DESIGN ENGINEERING

Design is a core engineering activity. Design creates a model of the software. Design engineering encompasses the set of principles, concepts and practices that lead to the development of a high quality system or product.

The goal of design engineering is to produce a model or representation that exhibits firmness, commodity and delight. Design engineering for computer software changes continually as new methods, better analysis and broader understanding evolve.

❖ A product should be ***designed*** in a ***flexible*** manner to develop quality software.

DESIGN WITHIN THE CONTEXT OF SOFTWARE ENGINEERING:

Software design is the last software engineering action within modeling activity and sets the stage for development. The analysis model, manifested by scenario-based, class- based, flow-oriented and behavior elements, feed the design task.

Design model produces a data/class design, an architectural design, an interface design, a component design and a deployment design.

- ✓ The data/class design transforms the analysis class models into design class realizations and data structures require implementing the software. Part of class design may occur as each software component is designed.
- ✓ The architectural design defines the relationship between major structural elements of software, the architectural styles and design patterns and the constraints that affect the way in which architectural can be derived from system specifications, the analysis model and interaction of subsystems defined within analysis model.

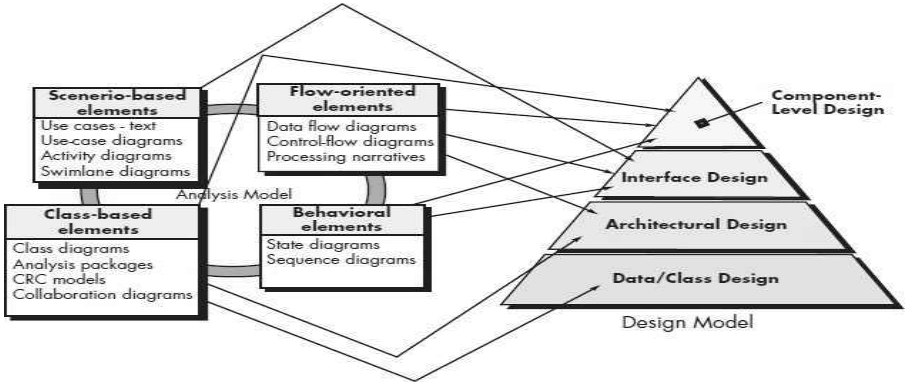


Fig: Translating the Requirements model to Design model

- ✓ The interface design describes how the software communicates with systems that interoperate with it, and with humans who use it. Usage scenarios and behavioral models provide much of information required for the interface design.
- ✓ The component-level design transforms structural element of the software architecture into a procedural description of software components. Information from class-based models, flow models, behavioral models serve as basis for component design.

DESIGN PROCESS AND DESIGN QUALITY:

Importance of software design can be stated with one word QUALITY. Software design serves as foundation for all software engineering and software support activities that follow.

Software design is an interactive process through which requirements are translated into a "blueprint" for constructing the software. Initially, design is represented at a high level of abstraction. As iteration occurs, subsequent refinement leads to design representations at lower levels of abstraction.

The following characteristics serve as a guide for evaluation of good design:

1. The design must implement all explicit requirements contained in analysis model, and accommodate all implicit requirements desired by customer.
2. Design must be a readable, understandable guide for those who generate code, who test and support the software.
3. Design should provide a complete picture of the software, addressing data, functional and behavioral domains.

- ✓ Each of these characteristics is goal of the design process.

Quality Guidelines: Guidelines for quality design are:

1. A design should exhibit an architecture that
   a) has been created using recognizable architectural styles/patterns.
   b) composed of components that exhibit good design characteristics.
   c) can be implemented in an evolutionary fashion.
2. A design should be modular; software should be logically partitioned into elements or sub-systems.
3. It should contain distinct representations of data, architecture, interfaces and components.
4. It should lead to data structures that are appropriate for the classes to be implemented.
5. It should lead to components that exhibit independent functional characteristics.

6. It should lead to interfaces that reduce complexity of connections between components and external environment.
7. It should be represented using a repeatable (iterative) method.
8. It should be represented using a notation that effectively communicates its meaning.

❖ Design engineering encourages good design through the application of fundamental design principles, systematic methodology and through review.

Quality Attributes: Hewlett-Packard (HP) developed a set software quality attributes, given by the acronym ***FURPS***:

1. ***Functionality:*** It is assessed by evaluating feature set and capabilities of the program, generality of functions and security of overall system.
2. ***Usability:*** It is assessed by considering human factors, overall aesthetics, consistency and documentation.
3. ***Reliability:*** It is evaluated by measuring frequency & severity of failure, ability to recover, accuracy of output results, Mean- Time-To-Failure (MTTF) and predictability of the program.
4. ***Performance:*** It is measured by processing speed, response time, resource consumption, throughout and efficiency.
5. ***Supportability:*** It combines the ability to extend the program (extensibility), adaptability, serviceability, which represent maintainability of the project.

These quality attributes must be considered as soon as design commences, but not after the design is complete and construction has begun.

DESIGN CONCEPTS

Fundamental software design concepts provide necessary framework for "getting it right".

1. **Abstraction:** At highest level of abstraction, a solution for design problem is stated in broad terms using language of the problem environment. At lower levels of abstraction, a more detailed description of solution is provided.

✓ A *data abstraction* is a named collection of data that describes a data object.

✓ A *procedural abstraction* refers to a sequence of instructions that have a specific and limited function. Name of procedural abstraction implies these functions, but specific details are suppressed.

2. **Architecture:** It is the structure or organization of program components (modules), their interaction, and structure of data that are used by components. "Components can be generalized to represent major system elements & their interactions.

A set of architectural patterns enable a software engineer to reuse design level concepts. One goal of software design is to derive an architectural rendering of a system, which serves as a framework to conduct detailed design activities.

Architectural design can be represented using one or more of a number of different models:

1. Structural models: Represent architecture as collection of programcomponents.
2. Framework models: Increase the level of design abstraction by attempting to identify repeatable architectural design frameworks that are encountered in similar types of applications.
3. Dynamic models: Address behavioral change aspects of programarchitecture.
4. Process models: Focus on design of business or technical process that the system must accommodate.
5. Functional models: Can be used to represent functional hierarchy of a system.

3. **Patterns:**"A design pattern describes a design structure that solves a particular design problem within a specific context amid "forces"(constraints) that may have an impact on the manner in which pattern is applied and used."

Intent of each design pattern is to provide a description that enables a designer to determine:

i. Whether pattern is applicable to the current work.
ii. Whether pattern can be reused (hence, saving design time)
iii. Whether pattern can serve as a guide for developing a similar, but functionally or structurally different pattern.

4. **Modularity:** Software architecture and design patterns embody modularity, i.e., software is divided into separately named and addressable components, sometimes called modules that are integrated to satisfy problem requirements.

Modularity is the single attribute of software that allows a program to the intellectually manageable (by breaking big process into modules). Modularity leads to a "divide and conquer" strategy, it's easier to solve a complex problem when you break it into manageable pieces, hence effort required to develop becomes negligibly small.

We modularize a design, so that development can be more easily planned, software increments can be defined and delivered, changes can be more easily accommodated, testing and debugging can be conducted more efficiently and long- term maintenance can be conducted without serious side effects.

5. **Information Hiding:** It suggests that "modules should be specified and designed so that information (algorithms, data) contained within a module is inaccessible to other modules that have no need for such

information."

Hiding defines and enforces access constraints to both procedural detail within a module and any local data structure used by the module. As most data and procedure are hidden from other parts of the software, errors during modification are less likely to propagate to other locations within the software.

6. **Functional Independence:** It is achieved by developing modules with "single-minded" function and an "aversion" to excessive interaction with other modules.

Functional independence is a key to good design and design is the key to software quality. Independence is assessed using two qualitative criteria:

- ***Cohesion:*** Cohesion is an indication of relative functional strength of a module. A cohesive module performs a single task, requiring little interaction with other components.
- ***Coupling:*** Coupling is an indication of interconnection among modules in software architecture. Coupling depends on the interface complexity between modules.

7. **Refinement:** Stepwise refinement is a top-down design strategy, which is actually a process of elaboration. A program is developed by successively refining levels of procedural detail.

   It defines/begins with a statement of function that is defined at a high level of abstraction. Refinement helps the designer to reveal low-level details as design progresses, thus in creating a complete design model.

8. **Refactoring:** "Refactoring is the process of changing a software system in such a way that it does not alter external behavior of the code (design) yet improves its internal structure".

   When software is refactored, the existing design is examined for redundancy, unused design elements, poorly constructed data structures, unnecessary algorithms etc for better design.

9. **Design Classes:** As the design model evolves, the software team must define a set of design classes that:
   - Refine analysis classes by providing design detail that will enable the classes to be implemented.
   - Create a new set of design classes that implement a software infrastructure to support the business solution.

   Design classes provide more technical detail as a guide for implementation. Five different types of design classes, each representing a different layer of design architecture are suggested. They are:

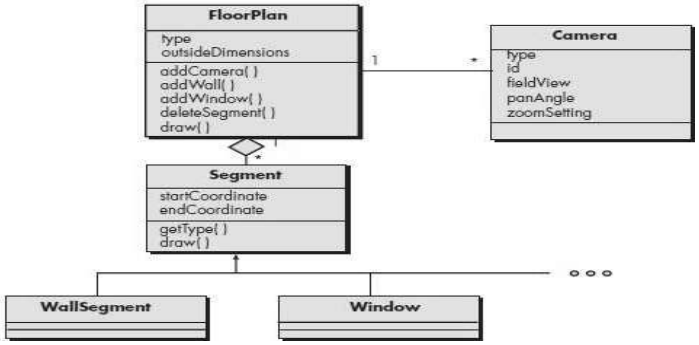
1. *User Interface Classes:* These define all abstractions that are necessary for Human Computer Interaction (HCI). HCI occurs within context of a metaphor (Ex: Order form, a checkbook) and design classes for interface may be visual representations of elements of metaphor.
2. *Business Domain Classes:* These are often refinements of the analysis classes defined earlier. The classes identify attributes and services (operations) that are required to implement some element of the business domain.
3. *Process Classes:* These implement lower-level business abstractions required to fully manage business domain classes.
4. *Persistent Classes:* These represent data stores (DBs) that will persist beyond execution of the software.
5. *System Classes:* These implement software management and control functions that enable system to operate and communicate. These are also known as supporting classes.

As design model evolves, software team must develop a complete set of attributes and operations for each design class.

Four characteristics of a well-formed design class:

1. ***Complete and Sufficient*:** A design class should be the complete encapsulation of all attributes and methods that can reasonably be expected to exist for the class. Sufficiency ensures that the design class contains only those methods that are sufficient to achieve the intent of the class.(No more and No less).
2. ***Primitiveness*:** Methods associated with a design class should be focused on accomplishing one service the class. Once the service has been implemented, with a method, the class should not provide another way to accomplish same thing.
3. ***High Cohesion*:** A cohesion design class has a small, focused set of responsibilities and single- mindedly applies attributes and methods to implement those responsibilities.
4. ***Low Coupling*:** Collaboration between design classes should be kept to an acceptable minimum. If a design model is highly coupled, system is difficult to implement, test & maintain. So, design classes in a subsystem should have only limited knowledge of classes in other subsystems. It is also called as "*Law of Demeter*", suggests that a method should only send messages to methods in neighboring classes.

FIGURE 8.3
Design class for FloorPlan and composite aggregation for the class (see sidebar discussion)



THE DESIGN MODEL

The design model can be viewed in two different dimensions:

- ✓ The process dimension indicates evolution of design model as design tasks are executed as part of the software process.
- ✓ The abstraction dimension represents level of detail as each element of analysis model is transformed into a design equivalent and then refined iteratively.

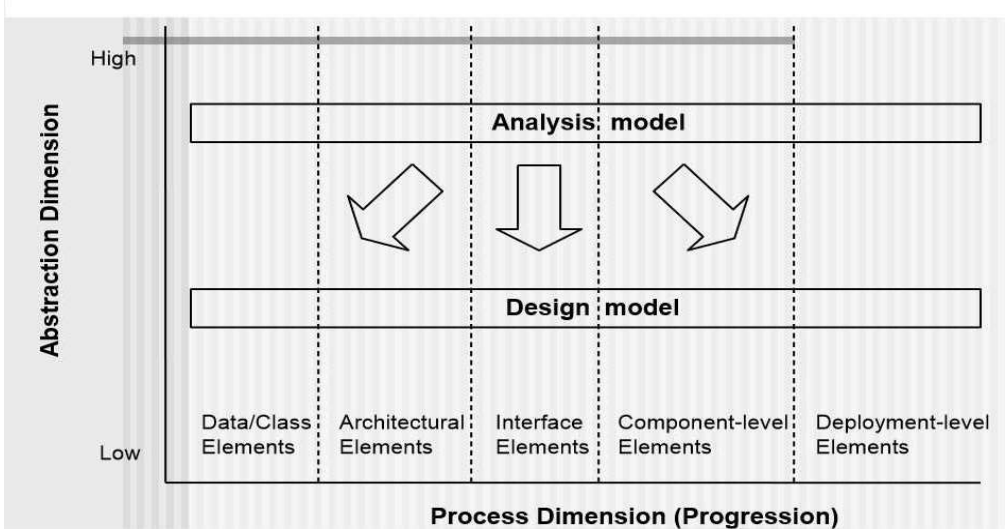


Fig: Dimensions of the Design Model

The elements of design model use many of same UML diagrams that were used in analysis model..The difference is that these diagrams are refined and elaborated as part of design, more implementation- specific detail is provided and emphasis is on architectural structure and style, components & interfaces.

Elements of design model:

1. **Data Design Elements:** Data design also sometimes referred as "Data Architecting". Data design creates a model of data and/or information that is represented at a high level of abstraction.

   In many software applications, architecture of data will have a profound influence on architecture of software that must process it. Structure of data always plays important role in software design.

   - At program component level, design of data structures and associated algorithms required to manipulate them is essential to the creation of high-quality applications.
   - At application level, translation of data model into a DB is important to achieve business objectives.
   - At business level, collection of information stored in DBs and reorganized into a "data warehouse" enables data mining or knowledge discovery.

2. **Architectural Design Elements:** These give us an overall view of the software. It is derived from 3 sources:

   i. Information about application domain for software to be built.

   ii. Specific analysis model elements such as DFDs or analysis classes, their relationships and collaborations for the problem.

   iii. Availability of architectural patterns and styles.

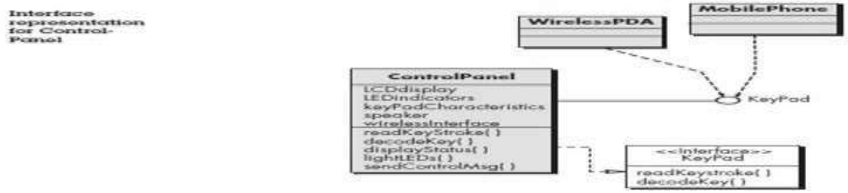
3. **Interface Design Elements:** These tell how information flows into and out of the system and how it is communicated among components designed as part of the architecture. There are 3 important elements of interface design:

   i. **User Interface (UI):** Design of a UI incorporates aesthetic elements (Ex: color, layout, graphics), ergonomic elements (information layout and placement, navigation), and technical elements (UI patterns, reusable components). In general, UI is a unique subsystem within overall application architecture.

   ii. **External interfaces to other systems, devices, networks, other producers/consumers of information:** The design of external interfaces requires definitive information about the entity to which information is sent or received. In every case, this information should be collected during Requirement Engineering and verified. This design should incorporate error checking and appropriate security features.

   iii. **Internal interfaces between various design components:** It is closely aligned with component level design. Design realizations of analysis classes represent all operations and messaging schemes required to enable communication and collaboration between operations in various classes.

In some cases, an interface is modeled in same way as a class."An interface is a set of operations that describes some part of the behavior of a class and provides access to those operations."



4. **Component-Level Design Elements:** Component level for software fully describes the internal detail of each software component. To accomplish this, component-level design defines detail for all processing that occurs within a component and an interface that allows access to all component operations.

A UML component diagram

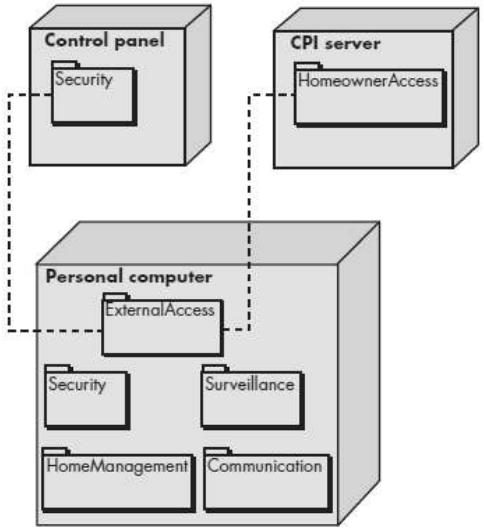
Design details of a component can be modeled at many different levels of abstraction. An activity diagram can be used to represent processing logic. Detailed procedural flow for a component can be represented using either pseudo code or some diagrammatic form.

5. **Deployment-Level Design Elements:** These indicate how software functionality and subsystems will be allocated within the physical computing environment that will support the software.

Deployment diagram shows the computing environment but does not explicitly indicate configuration details. Each instance of deployment is identified.

During design, a UML deployment diagram is first developed, and then refined. In a deployment diagram, each subsystem would be elaborated to indicate components that it implements.

A UML deployment diagram



PATTERN-BASED SOFTWARE DESIGN

Throughout the design process, a software engineer should look for every opportunity to reuse existing design patterns (when they meet needs of the design) rather than creating new ones.

Describing a Design Pattern: Mature engineering disciplines make use of thousands of design patterns, for things such as buildings, highways, electrical circuits, factories, weapons, computers etc. A description of design pattern may also consider a set of design forces.

Design forces describe non-functional requirements (Ex: Portability) associated the software for which the pattern is to be applied. These also define the constraints that may restrict the manner in which design is to be implemented. Design forces describe the environment and constraints that must exist to make design pattern applicable.

- ✓ Pattern characteristics (classes, responsibilities & collaborations) indicate the attributes of the design that may be adjusted to enable the pattern to accommodate a variety of problems.

The names of design patterns should be chosen with care and should have a meaningful name.

Using Patterns in Design: Design patterns can be used throughout software design. The problem description is examined at various levels of abstraction to determine if it is amenable to one or more following types of patterns:

1. **Architectural Patterns:** These patterns,
   - ✓ Define overall structure of the software,
   - ✓ Indicate relationships among subsystems & software components,
   - ✓ Define rules for specifying relationships among the elements (class, components, packages, subsystems) of the architecture.
2. **Design Patterns:** These patterns address a specific element of the design such as an aggregation of components to solve some design problem, relationships among components, or mechanisms for effecting component-to component communication.
3. **Coding Patterns:** These are also called idioms; these language- specific patterns generally implement an algorithmic element of a component, a specific interface protocol, or a mechanism for communication among components. Each of these pattern types differs in the level of abstraction and degree to which it provides direct guidance for construction activity of software process.

Frameworks: "A framework is not an architectural pattern, but rather a skeleton with a collection of "plug points" (also called hooks and slots) that enable it to be adapted to a specific problem-domain."

Plug points enable designer to integrate problem specific classes or functionality within the skeleton.

In O-O context, framework is collection of co-operating classes.

- ✓ To be most effective, frameworks are applied with no changes, additional design elements may be added, but only via plug points that allow designer to flesh out the framework selection.

UNIT IV

Testing Strategies: A strategic approach to software testing, test strategies for conventionalsoftware, Black-Box and White-Box testing, Validation testing, System testing, the art of Debugging.

Product metrics: Software Quality, Metrics for Analysis Model, Metrics for Design Model,Metrics for source code, Metrics for testing, Metrics for maintenance.

Metrics for Process and Products: Software Measurement, Metrics for software quality.

Testing Strategies

Software is tested to uncover errors introduced during design and construction. Testingoften accounts for

More project effort than other s/e activity. Hence it has to be done carefully using a testingstrategy.
The strategy is developed by the project manager, software engineers and testing specialists.
Testing is the process of execution of a program with the intention of finding errors Involves 40% of total project cost
Testing Strategy provides a road map that describes the steps to be conducted as part oftesting.
It should incorporate test planning, test case design, test execution and resultant data collection and execution
Validation refers to a different set of activities that ensures that the software is traceable tothe Customer requirements.
V&V encompasses a wide array of Software Quality Assurance

A strategic Approach for Software testing

Testing is a set of activities that can be planned in advance and conducted systematically.Testing strategy
Should have the following characteristics:
-- usage of Formal Technical reviews(FTR)
-- Begins at component level and covers entire system
-- Different techniques at different points
-- conducted by developer and test group
-- should include debugging
Software testing is one element of verification and validation.
Verification refers to the set of activities that ensure that software correctly implements aspecific function.
(Ex: Are we building the product right?)
Validation refers to the set of activities that ensure that the software built is traceable tocustomer requirements.

(Ex: Are we building the right product ?)

Testing Strategy

Testing can be done by software developer and independent testing group. Testing and debugging are different activities. Debugging follows testing

Low level tests verifies small code segments. High level tests validate major systemfunctions against customer requirements

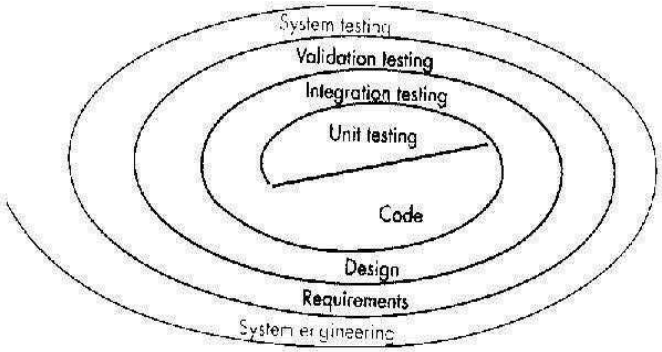
Test Strategies for Conventional Software:

Testing Strategies for Conventional Software can be viewed as a spiral consisting of four levelsof testing:

1) Unit Testing
2)Integration Testing
3)Validation Testing
and

4) System Testing

Spiral Representation of Testing for Conventional Software



Unit Testing begins at the vortex of the spiral and concentrates on each unit of software insource code.

It uses testing techniques that exercise specific paths in a component and its control structure to ensure complete coverage and maximum error detection. It focuses on the internal processing logicand data structures. Test cases should uncover errors.

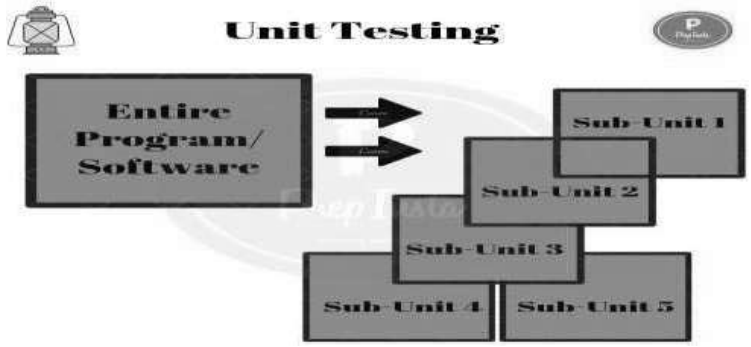


Fig: Unit Testing

Boundary testing also should be done as s/w usually fails at its boundaries. Unit tests can be designed before coding begins or after source code is generated.

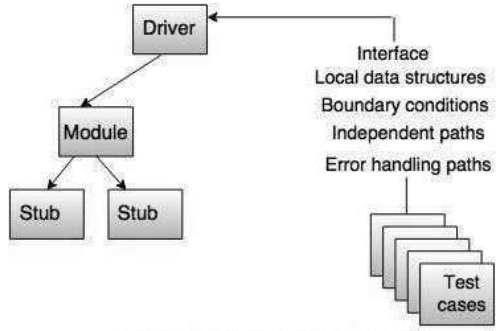


Fig. - Unit test environment

Integration testing: In this the focus is on design and construction of the software architecture. It addresses the issues associated with problems of verification and program construction by testing inputs and outputs. Though modules function independently problems may arise because of interfacing. This technique uncovers errors associated with interfacing. We can use top-down integration wherein modules are integrated by moving downward through the control hierarchy, beginning with the main control module. The other strategy is bottom –up which begins construction and testing with atomic modules which are combined into clusters as we move up the hierarchy. A combined approach called Sandwich strategy can be used i.e., top- down for higher level modules and bottom-up for lower level modules.

Validation Testing: Through Validation testing requirements are validated against s/wconstructed. These are high-order tests where validation criteria must be evaluated to assure that s/w meets all functional, behavioural and performance requirements. It succeeds when the software functionsin a manner that can be reasonably expected by the customer.

1)Validation Test

Criteria2)Configuration

Review 3)Alpha And

Beta Testing

The validation criteria described in SRS form the basis for this testing. Here, Alpha and Beta testing is performed. Alpha testing is performed at the developers site by end users in a natural setting and with a controlled environment. Beta testing is conducted at end-user sites. It is a "live" application and environment is not controlled.

End-user records all problems and reports to developer. Developer then makes modifications and releases the product.

System Testing: In system testing, s/w and other system elements are tested as a whole. This is the last high-order testing step which falls in the context of computer system engineering. Software is combined with other system elements like H/W, People, Database and the overall functioning is checked by conducting a series of tests. These tests fully exercise the computer based system. The types of tests are:

1. Recovery testing: Systems must recover from faults and resume processing within a prespecified time.

It forces the system to fail in a variety of ways and verifies that recovery is properly performed. Here the Mean Time To Repair (MTTR) is evaluated to see if it is within acceptable limits.

2. Security Testing: This verifies that protection mechanisms built into a system will protect it from improper penetrations. Tester plays the role of hacker. In reality given enough resources and time it is possible to ultimately penetrate any system. The role of system designer is to make penetration cost more than the value of the information that will be obtained.

3. Stress testing: It executes a system in a manner that demands resources in abnormal quantity, frequency or volume and tests the robustness of the system.

4. Performance Testing: This is designed to test the run-time performance of s/w within the context of an integrated system. They require both h/w and s/w instrumentation.

Testing Tactics:

The goal of testing is to find errors and a good test is one that has a high probability of findingan error.

A good test is not redundant and it should be neither too simple nor too complex.Two major categories of software testing

- Black box testing: It examines some fundamental aspect of a system, tests whether eachfunction of product is fully operational.
- White box testing: It examines the internal operations of a system and examines theprocedural detail.

Black box testing

This is also called behavioural testing and focuses on the functional requirements of software. It fully exercises all the functional requirements for a program and finds incorrect or missing functions,interface errors, database errors etc. This is performed in the later stages in the testingprocess. Treatsthe system as black box whose behaviour can be determined by studying its input and related output Not concerned with the internal. The various testing methods employedhere are:

1) Graph based testing method: Testing begins by creating a graph of important objects and theirrelationships
and then devising a series of tests that will cover the graph so that each object and relationshipis exercised and errors are uncovered.

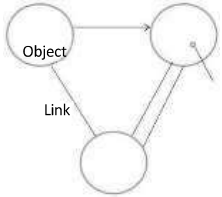


Fig: O-R graph.

2) Equivalence partitioning: This divides the input domain of a program into classes of datafrom which test
Cases can be derived. Define test cases that uncover classes of errors so that no. of test cases are reduced.This is based on equivalence classes which represents a set of valid or invalid states forinputconditions. Reduces the cost of testing

Example

Input consists of 1 to 10

Then classes are n<1,1<=n<=10,n>10

Choose one valid class with value within the allowed range and two invalid classes wherevalues are greater than maximum value and smaller than minimum value.

3) Boundary Value analysis

Select input from equivalence classes such that the input lies at the edge of the equivalence classes. Set of

data lies on the edge or boundary of a class of input data or generates the data that lies at the boundary of a class of output data. Test cases exercise boundary values to uncover errors at the boundaries of the input domain.

Example

If 0.0<=x<=1.0

Then test cases are (0.0,1.0) for valid input and (-0.1 and 1.1) for invalid input

4) Orthogonal array Testing
This method is applied to problems in which input domain is relatively small but too large for exhaustive testing

Example

Three inputs A,B,C each having three values will require 27 test cases. **O**rthogonal testing will reduce the number of test case to 9 as shown below

White Box testing
Also called glass box testing. It uses the control structure to derive test cases. It exercises all independent paths, Involves knowing the internal working of a program, Guarantees that all independent paths will be exercised at least once .Exercises all logical decisions on their true and false sides, Executes all loops,Exercises all data structures for their validity. White box testingtechniques

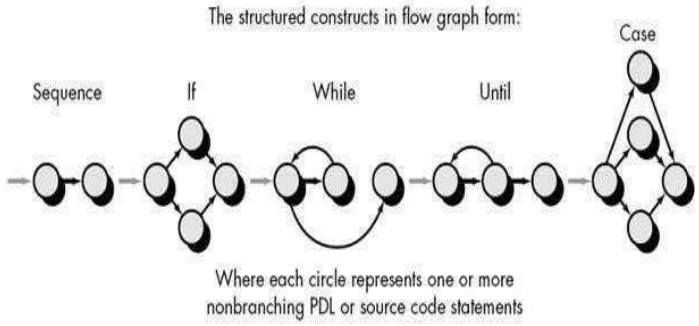
1. Basis path testing
2.Control structure
testing1.Basis path
testing

Proposed by Tom McCabe. Defines a basic set of execution paths based on logicalcomplexity of a procedural design. Guarantees to execute every statement in the program at least once Steps of Basis Path Testing

1. Draw the flow graph from flow chart of the program
2.Calculate the cyclomatic complexity of the resultant flow graph3.Prepare test cases that will force execution of each path

Two methods to compute Cyclomatic complexity number
1.V(G)=E-N+2 where E is number of edges, N is number of nodes2.V(G)=Number of regions

The structured constructs used in the flow graph are:



effective
It is not sufficient in

itself2.Control

Structure testing

This broadens testing coverage and improves quality of testing. It uses the following methods:
a) Condition testing: Exercises the logical conditions contained in a program module.
Focuses on testing each condition in the program to ensure that it does not contain errorsSimple condition
E1<relation operator>E2 Compound condition
simple condition<Boolean operator>simple condition
Types of errors include operator errors, variable errors, arithmetic expression errors etc.
b) Data flow Testing
This selects test paths according to the locations of definitions and use of variables in aprogram Aims to ensure that the definitions of variables and subsequent use is tested
First construct a definition-use graph from the control flow of a program
DEF(definition):definition of a variable on the left-hand side of an assignment statement
USE: Computational use of a variable like read, write or variable on the right hand of
assignment statement Every DU chain be tested at least once.

c) Loop Testing
This focuses on the validity of loop constructs. Four categories can be defined

1. Simple loops
2.Nested loops

3.Concatenated
loops
4.Unstructured
loops

Testing of simple loops
N is the maximum number of allowable passes through the loop
1.Skip the loop entirely
2.Only one pass through the
loop3.Two passes through
the loop
4.m passes through the loop where
m>N5.N-1,N,N+1 passes the loop

The Art of Debugging

Debugging occurs as a consequence of successful testing. It is an action that results in theremoval of errors.
It is very much an art.

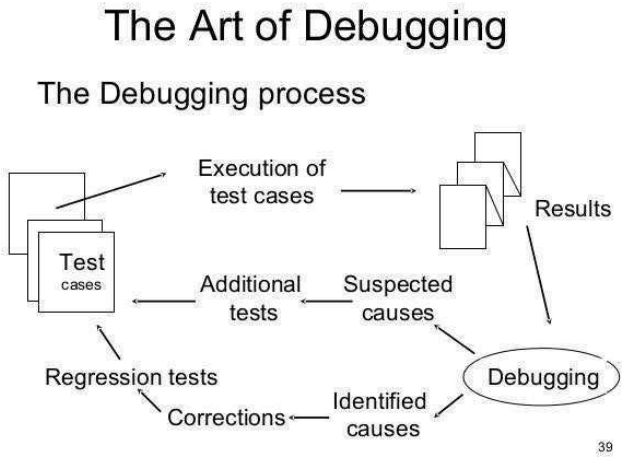


Fig: Debugging process

Debugging has two outcomes:
- cause will be found and corrected
- cause will not be
foundCharacteristics of
bugs:
- symptom and cause can be in different locations

- Symptoms may be caused by human error or timing problems

Debugging is an innate human trait. Some are good at it and some are

not.

Debugging Strategies:

The objective of debugging is to find and correct the cause of a software error which is realizedby a combination of systematic evaluation, intuition and luck. Three strategies are proposed: 1)Brute Force Method.

2) Back Tracking
3)Cause
Elimination

Brute Force: Most common and least efficient method for isolating the cause of a s/w error.

This is applied

when all else fails. Memory dumps are taken, run-time traces are invoked and program is loaded with output statements. Tries to find the cause from the load of information Leads towaste of time and effort.

Back tracking: Common debugging approach. Useful for small programs

Beginning at the system where the symptom has been uncovered, the source code is tracedbackward until the site of the cause is found. More no. of lines implies no. of paths are unmanageable.

Cause Elimination: Based on the concept of Binary partitioning. Data related to error occurenec are organized to isolate potential causes. A "cause hypothesis" is devised and data is used to prove or disprove it. A list of all possible causes is developed and tests areconducted to eliminate each

Automated Debugging: This supplements the above approaches with debugging tools thatprovide semi-automated support like debugging compilers, dynamic debugging aids, test casegenerators, mapping tools etc.

Regression Testing: When a new module is added as part of integration testing the software changes.

This may cause problems with the functions which worked properly before. This testing is there-execution of some subset of tests that are already conducted to ensure that changes have not propagatedunintended side effects. It ensures that changes do not introduce unintended behaviour or errors. This can be done manually or automated. Software Quality Conformanceto explicitly stated functional andperformance requirements, explicitly documented development standards, and implicit characteristics that are expected of

All professionally developed software.

Factors that affect software quality can be categorized in two broad groups:
Factors that can be directly measured (e.g. defects uncovered during testing)
Factors that can be measured only indirectly (e.g. usability or maintainability)

McCall's quality factors

1. ProductoperationCorrectness

Reliability
Efficiency
Integrity
Usability

2. Product Revision
Maintainability
Flexibility
3. Product
Transition
Portability
Reusability
Interoperability

ISO 9126 Quality Factors

1. Functionality
2. Reliability
3. Usability
4. Efficiency
5. Maintainability
6. Portability

Product metrics

Product metrics for computer software helps us to assess quality.
Measure Provides a quantitative indication of the extent, amount, dimension, capacity or size of someattribute of a product or processMetric(IEEE 93 definition)

A quantitative measure of the degree to which a system, component or process possess agiven attribute Indicator

A metric or a combination of metrics that provide insight into the software process, asoftware project or a product itself

Product Metrics for analysis, Design, Test and maintenance

Product metrics for the Analysis model

- Function point Metric
  - First proposed by Albrecht

Measures the functionality delivered by the systemFP computed from the following parameters

1) Number of external inputs(EIS)
2) Number external outputs(EOS)

Product metrics for the Analysis model

Number of external Inquiries(EQS)
Number of Internal Logical Files(ILF)
Number of external interface files(EIFS)
Each parameter is classified as simple, average or complex and weights are assigned as follows

Product metrics for the Analysis model

• Information Domain	Count	Simple	avg	Complex
EIS		3	4	6
EOS		4	5	7
EQS		3	4	6
ILFS		7	10	15
EIFS		5	7	10

FP=Count total *[0.65+0.01*E(Fi)]

Metrics for Design Model

DSQI(Design Structure Quality Index)US air force has designed the DSQI
Compute s1 to s7 from data and architectural design

S1:Total number of modules
S2:Number of modules whose correct function depends on the data inputS3:Number of modules whose function depends on prior processing S4:Number of data base items

S5:Number of unique database itemsS6: Number of database segments

S7:Number of modules with single entry and exit

Calculate D1 to D6 from s1 to s7 as follows:

D1=1 if standard design is followed otherwise D1=0D2(module independence)=(1-(s2/s1))
D3(module not depending on prior processing)=(1-(s3/s1))D4(Data base size)=(1-(s5/s4))
D5(Database compartmentalization)=(1-(s6/s4) D6(Module entry/exit characteristics)=(1-(s7/s1))

DSQI=sigma of WiDi

i=1 to 6,Wi is weight assigned to Di

If sigma of wi is 1 then all weights are equal to 0.167

DSQI of present design be compared with past DSQI. If DSQI is significantly lower than the average,further design work and review are indicated

METRIC FOR SOURCE CODE

HSS(Halstead Software science)

Primitive measure that may be derived after the code is generated or estimated once design is complete

n_1 = the number of distinct operators that appear in a program n_2 = the number of distinct operands that appear in a program N_1 = the total number of operator occurrences.
N_2 = the total number of operand occurrence.Overall program length N can be computed:

$N = n_1 \log2\ n_1 + n_2 \log2\ n_2$ $V = N \log_2(n_1 + n_2)$

METRIC FOR TESTING

n_1 = the number of distinct operators that appear in a program
n_2 = the number of distinct operands that appear in a program
N_1 = the total number of operator occurrences.
N_2 = the total number of operand occurrence.Program Level and Effort
$PL = 1/[(n_1 / 2) \times (N_2 / n_2$ l)]e = V/PL

METRICS FOR MAINTENANCE

M_t = the number of modules in the current release
F_c = the number of modules in the current release that have been changedF_a = the number of modules in the current release that have been added.
F_d = the number of modules from the preceding release that were deleted in the currentrelease

The Software Maturity Index, SMI, is defined as:

$$SMI = [M_{t-}(F_c + F_a + F_{d)/} M_t]$$

Metrics for Process And Product

Software Measurement:
Software measurement can be categorized as
1) Direct Measure and
2) Indirect Measure

Metrics for Process And ProductDirect Measurement

Direct measure of software process include cost and effort
Direct measure of product include lines of code, Execution speed, memory size, defects per reporting time period.

Indirect Measurement

Indirect measure examines the quality of software product itself(e.g. :- Functionality,complexity, efficiency, reliability and maintainability)

Reasons for measurement

To gain baseline for comparison with future assessmentTo determine status with respect to plan
To predict the size, cost and duration estimate
To improve the product quality and process improvement

Software Measurement

The metrics in software Measurement areSize oriented metrics
Function oriented metricsObject oriented metrics
Web based application metric

Size Oriented Metrics

It totally concerned with the measurement of software.
A software company maintains a simple record for calculating the size of the software.It includes LOC, Effort,$$,PP document,Error,Defect ,People.

Function oriented metrics

Measures the functionality derived by the application
The most widely used function oriented metric is Function pointFunction point is independent of programming language
Measures functionality from user point of view

Object oriented metric

Relevant for object oriented programmingBased on the following

- Number of scenarios (Similar to use cases)
- Number of key classesNumber of support
- classesNumber of average support class per
- key class Number of subsystem

Web based application metric

Metrics related to web based application measure the following

1. Number of static pages(NSP)
2. Number of dynamic pages(NDP)

Customization(C)=NSP/NSP+NDPC should approach 1

MetricsforSoftware Quality

Measuring Software Quality

1.Correctness=defects/KLOC

2. Maintainability=MTTC(Mean-time to change)3.Integrity=Sigma[1-(threat(1-security))]

Threat : Probability that an attack of specific type will occur within a given time

Security : Probability that an attack of a specific type will be repelled Metrics for SoftwareQuality Usability: Ease of use

Defect Removal Efficiency(DRE) DRE=E/(E+D)

E is the no. of errors found before delivery and D is no. of defects reported after deliveryIdeal value of DRE is 1

UNIT – V:

Risk Management: Reactive vs. Proactive risk strategies, Software risks, Risk identification, Risk projection, RMMM plan

Quality Management: Quality concepts, Software quality assurance, Formal technical reviews, ISO 9000 Quality standards.

Risk Management

Risk is an undesired event or circumstance that occur while a project is underway It is necessary for the project manager to anticipate and identify different risks that a project may be susceptible to Risk Management. It aims at reducing the impact of all kinds of risk that may effect a project by identifying, analyzing and managing them

Reactive Vs Proactive risk

Reactive : It monitors the projects likely risk and resources are set aside.

Proactive: Risk are identified, their probability and impact is accessed

Software Risk

It involve 2 characteristics

Uncertainty : Risk may or may not happen

Loss : If risk is reality unwanted loss or consequences will occur It includes

1)Project Risk 2)Technical Risk 3)Business Risk 4)Known Risk 5)Unpredictable Risk 6) Predictable risk

Project risk: Threaten the project plan and affect schedule and resultant cost

Technical risk: Threaten the quality and timeliness of software to be produced

Business risk: Threaten the viability of software to be built

Known risk: These risks can be recovered from careful evaluation

Predictable risk: Risks are identified by past project experience

Unpredictable risk: Risks that occur and may be difficult to identify

Risk Identification

It concerned with identification of riskStep1: Identify all possible risks Step2: Create item check list

Step3: Categorize into risk components-Performance risk, cost risk, support risk and schedule risk

Step4: Divide the risk into one of 4 categoriesNegligible-0

Marginal-1Critical-2

Risk Identification

Risk Identification includes Product size

Business impact Development environment Process definition Customer characteristicsTechnology to be built Staff size and experience

Risk Projection

Also called risk estimation. It estimates the impact of risk on the project and the product.

Estimation is done by using Risk Table. Risk projection addresses risk in 2 ways

Risk	Category	Probability	Impact	RMMM
Size estimate may be Significantly low	PS	60%	2	
Larger no. of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End user resist system	BU	40%	3	

Likelihood or probability that the risk is real

(Li) Consequences (X_i)

Risk Projection

Steps in Risk projection

- Estimate L_i for each risk
- Estimate the consequence X_i
- Estimate the impact
- Draw the risk table

Ignore the risk where the management concern is low i.e., risk having impact high or low with low probability of occurrence

Consider all risks where management concern is high i.e., high impact with high or moderate probability of occurrence or low impact with high probability of occurrence

Risk Projection Projection The impact of each risk is assessed by Impact valuesCatastrophic-1 Critical-2 Marginal-3Negligible-4

Risk Refinement

Also called Risk assessment

Refines the risk table in reviewing the risk impact based on the following three factorsa. Nature:

Likely problems if risk occurs

b. Scope: Just how serious is it?c. Timing: When and how long

It is based on Risk Elaboration Calculate Risk exposure RE=P*C

Where P is probability and C is cost of project if risk occurs Risk Mitigation Monitoring And Management (RMMM)

Its goal is to assist project team in developing a strategy for dealing with risk There are three issues of RMMM

1.Risk Avoidance

2.Risk Monitoring and

3.Risk Management

Risk Mitigation Monitoring And Management (RMMM)

Risk Mitigation

Proactive planning for risk avoidance Risk Monitoring

Assessing whether predicted risk occur or not Ensuring risk aversion steps are being properly applied Collection of information for future risk analysis Determine which risks caused which problems

Risk Mitigation Monitoring And Management (RMMM)Risk Management Contingency planning

Actions to be taken in the event that mitigation steps have failed and the risk has become a live problem Devise RMMP(Risk Mitigation Monitoring And Management Plan)

RMMM plan

It documents all work performed as a part of risk analysis.

Each risk is documented individually by using a Risk Information Sheet. RIS is maintained by using a database system Quality Management

QUALITY CONCEPTS

Variation control is the heart of quality control

Form one project to another, we want to minimize the difference between the predicted resources needed to complete a project and the actual resources used, including staff, equipment, and calendar time

1. Quality of design

Refers to characteristics that designers specify for the end product Quality Management

2. Quality of conformance

Degree to which design specifications are followed in manufacturing the product

3.Quality control

Series of inspections, reviews, and tests used to ensure conformance of a work product to its
specifications

4.Quality assurance

Consists of a set of auditing and reporting functions that assess the effectiveness and completeness of quality control activities

COST OF QUALITY

Prevention costs

Quality planning, formal technical reviews, test equipment, training Appraisal costs In-process and inter-process inspection, equipment calibration and maintenance, testing

Failure costs

rework, repair, failure mode analysis

External failure costs

Complaint resolution, product return and replacement, help line support, warranty work

Software Quality Assurance

Software quality assurance (SQA) is the concern of every software engineer to reduce cost and improve product time-to-market.

A Software Quality Assurance Plan is not merely another name for a test plan, though test plans are included in an SQA plan.

SQA activities are performed on every software project.

Use of metrics is an important part of developing a strategy to improve the quality of both

software processes and work products.

SOFTWARE QUALITY ASSURANCE

Definition of Software Quality serves to emphasize:

Conformance to software requirements is the foundation from which software quality is measured. Specified standards are used to define the development criteria that are used to guide the manner in which software is engineered. Software must conform to implicit requirements (ease of use, maintainability, reliability, etc.) as well as its explicit requirements.

SQA Activities

- Prepare SQA plan for the project.
- Participate in the development of the project's software process description.
- Review software engineering activities to verify compliance with the defined software process.
- Audit designated software work products to verify compliance with those defined aspart of the software process.
- Ensure that any deviations in software or work products are documented and handled according to a documented procedure.
- Record any evidence of noncompliance and reports them to management.

SOFTWARE REVIEWS

- Purpose is to find errors before they are passed on to another software engineering activity or released to the customer.
- Software engineers (and others) conduct formal technical reviews (FTRs) for software quality assurance.
- Using formal technical reviews (walkthroughs or inspections) is an effective means for improving software quality.

FORMAL TECHNICAL REVIEW

A FTR is a software quality control activity performed by software engineers and others. The objectives are:

- To uncover errors in function, logic or implementation for any representation ofthe software.
- To verify that the software under review meets its requirements.
- To ensure that the software has been represented according to predefined standards. To achieve software that is developed in a uniform manner and
- To make projects more manageable.

Review meeting in FTR

The Review meeting in a FTR should abide to the following constraints :

- Review meeting members should be between three and five.
- Every person should prepare for the meeting and should not require more than two hours of work for each person.
- The duration of the review meeting should be less than two hours.
- The focus of FTR is on a work product that is requirement specification, a detailed component design, a source code listing for a component.
- The individual who has developed the work product i.e, the producer informs the project leader that the work product is complete and that a review is required.

The project leader contacts a review leader, who evaluates the product for readiness, generates copy of product material and distributes them to two or three review members for advance preparation.

Each reviewer is expected to spend between one and two hours reviewing the product, making notes

The review leader also reviews the product and establish an agenda for the reviewmeeting

The review meeting is attended by review leader, all reviewers and the producer.

One of the reviewer act as a recorder, who notes down all important points discussed in the meeting.

The meeting (FTR) is started by introducing the agenda of meeting and then **the producer introduces his product. Then the producer "walkthrough" the** product, the reviewers raise issues which they have prepared in advance.

If errors are found the recorder notes down

Review reporting and Record keeping

During the FTR, a reviewer(recorder) records all issues that have been raised A review summary report answers three questions

What was reviewed? Who reviewed it?

Whatwere the findings and conclusions?

Review summary report is a single page form with possible attachments

The review issues list serves two purposes To identify problem areas in the product To serve as an action item checklist that guides the producer as corrections are made

Review Guidelines

- Review the product, not the producer
- Set an agenda and maintain it
- Limit debate and rebuttal
- Enunciate problem areas, but don't attempt to solve **every problem** noted
- Take return notes
- Limit the number of participants and insist upon advance preparation.
- Develop a checklist for each product i.e likely to be reviewed
- Allocate resources and schedule time for FTRS
- Conduct meaningful training for all reviewer
- Review your early reviews Software Defects
- Industry studies suggest that design activities introduce 50-65% of all defects or errors during the software process
- Review techniques have been shown to be up to 75% effective in uncovering design flaws which ultimately reduces the cost of subsequent activities in the software process

Statistical Quality Assurance Information about software defects is collected and categorized. Each defect is traced back to its cause

Using the Pareto principle (80% of the defects can be traced to 20% of the causes) isolate the "vital few" defect causes.

Move to correct the problems that caused the defects in the "vital few"

Six Sigma for Software Engineering

The most widely used strategy for statistical quality assurance

Three core steps:

- Define customer requirements, deliverables, and project goals via well-defined methods of customer communication.
- Measure each existing process and its output to determine current quality performance (e.g., compute defect metrics)
- Analyze defect metrics and determine vital few causes.

For an existing process that needs improvement

1. Improve process by eliminating the root causes for defects
2. Control future work to ensure that future work does not reintroduce causes of defectsIf new processes are being developed

1. Design each new process to avoid root causes of defects and to meet customer requirements

2.Verify that the process model will avoid defects and meet customer requirements

SOFTWARE RELIABILITY

Defined as the probability of failure free operation of a computer program in a specified environment for a specified time period

Can be measured directly and estimated using historical and developmental data Software reliability problems can usually be traced back to errors in design or implementation.

Measures of Reliability

Mean time between failure (MTBF) = MTTF + MTTRMTTF = mean time to failure MTTR = mean time to repair

Availability = [MTTF / (MTTF + MTTR)] x 100%

ISO 9000 Quality Standards

ISO (International Standards Organization) is a group or consortium of 63 countries established to plan and fosters standardization. ISO declared its 9000 series of standards in 1987. It serves as a reference for the contract between independent parties. The ISO 9000 standard determines the guidelines for maintaining a quality system. The ISO standard mainly addresses operational methods and organizational methods such as responsibilities, reporting, etc. ISO 9000 defines a set of guidelines for the production process and is not directly concerned about the product itself.

Types of ISO 9000 Quality Standards

The ISO 9000 series of standards is based on the assumption that if a proper stage is followed for production, then good quality products are bound to follow automatically. The types of industries to which the various ISO standards apply are as follows.

ISO 9001: This standard applies to the organizations engaged in design, development, production, and servicing of goods. This is the standard that applies to mostsoftware development organizations.

ISO 9002: This standard applies to those organizations which do not design products but are only involved in the production. Examples of these category industries contain steel and car manufacturing industries that buy the product and plants designs from external sources and are engaged in only manufacturing those products. Therefore, ISO 9002 does not apply to software development organizations.

ISO 9003: This standard applies to organizations that are involved only in theinstall action and testing of the products. For example, Gas companies.

An organization determines to obtain ISO 9000 certification applies to ISO registrar office for registration. The process consists of the following stages:

1. **Application:** Once an organization decided to go for ISO certification, itapplies to the registrar for registration.
2. **Pre-Assessment:** During this stage, the registrar makes a rough assessment of the organization.
3. **Document review and Adequacy of Audit:** During this stage, the registrar reviews the document submitted by the organization and suggest an improvement.
4. **Compliance Audit:** During this stage, the registrar checks whether the organization has compiled the suggestion made by it during the review or not.
5. **Registration:** The Registrar awards the ISO certification after the successful completionof all the phases.
6. **Continued Inspection:** The registrar continued to monitor the organization time by time.