Once a program code is written, it must be tested to detect and subsequently handle all errors in it. A number of schemes are used for testing purposes.

Another important aspect is the fitness of purpose of a program that ascertains whether the program serves the purpose which it aims for. The fitness defines the software quality.

**Testing Object-Oriented Systems**

Testing is a continuous activity during software development. In object-oriented systems, testing encompasses three levels, namely, unit testing, subsystem testing, and system testing.

**Unit Testing**

In unit testing, the individual classes are tested. It is seen whether the class attributes are implemented as per design and whether the methods and the interfaces are error-free. Unit testing is the responsibility of the application engineer who implements the structure.

**Subsystem Testing**

This involves testing a particular module or a subsystem and is the responsibility of the subsystem lead. It involves testing the associations within the subsystem as well as the interaction of the subsystem with the outside. Subsystem tests can be used as regression tests for each newly released version of the subsystem.

**System Testing**

System testing involves testing the system as a whole and is the responsibility of the quality-assurance team. The team often uses system tests as regression tests when assembling new releases.

**Object-Oriented Testing Techniques**

**Grey Box Testing**

The different types of test cases that can be designed for testing object-oriented programs are called grey box test cases. Some of the important types of grey box testing are:

- **State model based testing**: This encompasses state coverage, state transition coverage, and state transition path coverage.

- **Use case based testing**: Each scenario in each use case is tested.

- **Class diagram based testing**: Each class, derived class, associations, and aggregations are tested.

- **Sequence diagram based testing**: The methods in the messages in the sequence diagrams are tested.

**Techniques for Subsystem Testing**

The two main approaches of subsystem testing are:

- **Thread based testing**: All classes that are needed to realize a single use case in a subsystem are integrated and tested.

- **Use based testing**: The interfaces and services of the modules at each level of hierarchy are tested. Testing starts from the individual classes to the small modules comprising of classes, gradually to larger modules, and finally all the major subsystems.
Categories of System Testing

- **Alpha testing**: This is carried out by the testing team within the organization that develops software.
- **Beta testing**: This is carried out by select group of co-operating customers.
- **Acceptance testing**: This is carried out by the customer before accepting the deliverables.

Software Quality Assurance

Software Quality

Schulmeyer and McManus have defined software quality as “the fitness for use of the total software product”. A good quality software does exactly what it is supposed to do and is interpreted in terms of satisfaction of the requirement specification laid down by the user.

Quality Assurance

Software quality assurance is a methodology that determines the extent to which a software product is fit for use. The activities that are included for determining software quality are:

- Auditing
- Development of standards and guidelines
- Production of reports
- Review of quality system

Quality Factors

- **Correctness**: Correctness determines whether the software requirements are appropriately met.
- **Usability**: Usability determines whether the software can be used by different categories of users *beginners, non-technical, and experts*.
- **Portability**: Portability determines whether the software can operate in different platforms with different hardware devices.
- **Maintainability**: Maintainability determines the ease at which errors can be corrected and modules can be updated.
- **Reusability**: Reusability determines whether the modules and classes can be reused for developing other software products.

Object-Oriented Metrics

Metrics can be broadly classified into three categories: project metrics, product metrics, and process metrics.

Project Metrics

Project Metrics enable a software project manager to assess the status and performance of an ongoing project. The following metrics are appropriate for object-oriented software projects:

- Number of scenario scripts
- Number of key classes
- Number of support classes
- Number of subsystems

Product Metrics
Product metrics measure the characteristics of the software product that has been developed. The product metrics suitable for object-oriented systems are:

- **Methods per Class**: It determines the complexity of a class. If all the methods of a class are assumed to be equally complex, then a class with more methods is more complex and thus more susceptible to errors.

- **Inheritance Structure**: Systems with several small inheritance lattices are more well-structured than systems with a single large inheritance lattice. As a thumb rule, an inheritance tree should not have more than $7 \pm 2$ number of levels and the tree should be balanced.

- **Coupling and Cohesion**: Modules having low coupling and high cohesion are considered to be better designed, as they permit greater reusability and maintainability.

- **Response for a Class**: It measures the efficiency of the methods that are called by the instances of the class.

### Process Metrics

Process metrics help in measuring how a process is performing. They are collected over all projects over long periods of time. They are used as indicators for long-term software process improvements. Some process metrics are:

- **Number of KLOC** *KiloLines of Code*

- **Defect removal efficiency**

- **Average number of failures detected during testing**

- **Number of latent defects per KLOC**

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js