Integration Testing

- It combines all unit-tested modules and performs a test on their aggregation
- unit modules are not independent, and are related to each other by interface specifi cations
- Interfacing between units must be tested.
- It is actually a systematic technique for combining modules.
- We integrate the units according to the design and availability of units.
- Tester must be aware of the system design.

Integration Testing

- It exposes inconsistency between the modules such as
- Data can be lost across an interface.
- module Integration may not give the desired result.
- Data types and their valid ranges may mismatch between the modules.

Integration Testing

- Nodes represent the modules present in the system
- Links/edges between the two modules
- The nodes on the last level in the tree are *leaf nodes*
- integrate all the modules together and then test it. (Non Incremental)
- Another method is to integrate the modules one by one and test them incrementally.
- Based on these methods, integration testing methods are classified into two categories:
- (a) non-incremental
- (b) incremental.

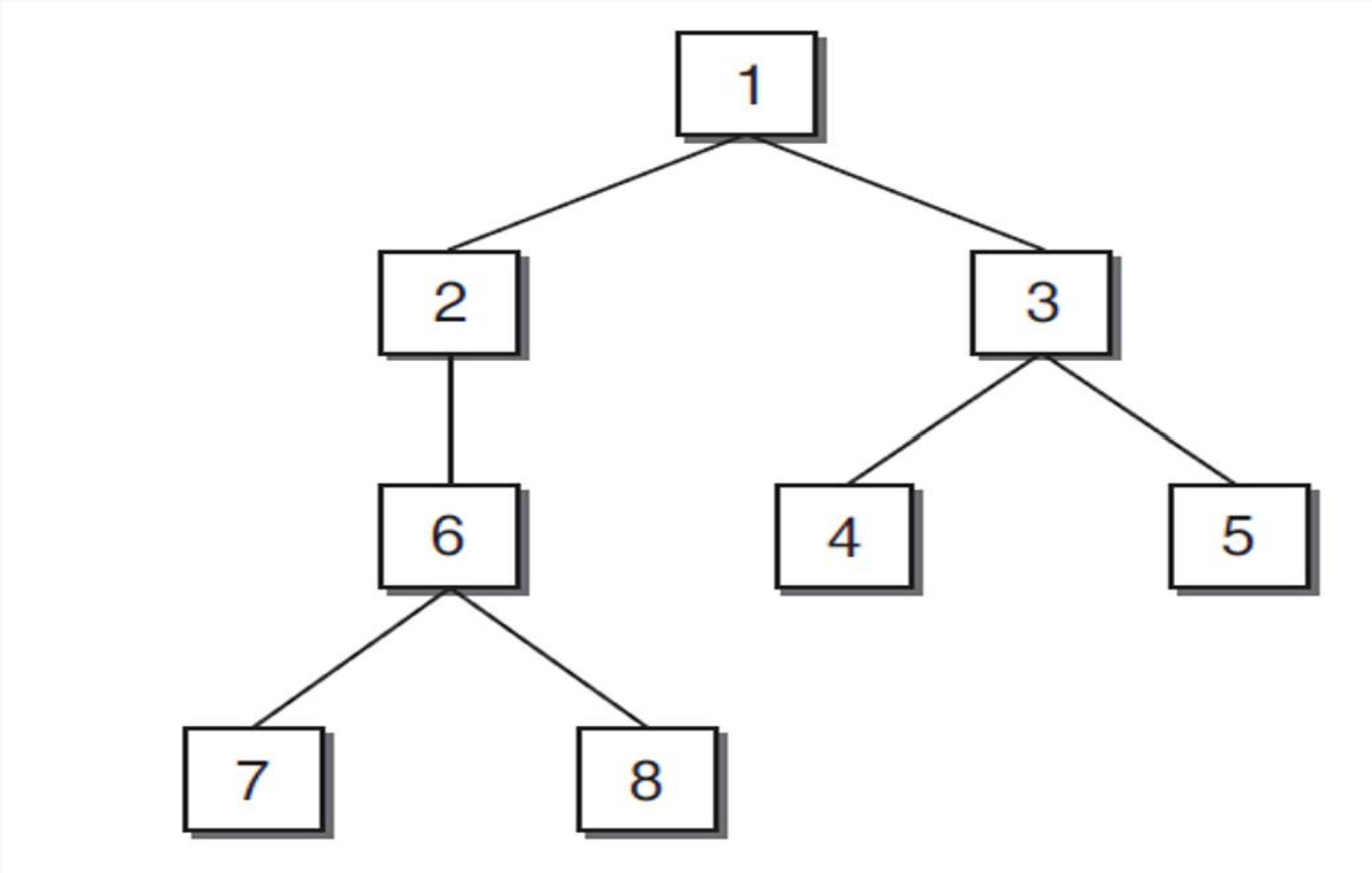
Non-incremental Integration Testing : Big-bang Integration Testing

- untested modules are combined together and then tested are combined together. It is also known as Big-Bang integration testing

Drawbacks

- Big-Bang method cannot be adopted practically
- Big-Bang requires more work.

Non-incremental Integration Testing : Big-bang Integration Testing



Incremental Integration Testing

- Start with one module and unit test it.
- Then combine with another module
- modules are combined one by one
- perform test on both the modules.
- incrementally keep on adding the modules
- It test the recent environment.
- Thus, an integrated tested software system is achieved

Types Of Incremental Integration Testing

Incremental Integration Testing Is Divided Into Two Categories.

- **1. Top-down Integration Testing**
- A. ***Depth First Integration***
- *B* ***Breadth First Integration***
- **2. Bottom-up Integration Testing**

Top-down Integration Testing

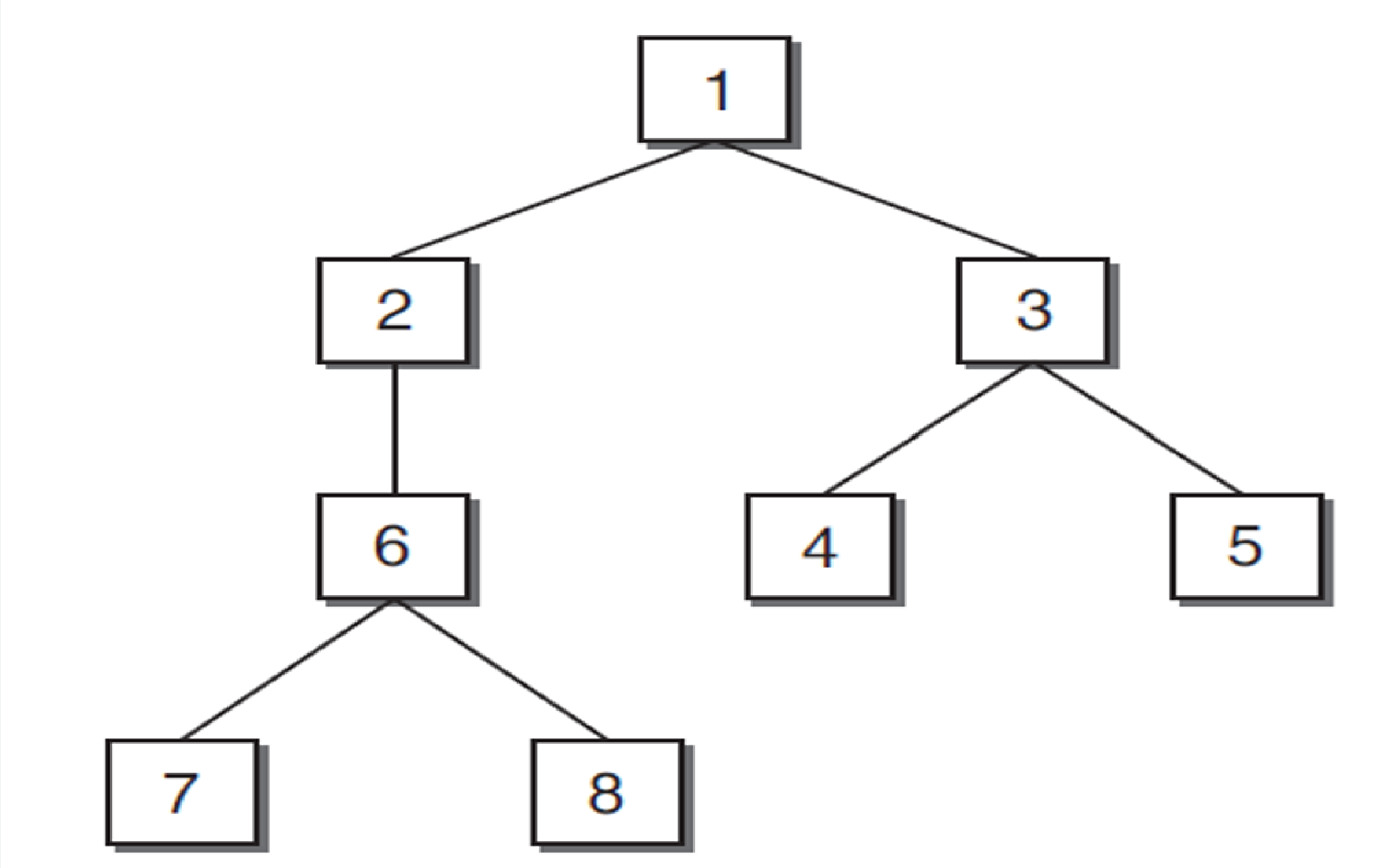
- Start with the top or initial module in the software.
- Substitute the stubs for all the subordinate modules of top module. Test the top module.
- After testing the top module, stubs are replaced one at a time with the actual modules for integration.
- Perform testing on this recent integrated environment.
- Look at the design hierarchy from top to bottom.
- Start with the high-level modules and move downward through the design hierarchy

Top-down Integration Testing : Depth First Integration

- Modules subordinate to the top module are integrated in the following two ways:
- ***Depth fi rst integration*** ***1*** 2, 6, 7/8 will be integrated fi rst.

  Next, modules 1, 3, 4/5 will be integrated

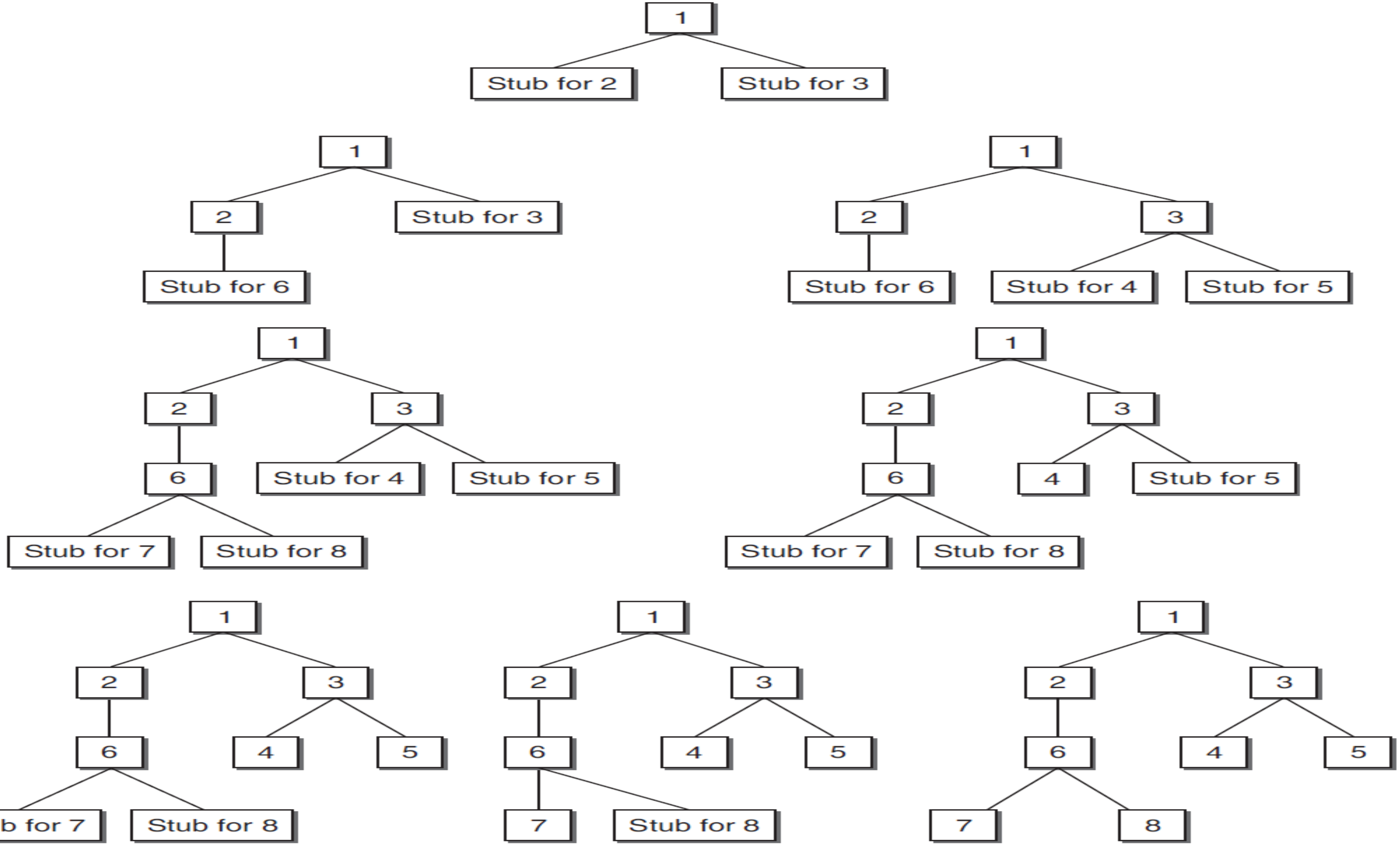
Top-down Integration Testing : Depth First Integration



Top-down Integration Testing : Breadth First Integration

- All modules directly subordinate at each level, moving across the design hierarchy horizontally, are integrated fi rst
- modules 2 and 3 will be integrated fi rst. Next, modules 6, 4, and 5 will be integrated. Modules 7 and 8 will be integrated last.

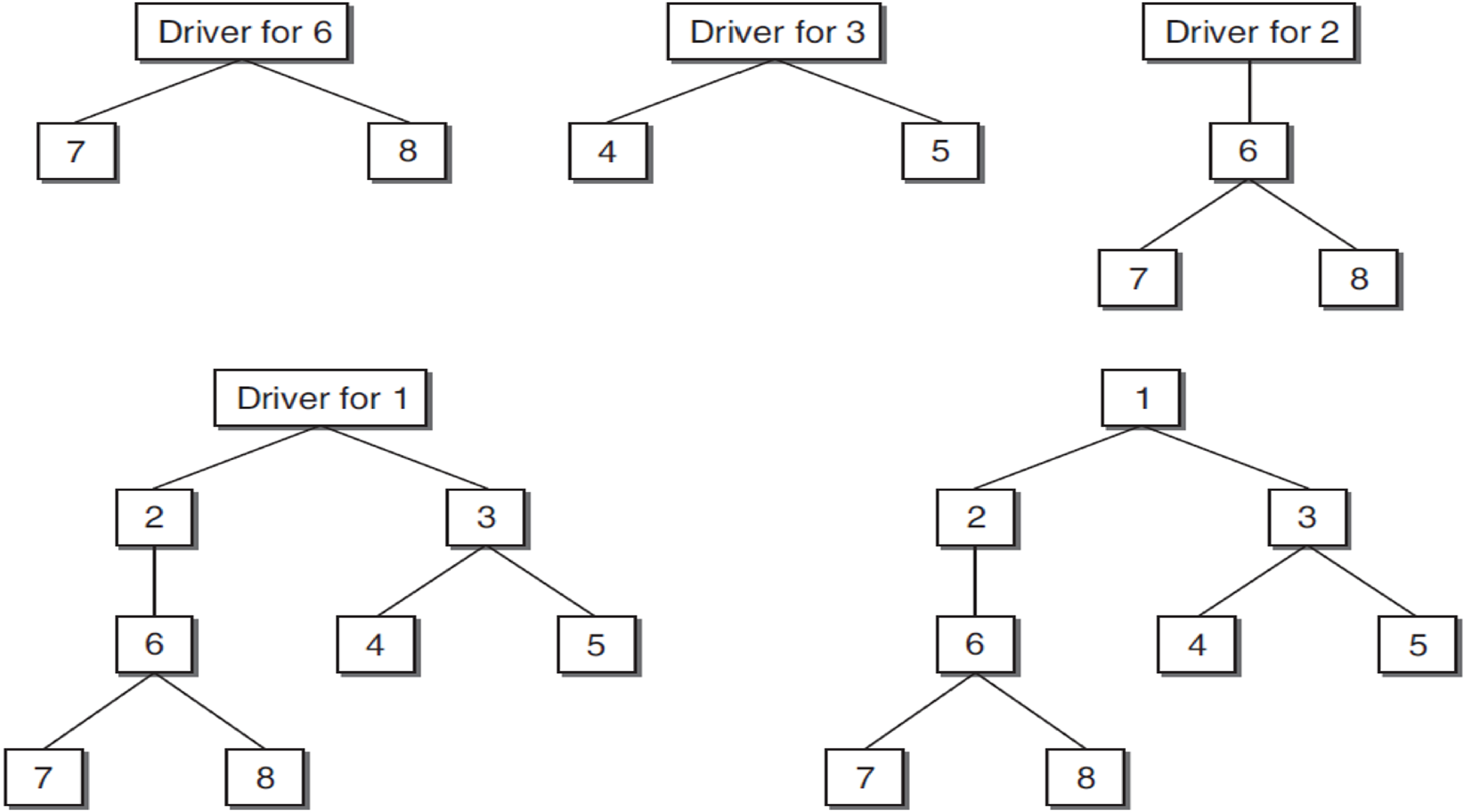
Top-down Integration Testing : Breadth First Integration



Bottom-up Integration Testing

- Begins with the modules at the lowest level in the software structure.
- After testing these modules, they are integrated and tested moving from bottom to top level
- Bottom-up integration can be considered as the opposite of top-down approach
- Start with the lowest level modules in the design hierarchy.
- Look for the super-ordinate
- Design the driver module for this super-ordinate module.
- Test the module selected in step 1 with the driver designed in step 2.
- Repeat steps and move up in the design hierarchy.
- Whenever, the actual modules are available, replace stubs and drivers with the actual one and test again.

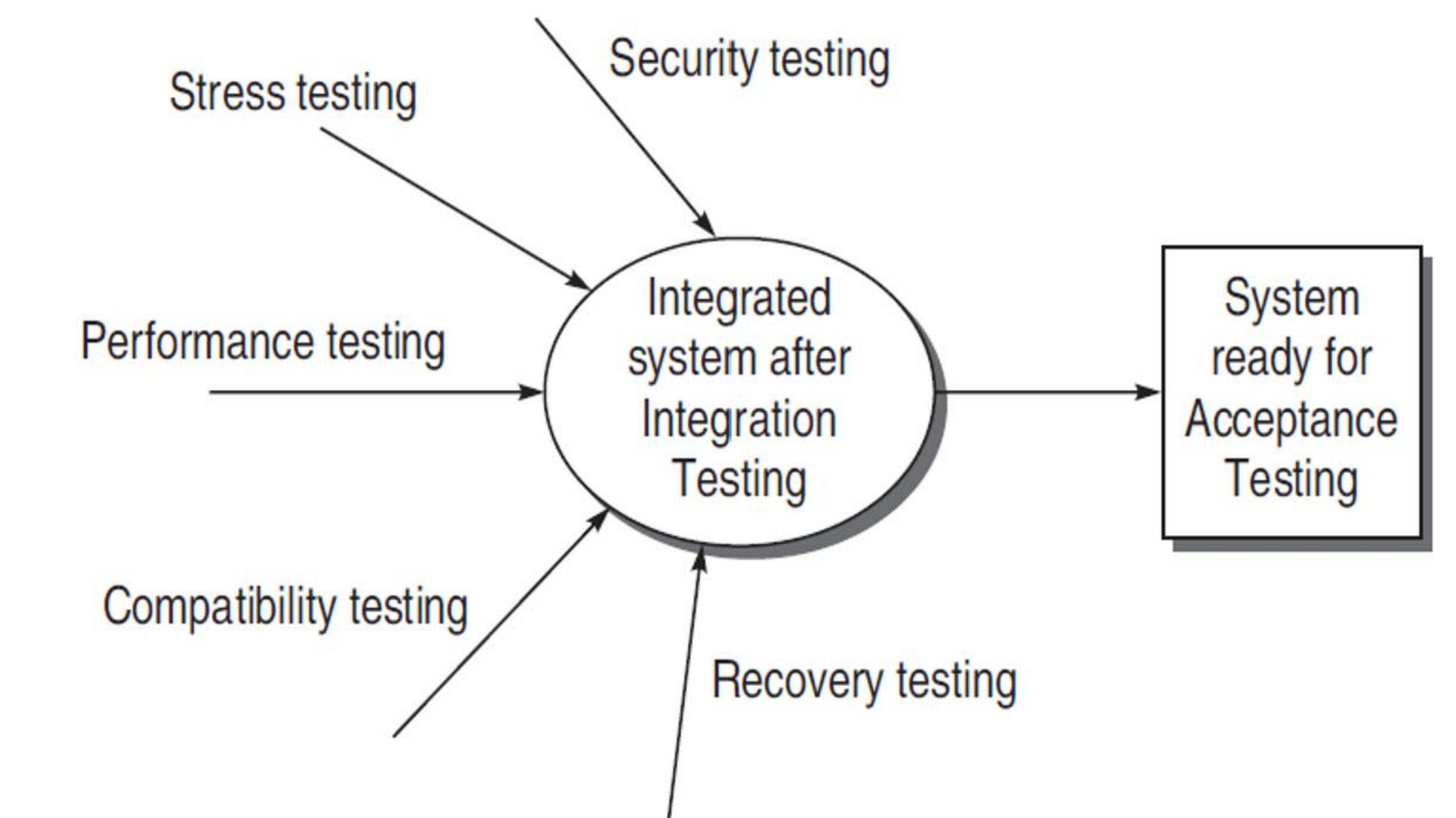
Bottom-up Integration Testing



System Testing

- Test the whole system on various grounds
- The ground can be performance, security, maximum load etc
- It checks if program or system does not meet its original requirements an objectives as stated in the requirement specification.
- It focuses on testing the entire integrated system.

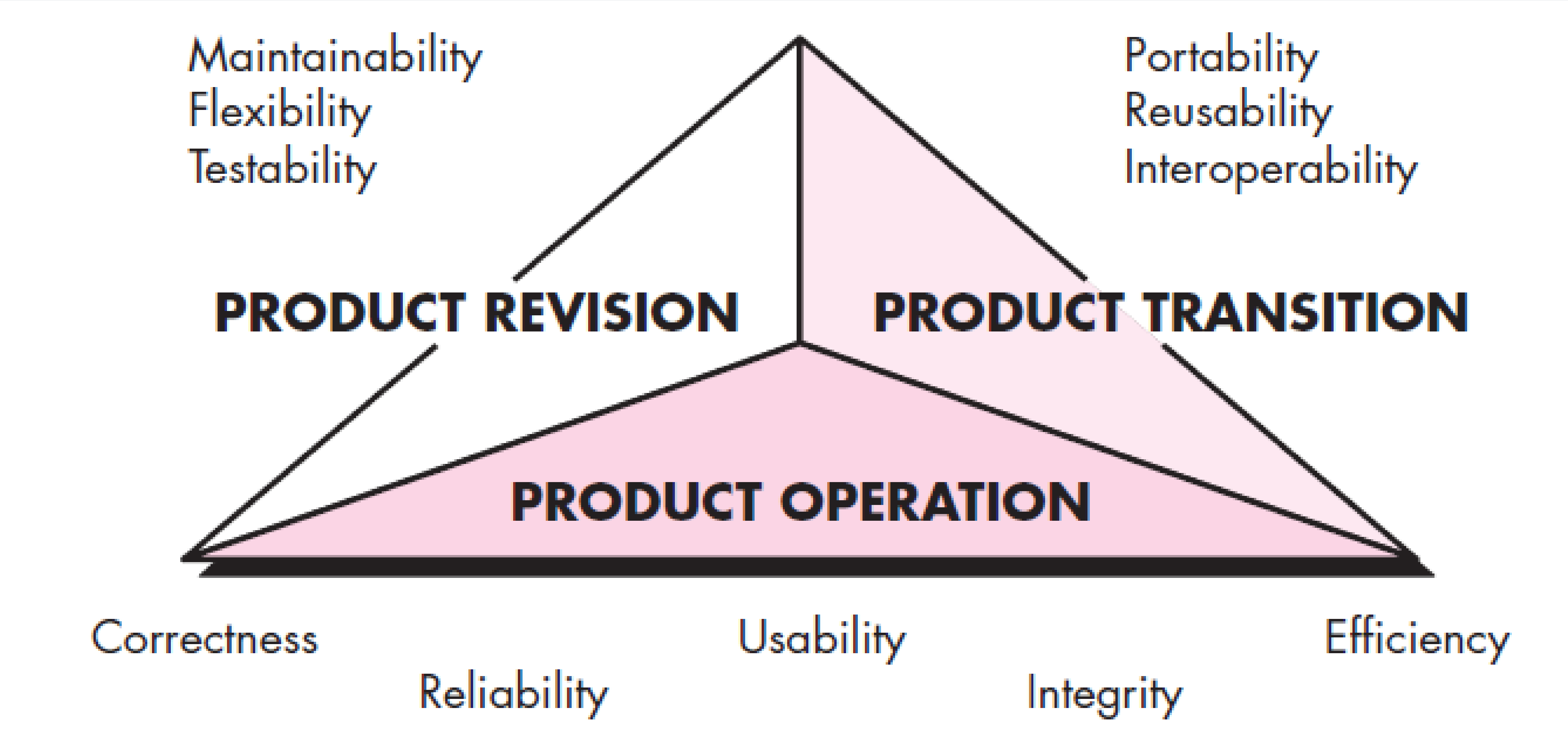
System Testing



SOFTWARE QUALITY MODELS: MCCALL'S QUALITY FACTORS

- McCall software quality model was introduced in 1977.
- It incorporates many attributes
- These software quality factors, as shown in the above figure, here focus on three important aspects
- software product operational characteristics
- It includes requirements that directly affect the operation of the software
- These factors help in providing a better user experience.
- Product Revision Factors: It includes software quality factors, which are required for testing and maintenance of the software.
- Product Transition Factors: It includes three software quality factors, that allows the software to adapt to the change of environments

Software Quality Models: MCcall's Quality Factors

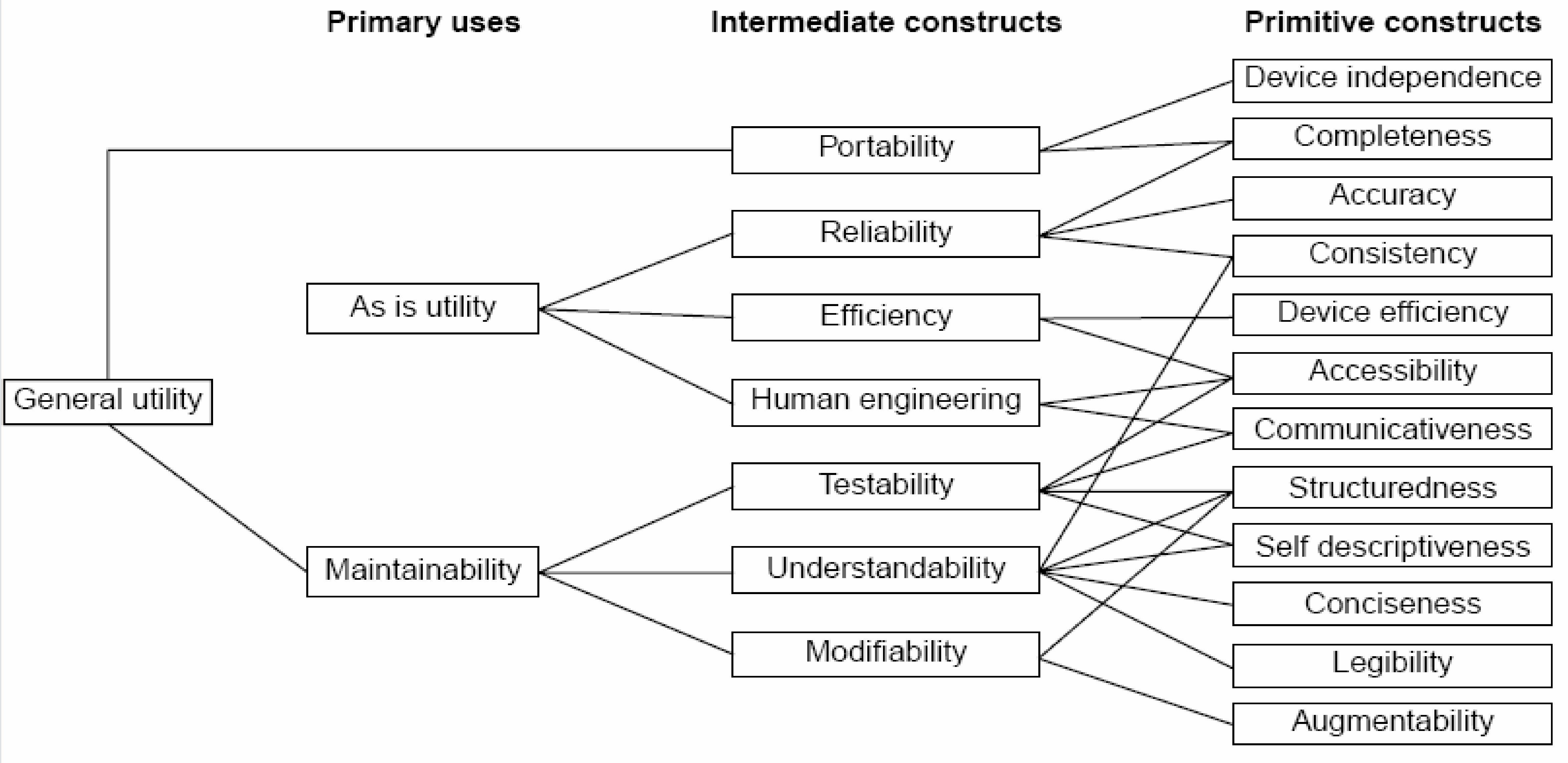


QUALITY FACTORS		
Product operation	Correctness	Ability To Fulfill Specification And Customer Requirements
	Reliability	Degree By Which Software Should Work As Per Requirement Precision
	Usability	Ability To Prepare The Valid Input And Interpret Correct Output
	Efficiency	Measure Of Computing Resources And Time Required By Program
	Integrity	Controlling Ability By Which Unauthorized Acces Can Be Prevented
Product Revision	Maintainability	Ability Required To Fix Bugs In Software
	Flexibility	Extent By Which It Is Allowed To Make Certain Modification In Program
	Testability	Ability To Check Function Is Working As Per Requirement
Product transition	Reusability	Ability For Software Component Can Be Reused By Other Program
	Portability	Ability For Software Work Even If Environment Gets Changed
	Interoperability	Ability Of System To Work With Other System

BOEHM'S SOFTWARE QUALITY MODEL

- In 1978, B.W. Boehm introduced his software quality model.
- The model represents a hierarchical quality model
- Boehm's quality model is based on a wider range of characteristics.
- The Boehm's model has three levels for quality attributes.
- These levels are divided based on their characteristics.
- primary uses (high level characteristics),
- intermediate constructs(mid-level characteristics)
- primitive constructs(primitive characteristics).
- As is utility –Extent to which, we can use software as-is.
  - Maintainability –Effort required to detect and fix an error during maintenance.
  - Portability –Effort required to change software to fit in a new environment.

Boehm Software Quality Model



Boehm's Software Quality Model

As-is Utility	Reliability	Self Containedness Accuracy Completeness Robustness/Integrity Consistency
	Efficiency	Accountability Device Efficiency Accessibility
	Human Engineering	Robustness/Integrity Accessibility Communicativeness
Portability		Device Independence Self Containedness
Maintainability	Testability	Accountability Communicativeness Self Descriptiveness Structuredness
	Understandability	Consistency Structuredness Conciseness Legibility
	Modifiability	Structuredness Augmentability
3 High-Level Characteristics	**7 Intermediate-Level Characteristics**	**15 Distinct Primitive Characteristics**

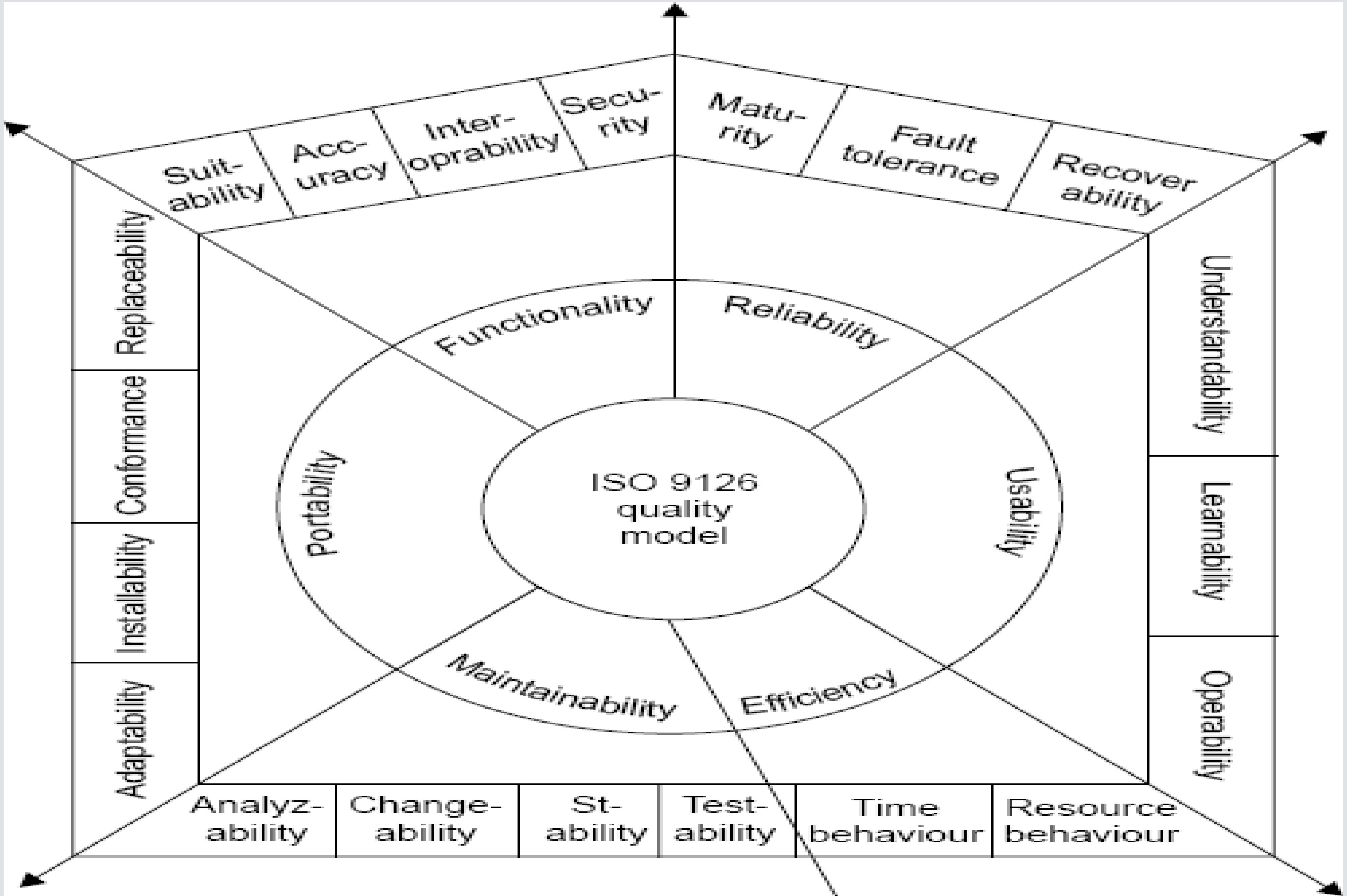
BOEHM'S SOFTWARE QUALITY MODEL

- The next level of Boehm's hierarchical model consists of seven quality factors associated with three primary uses, stated as below –
- Portability – Effort required to change software to fit in a new environment.
- Reliability – Extent to which software performs according to requirements.
- Efficiency – Amount of hardware resources and code required to execute a function.
- Usability – Extent of effort required to learn, operate and understand functions of the software.
- Testability – Effort required to verify that software performs its intended functions.
- Understandability – Effort required for a user to recognize logical concept and its applicability.
- Modifiability – Effort required to modify a software during maintenance phase.

Boehm's Software Quality Model

- Boehm further classified characteristics into Primitive constructs
- Boehm's model is an improvised version of McCall's model
- Testability is broken down into:- accessibility, communicativeness, structuredness and self descriptiveness.
- Advantages :
  - It satisfy the needs of the user.
  - It focuses on software maintenance cost effectiveness.

ISO 9126 Quality Model

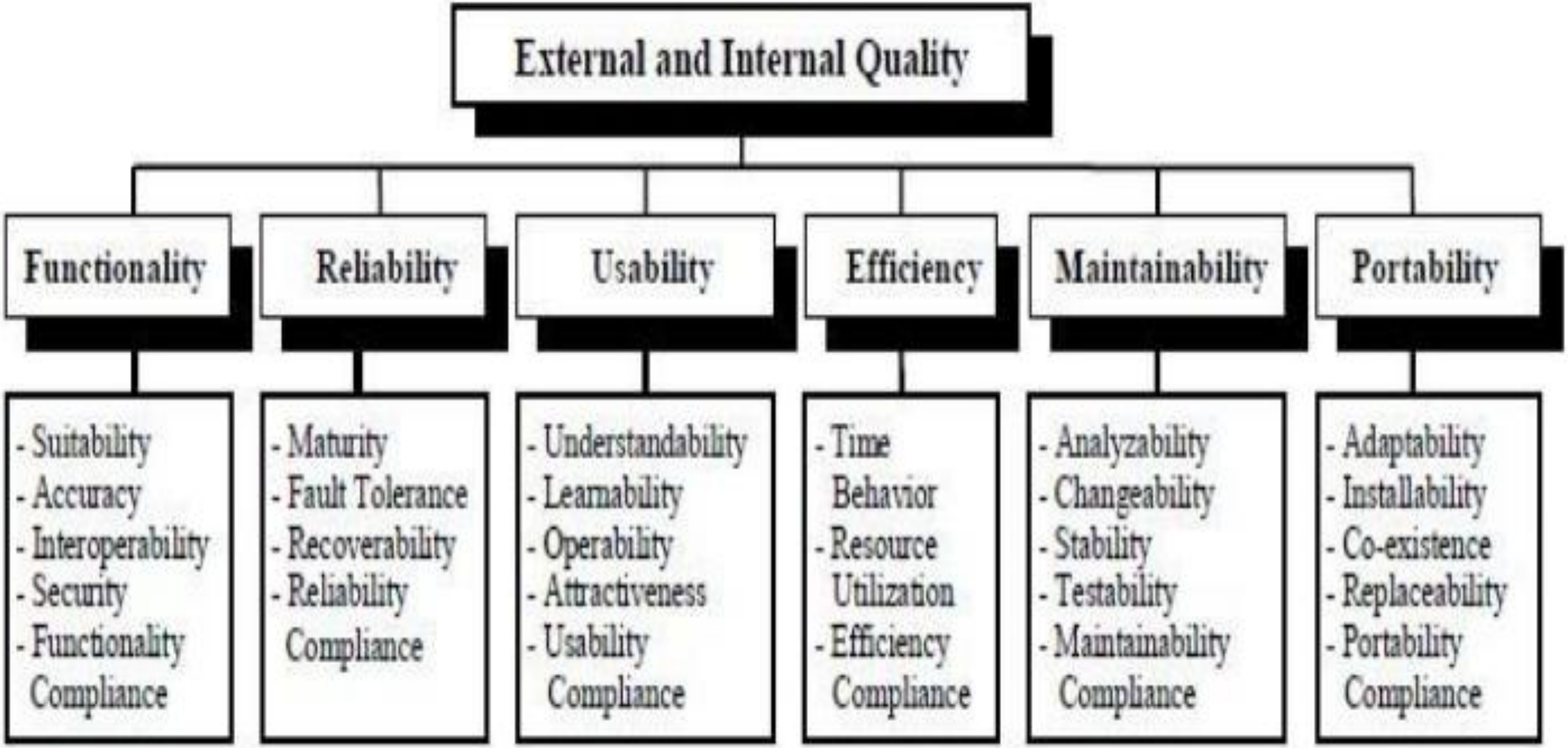


ISO 9126 QUALITY FACTORS

The ISO 9126 standard was developed in an attempt to identify the key quality attributes for computer software. The standard identifies six key quality attributes:

ISO 9126 HAS DEVELOPED FOLLOWING ATTRIBUTES FOR SOFTWARE QUALITY		
FACTOR NAME	**QUALITY FACTORS**	**DESCRIPTION**
Functionality	Sustainability, Accuracy, Compliance, Security	Ability By Which Software Satisfies Needs
Reliability	Maturity, Fault Tolerance, Recoverability	Degree By Which Software Will Be Available
Usability	Understandability, Learnability, Operability	Indicated Usefullness Of Software
Efficiency	Time Behaviour, Resource Behaviour	Measure Of Computing Resource And Time Required By Program To Perform
Maintainability	Anayzability,stability,testability, Changability	Ability To Fix Bug In Software
Portability	Adaptability ,Instanability, Replacability	Ability Of Software To Work Properly

ISO 9126 Quality Factors



ISO 9126 Quality Factors

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

ISO 9126 Quality Factors

• 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.

ISO 9126 Quality Factors

• 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.

ISO 9126 Quality Factors

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.

CAPABILITY MATURITY MODELS (CMM)

- The U.S. Software Engineering Institute (SEI) initiated a study of ways to assess the capabilities o software contractors.
- The outcome of this capability assessment was the SEI Software Capability Maturity Model (CMM).
- The Software CMM was followed by a range of other capability maturity models, including the Peopl Capability Maturity Model (P-CMM) and the Systems Engineering Capability Model.
- Other organizations have also developed comparable process maturity models.
- The SPICE (Software Process Improvement and Capability dEtermination) approach to capabilit assessment and process improvement is more flexible than the SEI model.

CAPABILITY MATURITY MODEL (CMM)

- measure the quality of a process by recognizing it on a maturity scale.
- maturity of a process directly affects the progress of a project and its results
- SEI developed the CMM

CMM Structure

- CMM consists of fi ve maturity levels.
- Maturity levels consist of key process areas (KPAs) which are organized by common features.
- KPA identifi es a cluster of related activities
- features in turn consist of key practices
- Key practices describe the activities to be done

CMM STRUCTURE

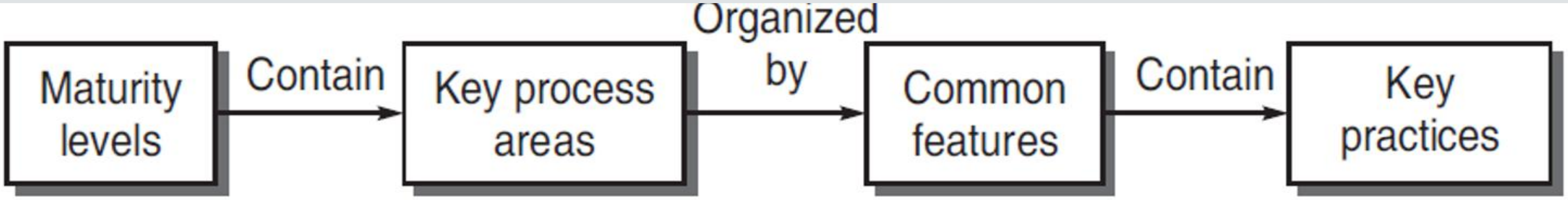


Figure 13.7(a) CMM Structure

Maturity levels	Indicate	Process capability
Key process areas	Achieve	Goals
Common features	Address	Implementation/Institutionalization
Key practices	Describe	Infrastructure/Activities

CMM Structure

- structure of CMM is hierarchical in nature
- such that if one organization wishes to be on maturity level 3, then it must fi rst achieve maturity level 2

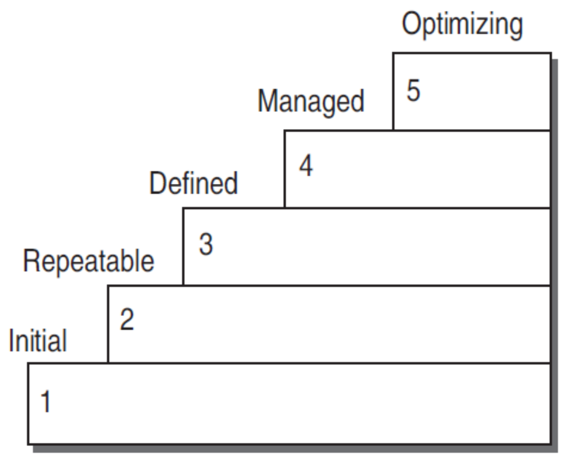


Figure 13.8 Maturity levels

Table 13.1 Process capability at each maturity level

Maturity level	Process Capability
1	—
2	Disciplined process
3	Standard consistent process
4	Predictable process
5	Continuously improving process

CMM MATURITY LEVELS

- A brief description of the fi ve maturity levels

Initial : At this level, there is no control on development progress

- success is purely dependent on individual efforts.
- There is no stable environment for development.
- The projects exceed the budget and schedule.

Repeatable Earlier project successes are used here

- Basic project management processes are established
- cost, schedule, and other parameters can be tracked. organization-wide processes still do not exist.

Defi ned: Project management as well as process management starts.

- The standard processes are defi ned with documentations.

CMM MATURITY LEVELS

- All level 2 criteria have been achieved.
- Process area is defined as per organizational policy

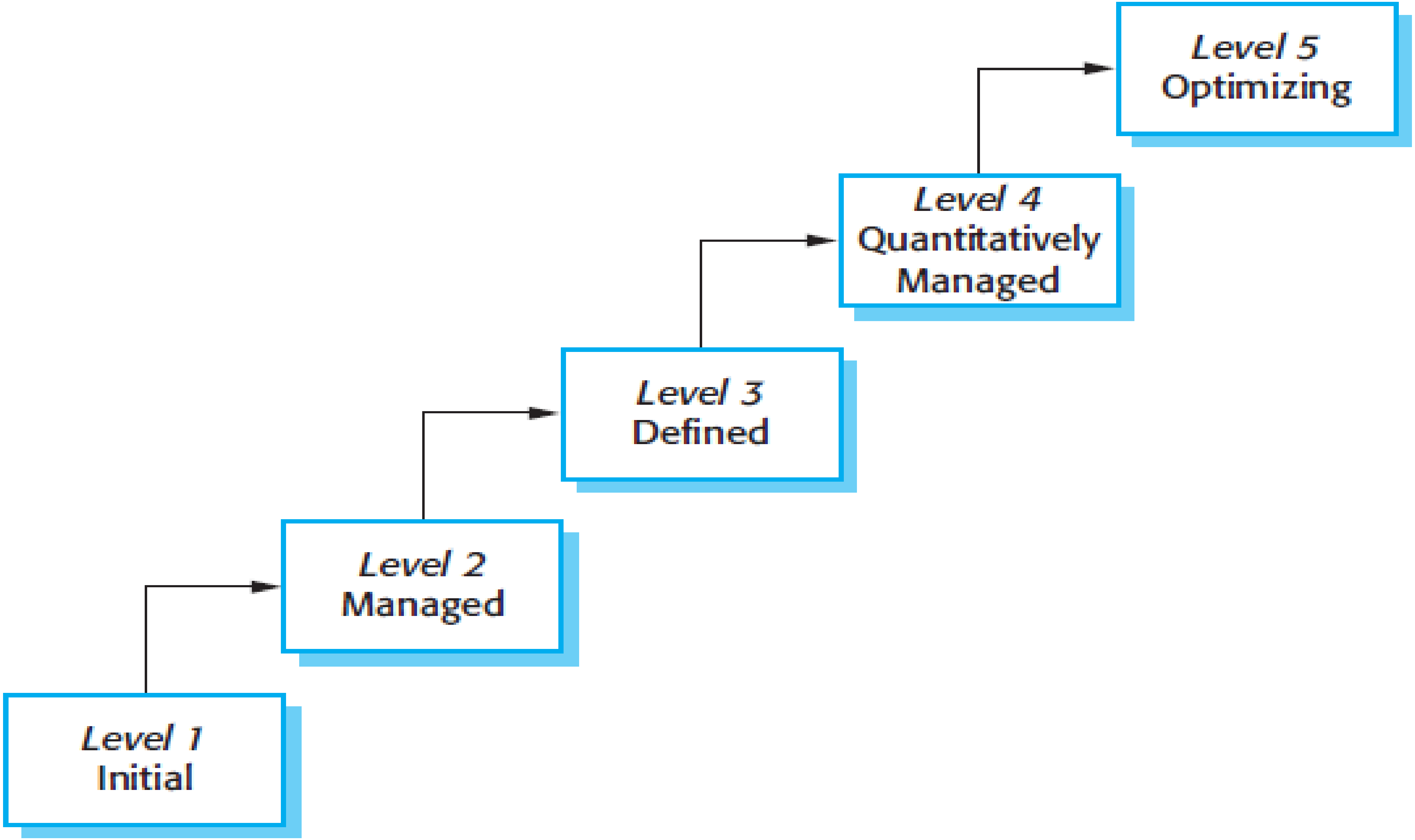
Managed: processes can be monitored and controlled

- quality goals are also set for the processes.
- processes are predictable and controlled.
- All level 3 criteria have been satisfied.
- Process area is controled and improved
- The main objective of quality is established

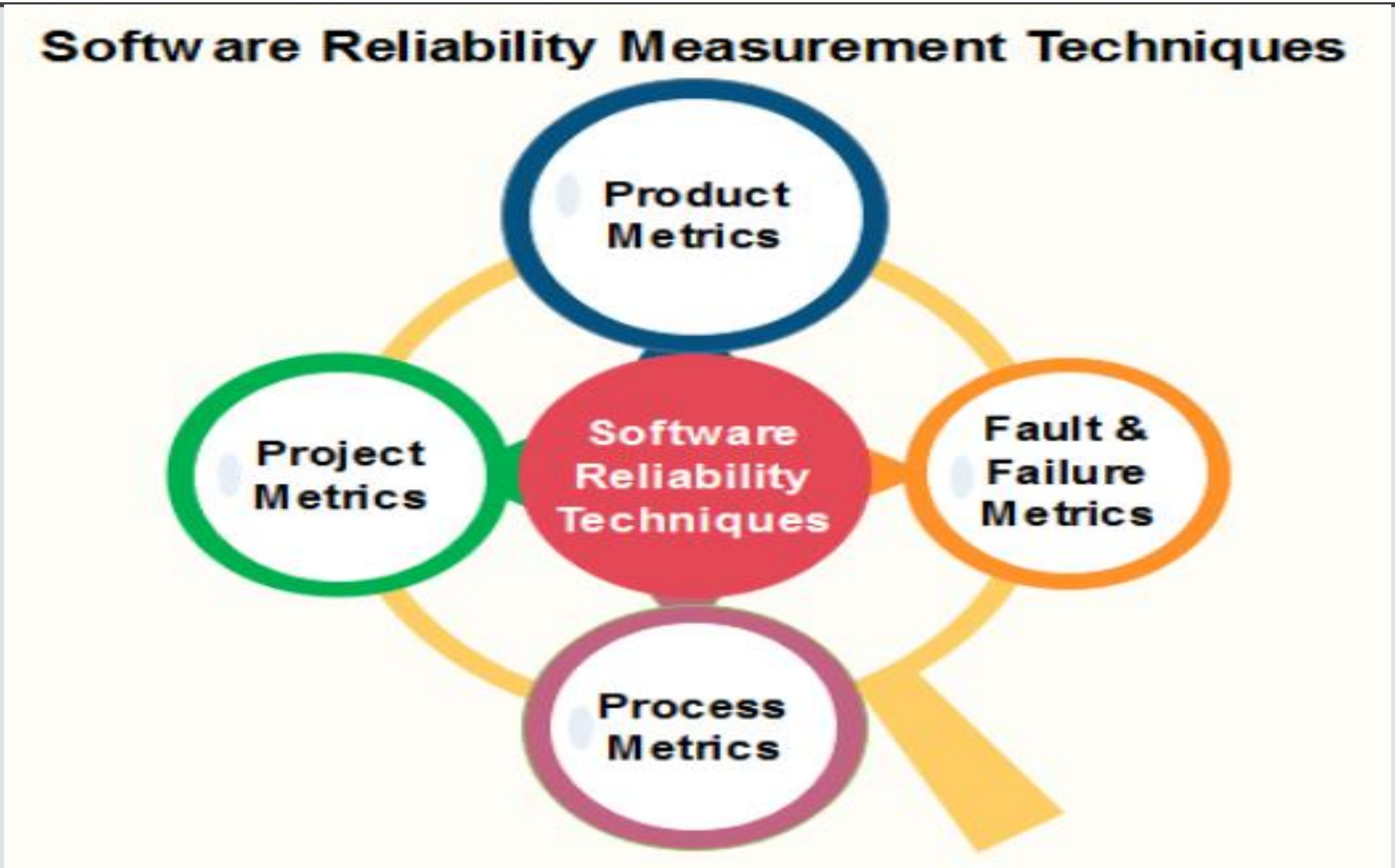
Optimizing : The process is continually improved through incremental and innovative technological changes or improvements.

- process improvement objectives are identifi ed, evaluated, and deployed.

Level	Focus	Process Areas
Optimizing	*Continuous process improvement*	Organizational Innovation and Deployment Causal Analysis and Resolution
Quantitatively managed	*Quantitative management*	Organizational Process Performance Quantitative Project Management
Defined	*Process standardization*	Requirements Development Technical Solution Product Integration Verification Validation Organizational Process Focus Organizational Process Definition Organizational Training Integrated Project Management Integrated Supplier Management Risk Management Decision Analysis and Resolution Organizational Environment for Integration Integrated Teaming
Managed	*Basic project management*	Requirements Management Project Planning Project Monitoring and Control Supplier Agreement Management Measurement and Analysis Process and Product Quality Assurance Configuration Management
Performed		


Level 5
Optimizing
Level 4
Quantitatively
Managed
Level 3
Defined
Level 2
Managed
Level 1
Initial

SOFTWARE METRICS



PRODUCT METRICS

1.Software size. Lines of Code (LOC), or LOC in thousands (KLOC),

2.Function point metric is a technique to measure the functionality of proposed software development based on the count of inputs, outputs, master files, inquires, and interfaces.

3.Test coverage metric It is portion of software that is successfully verified or tested.

4.Complexity It determine the complexity of a program's control structure by simplifying the code into a graphical representation.

5.Quality metrics measure the quality at various steps of software product development. An vital quality metric is Defect Removal Efficiency (DRE)..

Function Point Analysis (FPA)

Alan Albrecht while working for IBM, recognized the problem in size measurement in the 1970s, and developed a technique (which he called Function Point Analysis), which appeared to be a solution to the size measurement problem.

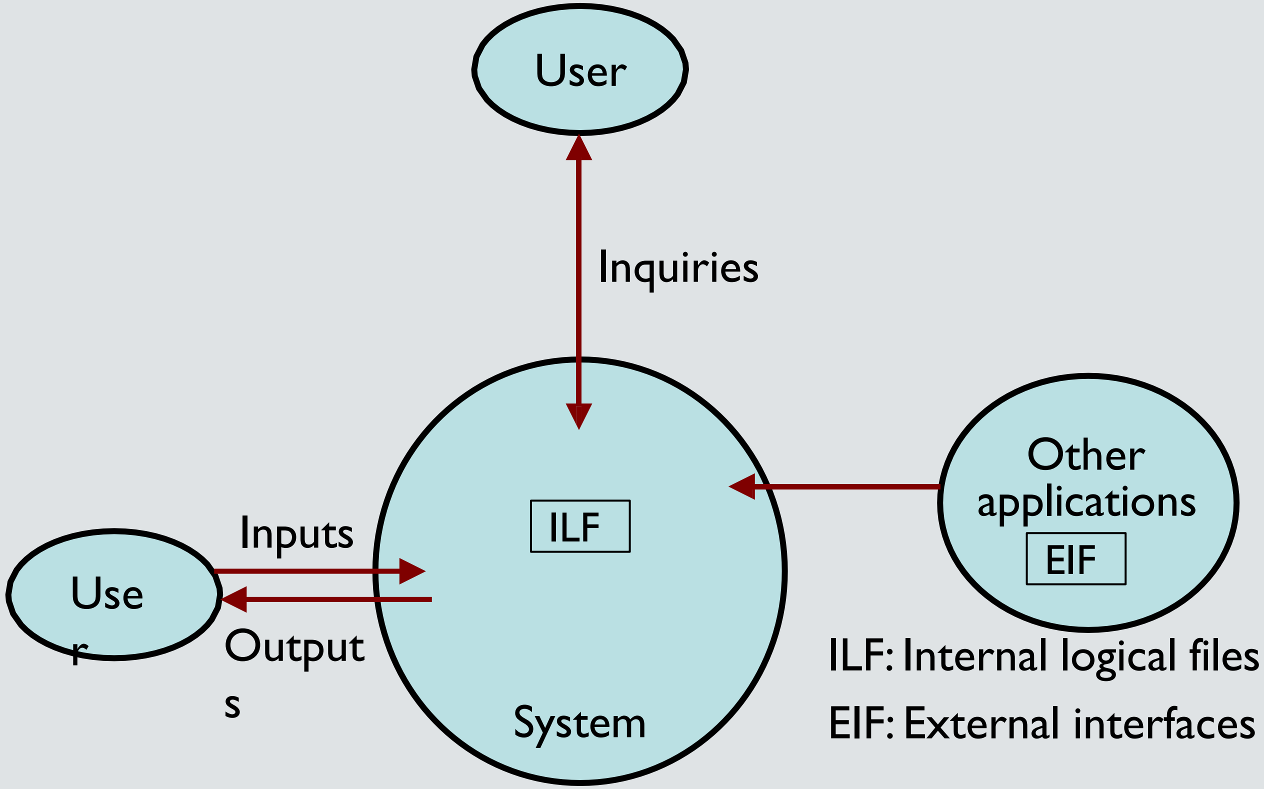
Function Point Analysis (FPA)

The principle of Albrecht's function point analysis (FPA) is that a system is decomposed into functional units.

▪ Inputs	:	information entering the system
▪ Outputs	:	information leaving the system
▪ Enquiries	:	requests for instant access to information
▪ Internal logical files	:	information held within the system
▪ External interface files	:	information held by other system that is used by the system being analyzed.

Function Point Analysis (FPA)

The FPA functional units are shown in figure given below:



ILF: Internal logical files

EIF: External interfaces

Functional Units With Weighting Factors

Functional Units	Weighting factors		
	Low	Average	High
External Inputs (EI)	3	4	6
External Output (EO)	4	5	7
External Inquiries (EQ)	3	4	6
External logical files (ILF)	7	10	15
External Interface files (EIF)	5	7	10

Function Point Analysis (FPA)

Table 2: UFP calculation table

Functional Units	Count Complexity	Complexity Totals	Functional Unit Totals
External Inputs (EIs)	Low x 3 Average x 4 High x 6	= = =	
External Outputs (EOs)	Low x 4 Average x 5 High x 7	= = =	
External Inquiries (EQs)	Low x 3 Average x 4 High x 6	= = =	
External logical Files (ILFs)	Low x 7 Average x 10 High x 15	= = =	
External Interface Files (EIFs)	Low x 5 Average x 7 High x 10	= = =	
Total Unadjusted Function Point Count			

Weighting Factors & Unadjusted Function Point (UFP)

The weighting factors are identified for all functional units and multiplied with the functional units accordingly.

The procedure for the calculation of Unadjusted Function Point (UFP) is given in table shown above.

Unadjusted Function Point (UFP)

The procedure for the calculation of UFP in mathematical form is given below:

$$UFP = \sum_{i=1}^{5} \sum_{J=1}^{3} Z_{ij} w_{ij}$$

Where i indicate the row and j indicates the column of Table 1

W_{ij} : It is the entry of the i^{th} row and j^{th} column of the table 1

Zij : It is the count of the number of functional units of Type *i* that have been classified as having the complexity corresponding to column *j*.

Function Point Analysis (FPA)

Organizations that use function point methods develop a criterion for determining whether a particular entry is Low, Average or High. Nonetheless, the determination of complexity is somewhat subjective.

$$FP = UFP * CAF$$

Where CAF is complexity adjustment factor and is equal to [0.65 + 0.01 x ΣF_i]. The F_i (i=1 to 14) are the degree of influence and are based on responses to questions noted in table 3.

Function Point Analysis (FPA)

Table 3 : Computing function points.

Rate each factor on a scale of 0 to 5.

0	*1*	*2*	*3*	*4*	*5*
No Influence	*Incidental*	*Moderate*	*Average*	*Significant*	*Essential*

Number of factors considered (F_i)

1. Does the system require reliable backup and recovery ?
2. Is data communication required ?
3. Are there distributed processing functions ?
4. Is performance critical ?
5. Will the system run in an existing heavily utilized operational environment ?
6. Does the system require on line data entry ?
7. Does the on line data entry require the input transaction to be built over multiple screens or operations ?
8. Are the master files updated on line ?
9. Is the inputs, outputs, files, or inquiries complex ?
10. Is the internal processing complex ?
11. Is the code designed to be reusable ?
12. Are conversion and installation included in the design ?
13. Is the system designed for multiple installations in different organizations ?
14. Is the application designed to facilitate change and ease of use by the user ?

Function Points Compute The Following Important Metrics

Functions points may compute the following important metrics:

Productivity	=	FP / persons-months
Quality	=	Defects / FP
Cost	=	Rupees / FP
Documentation	=	Pages of documentation per FP

These metrics are controversial and are not universally acceptable. There are standards issued by the International Functions Point User Group (IFPUG, covering the Albrecht method) and the United Kingdom Function Point User Group (UFPGU, covering the MK11 method). An ISO standard for function point method is also being developed.

Function Points Example

Consider a project with the following functional units:

Number of user inputs	= 50
Number of user outputs	= 40
Number of user enquiries	= 35
Number of user files	= 06
Number of external interfaces	= 04

Assume all complexity adjustment factors and weighting factors are average. Compute the function points for the project.

Function Points Example

Solution

We know

$$UFP = \sum_{i=1}^{5} \sum_{J=1}^{3} Z_{ij} w_{ij}$$

UFP = 50 x 4 + 40 x 5 + 35 x 4 + 6 x 10 + 4 x 7

= 200 + 200 + 140 + 60 + 28 = 628

CAF = (0.65 + 0.01 ΣF_i)

= (0.65 + 0.01 (14 x 3)) = 0.65 + 0.42 = 1.07

FP = UFP x CAF

= 628 x 1.07 = 672

Function Points Example

An application has the following:

10 low external inputs, 12 high external outputs, 20 low

internal logical files, 15 high external interface files, 12 average external inquiries, and a value of complexity adjustment factor of 1.10.

What are the unadjusted and adjusted function point counts ?

Function Points Example

Solution

Unadjusted function point counts may be calculated using as:

$$UFP = \sum_{i=1}^{5} \sum_{J=1}^{3} Z_{ij} w_{ij}$$

= 10 x 3 + 12 x 7 + 20 x 7 + 15 + 10 + 12 x 4

= 30 + 84 +140 + 150 + 48

= 452

FP = UFP x CAF

= 452 x 1.10 = 497.2.

Function Points Example

Consider a project with the following parameters.

(i) External Inputs:

(a)10 with low complexity

(b)15 with average complexity

(c) 17 with high complexity

(ii) External Outputs:

(a)6 with low complexity

(b)13 with high complexity

(iii) External Inquiries:

(a) 3 with low complexity

(b) 4 with average complexity

(c) 2 high complexity

Function Points Example

(iv) Internal logical files:

(a) 2 with average complexity

(b) 1 with high complexity

(v) External Interface files:

(a) 9 with low complexity

In addition to above, system requires

i. Significant data communication
ii. Performance is very critical
iii. Designed code may be moderately reusable
iv. System is not designed for multiple installation in different organizations.

Other complexity adjustment factors are treated as average. Compute the function points for the project.

Function Points Example

Solution: Unadjusted function points may be counted using table 2

Functional Units	Count	Complexity		Complexity Totals	Functional Unit Totals
External Inputs (EIs)	10	Low x 3	=	30	
	15	Average x 4	=	60	
	17	High x 6	=	102	192
External Outputs (EOs)	6	Low x 4	=	24	
	0	Average x 5	=	0	
	13	High x 7	=	91	115
External Inquiries (EQs)	3	Low x 3	=	9	
	4	Average x 4	=	16	
	2	High x 6	=	12	37
External logical Files (ILFs)	0	Low x 7	=	0	
	2	Average x 10	=	20	
	1	High x 15	=	15	35
External Interface Files (EIFs)	9	Low x 5	=	45	
	0	Average x 7	=	0	
	0	High x 10	=	0	45
		Total Unadjusted Function Point Count			424

Function Points Example

$$\sum_{i=1}^{14} F_i = 3+4+3+5+3+3+3+3+3+3+2+3+0+3=41$$

$$\begin{aligned}
\text{CAF} &= (0.65 + 0.01 \times \Sigma F_i) \\
&= (0.65 + 0.01 \times 41) \\
&= 1.06 \\
\text{FP} &= \text{UFP} \times \text{CAF} \\
&= 424 \times 1.06 \\
&= 449.44
\end{aligned}$$

Hence $\boxed{\text{FP} = 449}$

LOGARITHMIC POISSON EXECUTION TIME MODEL

- It is one of the **Model** is a **software reliability growth model (SRGM)**
- It predict software failure rates over time.
- It helps estimate software reliability based on observed failure data during testing.
- **Overview of the Logarithmic Poisson Model**
- The model assumes that **software failures occur according to a Poisson process**.
- It considers that **the failure intensity decreases logarithmically as the number of detected failures increases**.
- It is useful for analyzing software reliability **during testing and operational phases**.

LOGARITHMIC POISSON EXECUTION TIME MODEL

2. Mathematical Representation

The expected cumulative number of failures at time t, denoted as $m(t)$, is given by:

$$m(t) = a \log(1 + bt)$$

Where:

- a = Total number of expected failures in the software
- b = Model parameter representing the rate of failure detection
- t = Time or test execution time

The failure intensity function $\lambda(t)$ is:

$$\lambda(t) = \frac{ab}{(1 + bt)}$$

ASSUMPTIONS OF LOGARITHMIC POISSON EXECUTION TIME MODEL

- Failures occur independently and follow a **Poisson distribution**.
- The failure rate **decreases** as more bugs are fixed.
- The number of failures detected follows a **logarithmic growth** pattern.

APPLICATIONS OF LOGARITHMIC POISSON EXECUTION TIME MODEL

- **Software Testing**: Predicting software reliability during the testing phase.
- **Reliability Prediction**: Estimating remaining failures before release.
- **Resource Allocation**: Planning testing efforts efficiently