Quality Management

Software quality management

- Concerned with **ensuring that the required level of quality is achieved in a software product**.
- **Three principal concerns**:
  - **At the organizational level**, quality management is concerned with **establishing a framework of organizational processes and standards** that will lead to high-quality software.
  - **At the project level**, quality management involves
    - **Establishing a quality plan for a project**. The quality **plan should set out the quality goals** for the project **and define what processes and standards are to be used**.
    - **Application of specific quality processes and checking that these planned processes have been followed**.

Quality management activities

- Quality management provides **an independent check on the software development process**.
- The quality management process **checks the project deliverables to ensure that they are consistent with organizational standards and goals**
- **The quality team should be independent from the development team** so that they can take an objective view of the software.

  This allows them to report on software quality without being influenced by software development issues.

Quality management and software development

Quality planning

- **A quality plan sets out the desired product qualities and how these are assessed and defines the most significant quality attributes**.
- The quality plan should **define the quality assessment process.**
- It should **set out which organisational standards should be applied** and, where necessary, define new standards to be used.

Quality plans

- **Quality plan structure**
  - **Product introduction;**
  - **Product plans;**
  - **Process descriptions;**
  - **Quality goals;**
  - **Risks and risk management.**
- Quality plans should be short, succinct documents
  - If they are too long, no-one will read them.

Scope of quality management

- **Quality management** is particularly **important for large, complex systems.** The **quality documentation is a record of progress and supports continuity of development as the development team changes**.
- **For smaller systems, quality management needs less documentation** and **should focus on establishing a quality culture.**

Software quality

- **Quality**, simplistically, **means that a product should meet its specification.**
- **This is problematical for software systems**
  - **There is a tension between customer quality requirements (efficiency, reliability**, etc.) **and developer quality requirements (maintainability, reusability**, etc.);
  - **Some quality requirements are difficult to specify in an unambiguous way**;
  - **Software specifications are usually incomplete and often inconsistent.**
- The **focus may be 'fitness for purpose' rather than specification conformance.**

Software fitness for purpose

- **Have programming and documentation standards been followed** in the development process?
- **Has the software been properly tested**?
- **Is the software sufficiently dependable** to be put into use?
- **Is the performance of the software acceptable** for normal use?
- **Is the software usable**?
- **Is the software well-structured and understandable**?

Software quality attributes

Safety	**Understandability**	**Portability**
Security	**Testability**	**Usability**
Reliability	**Adaptability**	**Reusability**
Resilience	**Modularity**	**Efficiency**
Robustness	**Complexity**	**Learnability**

Resilience:
the quality of being able to return quickly to a previous good condition after problems:

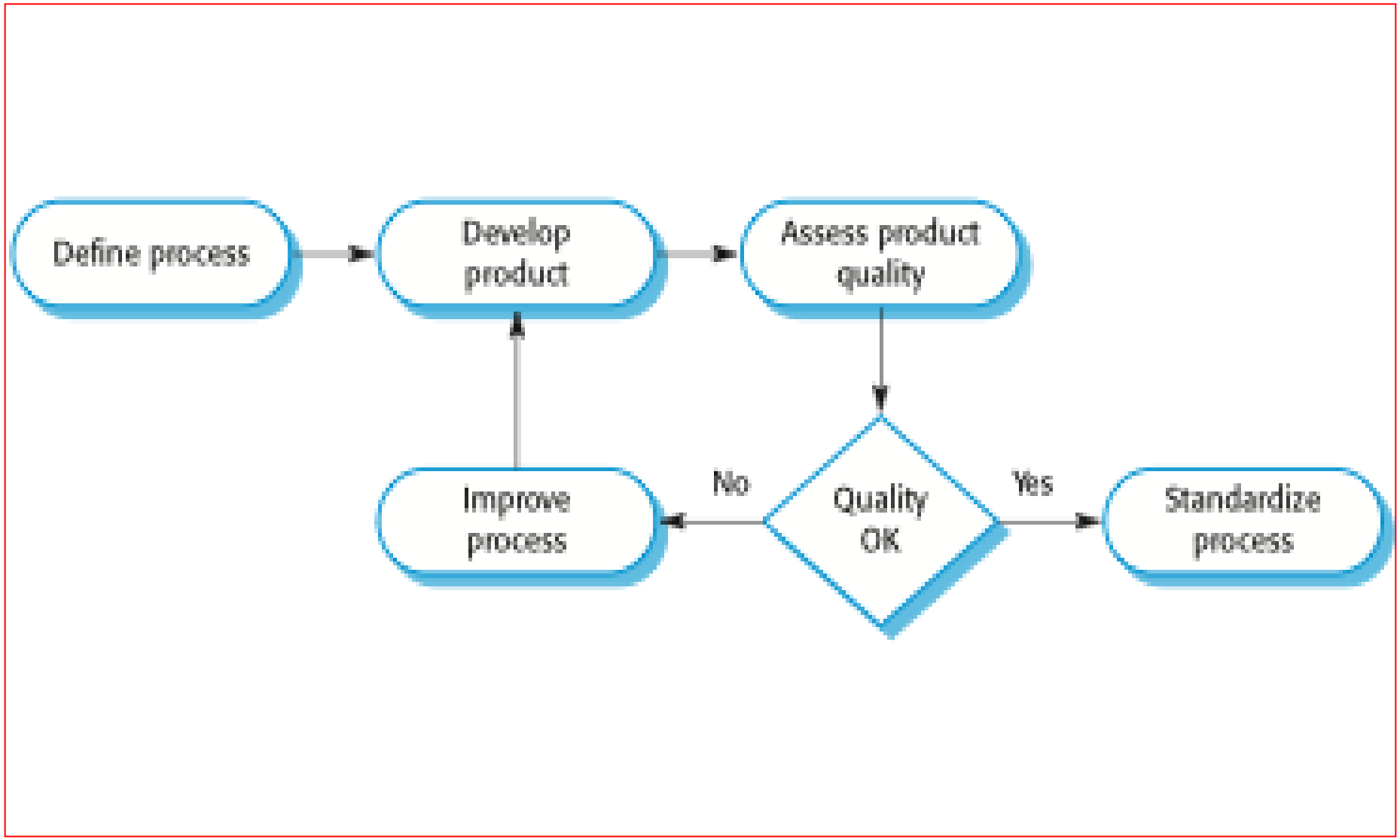
Quality conflicts

- **It is not possible for any system to be optimized for all of these attributes** – for example, **improving robustness may lead to loss of performance**.
- The **quality plan should therefore define the most important quality attributes** for the software that is being developed.
- **The plan should also include a definition of the quality assessment process**, an agreed way of **assessing whether** some quality, such as **maintainability or robustness, is present in the product.**

Process and product quality

- The **quality of a developed product is influenced by the quality of the production process**.
- This is important in software development **as some product quality attributes are hard to assess**.
- However, there is a very **complex** and poorly understood **relationship between software processes and product quality**.
  - The **application of individual skills and experience** is particularly **important in software development**;
  - **External factors** such as the **novelty of an application or** the need for an **accelerated development** schedule **may impair product quality**.

Process-based quality



Software standards

- **Standards define the required attributes of a product or process**. They play an important role in quality management.
- Standards may be **international**, **national**, **organizational** or **project standards**.
- **Product standards define characteristics that all software components should exhibit**

  e.g. **a common programming style**.
- **Process standards define how the software process should be enacted**.

Importance of standards

- Encapsulation of **best practice- avoids** repetition of **past mistakes**.
- **They are a framework for defining what quality means** in a particular setting i.e. that organization's view of quality.
- **They provide continuity - new staff can understand the organisation by understanding the standards that are used**.

Product and process standards

Product standards	Process standards
Design review form	Design review conduct
Requirements document structure	Submission of new code for system building
Method header format	Version release process
Java programming style	Project plan approval process
Project plan format	Change control process
Change request form	Test recording process

Problems with standards

- They **may not be seen as relevant and up-to-date** by software engineers.
- They **often involve too much bureaucratic form filling.**
- **If they are unsupported by software tools, tedious form filling work is often involved to maintain the documentation associated with the standards**.

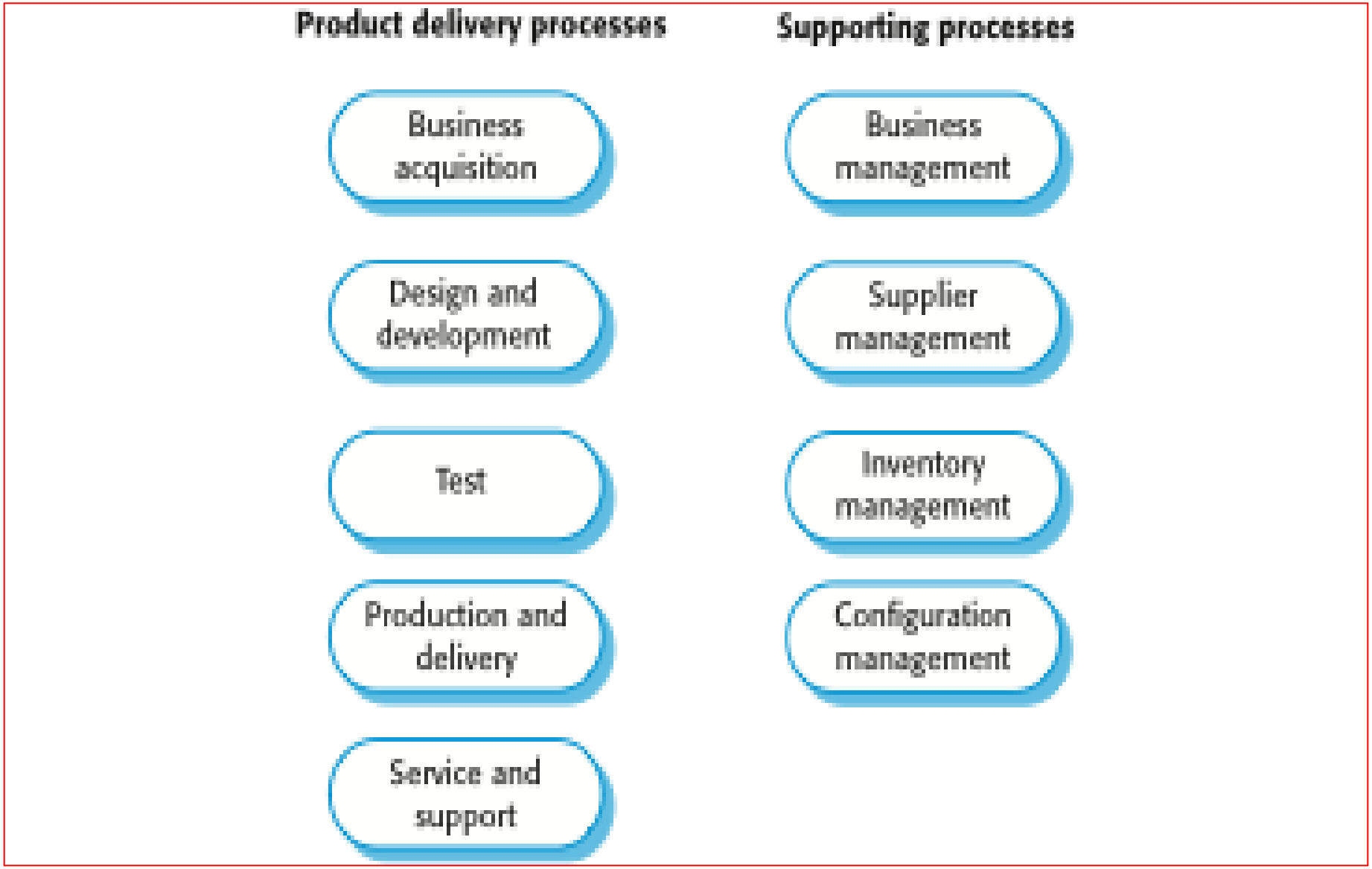
Standards development

- **Involve practitioners in development**. Engineers should understand the rationale underlying a standard.
- **Review standards and their usage regularly. Standards can quickly become outdated** and this reduces their credibility amongst practitioners.
- **Detailed standards should have specialized tool support**. Excessive clerical work is the most significant complaint against standards.
  - Web-based forms are not good enough.

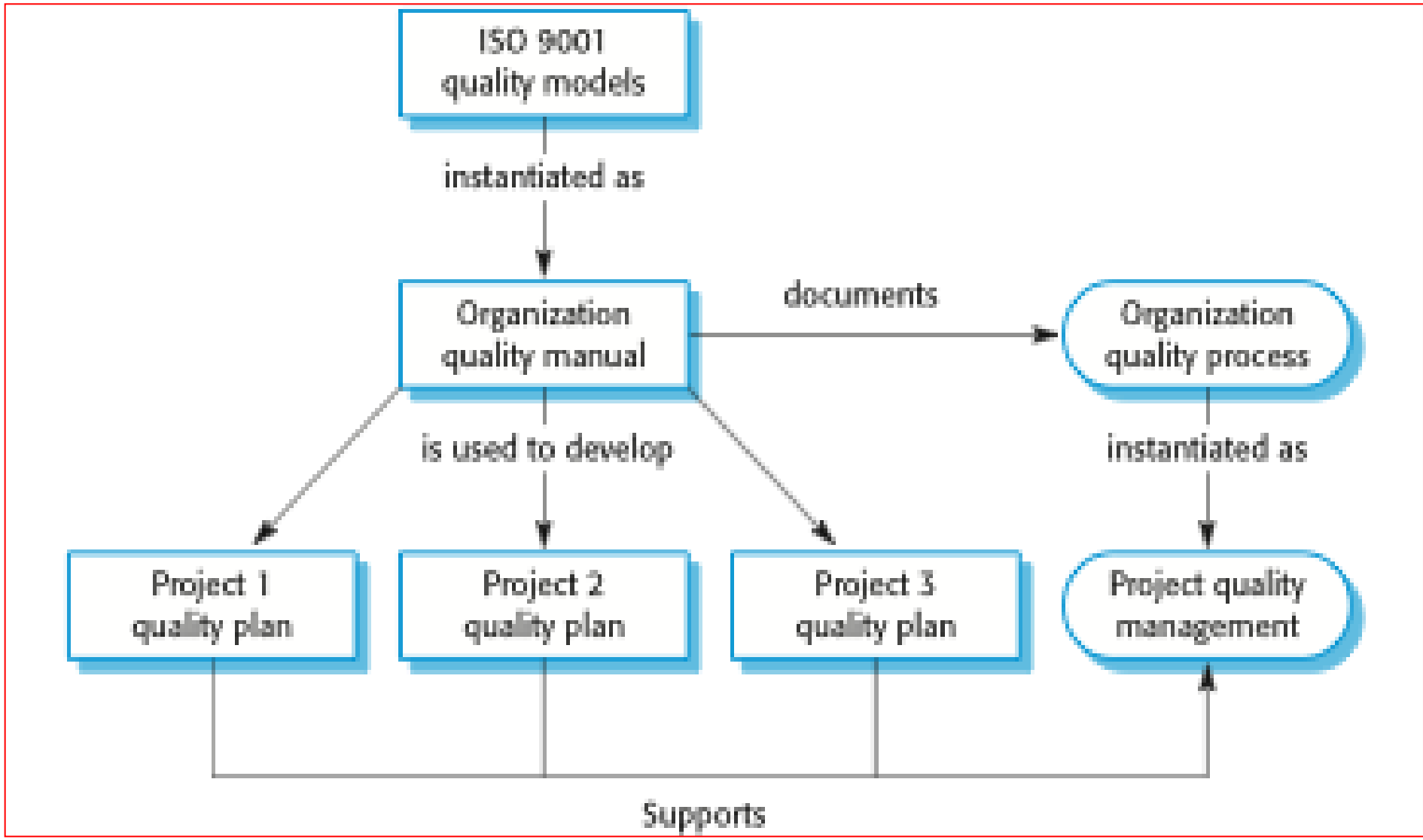
ISO 9001 standards framework

- An international set of standards that can be used **as a basis for developing quality management systems.**
- **ISO 9001**, the most general of these standards, **applies to organizations that design, develop and maintain products, including software**.
- **The ISO 9001 standard is a framework for developing software standards**.
  - **It sets out general quality principles, describes quality processes in general** and **lays out the organizational standards and procedures** that should be defined. These should be **documented in an organizational quality manual.**

ISO 9001 core processes



ISO 9001 and quality management



ISO 9001 certification

- **Quality standards and procedures should be documented in an organisational quality manual**.
- **An external body may certify that an organisation's quality manual conforms to ISO 9000 standards.**
- **Some customers require suppliers to be ISO 9000 certified** although the need for flexibility here is increasingly recognised.

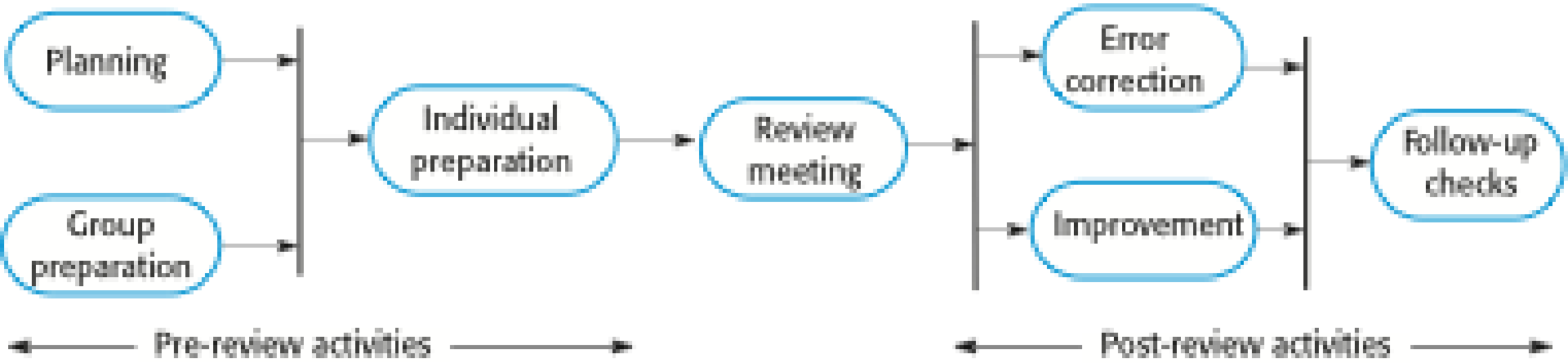
Reviews and inspections

- **A group examines part or all of a process or system and its documentation** to find potential problems.
- Software or documents may be 'signed off' at a review which signifies that progress to the next development stage has been approved by management.
- There are **different types of review** with different objectives
  - **Inspections for defect removal (product);**
  - **Reviews for progress assessment (product and process);**
  - **Quality reviews (product and standards)**.

Quality reviews

- **A group of people carefully examine part or all of a software system and its associated documentation.**
- **Code, designs, specifications, test plans, standards, etc. can all be reviewed.**
- **Software or documents may be 'signed off' at a review** which signifies that **progress to the next development stage has been approved** by management.

The software review process



Reviews and agile methods

- The **review process in agile software development** is **usually informal**.
  - **In Scrum**, for example, **there is a review meeting after each iteration of the software has been completed (a sprint review)**, where quality issues and problems may be discussed.
- **In extreme programming, pair programming ensures that code is constantly being examined and reviewed by another team member.**
- **XP relies on individuals taking the initiative to improve and refactor code**. **Agile approaches are not usually standards-driven**, so issues of standards compliance are not usually considered.

Program inspections

- These **are peer reviews where engineers examine the source of a system with the aim of discovering anomalies** and defects.
- **Inspections do not require execution of a system** so may be **used before implementation**.
- They **may be applied to any representation of the system** (**requirements, design, configuration data, test data**, etc.).
- They have been shown to be an **effective technique for discovering program errors**.

Inspection checklists

- **Checklist of common errors** should be **used to drive the inspection.**
- **Error checklists** are **programming language dependent** and **reflect the characteristic errors** that are likely to arise **in the language**.
- In general, the 'weaker' the type checking, the larger the checklist.
- **Examples: Initialisation, Constant naming, loop termination, array bounds**, etc.

An inspection checklist (a)

Fault class	Inspection check
Data faults	• Are all program variables initialized before their values are used? • Have all constants been named? • Should the upper bound of arrays be equal to the size of the array or Size -1? • If character strings are used, is a delimiter explicitly assigned? • Is there any possibility of buffer overflow?
Control faults	• For each conditional statement, is the condition correct? • Is each loop certain to terminate? • Are compound statements correctly bracketed? • In case statements, are all possible cases accounted for? • If a break is required after each case in case statements, has it been included?
Input/output faults	• Are all input variables used? • Are all output variables assigned a value before they are output? • Can unexpected inputs cause corruption?

An inspection checklist (b)

Fault class	Inspection check
Interface faults	• Do all function and method calls have the correct number of parameters? • Do formal and actual parameter types match? • Are the parameters in the right order? • If components access shared memory, do they have the same model of the shared memory structure?
Storage management faults	• If a linked structure is modified, have all links been correctly reassigned? • If dynamic storage is used, has space been allocated correctly? • Is space explicitly deallocated after it is no longer required?
Exception management faults	• Have all possible error conditions been taken into account?

Agile methods and inspections

- **Agile processes rarely use formal inspection or peer review processes**.
- **Rather, they rely on team members cooperating to check each other's code**, and informal guidelines, such as 'check before check-in', which suggest that programmers should check their own code.
- **Extreme programming** practitioners **argue that pair programming is an effective substitute for inspection** as this is, in effect, a continual inspection process.
- **Two people look at every line of code and check it before it is accepted.**

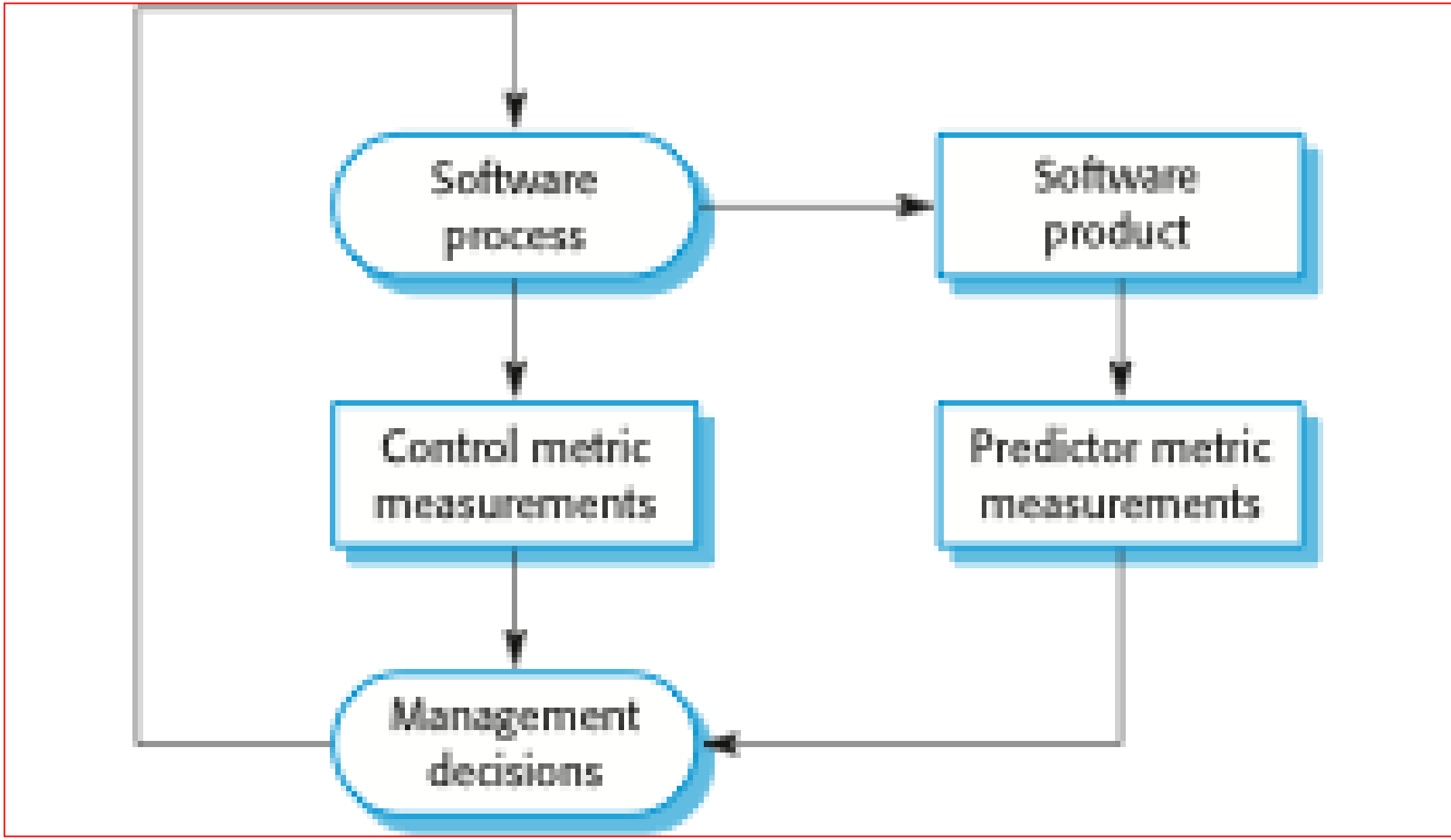
Software measurement and metrics

- **Software measurement** is concerned with **deriving a numeric value for an attribute of a software product or process**.
- This allows for objective comparisons between techniques and processes.
- Although some companies have introduced measurement programmes, most organisations still don’t make systematic use of software measurement.
- There are few established standards in this area.

Software metric

- **Any type of measurement which relates to a software system**, process or related documentation
  - **Lines of code** in a program, the **Fog index**, number of **person-days** required to develop a component.
- Allow the software and the software process to be quantified.
- **May be used to predict product attributes or to control the software process**.
- Product metrics can be used for general predictions or to identify anomalous components.

Predictor and control measurements



Use of measurements

✧ To assign a value to system quality attributes

- **By measuring the characteristics of system components**, such as their **cyclomatic complexity**, and then aggregating these measurements, **you can assess system quality attributes**, **such as maintainability.**

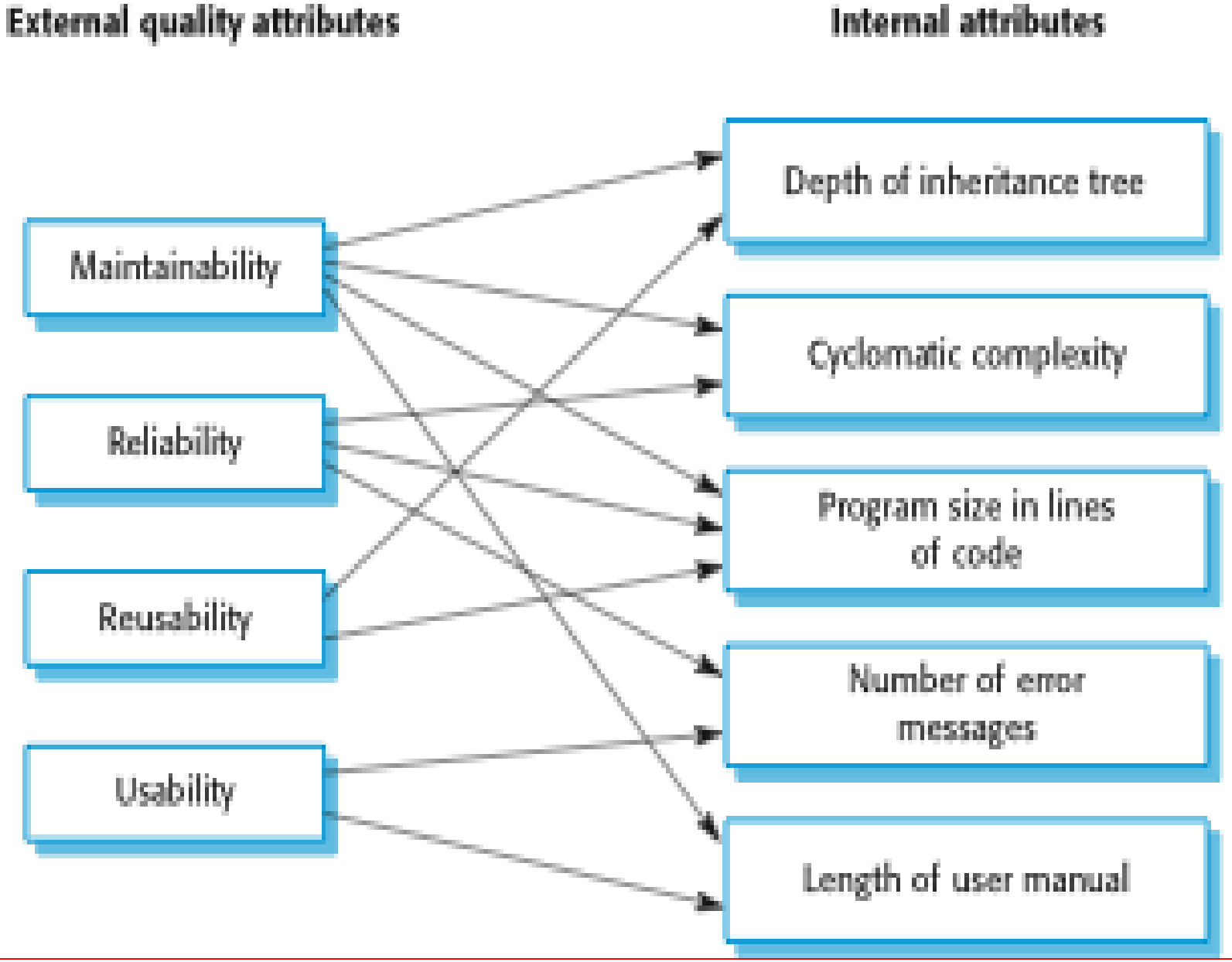
✧ To identify the system components whose quality is sub-standard

- Measurements can identify individual components with characteristics that deviate from the norm. For example, you can measure **components** to discover those **with the highest complexity**. These are **most likely to contain bugs** because the complexity makes them harder to understand.

Metrics assumptions

- A software property can be measured.
- **The relationship exists between what we can measure and what we want to know**. **We can only measure internal attributes** but are often more interested in external software attributes.
- This relationship has been formalised and validated.
- It may be difficult to relate what can be measured to desirable external quality attributes.

Relationships between internal and external software



Problems with measurement in industry

- It is **impossible to quantify the return on investment of introducing an organizational metrics program**.
- **There are no standards for software metrics or standardized processes for measurement and analysis**.
- In many companies, software processes are not standardized and are poorly defined and controlled.
- Most work on software measurement has focused on code-based metrics and plan-driven development processes. However, more and more software is now developed by configuring ERP systems or COTS.
- **Introducing measurement adds additional overhead to processes.**

Product metrics

- A **quality metric** should be a **predictor of product quality.**
- **Classes of product metric**
  - **Dynamic metrics** which are **collected by measurements made of a program in execution**;
  - **Static metrics** which are **collected by measurements made of the system representations**;

Dynamic metrics help **assess efficiency and reliability**

Static metrics help **assess complexity, understandability and maintainability.**

Dynamic and static metrics

- **Dynamic metrics** are closely **related to software quality attributes**
  - It is relatively **easy to measure** the **response time of a system (performance attribute)** or the **number of failures (reliability attribute).**
- **Static metrics have an indirect relationship with quality attributes**
  - You need to try and **derive a relationship between these metrics and properties such as complexity, understandability and maintainability.**

Static software product metrics

Software metric	Description
Fan-in/Fan-out	Fan-in is a measure of the number of functions or methods that call another function or method (say X). Fan-out is the number of functions that are called by function X. A high value for fan-in means that X is tightly coupled to the rest of the design and changes to X will have extensive knock-on effects. A high value for fan-out suggests that the overall complexity of X may be high because of the complexity of the control logic needed to coordinate the called components.
Length of code	This is a measure of the size of a program. Generally, the larger the size of the code of a component, the more complex and error-prone that component is likely to be. Length of code has been shown to be one of the most reliable metrics for predicting error-proneness in components.

Static software product metrics

Software metric	Description
Cyclomatic complexity	This is **a measure of the control complexity of a program.** This control complexity may be related to program understandability. I discuss cyclomatic complexity in Chapter 8.
Length of identifiers	This is **a measure of the average length of identifiers (names for variables, classes, methods, etc.)** in a program. The longer the identifiers, the more likely they are to be meaningful and hence the more understandable the program.
Depth of conditional nesting	This is **a measure of the depth of nesting of if-statements in a program**. Deeply nested if-statements are hard to understand and potentially error-prone.
Fog index	This is **a measure of the average length of words and sentences in documents**. The higher the value of a document's Fog index, the more difficult the document is to understand.

The CK object-oriented metrics suite

Object-oriented metric	Description
Weighted methods per class (WMC)	This is **the number of methods in each class**, **weighted by the complexity of each method**. Therefore, a simple method may have a complexity of 1, and a large and complex method a much higher value. **The larger the value for this metric, the more complex the object class**. Complex objects are more likely to be **difficult to understand.** They may not be logically cohesive, so cannot be reused effectively as superclasses in an inheritance tree.
Depth of inheritance tree (DIT)	This represents **the number of discrete levels in the inheritance tree where subclasses inherit attributes and operations (methods) from superclasses. The deeper the inheritance tree, the more complex the design**. Many object classes may have to be understood to understand the object classes at the leaves of the tree.
Number of children (NOC)	This is **a measure of the number of immediate subclasses in a class. It measures the breadth of a class hierarchy, whereas DIT measures its depth. A high value for NOC may indicate greater reuse**. It may mean that more effort should be made in validating base classes because of the number of subclasses that depend on them.

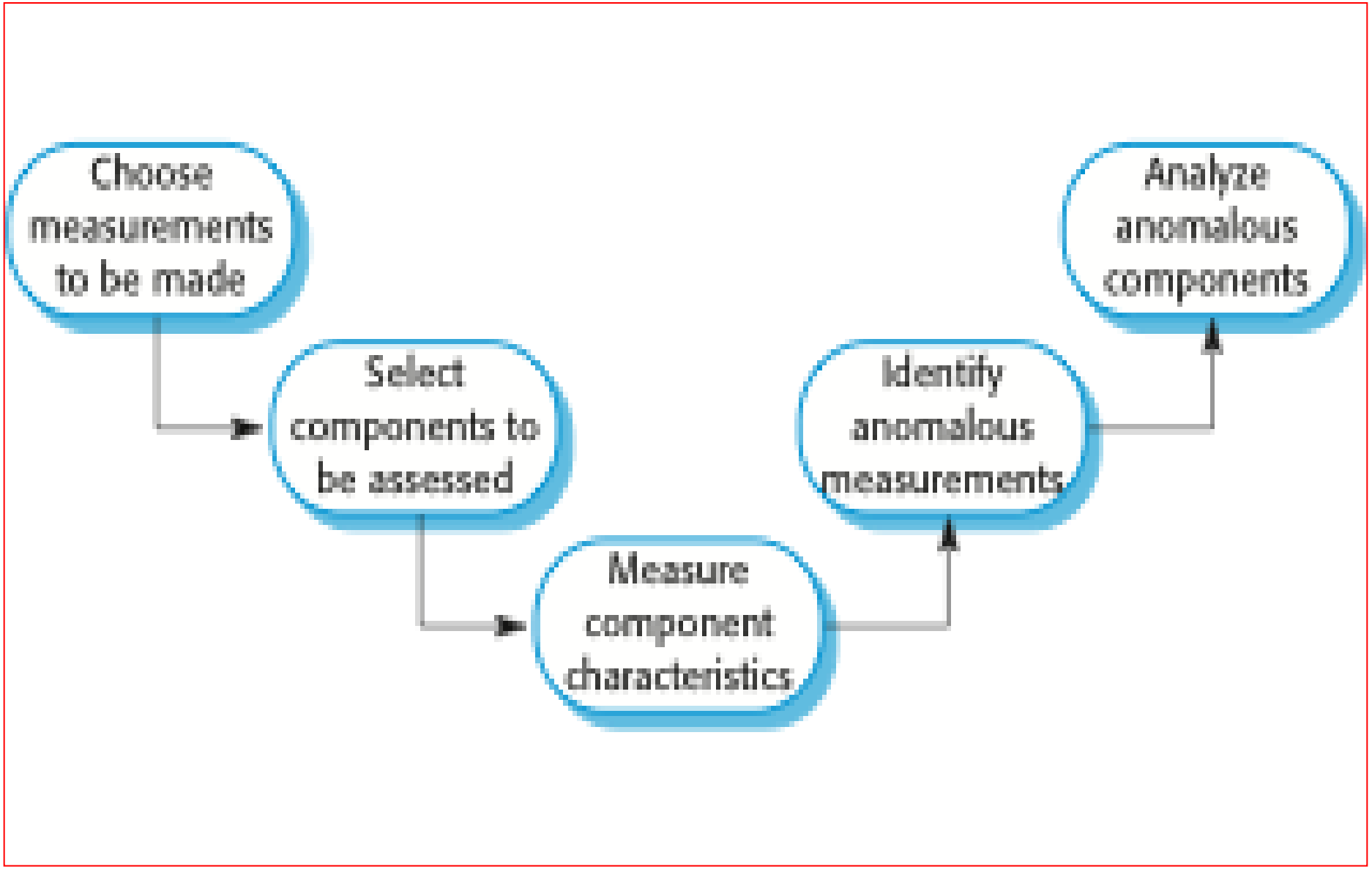
The CK object-oriented metrics suite

Object-oriented metric	Description
Coupling between object classes (CBO)	**Classes are coupled when methods in one class use methods or instance variables defined in a different class**. **CBO is a measure of how much coupling exists**. **A high value for CBO means that classes are highly dependent**, and therefore it is more likely that changing one class will affect other classes in the program.
Response for a class (RFC)	**RFC is a measure of the number of methods that could potentially be executed in response to a message received by an object of that class**. Again, RFC is related to complexity. **The higher the value for RFC, the more complex a class** and hence the more likely it is that it will include errors.
Lack of cohesion in methods (LCOM)	LCOM is calculated by considering pairs of methods in a class. **LCOM is the difference between the number of method pairs without shared attributes and the number of method pairs with shared attributes**. The value of this metric has been widely debated and it exists in several variations. It is not clear if it really adds any additional, useful information over and above that provided by other metrics.

Software component analysis

- System component can be analyzed separately using a range of metrics.
- The values of these metrics may then compared for different components and, perhaps, with historical measurement data collected on previous projects.
- Anomalous measurements, which deviate significantly from the norm, may imply that there are problems with the quality of these components.

The process of product measurement



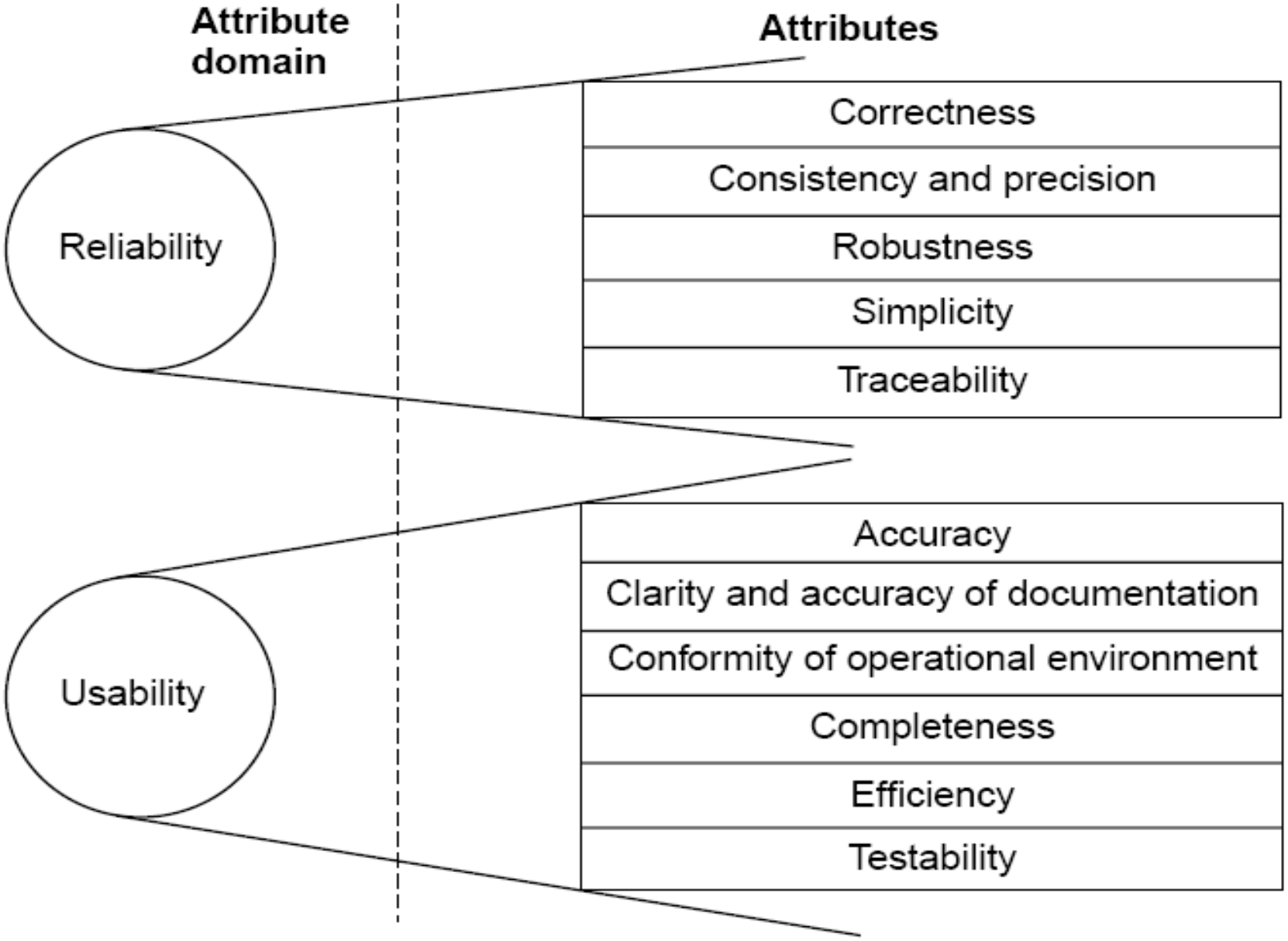
Measurement surprises

- Reducing the number of faults in a program leads to an increased number of help desk calls
  - The program is now thought of as more reliable and so has a wider more diverse market. The percentage of users who call the help desk may have decreased but the total may increase;
  - A more reliable system is used in a different way from a system where users work around the faults. This leads to more help desk calls.

Software Quality

Different people understand different meanings of quality like:

- conformance to requirements
- fitness for the purpose
- level of satisfaction


Attribute domain
Attributes
Reliability
Correctness
Consistency and precision
Robustness
Simplicity
Traceability
Usability
Accuracy
Clarity and accuracy of documentation
Conformity of operational environment
Completeness
Efficiency
Testability

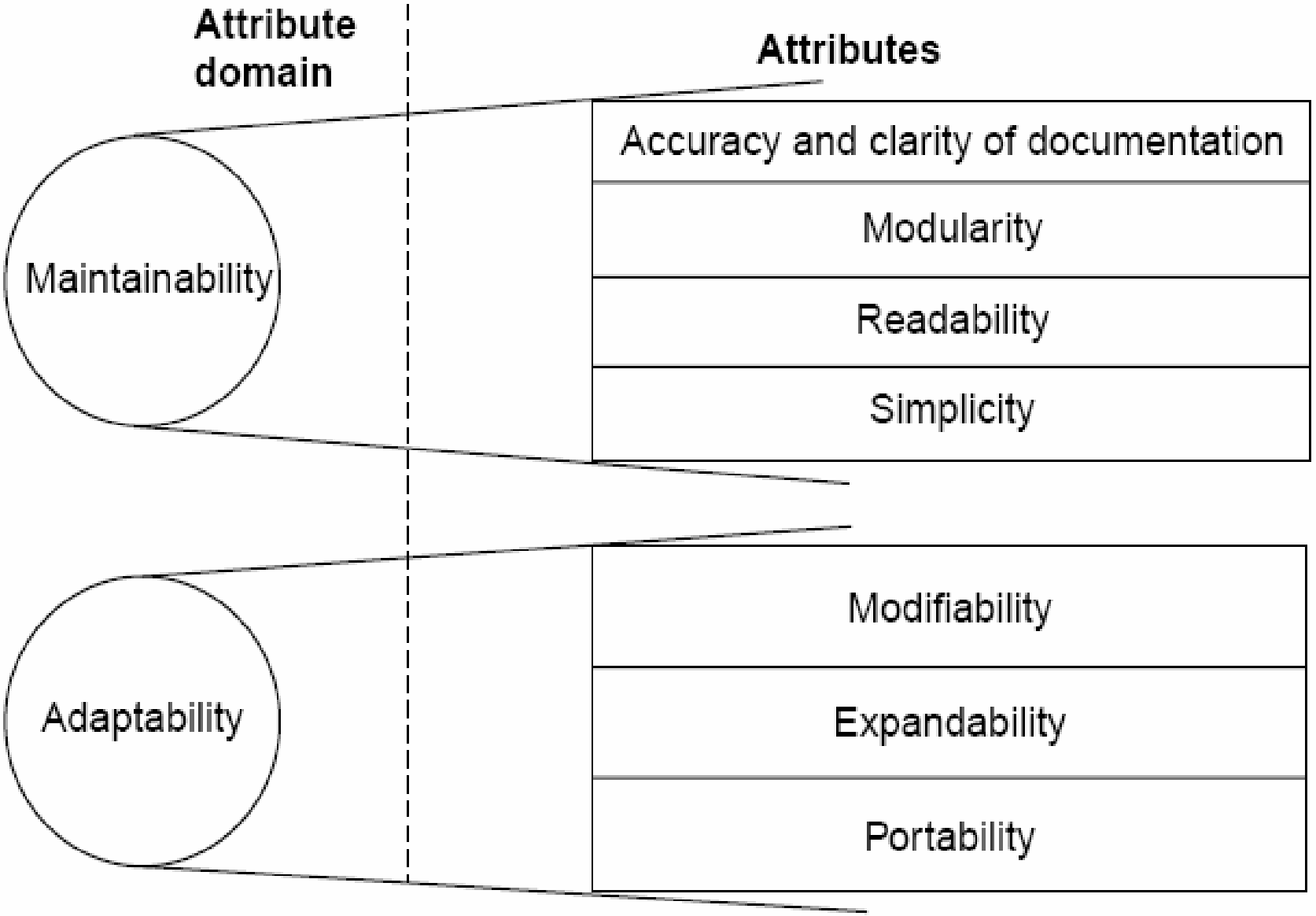


Fig 7.8: Software quality attributes

1	Reliability	The extent to which a software performs its intended functions without failure.
2	Correctness	The extent to which a software meets its specifications.
3	Consistency & precision	The extent to which a software is consistent and give results with precision.
4	Robustness	The extent to which a software tolerates the unexpected problems.
5	Simplicity	The extent to which a software is simple in its operations.
6	Traceability	The extent to which an error is traceable in order to fix it.
7	Usability	The extent of effort required to learn, operate and understand the functions of the software

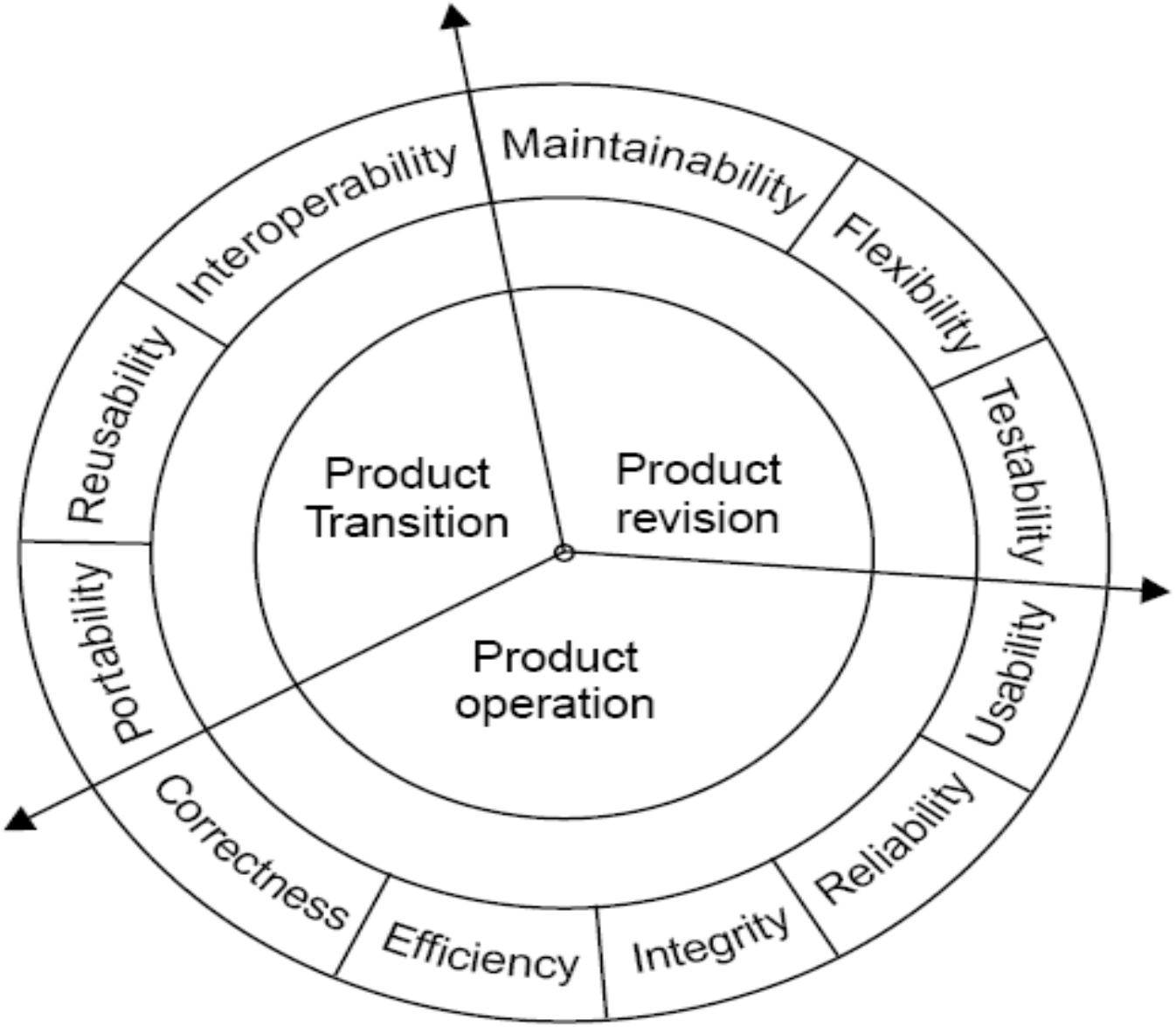
8	Accuracy	Meeting specifications with precision.
9	Clarity & Accuracy of documentation	The extent to which documents are clearly & accurately written.
10	Conformity of operational environment	The extent to which a software is in conformity of operational environment.
11	Completeness	The extent to which a software has specified functions.
12	Efficiency	The amount of computing resources and code required by software to perform a function.
13	Testability	The effort required to test a software to ensure that it performs its intended functions.
14	Maintainability	The effort required to locate and fix an error during maintenance phase.

15	Modularity	It is the extent of ease to implement, test, debug and maintain the software.
16	Readability	The extent to which a software is readable in order to understand.
17	Adaptability	The extent to which a software is adaptable to new platforms & technologies.
18	Modifiability	The effort required to modify a software during maintenance phase.
19	Expandability	The extent to which a software is expandable without undesirable side effects.
20	Portability	The effort required to transfer a program from one platform to another platform.

Table 7.4: Software quality attributes

- McCall Software Quality Model

Fig 7.9: Software quality factors



i. Product Operation

Factors which are related to the operation of a product are combined. The factors are:

- **Correctness**
- **Efficiency**
- **Integrity**
- **Reliability**
- **Usability**

These five factors are related to operational performance, convenience, ease of usage and its correctness. These factors play a very significant role in building customer's satisfaction.

ii. Product Revision

The factors which are required for testing & maintenance are combined and are given below:

- **Maintainability**
- **Flexibility**
- **Testability**

These factors pertain to the testing & maintainability of software. They give us idea about ease of maintenance, flexibility and testing effort. Hence, they are combined under the umbrella of product revision.

iii. Product Transition

We may have to transfer a product from one platform to an other platform or from one technology to another technology. The factors related to such a transfer are combined and given below:

- **Portability**
- **Reusability**
- **Interoperability**

Most of the quality factors are explained in table 7.4. The remaining factors are given in table 7.5.

Sr.No.	Quality Factors	Purpose
1	Integrity	The extent to which access to software or data by the unauthorized persons can be controlled.
2	Flexibility	The effort required to modify an operational program.
3	Reusability	The extent to which a program can be reused in other applications.
4	Interoperability	The effort required to couple one system with another.

Table 7.5: Remaining quality factors (other are in table 7.4)

Quality criteria

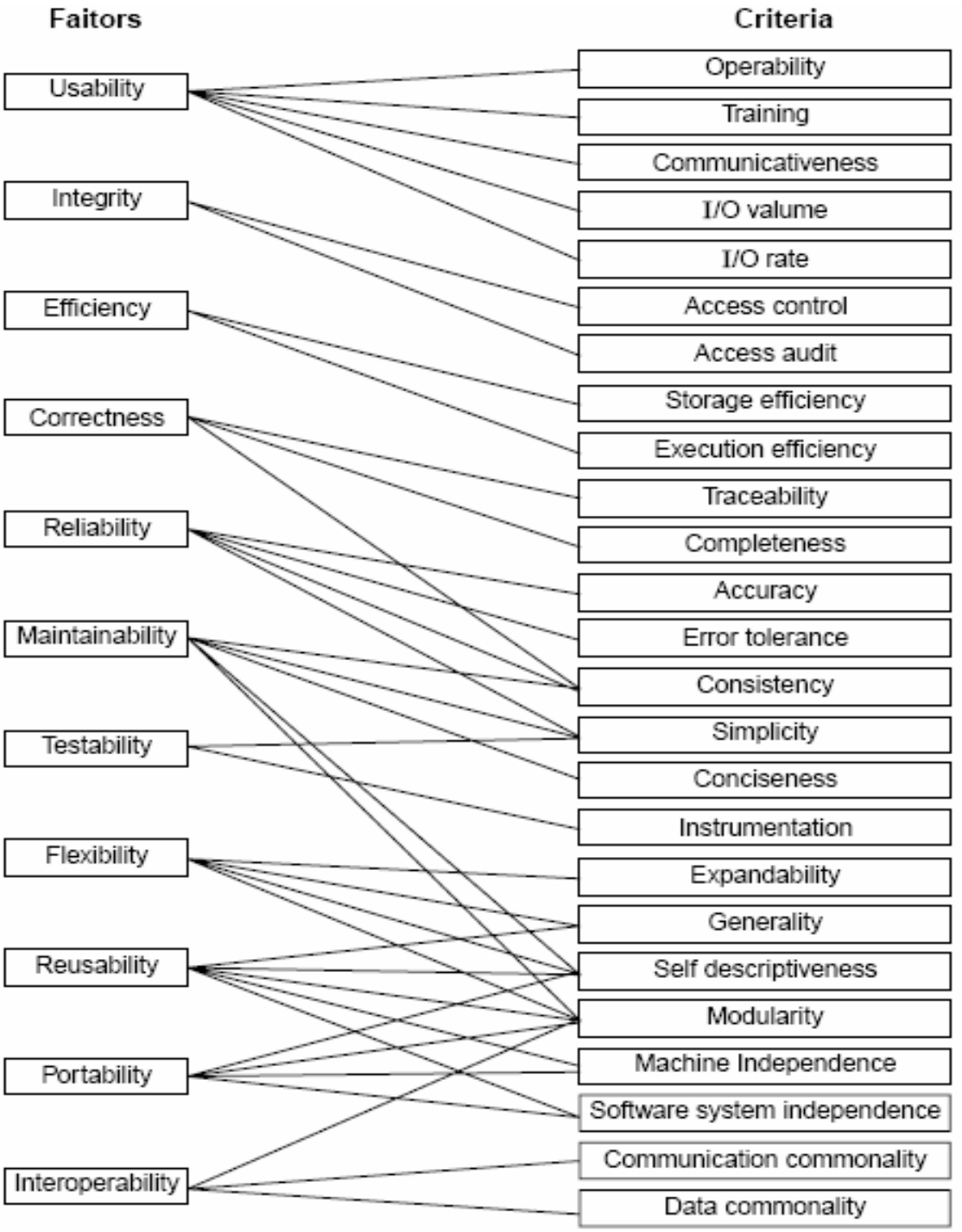


Fig 7.10: McCall's quality model

Sr. No.	*Quality Criteria*	*Usability*	*Integrity*	*Efficiency*	*Correctness*	*Reliability*	*Maintainability*	*Testability*	*Flexibility*	*Reusability*	*Portability*	*Interoperability*
1.	Operability	×										
2.	Training	×										
3.	Communicativeness	×										
4.	I/O volume	×										
5.	I/O rate	×										
6.	Access control		×									
7.	Access Audit		×									
8.	Storage efficiency			×								
9.	Execution Efficiency			×								
10.	Traceability				×							
11.	Completeness				×							
12.	Accuracy					×						
13.	Error tolerance					×						
14.	Consistency				×	×	×					
15.	Simplicity					×	×	×				
16.	Conciseness						×					
17.	Instrumentation							×				
18.	Expandability								×			
19.	Generality								×	×		
20.	Self-descriptiveness						×		×	×	×	
21.	Modularity						×		×	×	×	×
22.	Machine independence									×	×	
23.	S/W system independence									×	×	
24.	Communication commonality											×
25.	Data commonality											×

Table 7.5(a): Relation between quality factors and quality criteria

1	Operability	The ease of operation of the software.
2	Training	The ease with which new users can use the system.
3	Communicativeness	The ease with which inputs and outputs can be assimilated.
4	I/O volume	It is related to the I/O volume.
5	I/O rate	It is the indication of I/O rate.
6	Access control	The provisions for control and protection of the software and data.
7	Access audit	The ease with which software and data can be checked for compliance with standards or other requirements.
8	Storage efficiency	The run time storage requirements of the software.
9	Execution efficiency	The run-time efficiency of the software.

10	Traceability	The ability to link software components to requirements.
11	Completeness	The degree to which a full implementation of the required functionality has been achieved.
12	Accuracy	The precision of computations and output.
13	Error tolerance	The degree to which continuity of operation is ensured under adverse conditions.
14	Consistency	The use of uniform design and implementation techniques and notations throughout a project.
15	Simplicity	The ease with which the software can be understood.
16	Conciseness	The compactness of the source code, in terms of lines of code.
17	Instrumentation	The degree to which the software provides for measurements of its use or identification of errors.

18	Expandability	The degree to which storage requirements or software functions can be expanded.
19	Generability	The breadth of the potential application of software components.
20	Self-descriptiveness	The degree to which the documents are self explanatory.
21	Modularity	The provision of highly independent modules.
22	Machine independence	The degree to which software is dependent on its associated hardware.
23	Software system independence	The degree to which software is independent of its environment.
24	Communication commonality	The degree to which standard protocols and interfaces are used.
25	Data commonality	The use of standard data representations.

Table 7.5 (b): Software quality criteria

- Boehm Software Quality Model

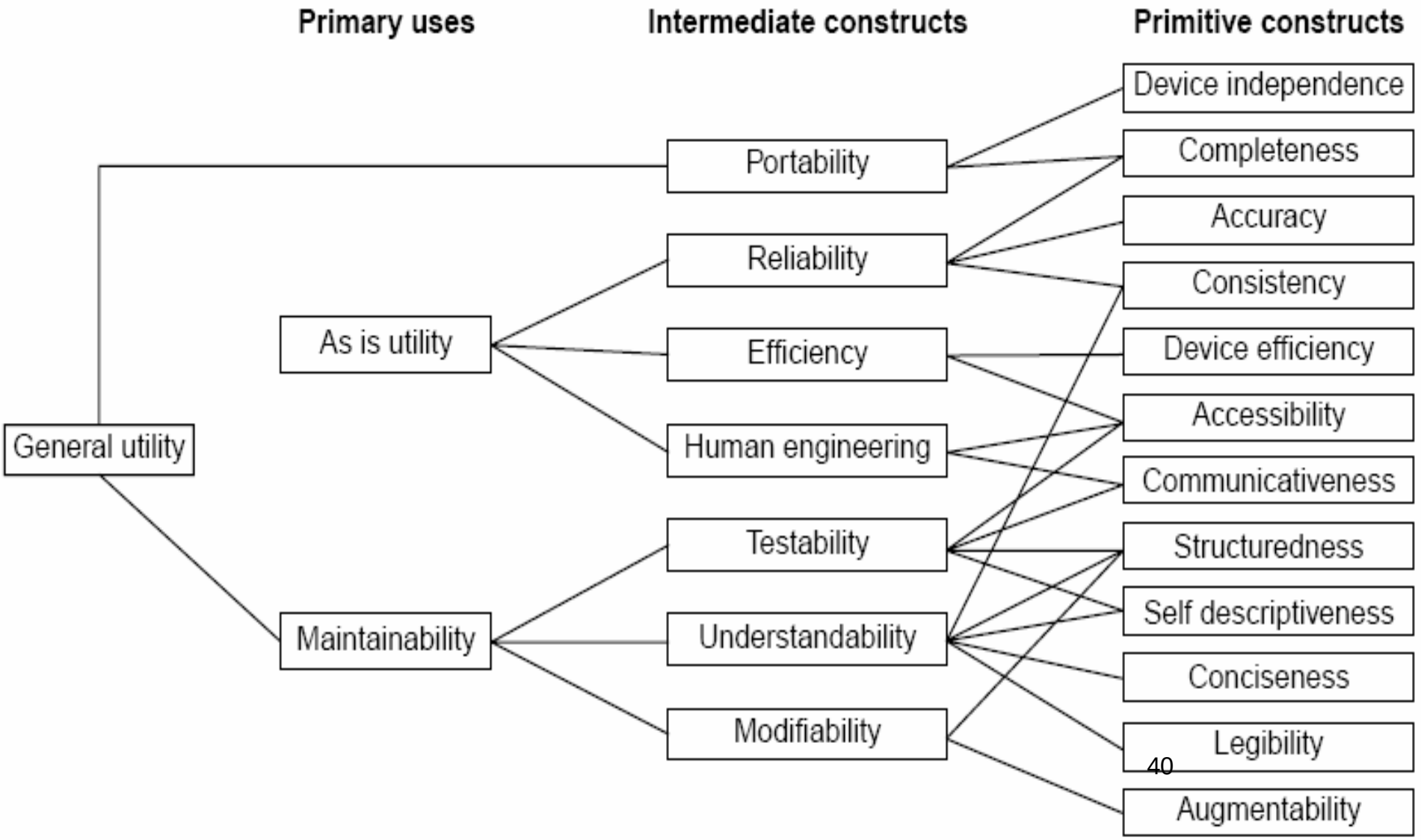


Fig.7.11: The Boehm software quality model

ISO 9126

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Characteristic/ Attribute	Short Description of the Characteristics and the concerns Addressed by Attributes
Functionality	Characteristics relating to achievement of the basic purpose for which the software is being engineered
• Suitability	The presence and appropriateness of a set of functions for specified tasks
• Accuracy	The provision of right or agreed results or effects
• Interoperability	Software's ability to interact with specified systems
• Security	Ability to prevent unauthorized access, whether accidental or deliberate, to program and data.
Reliability	Characteristics relating to capability of software to maintain its level of performance under stated conditions for a stated period of time
• Maturity	Attributes of software that bear on the frequency of failure by faults in the software

• Fault tolerance	Ability to maintain a specified level of performance in cases of software faults or unexpected inputs
• Recoverability	Capability and effort needed to reestablish level of performance and recover affected data after possible failure.
Usability	Characteristics relating to the effort needed for use, and on the individual assessment of such use, by a stated implied set of users.
• Understandability	The effort required for a user to recognize the logical concept and its applicability.
• Learnability	The effort required for a user to learn its application, operation, input and output.
• Operability	The ease of operation and control by users.
Efficiency	Characteristic related to the relationship between the level of performance of the software and the amount of resources used, under stated conditions.

• Time behavior	The speed of response and processing times and throughout rates in performing its function.
• Resource behavior	The amount of resources used and the duration of such use in performing its function.
Maintainability	Characteristics related to the effort needed to make modifications, including corrections, improvements or adaptation of software to changes in environment, requirements and functions specifications.
• Analyzability	The effort needed for diagnosis of deficiencies or causes of failures, or for identification of parts to be modified.
• Changeability	The effort needed for modification, fault removal or for environmental change.
• Stability	The risk of unexpected effect of modifications.
• Testability	The effort needed for validating the modified software.

Portability	Characteristics related to the ability to transfer the software from one organization or hardware or software environment to another.
• Adaptability	The opportunity for its adaptation to different specified environments.
• Installability	The effort needed to install the software in a specified environment.
• Conformance	The extent to which it adheres to standards or conventions relating to portability.
• Replaceability	The opportunity and effort of using it in the place of other software in a particular environment.

Table 7.6: Software quality characteristics and attributes – The ISO 9126 view

Fig.7.12: ISO 9126 quality model