Configuration management
- – Change management
- – Commissioning management

Five processes for service operation:

– Service level management

– Financial management of IT services

– Capacity management

– IT service continuity management

– Availability management

Given the major recognition of ITIL worldwide, an international standard based on ITIL came into being: ISO/IEC 20000-1.

The principles of ITIL were successfully conveyed to many companies of all sizes and from all sectors of activity

The main subjects handled under ITIL

– user support, which includes the management of incidents and is an extension of the concept of a Helpdesk;

– provision of services which involves managing processes that are dedicated to the daily operations of IT (cost control, management of service levels);

– management of the production environment infrastructure which involves implementing the means for network management and production tools (scheduling, backup, and monitoring);

– application management which consists of managing the support of an operational program;

– security management (confidentiality, data integrity, data availability, etc.) of the SI (security process) innovate achieve lead

THANK YOU