5.Levels of Testing

IT6206 – Software Quality Assurance

Level III - Semester 6

Overview

- Test type are introduced as a means of clearly defining the objective of a certain test level for a program or project.
- This section will discuss about both tests that focus on the functionality of the system (Functional Testing) and tests that focus on non-functional attributes of a system (Non-Functional Testing)
- It is important to think about different types of testing because testing functionality of the component or system may not be sufficient at each level to meet overall test objectives.

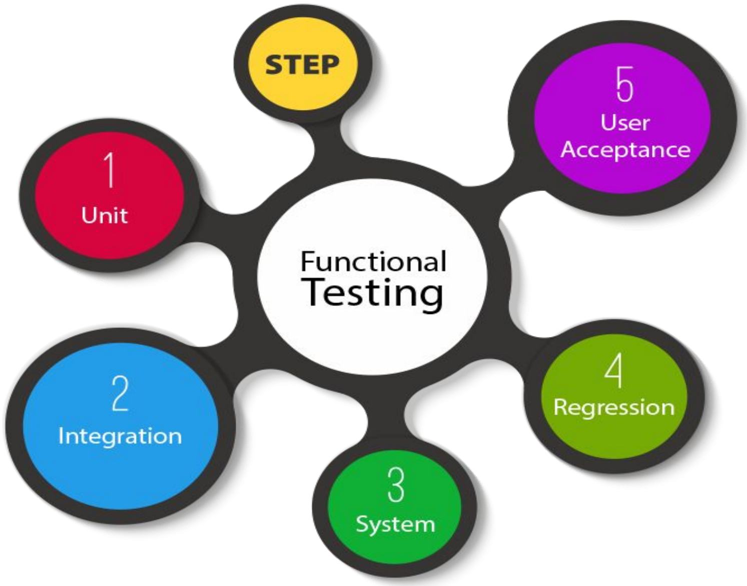
Intended Learning Outcomes

At the end of this lesson, you will be able to;

- Compare functional and non-functional testing
- Recognize that functional and non-functional tests occur at any level.
- Knowing the appropriate testing types for different stages of the software development life cycle.
- Understanding the challenges and limitations of software testing and how to address them.

List of sub topics

5.1 Functional Testing



Source: https://www.zentao.pm/blog/things-you-should-know-about-functional-testing-826.html

5.1 Functional Testing

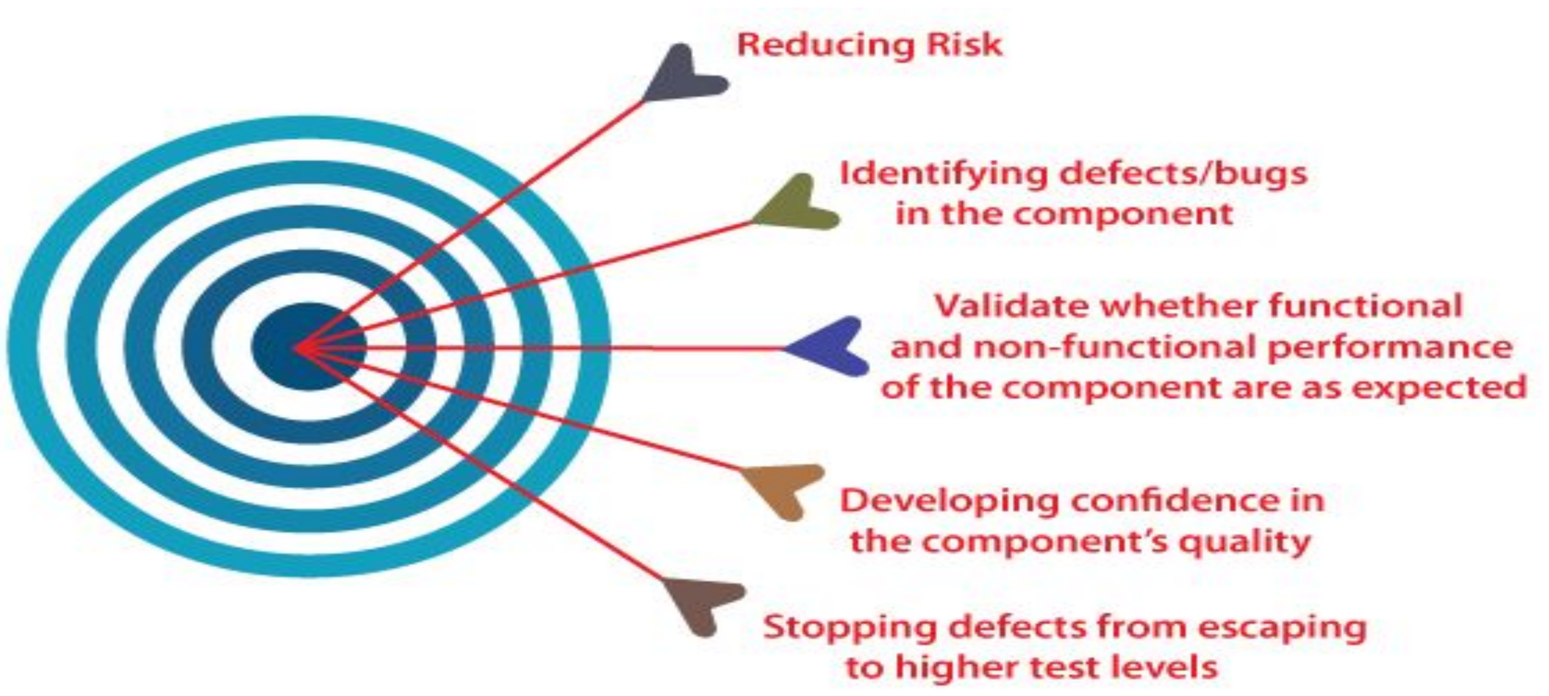
Functional Testing – *A type of software testing that validates the software system against functional requirements/specifications*.

- The main purpose of **Functional tests :** To test *each function* of the *software application*, by providing appropriate input, verifying the output against the Functional requirements.
- Functional testing can be done either manually or using automation.
- Functional testing checks;
  - User Interface
  - APIs
  - Database
  - Security
  - Client / Server communication and etc.

5.1.1 Component / Unit Testing

Component Testing also known as “unit” or “module” testing searches for defects in, and verifies the functioning of, software items that are separately testable.

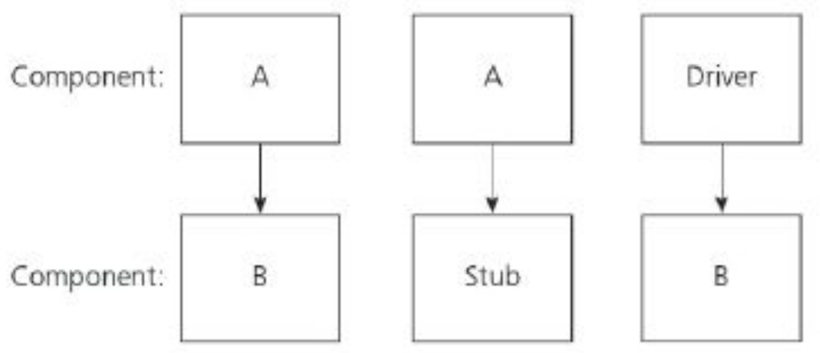
Goals of Component Testing


Reducing Risk
Identifying defects/bugs in the component
Validate whether functional and non-functional performance of the component are as expected
Developing confidence in the component's quality
Stopping defects from escaping to higher test levels

Source: https://static.javatpoint.com/tutorial/software-testing/images/component-testing2.png

5.1.1 Component / Unit Testing

- Component testing may be done in isolation from the rest of the system depending on the context of the development life cycle and the system.
- “Mock objects” or “stubs” and “drivers” are used to replace the missing software and simulate the interface between software components in a simple manner.
- Stub: A stub is called from software component to be tested.
- Driver: A driver calls the component to be tested.



Source: Foundations of Software Testing ISTQB Certification,Rex Black, Erik van Veenendaal, Dorothy Graham (page 49)

5.1.1 Component / Unit Testing

Component Testing: Test Basis

"What is this particular component supposed to do?"

Work products that can be used as a test basis for a component testing:

- Detailed design
- Code
- Data model
- Component specifications (if available).

5.1.1 Component / Unit Testing

Component Testing: Test Objects

"What are we actually testing at this level?"

Typical test objects for component testing:

- Components themselves, units or modules
- Code and data structures
- Classes
- Database models

5.1.1 Component / Unit Testing

Component testing: Typical Defects and Failures

Defects and failures that can typically be revealed by component testing:

- Incorrect functionality
- Data flow problems
- Incorrect code or logic

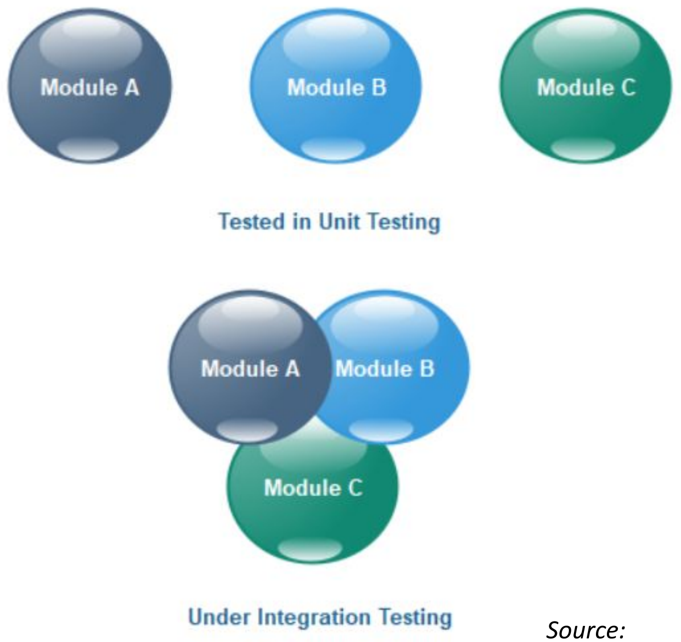
Defects are typically fixed as soon as they are found, without formally recording them in a defect management tool.

  - If such defects are recorded, it can provide useful information for root cause analysis.

5.1.2 Integration Testing

Integration Testing is a type of testing where software modules are integrated and logically tested as a group.

- A typical software project consists of multiple software modules created by various programmers.
- Objective of integration testing is to find issues with how various software modules interact when they are combined.



Source: https://static.javatpoint.com/tutorial/software-testing/images/integration-testing.png

5.1.2 Integration Testing

Levels of Integration Testing

- **Component integration testing**
    - Focuses on the interactions and interfaces between integrated components.
    - Performed after component testing.
    - Generally automated.
    - Often the responsibility of developers.
- **System integration testing**
    - Focuses on the interactions and interfaces between systems, packages and microservices.
    - Can also cover interactions with, and interfaces provided by external organizations (ex- web services)
    - May performed system testing or in parallel with ongoing system test activities.
    - Often the responsibility of testers.

5.1.2 Integration Testing

Integration testing: Objectives

- Reducing risk
  - Ex – Testing high risk integrations first.
- Verifying whether or not functional and non-functional behaviors of the interfaces are as they should be.
- Building confidence in the quality of the interfaces
- Finding defects in the interfaces themselves or in the components or system being tested together.
- Preventing defects from escaping to later testing.

5.1.2 Integration Testing

Integration testing: Test Basis

“How are these components or systems supposed to work together and communicate?”

Work products that can be used as a test basis:

- Software and system design
- Sequence diagrams
- Interface and communication protocol specifications
- Use cases
- Architecture at component of system level
- Workflows
- External interface definitions

5.1.2 Integration Testing

Integration testing: Typical defects and failures.

Defects and failures that can typically be revealed by component integration testing include:

- Incorrect data, missing data or incorrect data encoding
- Incorrect sequencing or timing of interface calls
- Interface mismatch
- Failures in communication between components
- Unhandled or improperly handled communication failures between components
- Incorrect assumptions about the meaning, units or boundaries of the data being passed between components

5.1.2 Integration Testing

Defects and failures that can typically be revealed by system integration testing include:

- Inconsistent message structures between systems.
- Incorrect data, missing data or incorrect data encoding.
- Interface mismatch.
- Failures in communication between systems
- Unhandled or improperly handled communication failures between systems.
- Failures to comply with mandatory security regulations.
- Incorrect assumptions about the meaning, units, or boundaries of the data being passed between systems.

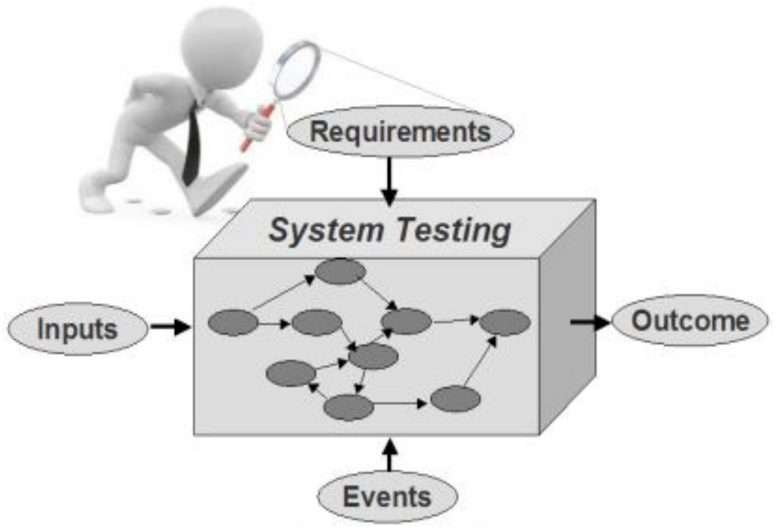
5.1.3 System Testing

System Testing is concerned with the behavior of the whole system/product as defined by the scope of a development project or product.

System testing may include tests based on,

- Risks
- requirement specifications
- business process
- use cases etc.

I



Source: https://artoftesters.files.wordpress.com/2014/04/systm-test.png?w=413&h=197

5.1.3 System Testing

Objectives of System Testing

- Reducing risk
- Verifying whether the design of the system’s functional and non-functional behaviors complies with the requirements of the customer.
- Validating that the system is complete and will work as expected.
- Building confidence in the quality of the system as a whole.
- Finding defects.
- Preventing defects from escaping to later testing or to production.

5.1.3 System Testing

System Testing: Test Basis

“What should the system as a whole would be able to do?”

Work products that can be used as a test basis for system testing

- System and software requirement specification.
- Risk analysis report.
- Use cases.
- State diagrams.
- Models of system behavior.
- System and user manuals.
- Epics and user stories

5.1.3 System Testing

System testing: Test Objects

“What are we actually testing at this level”

Typical test objects of system testing:

- Applications
- Hardware/Software Systems
- Operating Systems
- System Under Test (SUT)
- System configuration and configuration data.

These objects are fully integrated systems. System could be a software (like MS Office), or it could be an Operating System (like Windows 11), etc.

5.1.3 System Testing

System testing: Typical defects and failures

System testing is usually the final test from the development team which will ensure that the system delivered will meet specifications.

Examples of defects and typical failures for system testing:

- Incorrect calculations
- Incorrect control or data flow within the system
- Incorrect/Unexpected system functional or non-functional behavior
- Failure to carry out end to end tasks.
- Failure of the system to work properly in the production environments
- Failure of the system to work as described in the system and user manuals

5.1.3 System Testing

Approaches and Responsibilities

- System testing is most often carried out by specialist testers, that form a dedicated and sometimes independent test team within development.
- Testing team is involved from the start so they have a complete understanding of the system.
- System test scenarios re usually end-to-end tests that ca cove full system.
- Architects and product owners usually review these scenarios.

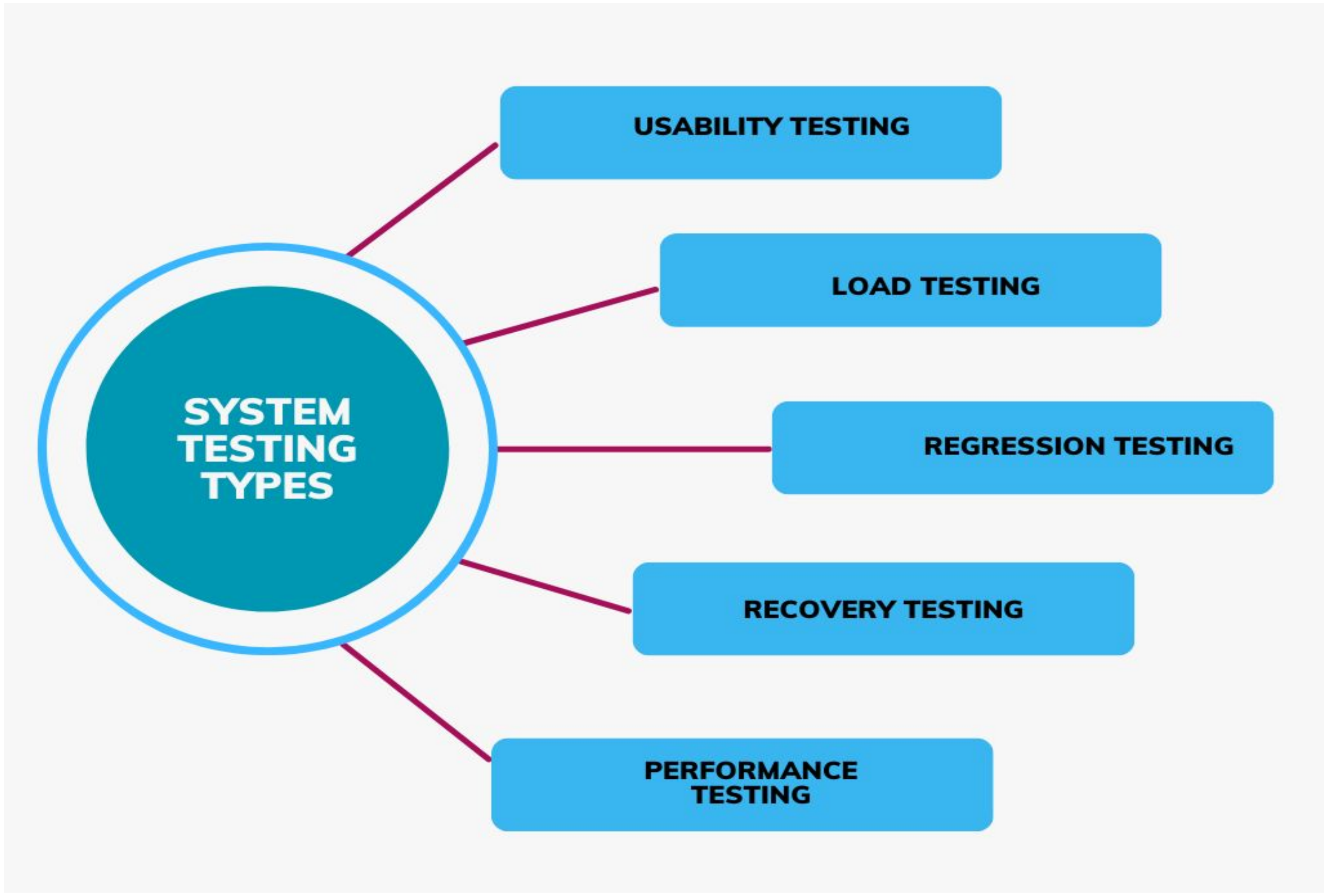
5.1.3 System Testing

Responsibilities of System testing team.

- Understanding system flows and create high-level user journey.
- Create detailed end-to-end system cases.
- Identify and generate any test data required to execute the test cases.
- Coordinate with Scrum teams to ensure adequate support is available to fix defects.

5.1.3 System Testing

Some of the system testing types



5.1.4 User Acceptance Testing

User Acceptance Testing conducted in a real or simulated operational environment by *intended users* focusing on their needs, requirements and business processes.

The aim of testing is to build confidence that the system will enable the users to do what they need to do in an efficient way.

Types of **Acceptance Testing**

- Contract Acceptance Testing (CAT)
- Regulations Acceptance Testing (RAT)
- Operational Acceptance Testing (OAT)
- Operational Acceptance Testing (OAT)
- Alpha and Beta Testing

5.1.4 User Acceptance Testing

Acceptance Testing: Test Basis

“How do we know that the system is ready to be used for real ?”

Examples of work products that can be used as a test basis

- Business process
- User/ Business requirements
- Use cases
- System requirements
- Risk analysis reports
- Regulations, legal contracts and standards

5.1.4 User Acceptance Testing

Acceptance Testing: Test Objects

“What are we actually testing at this level”

Typical test objects for acceptance testing:

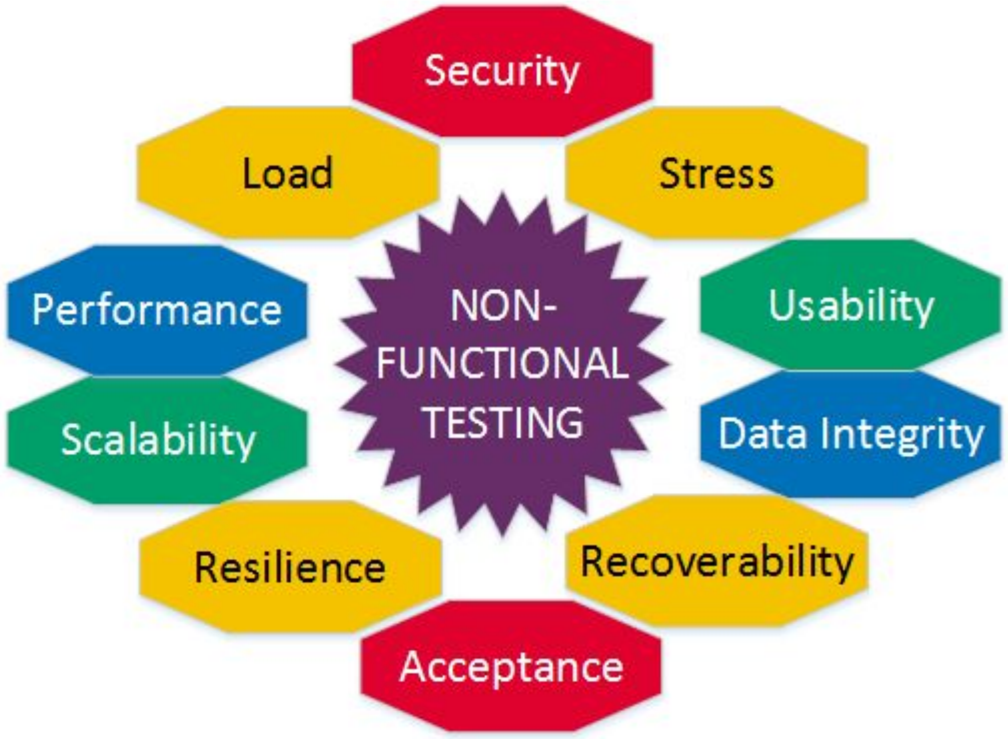
- System Under Test (SUT)
- System configuration and configuration data
- Business processes for fully integrated system
- Recovery systems and hot sites
- Operational and maintenance processes
- Forms
- Reports

5.1.4 User Acceptance Testing

Acceptance Testing: Typical defects and failures

- System workflows do not meet business or user requirements.
- Business rules are not implemented correctly
- System does not satisfy contractual or regulatory requirements.

5.2 Non-Functional Testing



Source: https://testinggenez.com/wp-content/uploads/2019/10/Non-Functional-Testing.png?x69794

5.2 Non-Functional Testing

Non – Functional testing conducted to evaluate the compliance of a component or a system with non-functional requirements.

Characteristics of non-functional testing

- Should be measurable (therefore no place for subjective characterization).
- Ensure the quality attributes are identified correctly .
- Important to prioritize the reqiurements

5.2 Non-Functional Testing

Non – functional testing parameters

- Interoperability
- Performance
- Scalability
- Security
- Loading Capacity
- Stability
- Reliability and etc.

5.2.1 Interoperability Testing

- **Interoperability Testing** is designed to ensure that the software can interact with other software components and systems.
- Testing means to prove that end-to-end functionality between two communicating systems is as specified by the requirements.
- Types of Interoperability Testing
  - Data type Interoperability Testing
  - Semantic Interoperability Testing
  - Physical Interoperability Testing
  - Protocol Interoperability Testing
  - Data format Interoperability Testing

5.2.1 Interoperability Testing

Interoperability Testing could be particularly significant for:

- Systems based on the Internet of Things (IoT)
- Web services with connectivity to other systems
- Applications based on systems of systems
- Commercial off-the-shelf (COTS) software products

5.2.2. Performance Testing

Performance tests are designed to determine the performance of the actual system compared to the expected one.

- Performance metrics needed to be measured vary from application to application.
- Tests are designed for testing the speed, response time, stability, reliability, scalability, and resource usage of a software application.
- Main purpose is to identify and eliminate the performance bottlenecks in the software application.

5.2.2. Performance Testing

Example Performance Test cases

- Check the maximum number of users that the application can handle before it crashes.
- Check CPU and memory usage of the application and the server under peak load conditions.
- Verify the response time of the application under low, normal, moderate and heavy load conditions.

5.2.3 Scalability Testing

Scalability Testing is designed to verify that the system can scale up to its engineering limits.

Scalability testing is done to determine:

- how the application scales with increasing workload.
- The user limit for a web application.
- End user experience under load
- Server-side robustness and degradation.

5.2.3 Scalability Testing

Few Scalability Testing Attributes:

- Response Time
- Throughput
- Time (Session time, transaction time, task execution time and etc.)
- Network usage
- Memory usage
- Performance measurement with number of users
- Performance measurement under load

5.2.4 Stress Testing

Stress Testing is deigned to evaluate and determine the behavior of a software component under extremely heavy load conditions.

Stress testing ensures that the system can perform acceptably under worst-case conditions.

The recovery mechanism should be invoked if the limit exceeds and system fails.

Tests are targeted to bring out the problems associated with:

- Memory Leak
- Buffer allocation and memory carving

5.2.4 Stress Testing

Best way to identify system bottlenecks is to perform stress testing from different locations inside and outside the system.

- Start with innermost components and progressively move outward.
- Then test from remote locations far outside the system.
- After all components are tested beyond their highest capacity, test the full system.
- Incrementally increase the load until the system fails.
- Observe the causes and locations of failures.

Source https://www.itij.com/sites/default/files/styles/og_image/public/2021-08/stress-testing-insurers-during-covid-19.jpg?h=3a490b08&itok=RsT60Yf3

5.2.5 Security Testing

Security Tests are designed to verify that the system meets security requirements: *confidentiality, integrity and availability.*

The objective of security testing is to demonstrate:

- The software behaves securely and consistently under all conditions.
- If the software fails, the failure does not leave the system, its data or its resources to attack.
- Obscure areas of code and dormant functions cannot be compromised or exploited.
- Interfaces and interactions among components at the application, middleware and operating system levels are consistently secure.

5.2.5 Security Testing

The popularity of the internet and wireless data communication technologies have created new types of security threats such as,

- Un-authorized access to the wireless data networks
- Eavesdropping on transmitted data
- Denial of service attack

Tests are designed to ensure new techniques developed to combat these kind of threats work.

5.2.5 Security Testing

Few useful types of security tests:

- Verify that only authorized accesses to the system are permitted. This may include authentication of user ID and password and verification of expiry of a password
- Verify that illegal reading of files, to which the perpetrator is not authorized, is not allowed.
- Ensure that virus checkers prevent or curtail entry of viruses into the system.

5.2.6 Load and Stability Testing

Load and Stability Tests are designed to ensure that the system remains stable for a long period under full load.

When a large number of users are introduced with incompatible systems and applications run for month without restarting following problems may occur

- The system slow down
- The system encounters functionality problems
- The system silently fails over
- The system crash altogether

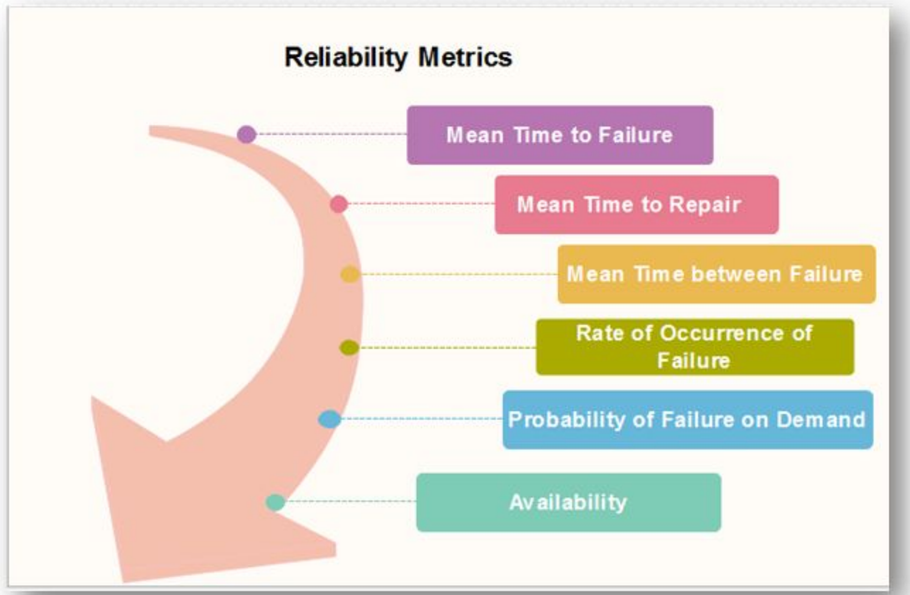
5.2.6 Load and Stability Testing

- Load and stability testing typically involves exercising the system with virtual users and measuring the performance to verify whether the system can support the anticipated load.
- Testing helps one to understand the ways the system will fare in real-life situations.
- With such an understanding, one can anticipate and even prevent load-related problems.
- In load and stability testing, the *objective* is to ensure that the system can operate on a large scale for several months,

5.2.7. Reliability Testing

Reliability tests are designed to measure the ability of the system to remain operational for long periods of time.

The reliability of a system is typically expressed in terms of **Mean Time to Failure (MTTF)**.


Reliability Metrics
Mean Time to Failure
Mean Time to Repair
Mean Time between Failure
Rate of Occurrence of Failure
Probability of Failure on Demand
Availability

Source: https://static.javatpoint.com/tutorial/software-engineering/images/software-engineering-reliability-metrics.png

5.2.7. Reliability Testing

Testing process

- Record the time intervals of successive failures ($t_1, t_2, \ldots, t_i$) as progressing through the system testing phase. The average of all the *i* time interval is called MTTF.
- After failure observed developers fix the defects which consumes time (repair time). Average of all repair times known as the **mean time to repair (MTTR)**
- Calculate **mean time between failures (MTBF)**

$$\boxed{\text{MTBF} = \text{MTTF} + \text{MTTR}}$$

5.2.8 Regression Testing

Regression Testing is to verify that no defect has been introduced into the unchanged portion of a system due to changes made elsewhere in the system.

During system testing, many defects are revealed and code is modified to fix those. There could be four different resulting scenarios

- The reported defect is fixed.
- The reported defect could not be fixed.
- The reported defect has been fixed, but something used to work before has been failing.
- The reported defect could not be fixed, and something used to work before has been failing.

5.2.8 Regression Testing

Considering the above scenarios it appears straightforward to re execute every test case from version n-1 to version n before testing anything new, Which is highly expensive.

Moreover, new software versions often feature many new functionalities in addition to the defect fixes.

Regression is an expensive task; a subset of test cases is carefully selected from existing test suite to,

- Maximize the likelihood of uncovering new defects
- Reduce the cost of testing

5.2.9 Documentation Testing

Documentation testing means verifying the technical accuracy and readability of the user manuals, including the tutorials and the on-line help.

Documentation Testing Levels

- Read Test: Documentation is reviewed for clarity, organization flow, and accuracy without executing the documented instructions on the system.
- Hands-On Test: The on-line help is exercised and the error messages verified to evaluate their accuracy and usefulness.
- Functional Test: The instructions in the documentation are followed to verify that the system works as it has been documented.

5.2.9 Documentation Testing

The following concrete tests are recommended for documentation testing:

- Read all documentation to verify
  - Correct use of grammar,
  - Consistent use of terminology
  - Appropriate use of graphics
- Verify that the glossary uses a standard, commonly accepted terminology
- Verify that there exists an index for each of the documents.
- Verify that there is no internal inconsistency within the documentation.

List Continued to Next Slide

5.2.9 Documentation Testing

- Verify that the on-line and printed versions of the documentation are same.
- Verify the troubleshooting guide by inserting error and then using the guide to troubleshoot the error.
- Verify the installation procedure by executing the steps described in the manual in a real environment.
- Verify the configuration section of the user guide by configuring the system as described in the documentation.

Summary

Functional Testing	• Focuses on verifying whether a software system meets its functional requirements and performs as intended.
Non-Functional Testing	• assesses the performance, usability, security, and other aspects of a system that are not related to its functionality.