Thakur Educational Trust's (Regd.)

THAKUR RAMNARAYAN COLLEGE OF ARTS & COMMERCE

ISO 21001:2018 Certified

PROGRAMME: B.Sc (I.T)

CLASS: S.Y.B.Sc (I.T)

SUBJECT NAME: SOFTWARE ENGINEERING

SEMESTER: IV

FACULTY NAME: Ms. SMRITI DUBEY

UNIT III

Chapter 4 – Quality Management

Concepts:

Introduction

Software quality product is defined in term of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document. Although "fitness of purpose" is a satisfactory interpretation of quality for many devices such as a car, a table fan, a grinding machine, etc. **For software products, "fitness of purpose" is not a wholly satisfactory definition of quality.**

Quality is meeting the requirement, expectation, and needs of the customer is free from the defects, lacks and substantial variants. There are standards needs to follow to satisfy the customer requirements**.**

Quality management techniques, in conjunction with new software technologies and better software testing, have led to significant improvements in the general level of software quality.

Software quality management for software systems has three principal concerns:

1.At the organizational level, quality management is concerned with establishing a framework of organizational processes and standards that will lead to high quality software. This means that the quality management team should take responsibility for defining the software development processes to be used and standards that should apply to the software and related documentation, including the system requirements, design, and code.

2. At the project level, quality management involves the application of specific quality processes, checking that these planned processes have been followed, and ensuring that the project outputs are conformant with the standards that are applicable to that project.

3. Quality management at the project level is also concerned with 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.

The terms 'quality assurance' and 'quality control' are widely used in manufacturing industry.

Quality assurance (QA) is the definition of processes and standards that should lead to high-quality products and the introduction of quality processes into the manufacturing process.

Quality Assurance is known as QA and focuses on preventing defect. Quality Assurance ensures that the approaches, techniques, methods and processes are designed for the projects are implemented correctly.

Quality assurance activities monitor and verify that the processes used to manage and create the deliverables have been followed and are operative.

Quality Assurance is a proactive process and is Prevention in nature. It recognizes flaws in the process. Quality Assurance has to complete before Quality Control.

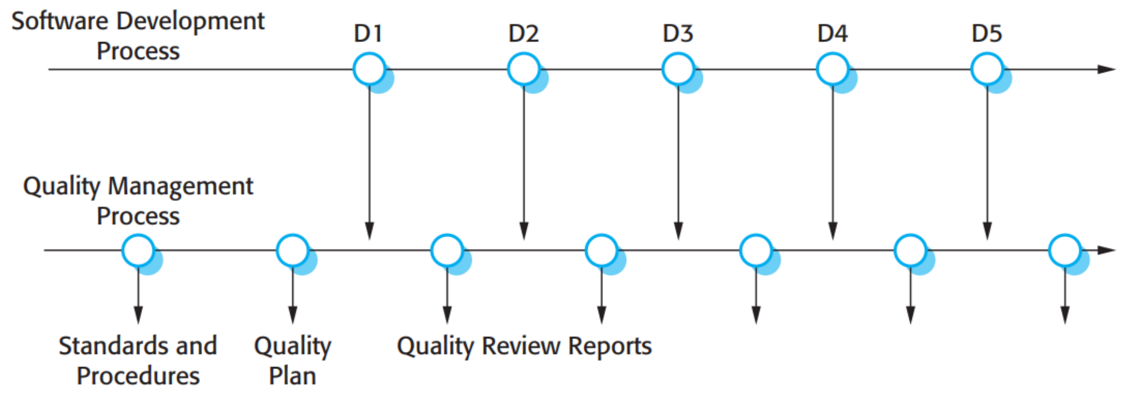
Quality control is the application of these quality processes to weed out products that are not of the required level of quality.

Quality Control is known as QC and focuses on identifying a defect. QC ensures that the approaches, techniques, methods and processes are designed in the project are following correctly. QC activities monitor and verify that the project deliverables meet the defined quality standards.

Quality Control is a reactive process and is detection in nature. It recognizes the defects. Quality Control has to complete after Quality Assurance.

The QA team in most companies is responsible for managing the release testing process. They are responsible for checking that the system tests provide coverage of the requirements and that proper records are maintained of the testing process.

Quality planning is the process of developing a quality plan for a project. The quality plan should set out the desired software qualities and describe how these are to be assessed. It therefore defines what 'high-quality' software actually means for a particular system



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 **(Figure**). The QA 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.

Ideally, the quality management team should not be associated with any particular development group, but should rather have organization-wide responsibility for quality management. They should be independent and report to management above the project manager level. The reason for this is that project managers have to maintain the project budget and schedule. If problems arise, they may be tempted to compromise on product quality so that they meet their schedule. An independent quality management team ensures that the organizational goals of quality are not compromised by short-term budget and schedule considerations.

Process and Product Quality Management

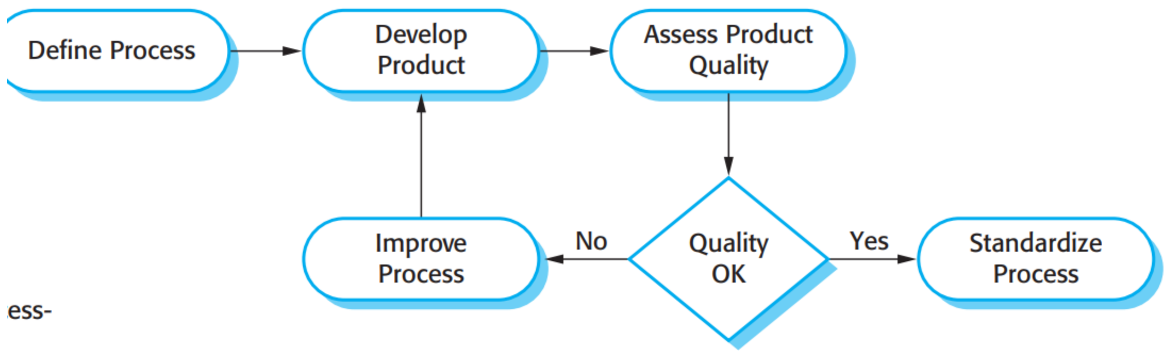
An assumption that underlies software quality management is that the quality of software is directly related to the quality of the software development process. This again comes from manufacturing systems where product quality is intimately related to the production process.

A manufacturing process involves configuring, setting up, and operating the machines involved in the process. Once the machines are operating correctly, product quality naturally follows. You measure the quality of the product and change the process until you achieve the quality level that you need. **Figure** illustrates this process-based approach to achieving product quality.

There is a clear link between process and product quality in manufacturing because the process is relatively easy to standardize and monitor. However, software is not manufactured—it is designed. In software development, therefore, the relationship between process quality and product quality is more complex. Software development is a creative rather than a mechanical process, so the influence of individual skills and experience is significant.

There is no doubt that the development process used has a significant influence on the quality of the software and that good processes are more likely to lead to good quality software.

Process quality management and improvement can lead to fewer defects in the software being developed. However, it is difficult to assess software quality attributes, such as maintainability, without using the software for a long period. Process quality management involves:



1. Defining process standards such as how and when reviews should be conducted

2.Monitoring the development process to ensure that the standards are being followed.

3.Reporting the software process to project management and to the buyer of the software.

Software standards

Software standards play a very important role in software quality management. An important part of quality assurance is the definition or selection of standards that should apply to the software development process or software product. Once standards have been selected for use, project-specific processes have to be defined to monitor the use of the standards and check that they have been followed.

Software standards are important for three reasons:

1. Standards capture wisdom that is of value to the organization. They are based on knowledge about the best or most appropriate practice for the company. This knowledge is often only acquired after a great deal of trial and error. Building it into a standard helps the company reuse this experience and avoid previous mistakes.

2. Standards provide a framework for defining what 'quality' means in a particular setting. Software quality is subjective, and by using standards you establish a basis for deciding if a required level of quality has been achieved. Of course, this depends on setting standards that reflect user expectations for software dependability, usability, and performance.

3. Standards assist continuity when work carried out by one person is taken up and continued by another. Standards ensure that all engineers within an organization adopt the same practices. Consequently, the learning effort required when starting new work is reduced.

There are two related types of software engineering standard that may be defined and used in software quality management:

1. Product standards: These apply to the software product being developed. They include document standards, such as the structure of requirements documents, documentation standards, such as a standard comment header for an object class definition, and coding standards, which define how a programming language should be used.

2. Process standards: These define the processes that should be followed during software development. They should encapsulate good development practice. Process standards may include definitions of specification, design and validation processes, process support tools, and a description of the documents that should be written during these processes.

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

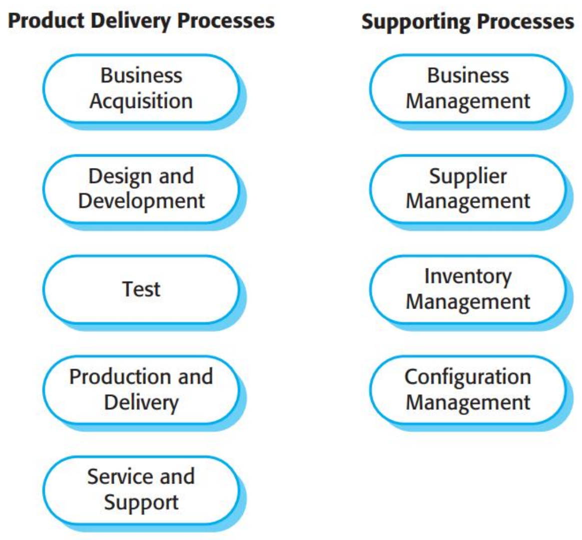
Standards have to deliver value, in the form of increased product quality. There is no point in defining standards that are expensive in terms of time and effort to apply that only lead to marginal improvements in quality. **Product standards have to be designed so that they can be applied and checked in a cost-effective way**, and **process standards should include the definition of processes that check that product standards have been followed.**

The ISO 9001 standards framework

There is an international set of standards that can be used in the development of quality management systems in all industries, called ISO 9000. ISO 9000 standards can be applied to a range of organizations from manufacturing through to service industries. ISO 9001, the most general of these standards, applies to organizations that design, develop, and maintain products, including software. The ISO 9001 standard was originally developed in 1987, with its most recent revision in 2008.

The ISO 9001 standard is not itself a standard for software development but 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.

The major revision of the ISO 9001 standard in 2000 reoriented the standard around nine core processes (Figure). If an organization is to be ISO 9001 conformant, it must document how its processes relate to these core processes. It must also define and maintain records that demonstrate that the defined organizational processes have been followed. The company quality manual should describe the relevant processes and the process data that has to be collected and maintained.



The ISO 9001 standard does not define or prescribe the specific quality processes that should be used in a company. To be conformant with ISO 9001, a company must have defined the types of process shown in **Figure** and have procedures in place that demonstrate that its quality processes are being followed. This allows flexibility across industrial sectors and company sizes. Quality standards can be defined that are appropriate for the type of software being developed.

Structure of Quality Planning

Quality planning is the process of developing a quality plan for a project. The quality plan should set out the desired software qualities and describe how these are to be assessed. It therefore defines what 'high-quality' software actually means for a particular system.

An outline structure for a quality plan. This includes:

1. Product introduction: A description of the product, its intended market, and the quality expectations for the product.

2. Product plans: The critical release dates and responsibilities for the product, along with plans for distribution and product servicing

3. Process descriptions: The development and service processes and standards that should be used for product development and management.

4. Quality goals: The quality goals and plans for the product, including an identification and justification of critical product quality attributes.

5. Risks and risk management: The key risks that might affect product quality and the actions to be taken to address these risks.

Reviews and inspections

Reviews and inspections are QA activities that check the quality of project deliverables. This involves examining the software, its documentation and records of the process to discover errors and omissions and to see if quality standards have been followed.

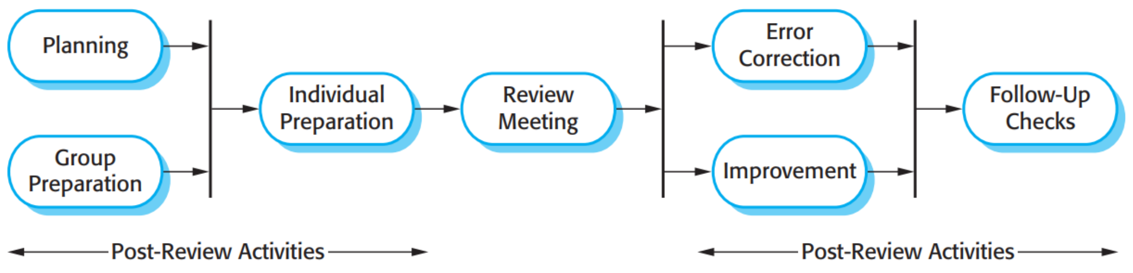
During a review, a group of people examine the software and its associated documentation, looking for potential problems and non-conformance with standards. The review team makes informed judgments about the level of quality of a system or project deliverable. Project managers may then use these assessments to make planning decisions and allocate resources to the development process.

Quality reviews are based on documents that have been produced during the software development process. As well as software specifications, designs, or code, process models, test plans, configuration management procedures, process standards, and user manuals may all be reviewed. The review should check the consistency and completeness of the documents or code under review and make sure that quality standards have been followed.

The purpose of reviews and inspections is to improve software quality, not to assess the performance of people in the development team. Reviewing is a public process of error detection, compared with the more private component-testing process. A quality review provides information for management about the software being developed, quality reviews are not the same as management progress reviews.

The review process

Although there are many variations in the details of reviews, the review process (Figure) is normally structured into three phases:



1. Pre-review activities: These are preparatory activities that are essential for the review to be effective. Typically, pre-review activities are concerned with review planning and review preparation. Review planning involves setting up a review team, arranging a time and place for the review, and distributing the documents to be reviewed. During review preparation, the team may meet to get an overview of the software to be reviewed. Individual review team members read and understand the software or documents and relevant standards. They work independently to find errors, omissions, and departures from standards. Reviewers may supply written comments on the software if they cannot attend the review meeting.

2. The review meeting: During the review meeting, an author of the document or program being reviewed should 'walk through' the document with the review team. The review itself should be relatively short—two hours at most. One team member should chair the review and another should formally record all review decisions and actions to be taken. During the review, the chair is responsible for ensuring that all written comments are considered. The review chair should sign a record of comments and actions agreed during the review.

3. Post-review activities: After a review meeting has finished, the issues and problems raised during the review must be addressed. This may involve fixing software bugs, refactoring software so that it conforms to quality standards, or rewriting documents. Sometimes, the problems discovered in a quality review are such that a management review is also necessary to decide if more resources should be made available to correct them. After changes have been made, the review chair may check that the review comments have all been taken into account. Sometimes, a further review will be required to check that the changes made cover all of the previous review comments.

Review teams should normally have a core of three to four people who are selected as principal reviewers. One member should be a senior designer who will take the responsibility for making significant technical decisions. The principal reviewers may invite other project members, such as the designers of related subsystems, to contribute to the review. They may not be involved in reviewing the whole document but should concentrate on those sections that affect their work.

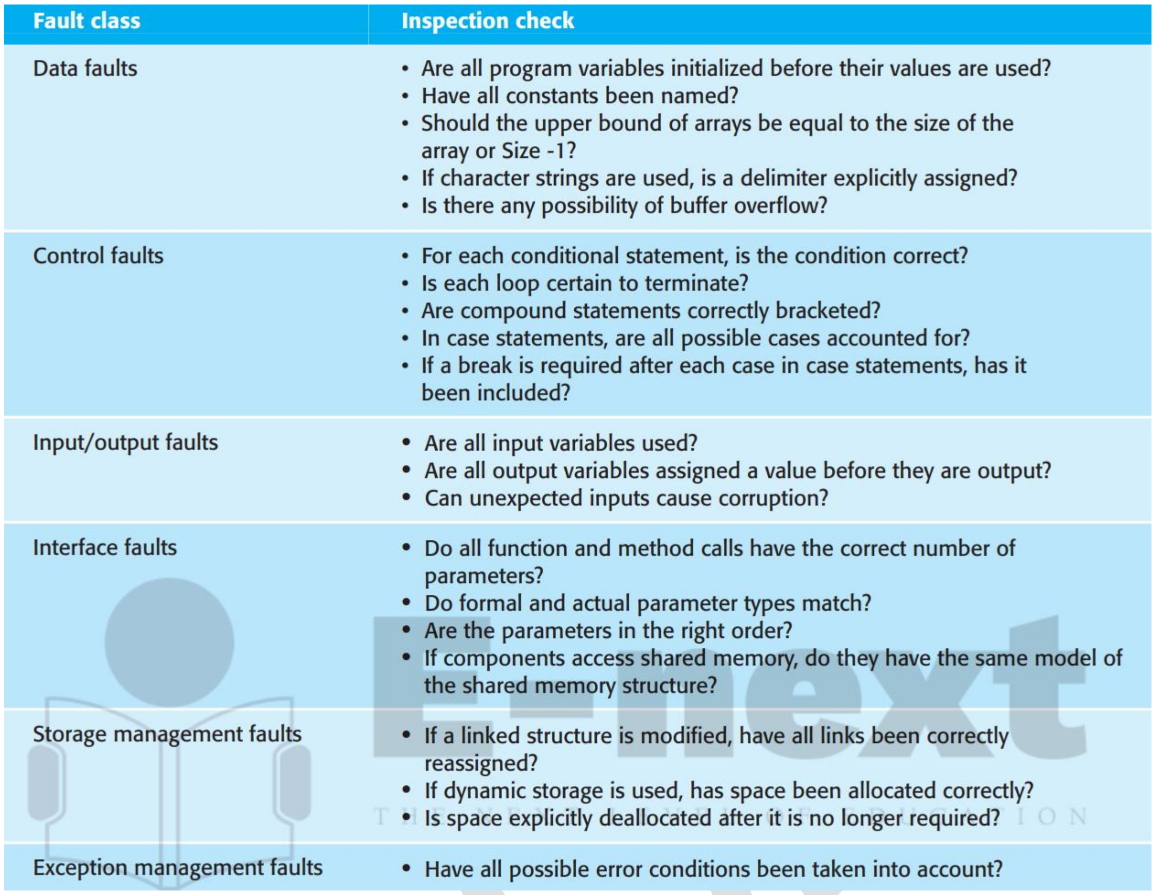
Program inspections

Program inspections are ‘peer reviews’ where team members collaborate to find
bugs in the program that is being developed. They complement testing as they do not require the program to be executed. This means that incomplete versions of the system can be verified and that representations such as UML models can be checked. Gilb and Graham (1993) suggest that one of the most effective ways to use inspections is to review the test cases for a system.

Inspections can discover problems with tests and so improve the effectiveness of these tests in detecting program bugs. Program inspections involve team members from different backgrounds who make a careful, line-by-line review of the program source code. They look for defects and problems and describe these at an inspection meeting. Defects may be logical errors, anomalies in the code that might indicate an erroneous condition or features that have been omitted from the code. The review team examines the design models or the program code in detail and highlights anomalies and problems for repair.

During an inspection, a checklist of common programming errors is often used to focus the search for bugs. This checklist may be based on examples from books or from knowledge of defects that are common in a particular application domain. You use different checklists for different programming languages because each language has its own characteristic errors.

Possible checks that might be made during the inspection process are shown in **Figure** .Gilb and Graham (1993) emphasize that each organization should develop its own inspection checklist based on local standards and practices. These checklists should be regularly updated, as new types of defects are found. The items in the checklist vary according to programming language because of the different levels of checking that are possible at compile-time. For example, a Java compiler checks that functions have the correct number of parameters; a C compiler does not.



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?
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?

Software measurement and metrics

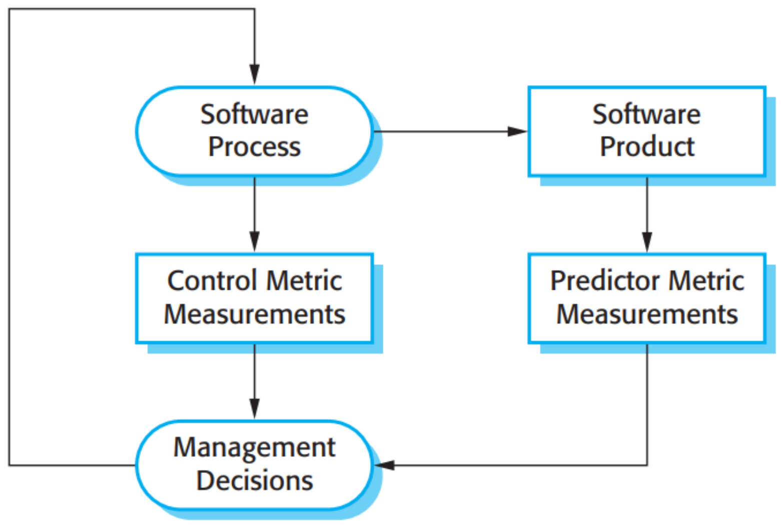
Software measurement is concerned with deriving a numeric value or profile for an attribute of a software component, system, or process. By comparing these values to each other and to the standards that apply across an organization, you may be able to draw conclusions about the quality of software, or assess the effectiveness of software processes, tools, and methods.

For example, say an organization intends to introduce a new software-testing tool. Before introducing the tool, you record the number of software defects discovered in a given time. This is a baseline for assessing the effectiveness of the tool. After using the tool for some time, you repeat this process. If more defects have been found in the same amount of time, after the tool has been introduced, then you may decide that it provides useful support for the software validation process.

The long-term goal of software measurement is to use measurement in place of reviews to make judgments about software quality. Using software measurement, a system could ideally be assessed using a range of metrics and, from these measurements, a value for the quality of the system could be inferred. If the software had reached a required quality threshold, then it could be approved without review. When appropriate, the measurement tools might also highlight areas of the software that could be improved.

A software metric is a characteristic of a software system, system documentation, or development process that can be objectively measured. Examples of metrics include the size of a product in lines of code; the Fog index (Gunning, 1962), which is a measure of the readability of a passage of written text; the number of reported faults in a delivered software product; and the number of person-days required to develop a system component.

Software metrics may be either control metrics or predictor metrics. As the names imply, control metrics support process management, and predictor metrics help you predict characteristics of the software. Control metrics are usually associated with software processes. Examples of control or process metrics are the average effort and the time required to repair reported defects.



Both control and predictor metrics may influence management decision making, as shown in **Figure**. Managers use process measurements to decide if process changes should be made, and predictor metrics to help estimate the effort required to make software changes. Predictor metrics, whose values are assessed by analyzing the code of a software system

There are two ways in which measurements of a software system may be used:

1. **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.

2. **To identify the system components** whose quality is substandard 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.

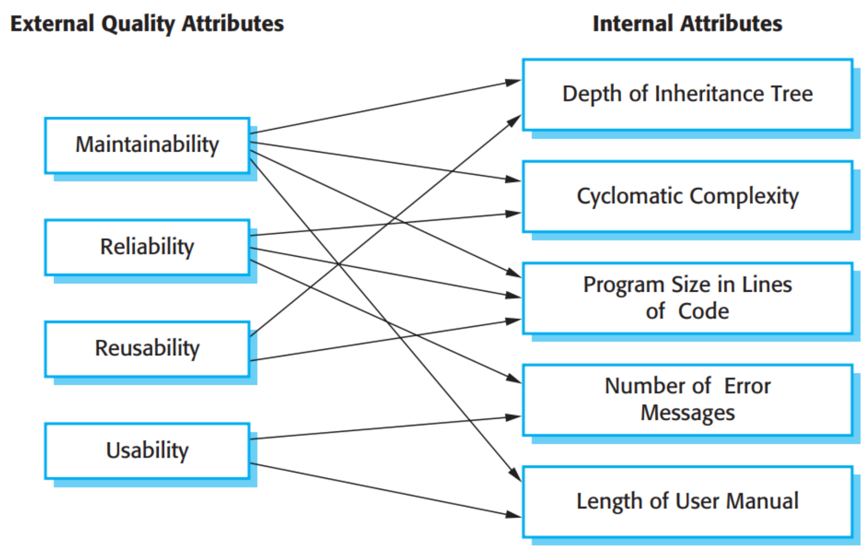


Figure shows some external software quality attributes and internal attributes that could, intuitively, be related to them. The diagram suggests that there may be relationships between external and internal attributes, but it does not say how these attributes are related. If the measure of the internal attribute is to be a useful predictor of the external software characteristic, three conditions must hold (Kitchenham, 1990):

1. The internal attribute must be measured accurately. This is not always straight forward and it may require special-purpose tools to make the measurements.

2. A relationship must exist between the attribute that can be measured and the external quality attribute that is of interest. That is, the value of the quality attribute must be related, in some way, to the value of the attribute than can be
measured.

3. This relationship between the internal and external attributes must be understood, validated, and expressed in terms of a formula or model. Model formulation involves identifying the functional form of the model (linear, exponential, etc.) by analysis of collected data, identifying the parameters that are to be included in the model, and calibrating these parameters using existing data.