Software Engineering

Unit 6: Software Quality and Testing

Software Quality and Testing

CE: 6.1
Software Quality

CE: 6.1 Software Quality

What is the Quality?

- IEEE defines quality as…

✓ The degree to which **a system**, **a component** or **a process** meets specific requirements.

or

✓ The degree to which **a system**, **a component** or **a process** meets ***customer or user needs or expectations***.

CE: 6.1 Software Quality (Conti…)

What is the Software Quality?

- A **software** must follow both ***functional*** and ***non-functional requirements*** which are specified by the customer or the user.

- If the software product fulfill all the customer's *requirements*, the customer *feels satisfied* and the product is *expected to be of high quality*.

- Therefore, ***the goal of software quality is to define:***
  1. ***How well is the design of the software?***
  2. ***How well the software follows to the developed design?***

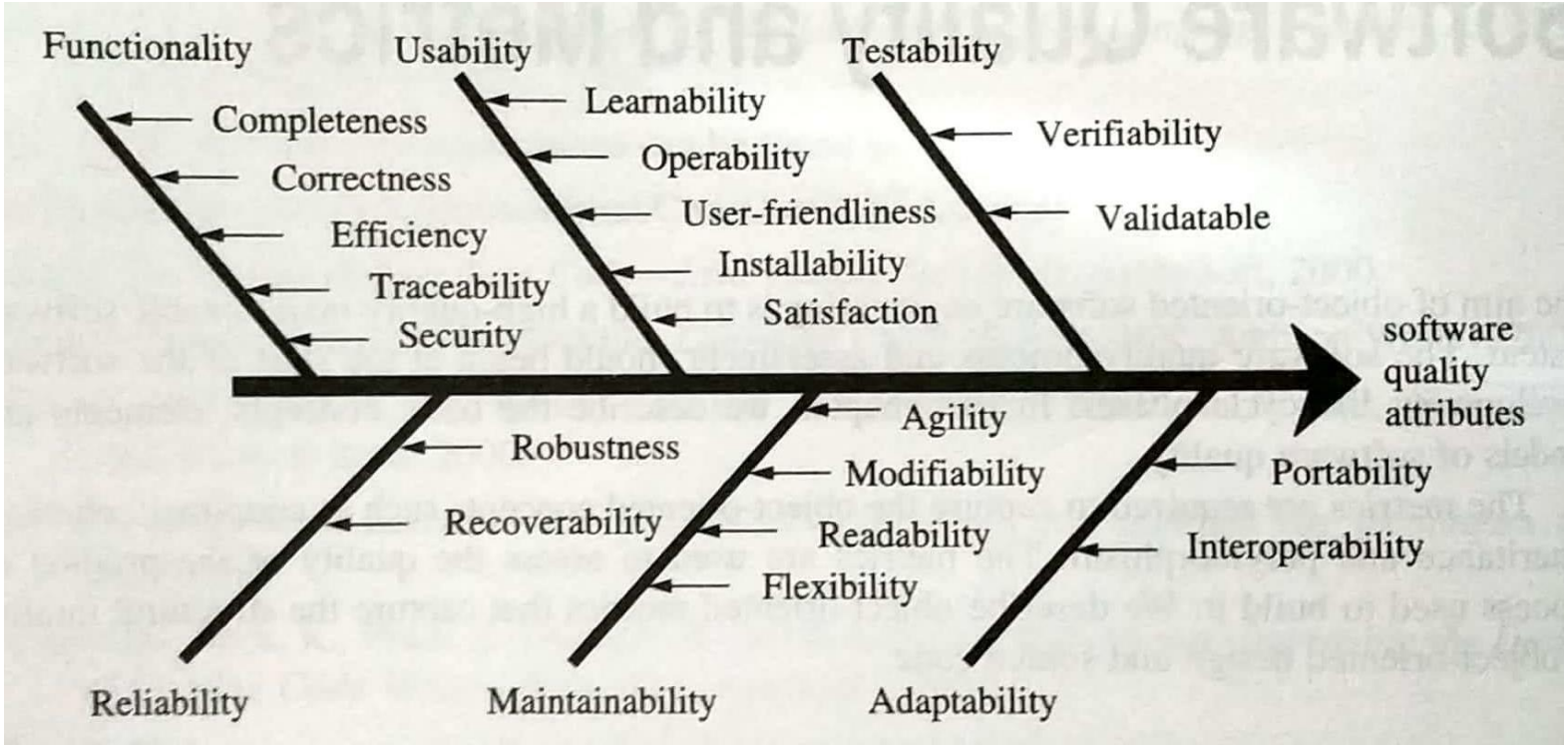
CE: 6.1.1 Software Quality Attributes

- To measure the quality of software, there are different domains of attribute that are required to define the software:

1. Functionality
2. Usability
3. Testability
4. Reliability
5. Maintainability
6. Adaptability

CE: 6.1.1 Software Quality Attributes (Conti…)

- The different domains attribute is further divided into attributes as…



CE: 6.1.1 Software Quality Attributes (Conti…)

1. Functionality:

- To which the purpose of the software is satisfied.

- It can be further divided into following attributes:

1. Completeness	To which the software is complete.
2. Correctness	To which the software is correct.
3. Efficiency	To which the software is requires resources to perform a software function.
4. Traceability	To which the requirement is traceable to the software design and source code.
5. Security	To which the software is able to prevent unauthorized access to the program data.

CE: 6.1.1 Software Quality Attributes (Conti…)

2. Usability:

- To which the software is easy to learn.

- It can be further divided into following attributes:

1. Learnability	To which the software is easy to learn.
2. Operability	To which the software is easy to operate.
3. User-friendliness	To which the interfaces of the software are easy to use and understand.
4. Installability	To which the software is easy to install.
5. Satisfaction	To which the user feels satisfied with the software.

CE: 6.1.1 Software Quality Attributes (Conti…)

3. Testability:

- The easiness with which the software can be tested to demonstrate the faults.

- It can be further divided into following attributes:

1. Verifiability	To which the software deliverable meets the specified standards, procedures and process.
2. Validatable	To ease with which the software can be executed to demonstrate whether the established testing criterion is met.

CE: 6.1.1 Software Quality Attributes (Conti…)

4. Reliability:

- To which the software performs failure-free functions.

- It can be further divided into following attributes:

1. Robustness	To which the software performs reasonably under unexpected circumstances.
2. Recoverability	The speed with which the software recovers after the occurrence of a failure.

CE: 6.1.1 Software Quality Attributes (Conti…)

5. Maintainability:

- To ease with which the faults can be located and fixed, quality of the software can be improved or software can be modified in the maintenance phase.

- It can be further divided into following attributes:

1. Agility	To which the software is quick to change or modify.
2. Modifiability	To which the software is easy to implement, modify and test in the maintenance phase.
3. Readability	To which the software documents and programs are easy to understand, so that the faults can be easily located and fixed in the maintenance phase.
4. Flexibility	The ease with which changes can be made in the software in the maintenance phase.

CE: 6.1.1 Software Quality Attributes (Conti…)

6. Adaptability:

- To which the software is adaptable to different technologies and platforms.

- It can be further divided into following attributes:

1. Portability	The ease with which the software can be transferred form one platform to another platform.
2. Interoperability	To which the system is compatible with other systems.

CE: 6.2
Software Quality Model

CE: 6.2 Software Quality Model

- There are many quality models available in literature. Some of the popular models are:

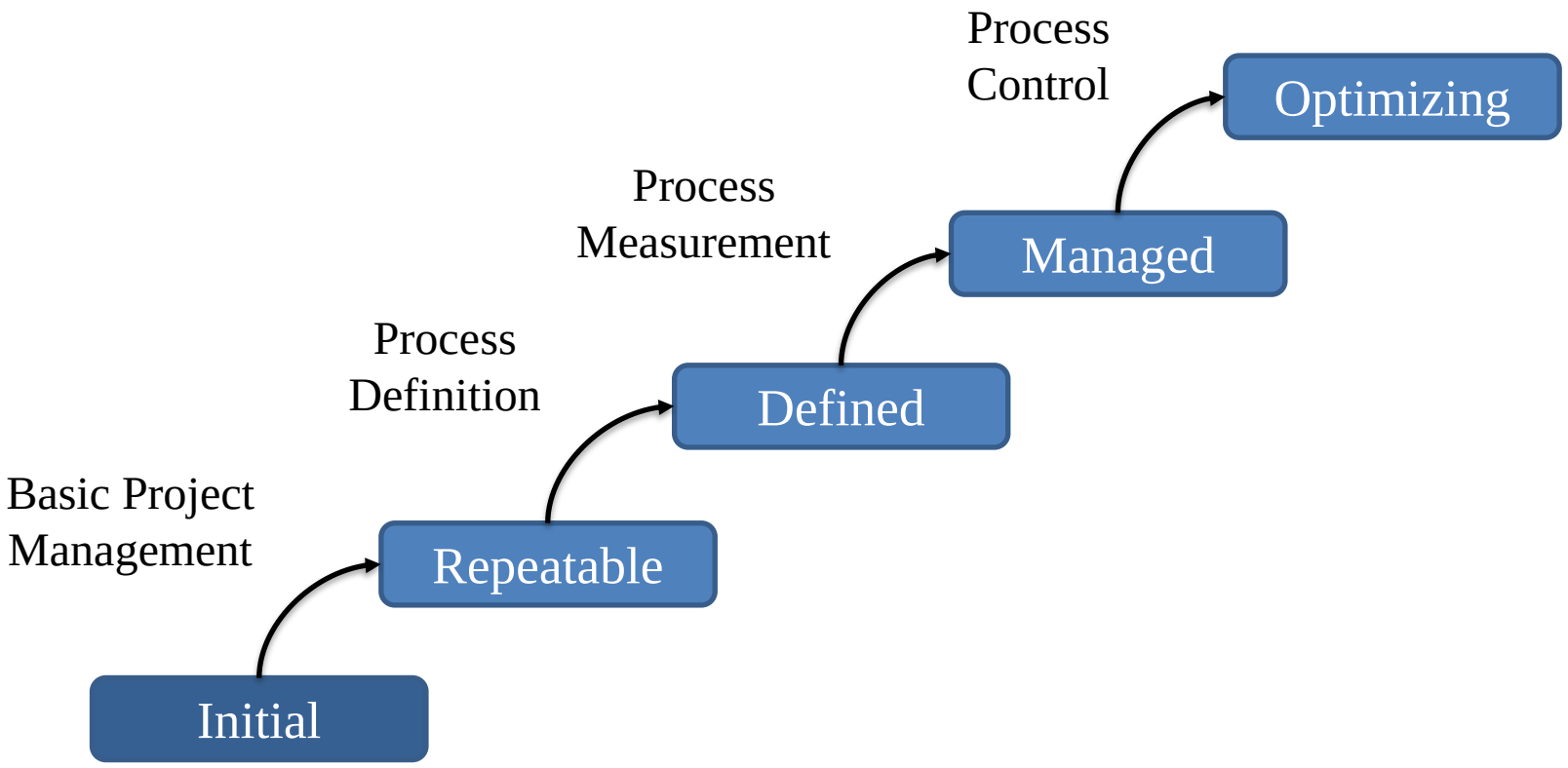
1. McCall's Software Quality Model
2. Boehm's Software Quality Model
3. ISO 9000
4. ISO 9126
5. **Capability Maturity Model**

CE: 6.2.1 Capability Maturity Model (CMM)

- CMM provides a framework to define the key elements of an effective and efficient software process.
- It covers the key practices that helps to ***plan, improve*** and ***manage*** Software Development Life Cycle activities.
- If any organization follow these software development practices, then that will ***improve the resource utilization*** and ***quality*** of software.
- The initial version of CMM was developed at Software Engineering Institute (SEI), Carnegie Mellon University, USA.

CE: 6.2.1 Capability Maturity Model (CMM) (Conti…)

- **The CMM is categorized into five maturity levels:**



Levels of CMM

CE: 6.2.1 Capability Maturity Model (CMM) (Conti…)

Maturity Level 1: Initial

- It is the lowest level and at this level, organization do not have a stable environment for software engineering and management practices.
- Reason???
  - ✓ Because of ineffective planning and management of system.

- **Everything is carried out on an ad-hoc basis.**

- The success of a project depends on a experienced manager and good software project team. But management principles cannot be covered by strong engineering principles.

- The capability of software process is also undisciplined, as the processes are continuously changing or being modified as the software development progresses.

CE: 6.2.1 Capability Maturity Model (CMM) (Conti…)

Maturity Level 1: Initial (Conti…)

- The performance is completely depended on individual rather than organization. So it changes when the staff changes.
- Therefore, Time and Cost of development are unpredictable.

CE: 6.2.1 Capability Maturity Model (CMM) (Conti…)

Maturity Level 2: Repeatable

- At this level, ***procedures*** and ***policies*** for managing the software product are established.
- Here, planning and managing of new software projects are based on the past similar projects, not on ad-hoc basis.
- Therefore, by repeating these past practices, an organization aim to develop effective software, will get into process.
- And an effective process is one which is ***efficient***, ***defined***, ***documented and measured***; and ***has the ability for improvement***.

CE: 6.2.1 Capability Maturity Model (CMM) (Conti…)

Maturity Level 2: Repeatable (Conti…)

- Unlike level 1, at this level, the managers identify problems and take corrective actions for the prevention.
- The managers also keep track of Time and Cost.
- The standards are also defined and the organizations ensure that these standards are actually followed by the project team.
- Therefore, at this level, the process can be **described as disciplined**.
  [Because it is well planned, stable and past successful practices are repeatable.]

CE: 6.2.1 Capability Maturity Model (CMM) (Conti…)

Maturity Level 3: Defined

- At this level, the established software processes are documented.
- The software processes are used by project managers and technical staff to improve the performance more effectivity. These are known as ***Standard Software Processes***.
- The organizations also conduct a training programme for this, so that the staff and managers are well aware of the sources and understand their assigned tasks.
- Different projects in the organizations modify the Standard Processes and Practices, to construct their own defined Software Processes and Practices.

CE: 6.2.1 Capability Maturity Model (CMM) (Conti…)

Maturity Level 3: Defined (Conti…)

- These defined Software Processes are specific to the requirement of individual characteristics of the given project.
- A well-defined process includes inputs, standards, practices and procedures to carry out the work; verification procedures, outputs and criteria of completion.
- At this level, a process can be described as **standard and completed**.

CE: 6.2.1 Capability Maturity Model (CMM) (Conti…)

Maturity Level 4: Managed

- At this level, the goals for quality of the software product are set.
- The *organization's measurement program*, measures the quality of the *software process* and *standards*; that helps in giving indication of the trends of quality in processes and standards followed in the organization.
- Here, ***the limits specifying starting point are established***; and when these limits are beaten, then the ***corrective action is to be taken***.

CE: 6.2.1 Capability Maturity Model (CMM) (Conti…)

Maturity Level 5: Optimizing

- It is the highest level of maturity in CMM.
- At this level, the focus of the organization is on ***continuous improvement*** of the software processes.
- The software projects identify the causes of the ***software defects*** and ***evaluate the software process*** to prevent the defects from reoccurring.
- By using new tools, technologies and the advance innovations in existing processes, the improvement is occurred.
- Therefore, this level can be defined as "**continuously improving**" level.

CE: 6.2.1 Capability Maturity Model (CMM) (Conti…)

CE: 6.2.1 Capability Maturity Model (CMM) (Conti…)

- The past experience reports show that moving from **level 1 to 2** may take **3 to 5 years**.
- The CMM is becoming popular and many software organization are aspiring to achieve CMM **level 5**.
- The acceptability and pervasiveness (extensiveness) of the CMM activities are helping the organizations to produce a ***quality software***.

CE: 6.3
Software Testing

CE: 6.3 Software Testing

- Software testing is a very important, challenging and essential activity.
- It starts along with the design of SRS document and ends with the delivery of software product to the customer.
- It consumes significant effort and maximum time of software development life cycle (without including the maintenance phase.)
- Any product cannot be imagine to deliver without adequate testing.
- However, an adequate testing has different meaning to different software testers.

CE: 6.3 Software Testing (Conti…)

But….

What is Software Testing?

- ***Software Testing*** is an activity to check whether the actual results match the expected results or not; and ***to check that the software system is defect free***.

- There are many more definitions of testing. Some of them are:

1. The aim of testing is to show that a program performs its desired functions correctly.
2. Testing is the process of demonstrating that errors are not present.
3. Testing is the process of establishing confidence that a program does what it is supposed to do.

CE: 6.3 Software Testing (Conti…)

- Therefore, as per the above definitions, the purpose of testing is ***to show the correctness of the program***.

- And our objective to do testing should be to find the faults and find them as early as possible.

- Hence, one more appropriate definition is given by Myers (2004) as:

4. Testing is the process of executing a program with the intent of finding faults.

- This definition motivates to select those inputs, which have higher probability of finding faults, although this definition is also not complete (because it focuses only on the execution of the program).

CE: 6.3 Software Testing (Conti…)

- Nowadays, the more attention is given to the activities like…
  reviewing the documents and programs.

- *Reviewing the documents* (like SRS and SDD (Software Design Descriptions)) helps to find a good number of faults in the early phases of the software development.

- Hence, a testing is dividing in two parts:
  1. Verification
  2. Validation

- Therefore, the most appropriate definition of software testing is:

  "***Software Testing*** is the process of *verifying* the outcomes of every phase of software development and *validating* the program by executing it with the intention of finding faults."

CE: 6.3.1 Verification

- *Software verification* is also known as *static testing*.
- *Verification testing* is the testing, where ***activities are carried out without executing the program***.
- It may include inspections, walkthroughs and reviews, where documents and programs are reviewed with purpose of finding faults.
- Therefore, ***verification*** is the process of reviewing (rechecking) documents and program with the intention of finding faults.
- And as per IEEE (2001):
  "The verification is the process of evaluating the system or component to determine, whether the products of a given development phase satisfy the conditions imposed at the start of that phase."

CE: 6.3.2 Validation

- *Software validation* is also known as *dynamic testing*.
- *Validation testing* is the testing, where the ***program is executed with given input(s).***
- It involves execution of the program and usually exposes symptoms of errors.
- Therefore, ***validation*** are the activities which are carried out with the execution of the program.
- While doing programming, we experience failures and the reason of such failures are identified through validation.

CE: 6.3.2 Validation (Conti…)

- And as per IEEE (2001):
  "The validation is the process of evaluating a system or component during or at the end of development process to determine, whether it satisfies the specified requirements."

CE: 6.3 Software Testing (Conti…)

- Therefore, both ***verification*** and ***validation*** are important and complementary to each other.

- Effective verification activities find a good number of faults in the early phases of software development. And removing such faults will definitely provide better foundations for the implementation/construction phase.

- And the *validation* activities are possible only after the implementation of program/module, but the *verification* activities are possible in very phase of software development.

- In the initial days of programming, testing was the primarily validation orientated, but now-a-days both are equally important and carried out in most of the software organizations.

CE: 6.4
Functional and Structural Testing

CE: 6.4.1 Functional Testing

- ***Functional testing techniques*** are ***validation techniques***, because *execution of the program* is essential for *the purpose of testing*.

- When the ***inputs are given to the program*** during execution, an ***observed output*** is generated***,*** *and this* ***observed output*** is compared with the ***expected output***.

- If the *observed output* is different than the *expected output*, then the situation is treated as a failure of the program.

- In functional testing, the source code (or program) is not considered ***for designing the test cases***. So, they have higher chances of making the program fail.

- These **test cases are designed** as per the ***functionality*** of the program and **ignores** the ***internal logic*** (or internal structure) of the program (or source code).

CE: 6.4.1 Functional Testing (Conti…)

- It is also known as ***black box testing***, because internal logic of the program is completely ignored.

- For Example:

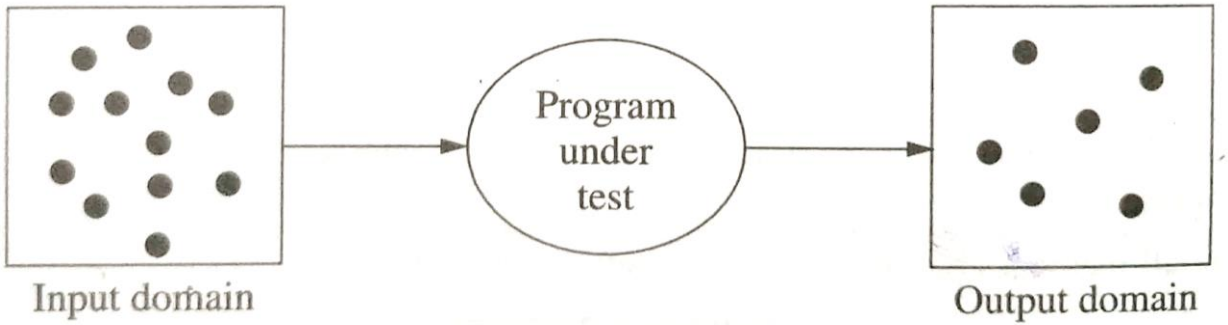


Figure 9.1 Functional (black box) testing.

- ✓ Every dot of the *input domain* represents input(s) and
- ✓ Every dot of the *output domain* represents output(s).
- ✓ Every dot of the *input domain* has a corresponding dot in the *output domain*.
- ✓ We can also observe the outputs behavior difference, due to the valid or invalid inputs for the purpose of program execution.

CE: 6.4.1 Functional Testing (Conti…)

- Ultimately, the main objective of *functional testing* is to provide the ways to design effective test cases to find errors in the program.

- It also includes…
  - ✓ Boundary Value Analysis
  - ✓ Equivalence Class Testing
  - ✓ Decision Table-Based Testing

- Other following testing are also considered as functional testing:
  - ✓ System Testing and Acceptance testing

- It is usually performed by the End user, Developer and Tester.

- And it is well suitable and efficient for the large code segments.

CE: 6.4.1 Functional Testing (Conti…)

- It is tough to automate. Because here the test and the programmer is only dependent on each other.

- But, if the modification is frequently in application, then the updation of automation test script is essential.

CE: 6.4.2 Structural Testing

- The ***structural testing*** is the opposite of ***functional testing***.
- In structural testing, the source code (or program) is considered ***for designing the test cases*** rather than ***the specifications of the inputs***.
- It **focuses on** the ***internal logic*** of the program and **ignores** the ***functionality*** of the program.
- While generating the test cases in structural testing, we may find complexity and weak area of the program, but with the help of it a clear and correct understanding of the source code is done.
- It is also known as ***white-box testing***, because it attempts to examine the source code thoroughly to understand it correctly.

CE: 6.4.2 Structural Testing (Conti…)

- It is also known as Clear box testing, Code-based testing, or Glass box testing.

- It includes…
  - ✓ Path testing,
  - ✓ Code coverage testing,
  - ✓ Analysis and Logic testing,
  - ✓ Nested loop testing, and similar techniques.

- From these, ***path testing*** is a popular structural testing technique.

CE: 6.4.2 Structural Testing (Conti…)

❖ **Path Testing:**

✓ In path testing, *the source code of the program* is converted into *a program graph*.

❑ **What is program graph?**

✓ A program graph is a graphical representation of the *source code*, where

- *statements of the program* are represented by ***nodes*** and
- *flow of control* is represented by ***edges***.

❑ **Joregenson (2007) has defined *program graph* as…**

✓ A program graph is a ***directed graph***, in which

- ✓ ***nodes*** are either *statements* or *fragments of a statement* and
- ✓ ***edges*** represent *flow of control*.

CE: 6.4.2 Structural Testing (Conti…)

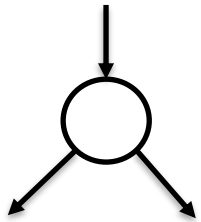
- ✓ The program graph provides a graphical view of the program and may become the foundation of many testing techniques.
- ✓ Further the program graph is converted into decision to decision (DD) path graphs.
- ✓ Both the graphs are commonly used in structural testing techniques for generating **test cases**.
- ✓ A program can be converted into a program graph using ***fundamental constructs***.

CE: 6.4.2 Structural Testing (Conti…)

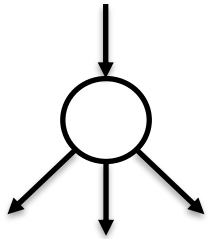
✓ The fundamental constructs of the program graph are:

Node

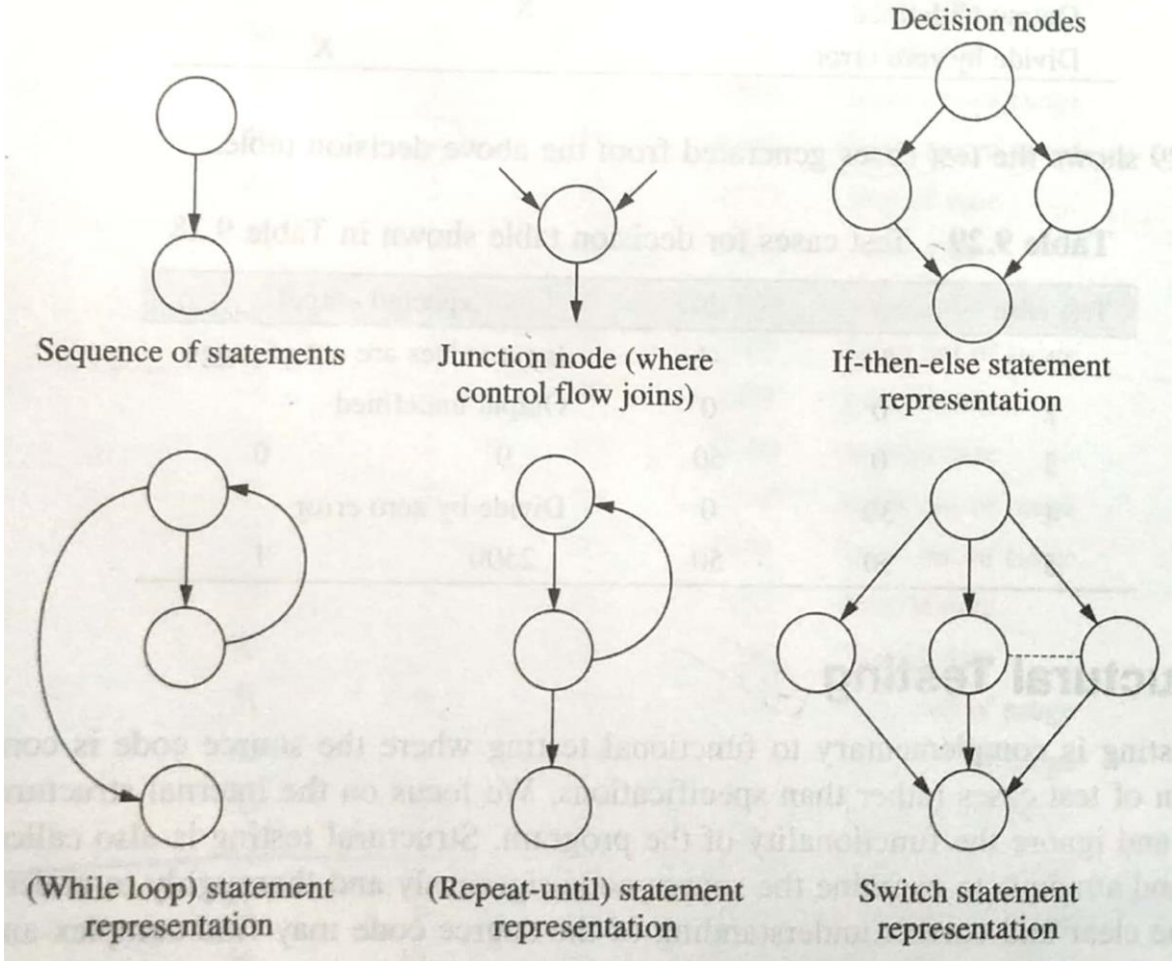
Edge



(One incoming edge and two outgoing edges)



(More than two outgoing edges)



CE: 6.4.2 Structural Testing (Conti…)

- Other following testing are also considered as structural testing:
  - ✓ Unit testing, Integration testing, Load testing, Stress testing and Performance testing.
- Ultimately, the main objective of *structural testing* is done to check the quality of the code.
- It is usually done by tester and developers.
- It also helps for removing the extra lines of code, which can bring in hidden defects.
- It is also easy to automate (by using automation tool), but an automated test cases can become useless if the code base is rapidly changing.

CE: 6.5
Levels of Testing

CE: 6.5 Levels of Testing

- There are four levels of testing. They are:

1. Unit Testing
2. Integration Testing
3. System Testing
4. Acceptance Testing

Software Testers are responsible

Customers are responsible

CE: 6.5.1 Unit Testing

- ***Unit testing*** is the first round of testing, where, an individual units (sections or parts of software or product) are tested using functional and structural testing technique.

- This type of testing is performed at developer's end to make sure that their program code is working well or not, to meet the user specifications.

- In unit testing, the focus is on encapsulation, because attributes and operations are combined in a class, and the operations within the class are the smallest available unit of testing.

- But, the operations as a unit are difficult to test, due to inheritance and polymorphism.

- Therefore, in unit testing, generally, ***classes are treated as a unit***; and functional and structural testing technique are equally applicable.

CE: 6.5.1 Unit Testing (Conti…)

- *Verification techniques* such as …
  - ✓ peer reviews,
  - ✓ inspections and
  - ✓ walkthroughs

  are easily applicable and may find a good number of faults.

- Other testing techniques like …
  - ✓ state-based testing,
  - ✓ path testing,
  - ✓ class testing,
  - ✓ boundary value analysis testing,
  - ✓ equivalence class testing and
  - ✓ decision table-based testing

  are also applicable.

CE: 6.5.1 Unit Testing (Conti…)

- Ultimately, the main aim of unit testing is to divide each and every part of the program; and test that the separate parts are working correctly or not.

CE: 6.5.2 Integration Testing

- At the integration level testing, two or more units are combined within a program and to test them as a group.

- There is no hierarchical control structure in object-oriented system. So the conventional integration testing techniques such as top down, bottom up and sandwitch integration may not be applicable.

- Basically, the meaning of integration testing is interclass testing.

- And there are three ways to carry out interclass testing. They are:
  1. Thread-based Testing
  2. Case-based Testing
  3. Cluster Testing

CE: 6.5.2 Integration Testing (Conti…)

1. Thread-based Testing:

- ✓ It integrates the set of classes that are required to respond to an input or a event for the system.
- ✓ When the input is given to the software, one or more classes are needed for execution, and such classes make a thread.
- ✓ There may be many such threads depending on the inputs.
- ✓ Each thread is integrated and tested individually.
- ✓ The expected output of every thread is calculated and is compared with the actual output. This technique is simple and easy to implement.

CE: 6.5.2 Integration Testing (Conti…)

2. Case-based Testing:

- ✓ It tests every basic and alternative paths of a **use case**.
- ✓ A path may require one or more classes for execution.
- ✓ Here, every use case path is tested, because of the involvement of many classes, an interclass issues are automatically tested.

3. Cluster Testing:

- ✓ In cluster testing, all classes are combined to show one collaboration.

- In all these three approaches, ***the classes are combined on the basis of logic*** and ***get executed to know the outcome***.
- The most popular and simple technique is the ***Thread-based Testing***.

CE: 6.5.3 System Testing

- *A **system testing*** is also known as a ***black box testing***.
- At the system level testing, a complete software system is tested.
- It is mostly done using *functional testing* techniques.
- It is performed after the *unit testing* and *integration testing only*.
- Here, a system is defined as the combination of software, hardware and other associated parts, which work together to provide the desire functionality.
- Like *functional testing*, *structural testing* techniques may also be used technically, but they are not very commonly used, due to the large size of the software.

CE: 6.5.3 System Testing (Conti…)

- For reviewing the source code and documents, *verification techniques* are normally used.
- The ***functional requirements*** of the software are tested under stated conditions.
- And the ***non-functional requirements*** such as stress, load, reliability, testability, performance, etc. are tested only at this level.
- Every stated functionally of the software is to be tested properly by keeping the customer's expectations in mind.
- After the whole completion of system testing only, the software gets ready for the customer.

CE: 6.5.4 Acceptance Testing

- The ***acceptance testing*** is carried out by the customer(s) or their authorized person to test the system as per their expectations.
- The customers testing strategy may range from *ad-hoc testing to a well-planned systematic testing*.
- The testing may be at the developer's site or the customer's site, that is depending on the mutual agreement. Generally, it is carried out at the customer's site only.
- If they use the software at the developer's site under the supervision of the developers, then it is known as ***alpha testing***.
- And if they distribute the software to the potential customers and ask them to use at their site in a free and independent environment, then it is known as ***beta testing***.

CE: 6.5.4 Acceptance Testing (Conti…)

- Ultimately, the purpose of acceptance testing is to test the software with an intention to accept and get the reasonable confidence about its usage and correctness.


Thank You