1. **Learning Objectives**
   1. **Primary Objectives**
2. To contribute to quality assurance of projects.
3. To manage work to meet requirements.
4. To work effectively with colleagues
5. To established interest in industrial/commercial activities.
6. To gain work experience enabling the student to apply what he learnt in college and acquire new skills.
   1. **Secondary Objectives**
7. Fundamentals of Testing.
8. Testing Throughout SDLC.
9. Static Techniques.
10. Test Design Techniques.
11. Test Management.
12. Tools Support.
13. **WEEKLY OVERVIEW OF OJT ACTIVITIES**

|  |  |  |  |
| --- | --- | --- | --- |
| **WEEK NO** | **DATE** | **DAY** | **NAME OF THE TOPIC/MODULE COMPLETED** |
| **WEEK**  **NO - 1** | 05-10-2021 | Tuesday | What is Software/Software Testing |
| 05-10-2021 | Tuesday | Need of Software Testing |
| 05-10-2021 | Tuesday | Seven Principle of Testing |
| 06-10-2021 | Wednesday | Psychology of Testing |
| 06-10-2021 | Wednesday | Code of Ethics |
| 07-10-2021 | Thursday | Fundamental Test Processes |
| 07-10-2021 | Thursday | Test Planning |
| 07-10-2021 | Thursday | Test Specification |
| 07-10-2021 | Thursday | Test Execution |
| 07-10-2021 | Thursday | Test Recording |
| 07-10-2021 | Thursday | Test Completion |
| 08-10-2021 | Friday | What is SDLC |
| 08-10-2021 | Friday | What is STLC |
| 08-10-2021 | Friday | SDLC Models - Waterfall, Spiral, V, Agile |
| 09-10-2021 | Saturday | How to Derive Expected Result |
| 09-10-2021 | Saturday | Test Case Format |
| 09-10-2021 | Saturday | Important Aspects of Test Cases |
| 09-10-2021 | Saturday | Software Testing Levels & Types |
| 09-10-2021 | Saturday | Maintenance Testing |
| 09-10-2021 | Saturday | Software Quality - QA (Static) & QC (Dynamic) |
| 09-10-2021 | Saturday | Review Process |
| 09-10-2021 | Saturday | Tools |
| **WEEK**  **NO - 2** | 11-10-2021 | Monday | Testing Methods Black box |
| 11-10-2021 | Monday | Black Box Techniques |
| 11-10-2021 | Monday | Decision Tables |
| 11-10-2021 | Monday | White Box Testing |
| 11-10-2021 | Monday | Grey Box Testing & Comparison |
| 12-10-2021 | Tuesday | Test Organisation |
| 12-10-2021 | Tuesday | Test Estimation |
| 12-10-2021 | Tuesday | How to Decide Priority |
| 12-10-2021 | Tuesday | Test Monitoring & Control |
| 12-10-2021 | Tuesday | Test Progress |
| 12-10-2021 | Tuesday | Configuration Management |
| 13-10-2021 | Wednesday | Risk & Testing |
| 13-10-2021 | Wednesday | What is Defect |
| 13-10-2021 | Wednesday | Defect Priority & Severity |
| 13-10-2021 | Wednesday | Defect Life Cycle |
| 13-10-2021 | Wednesday | Attributes of Defect |
| 14-10-2021 | Thursday | Stubs & Drivers |
| 14-10-2021 | Thursday | Types of Test Tools |
| 14-10-2021 | Thursday | Effective Use of Tools |
| 14-10-2021 | Thursday | Introducing Tool into Organisation |

1. **Introduction**

This Year’s OJT was on QA Engineer and to learn the role of QA Engineer, I learnt a lot of skills like Scenarios Writing, Test Cases, Test Management, Defect logging, Testing, Test Leading, Bug Reporting and FRS. As all of them were new skills for me they were quite challenging but, at the end, I managed to understand them properly thanks to our excellent lecturer Mr. Shashikant Karulkar.

On-the-job training (OJT) is training that is delivered while an individual is performing tasks or processes related to their particular occupation. The student typically performs tasks that are essential to their job function with the supervision of a manager, coach or mentor. This type of training is typically used to broaden a student’s skill set and to increase productivity.

On-the-job training is an important topic of human resource management. It helps develop the individual and the prosperous growth of the organization. On the job training is a form of training provided at the workplace. During the training, employees are familiarized with the working environment they will become part of organization. Employees also get a hands-on experience using machinery, equipment, tools, materials, etc. Part of on-the-job training is to face the challenges that occur during the performance of the job. An experienced employee or a manager is executing the role of the mentor who through written or verbal instructions and demonstrations are passing on his/her knowledge and company-specific skills to the new employee. Executing the training on at the job location, rather than the classroom, creates a stress-free environment for the employees. On-the-job training is the most popular method of training not only in the United States but in the most of the developed countries, such as the United Kingdom, China, Russia, etc. Its effectiveness is based on the use of existing workplace tools, machines, documents and equipment, and the knowledge of specialists who are working in this field. On-the-job training is easy to arrange and manage and it simplifies the process of adapting to the new workplace. On-the-job training is highly used for practical tasks. It is inexpensive, and it doesn’t require special equipment that is normally used for a specific job. Upon satisfaction of completion of the training, the employer is expected to retain participants as regular employees.

1. **OJT Discussion**

**Subsection 4.1: How the Objectives were achieved?**

The objectives were achieved by performing the job-related activities in a timely and systematic manner.

1. First, an introduction given about software testing.
2. Why software testing necessary – performed some activities like writing scenarios on ATM machine.
3. Introduction was given about test planning, test specification, test execution.
4. Activities related writing scenarios, execution, bug reporting were performed.
5. Introduction was given about white box, black box, grey box testing techniques.
6. Contribute to quality assurance of project.
7. Provide data/information in standard formats.

**Subsection 4.2: What skills were acquired as per the job role QP during the OJT?**

**4.2.1 What is Software Testing?**

1. Computer software or just software is any set of machine-readable instructions that directs computers processor to perform specific operations.
2. Software is a set of programs, which is design to perform a well-defined function. A program is a sequence of instruction written to meet a particular requirement.
3. In other words, Computer software is a set of instructions in the form of programs, procedures, functions, data etc. executed to meet desired requirements.
4. IEEE Definitions –
5. Software testing is the process of analysing a software item to detect the differences between existing and required conditions (that is bugs) and to evaluate the features of the software item.
6. Reliability is the ability of a system or component to perform its required functions under started conditions for specified period of time.
7. Software testing can also be started as the process of validating and verifying that a software program/application/product:
8. Meets the business and technical requirements that guided its design and development.
9. Works as expected.
10. Can be implemented with the same characteristics.
11. Testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test.
12. Testing involves any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results.

**4.2.2 Need of Software Testing**

1. Anything and everything created by humans is prone to have faults and defects. Some errors/faults do not have any severe impact but some faults are critical, which acts as a show-stopper, or will break the whole system.
2. So, in such cases it is important that all such high severity defects/faults are detected and resolved in advance, before the s/w is launched or implemented.
3. Perfection is very difficult.
4. People make errors, Errors can cause problems.
5. Deliverables can be defective.
6. Defects can cause failures; Failure can be a big problem.
7. To Err is human.
8. Error - mistake made, we are not perfect.
9. Fault - the result of an error. Also commonly known as bugs or defects.
10. Failure - the result of wrong behaviour, deviation from the expected.
11. The solution is to TEST (validate) at each stage.
12. Unfortunately, nobody is perfect and we all make mistakes. Sometimes this can be a misunderstanding of what is required of us, we are working under pressure such as delivery deadlines or sometimes we just get it wrong! Errors made early have a nasty habit of growing and getting worse.
13. If errors are present in software, they may cause problems immediately but they can also lie dormant and it may take a while before they surface. When errors have been made and lie undiscovered, the delivered software will be defective, which can lead to failures, which could mean severe problems for the business.
14. E.g. A leading supermarket chain had a '2 for 1' promotion on a hair conditioner. A customer noticed that when she had purchased two conditioners, not only had she only been charged for one of them, but also £3.50 had been deducted from her bill. She performed some further 'research' and discovered that for every one of the special purchases, £3.50 was deducted from the final bill.

Over the next days the woman visited all of the supermarket branches in her area and cleared Ore shelves of the conditioners. She estimated she saved herself £1000 ! This was caused by absence of a simple validation in software

**4.2.3 Seven Principles of Testing**

1. Testing shows presence of defects - Testing can show that defects are present, but cannot prove that there are no defects. Testing reduces the probability of undiscovered defects remaining in the software but, even if no defects are found, it is not a proof of correctness.
2. Exhaustive testing is impossible - Testing everything (all combinations of inputs and preconditions) is not feasible except for trivial cases. Instead of exhaustive testing, risk analysis and priorities should be used to focus testing efforts.
3. Early testing - To find defects early, testing activities shall be started as early as possible in the software or system development life cycle, and shall be focused on defined objectives.
4. Defect clustering - Testing effort shall be focused proportionally to the expected and later observed defect density of modules. A small number of modules usually contains most of the defects discovered during pre-release testing, or is responsible for most of the operational failures. (80:20 principle)
5. Pesticide paradox - If the same tests are repeated over and over again, eventually the same set of test cases will no longer find any new defects. To overcome this “pesticide paradox”, test cases need to be regularly reviewed and revised, and new and different tests need to be written to exercise different parts of the software or system to find potentially more defects.
6. Testing is context dependent - Testing is done differently in different contexts. For example, safety-critical software is tested differently from an e-commerce site. E.g. Testing Bill Payment, Rocket System, Mobile Game – Different considerations are applicable in each case.
7. Absence-of-errors fallacy - Finding and fixing defects does not help if the system built is unusable and does not fulfil the users’ needs and expectations.

**4.2.4 Psychology of Testing**

1. Why do we test?
2. Primarily to find faults in the software, rather than demonstrating correctness.
3. This can be perceived as being a destructive process, not constructive.
4. How does this fit with the mindsets of the developer and tester?
5. Developer Characteristics
6. Specialized
7. Trained
8. Creative
9. Sensitive to criticism
10. Tester Characteristics
11. Trained
12. Experienced
13. Methodical
14. Persistent
15. Happy when they find faults
16. Good Communicators
17. Independence
18. Independent testing is more effective. The author should not test their own work.
19. Assumptions made are carried into testing.
20. People see what they want to see.
21. There can be emotional attachment.
22. There can be a vested interest not to find faults.

**4.2.5 Code of Ethics**

Involvement in software testing enables individuals to learn confidential and privileged information. A code of ethics is necessary, among other reasons to ensure that the information is not put to inappropriate use. The ISTQB states the following code of ethics:

PUBLIC - Certified software testers shall act consistently with the public interest.

CLIENT AND EMPLOYER - Certified software testers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest.

PRODUCT - Certified software testers shall ensure that the deliverables they provide (on the products and systems they test) meet the highest professional standards possible.

JUDGMENT- Certified software testers shall maintain integrity and independence in their professional judgment.

Involvement in software testing enables individuals to learn confidential and privileged information. A code of ethics is necessary, among other reasons to ensure that the information is not put to inappropriate use. The ISTQB states the following code of ethics:

MANAGEMENT - Certified software test managers and leaders shall subscribe to and promote an ethical approach to the management of software testing.

PROFESSION - Certified software testers shall advance the integrity and reputation of the profession consistent with the public interest.

COLLEAGUES - Certified software testers shall be fair to and supportive of their colleagues, and promote cooperation with software developers.

SELF - Certified software testers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

**4.2.6 Fundamental Test Processes**

1. Test Planning
2. Determines the scope of the testing.
3. If company or industry policy, strategy or regulations refer to testing, the test plan shows how these rules are met identifies all exceptions to the rules. These rules may lay down what type of testing is required or how much testing is required.
4. Should be based on and derived from project base documentation: For example, planning of acceptance test: May be based on the business requirements; define the business processes to be tested; assign business risk to business processes. For example, planning of component test: May be based on the component specification; define the program path coverage; assign technical risk to the component processes.
5. Test Specification
6. Test Specification Is the design of Test Cases.
7. Test techniques defined in the test planning phase must be used. Test
8. A Test Case should encompass.
9. Test Execution
10. Having identified the test cases (during test specification) identify which tests are selected for this test run.
11. Dependent upon a release of software and what is contained within it.
12. Test execution is the execution of the identified tests.
13. Test Recording
14. The version of software being executed.
15. The actual outcome of the test.
16. The comparison of the actual outcome against the expected outcome.

* Identify the fault
* Make an initial assessment of whether this is a test error or an environment problem or a product problem
* Log the fault
* Assess the impact of the fault
* Record the failure of the test.
* If the actual outcome is the same as the expected outcome Record that the test has passed

**4.2.7 What is SDLC & STLC?**

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality software’s. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

The Software Testing Life Cycle (STLC) is a sequence of specific actions performed during the testing process to ensure that the software quality objectives are met. The STLC includes both verification and validation. Contrary to popular belief, software testing is not just a separate activity. It consists of a series of methodological activities to help certify your software product.

**4.2.8 SDLC Models - Waterfall, Spiral, V, Agile**

If you look at the project plan generated using the Waterfall model, you can see that none of the test planning or test creation as well as the test execution is performed until after the programs have been coded.

The Waterfall Model Testing tends to start with the delivery of the code, not the documentation.

Errors are more costly to correct due to testing starting later in the SDLC.

If the project is over budget and/or exceeding timescales, the testing phase will be reduced.

Testing is seen as a test execution process.

1. Advantages:
2. This model is simple and easy to understand and use.
3. It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
4. In this model phases are processed and completed one at a time. Phases do not overlap.
5. Waterfall model works well for smaller projects where requirements are very well understood.
6. Disadvantages:
7. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought.
8. No working software is produced until late during the life cycle.
9. High amounts of risk and uncertainty.
10. Not a good model for complex and object-oriented projects.
11. Poor model for long and ongoing projects.
12. Not suitable for the projects where requirements are at a moderate to high risk of changing.

It is an incremental approach to development and testing whereby the full system requirements may not be known at the start of the project (i.e. the users know their basic requirements but do not really know exactly what the complete system should look like).

The initial requirements are defined, designed, built and tested (with review points after each activity) and those requirements enhanced and built upon in further iterations of the define, design, build and test activities.

The system is implemented at the end of the required number of iterations.

1. Incremental approach
2. Build a little, test a little
3. Uses prototypes
4. Components/functions developed in parallel
5. Developments are time-boxed, delivered and assembled into a working prototype.
6. Advantages:
7. High amount of risk analysis hence, avoidance of Risk is enhanced.
8. Good for large and mission-critical projects.
9. Strong approval and documentation control.
10. Additional Functionality can be added at a later date.
11. Software is produced early in the software life cycle.
12. Disadvantages:
13. Can be a costly model to use.
14. Risk analysis requires highly specific expertise.
15. Project’s success is highly dependent on the risk analysis phase.
16. Doesn’t work well for smaller projects.

The V Model was introduced to address some of the problems associated with the Waterfall model. Within the V model testing is not seen as a phase that happens at the end of development, it is recognized that for every stage of development an equal stage of testing needs to occur.

It also recognizes that the test preparation, for example test planning and test creation can be separated from test execution. The test preparation is not dependant on the code being delivered and can occur much earlier in the SDLC.

The documentation shown on the (left-hand side of the V model is not rigid - organizations may call documents produced by different names, may merge the documents shown, or may have additional documents that they produce. The right-hand side of the V model is more rigid in terms of naming conventions, but organizations may choose to exclude certain levels of testing, depending upon the project in question

1. Advantages:
2. Simple and easy to use.
3. Testing activities like planning, test designing happen well before coding. This saves a lot of time. Hence higher chance of success over the waterfall model.
4. Proactive defect tracking – that is defects are found at early stage.
5. Avoids the downward flow of the defects.
6. Works well for small / Medium projects where requirements are easily understood.
7. Disadvantages:
8. Software is developed during the implementation phase, so no early prototypes of the software are produced.
9. If any changes happen in midway, then the test documents along with requirement documents has to be updated.

Agile method involves Iterative development. This model is based on Agile Manifesto (initially published in Feb 2001)

Agile testing is testing practice for a project using agile methodologies. It treats development as the customer of testing and emphasizing the test-first code later paradigm (TDD).

It focuses on peer code reviews (white box testing), extensive unit testing and regression testing. This implies high usage of automation of test cases.

**4.2.9 How to Derive Expected Result**

Expected Result is an ideal result that the tester should get after the test case is performed. It’s usually documented together with the test case. It’s usually compared with actual result, and if the actual result differs from the expected one, the difference is documented and called a bug.

**4.2.10 Test Case Format**

A **test case format** is a document that comes under the FRS which allows testers to develop the test cases for a particular test scenario in order to verify whether the features of an application are working as intended or not. Test cases are the set of positive and negative executable steps of a test scenario which has a set of pre-conditions, test data, expected results, post-conditions, and actual results.

**4.2.11 Important Aspects of Test Cases**

1. Objective: Here the tester mentions what he plans to achieve with that particular test case.
2. Steps to Follow: Here the tester mentions the steps that need to be followed to achieve the objective.
3. Expected Output: Here the tester mentions the output which is expected as per the requirements provided.
4. Actual Output: Here the tester mentions the actual output achieved by following the steps.
5. Pass/Fail: If the tester fails to achieve the ‘Expected Output’ by following the steps then he will mention ‘Fail’ against that particular test case. Similarly, if the tester is able to achieve the ‘Expected Output’ then he will mention ‘Pass’ against the test case.

**4.2.12 Software Testing Levels & Types**

1. Unit Testing
2. Unit testing is performed by the respective developers/Peer Developers on the individual units of source code assigned areas.
3. Unit Testing follows white box testing approach, where developer will test units of code like statements, Branches, Conditional Looping, Functions, methods etc.
4. The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality and works fine.
5. Integration Testing
6. The testing of combined parts of an application to determine if they function correctly together is Integration testing. There are two methods of doing Integration Testing Bottom-up Integration testing and Top Down Integration testing.
7. System Testing
8. This is the next level in the testing and tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets Quality Standards. This type of testing is performed by a specialized testing team.
9. Regression Testing
10. Whenever a change in a software application is made it is quite possible that other areas within the application have been affected by this change. To verify that a fixed bug hasn't resulted in another functionality or business rule violation is Regression testing.
11. The intent of Regression testing is to ensure that a change, such as a bug fix did not result in another fault being uncovered / introduced in the application.
12. Acceptance Testing
13. This is the formal type of software testing that is performed by End Customer, to check if software conforms to their business needs and to the requirements provided earlier.
14. Acceptance tests are usually documented. However, end customer may not document it for smaller versions or releases of a software.

**4.2.13 Maintenance Testing**

1. Once deployed, a system is often in service for many years. During this time the system and its operational environment is often corrected, changed or extended. Testing that is executed during this life cycle phase is called Maintenance testing.
2. Maintenance testing is different from maintainability testing, which defines how easy it is to maintain the system.
3. A major and important activity within maintenance testing is impact analysis. During impact analysis, together with stakeholders, a decision is made on what parts of the system may be unintentionally affected and therefore need careful regression testing.
4. Risk analysis will help to decide where to focus regression testing - it is unlikely that the team will have time to repeat all the existing tests.
5. If it is impossible to compile any specifications from which test cases can be written, including expected results, an alternative test basis, e.g. a test oracle (or Oracle assumptions) , should be sought by way of compromise. A search should be made for documentation which is closest to the specifications and which can be managed by developers as well as testers. In such cases it is advisable to draw the customer's attention to the lower test quality which may be achieved.

**4.2.14 Software Quality - QA (Static) & QC (Dynamic)**

1. Quality Assurance
2. It is a procedure that focuses on providing assurance that quality requested will be achieved
3. QA aims to prevent the defect
4. It is a method to manage the quality- Verification
5. It does not involve executing the program
6. It’s a Preventive technique
7. It’s a Proactive measure
8. It is the procedure to create the deliverables
9. QA involves in full software development life cycle
10. In order to meet the customer requirements, QA defines standards and methodologies
11. It is performed before Quality Control
12. It is a Low-Level Activity, it can identify an error and mistakes which QC cannot
13. Its main motive is to prevent defects in the system. It is a less time-consuming activity
14. QA ensures that everything is executed in the right way, and that is why it falls under verification activity
15. It requires the involvement of the whole team
16. The statistical technique applied on QA is known as SPC or Statistical Process Control (SPC)
17. Quality Control
18. The statistical technique applied on QA is known as SPC or Statistical Process Control (SPC)
19. QC aims to identify and fix defects
20. It is a method to verify the quality-Validation
21. It always involves executing a program
22. It’s a Corrective technique
23. It’s a Reactive measure
24. It is the procedure to verify that deliverables
25. QC involves in full software testing life cycle
26. QC confirms that the standards are followed while working on the product
27. It is performed only after QA activity is done
28. It is a High-Level Activity; it can identify an error that QA cannot
29. Its main motive is to identify defects or bugs in the system. It is a more time-consuming activity
30. QC ensures that whatever we have done is as per the requirement, and that is why it falls under validation activity
31. It requires the involvement of the Testing team
32. The statistical technique applied to QC is known as SQC or Statistical Quality Control

**4.2.15 Review Process**

Review is a form of static testing where a system/application is not executed. These can be informal like peer reviews or formal in the form of inspection for legal/regulatory requirements, audits. Informal review process is not documented.

Formal review process is well documented and involves one or more of following steps

1. Planning
2. Kick-off
3. Preparation
4. Review meeting
5. Rework
6. Follow-up

Typical roles associated with Formal review process –

1. Moderator
2. Author
3. Scribe
4. Reviewers
5. Manager

**4.2.16 Tools**

1. Selenium
2. Selenium is a testing framework to perform web application testing across various browsers and platforms like Windows, Mac, and Linux. Selenium helps the testers to write tests in various programming languages like Java, PHP, C#, Python, Groovy, Ruby, and Perl. It offers record and playback features to write tests without learning Selenium IDE.
3. TestingWhiz
4. TestingWhiz is a test automation tool with the code-less scripting by Cygnet Infotech, a CMMi Level 3 IT solutions provider. TestingWhiz tool’s Enterprise edition offers a complete package of various automated testing solutions like web testing, software testing, database testing, API testing, mobile app testing, regression test suite maintenance, optimization, and automation, and cross-browser testing.

**4.2.17 Testing Methods – Black Box**

The type of testing without having any knowledge of the interior workings of the application is Black Box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, when performing a black box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

**4.2.18 Black Box Techniques**

1. Equivalence partitioning (EP) is a specification-based or black-box technique.
2. It can be applied at any level of testing and is often a good technique to use first.
3. The idea behind this technique is to divide (i.e. partition) a set of test conditions into groups or sets that can be considered the same (i.e. the system should handle them equivalently), hence the name ‘equivalence partitioning’. Equivalence partitions are also known as equivalence classes – the two terms mean exactly the same thing.
4. In equivalence-partitioning (EP) technique we need to test only one condition from each partition. This is because we are assuming that all the conditions in one partition will be treated in the same way by the software.
5. If one condition in a partition works, we assume all of the conditions in that partition will work, and so there is little point in testing any other values. Similarly, if one of the conditions in a partition does not work, then we assume that none of the conditions in that partition will work so again there is little point in testing any more in that partition.

**4.2.18 Decision Tables**

1. Decision table is composed of rows and columns.
2. Each row corresponds to a single rule, with the columns defining the conditions of the rules. You can add new rows to a decision table and fill in its cells to create new rules.
3. It is a table which shows different combination inputs with their associated outputs, this is also known as cause effect table.
4. In EP and BVA we have seen that these techniques can be applied to only specific conditions or inputs however if we have different inputs which result in different actions being taken or in other words, we have a business rule to test where there are different combination of inputs which result in different actions.
5. For testing such rules or logic decision table testing is used.
6. It is a black box test design technique.

**4.2.19 White Box Testing**

White box testing is the detailed investigation of internal logic and structure of the code. White box testing is also called glass testing or open box testing. In order to perform white box testing on an application, the tester needs to possess knowledge of the internal working of the code.

The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

**4.2.20 Grey Box Testing & Comparison**

Grey Box testing is a technique to test the application with limited knowledge of the internal workings of an application. In software testing, the term the more you know the better carries a lot of weight when testing an application.

Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge. Unlike black box testing, where the tester only tests the application's user interface, in grey box testing, the tester has access to design documents and the database Design. Having this knowledge, the tester is able to better prepare test data and test scenarios when making the test plan.

1. Black Box Testing
2. The Internal Workings of an application are not required to be known
3. Also known as closed box testing, data driven testing and functional testing
4. Performed by end users and also by testers and developers
5. Testing is based on external expectations - Internal behaviour of the application is unknown
6. This is the least time consuming and exhaustive
7. Not suited to algorithm testing
8. This can only be done by trial and error method
9. Grey Box Testing
10. Some knowledge of the internal workings is known
11. Another term for grey box testing is translucent testing as the tester has limited knowledge of the insides of the application
12. Performed by end users and also by testers and developers
13. Testing is done on the basis of high-level database diagrams and data flow diagrams
14. Partly time consuming and exhaustive
15. Not suited to algorithm testing
16. Data domains and Internal boundaries can be tested, if known
17. White Box Testing
18. Tester has full knowledge of the Internal workings of the application
19. Also known as clear box testing, structural testing or code-based testing
20. Normally done by testers and developers
21. Internal workings are fully known and the tester can design test data accordingly
22. The most exhaustive and time-consuming type of testing
23. Suited for algorithm testing
24. Data domains and Internal boundaries can be better tested

**4.2.21 Test Organisation**

`Testing is an assessment of quality, and since that assessment is not always positive, many organizations strive to create an organizational climate where testers can deliver an independent, objective assessment of quality.

The degree of independence can vary. At one end of the spectrum lies the absence of independence, where the programmer performs testing within the programming team. On the other end of the spectrum lies complete independence. There can be a separate test team reporting into the organization at a point equal to the development or project team.

Typical roles will be listed below –

1. Test Leader
2. Test Analyst
3. Tester

**4.2.22 Test Estimation**

Two approaches for the estimation of test effort are:

1. The metrics-based approach: estimating the testing effort based on metrics of former or similar projects or based on typical values
2. The expert-based approach: estimating the tasks based on estimates made by the owner of the tasks or by experts

Once the test effort is estimated, resources can be identified and a schedule can be drawn up.

The testing effort may depend on a number of factors, including:

1. Characteristics of the product: the quality of the specification and other information used for test models (i.e., the test basis), the size of the product, the complexity of the problem domain, the requirements for reliability and security, and the requirements for documentation
2. Characteristics of the development process: the stability of the organization, tools used, test process, skills of the people involved, and time pressure
3. The outcome of testing: the number of defects and the amount of rework required

**4.2.23 How to Decide Priority**

Priority defines the order in which we should resolve a defect. Should we fix it now, or can it wait? This priority status is set by the tester to the developer mentioning the time frame to fix the defect. If high priority is mentioned then the developer has to fix it at the earliest.

The priority status is set based on the customer requirements. For example: If the company name is misspelled in the home page of the website, then the priority is high and severity is low.

**4.2.24 Test Monitoring & Control**

Test monitoring can serve various purposes during the project, including the following:

1. Give the test team and the test manager feedback on how the testing work is going, allowing opportunities to guide and improve the testing and the project.
2. Provide the project team with visibility about the test results.
3. Measure the status of the testing, test coverage and test items against the exit criteria to determine whether the test work is done.
4. Gather data for use in estimating future test efforts.

Test control is about guiding and corrective actions to try to achieve the best possible outcome for the project. The specific corrective or guiding actions depend, of course, on what we are trying to control.

**4.2.25 Test Progress**

Test progress monitoring is about gathering detailed test data; reporting test status is about effectively communicating our findings to other project stake-holders.

As with test progress monitoring, in practice there is wide variability observed in how people report test status, with the variations driven by the prefrences of the testers and stakeholders, the needs and goals of the project, regulatory requirements, time and money constraints and limitations of the tools available for test status reporting.

If you are doing risk-based testing, one main test objective is to subject the important product risks to the appropriate extent of testing. Table 5.1 shows an example of a chart that would allow you to report your test cover-age and unresolved defects against the main product risk areas you identified in your risk analysis. If you are doing requirements-based testing, you could measure coverage in terms of requirements or functional areas instead of risks.

**4.2.26 Configuration Management**

Configuration management is in part about determining clearly what the items are that make up the software or system. These items include source code, test scripts, third- party software, hardware, data and both development and test documentation. Configuration management is also about making sure that these items are managed carefully, thoroughly and attentively throughout the entire project and product life cycle.

It is very important to select configuration management procedures and tools.

**4.2.27 Risk & Testing**

Risk is the possibility of a negative or undesirable outcome. It has two parameters associated with it –

1. Probability – likelihood of event happening
2. Impact – consequences of the event

product risk as the possibility that the system or software might fail to satisfy some reasonable customer, user, or stakeholder expectation. (Some authors refer to 'product risks' as 'quality risks' as they are risks to the quality of the product.)

These risks may be related to functionality, security, reliability, usability, maintainability or performance.

Risk-based testing uses risk to prioritize and emphasize the appropriate tests during test execution, but it's about more than that. Risk-based testing starts early in the project, identifying risks to system quality and using that knowledge of risk to guide testing planning, specification, preparation and execution. Risk-based testing involves both mitigation - testing to provide opportunities to reduce the likelihood of defects, especially high-impact defects - and contingency - testing to identify work-arounds to make the defects that do get past us less painful. Risk-based testing also involves measuring how well we are doing at finding and removing defects in critical areas.

Risk-based testing starts with product risk analysis. One technique for risk analysis is a close reading of the requirements specification, design specifica-tions, user documentation and other items. Another technique is brainstorming with many of the project stakeholders. Another is a sequence of one-on-one or small-group sessions with the business and technology experts in the company.

You might have a checklist of typical or past risks that should be considered. You might also want to review the tests that failed and the bugs that you found in a previous release or a similar product.

1. testing is an activity like the rest of the project and thus it is subject to risks that endanger the project. To deal with the project risks that apply to testing, we can use the same concepts we apply to identifying, prioritizing and managing product risks.
2. Remembering that a risk is the possibility of a negative outcome, what project risks affect testing? There are direct risks such as the late delivery of the test items to the test team or availability issues with the test environment. There are also indirect risks such as excessive delays in repairing defects found in testing or problems with getting professional system administration support for the test environment.
3. To discover these risks, ask yourself and other project participants and stakeholders. Analysis process is similar to Product risks.
4. Mitigate: Take steps in advance to reduce the likelihood (and possibly the impact) of the risk.
5. Contingency: Have a plan in place to reduce the impact should the risk become an outcome.
6. Transfer: Convince some other member of the team or project stakeholder to reduce the likelihood or accept the impact of the risk.
7. Ignore: Do nothing about the risk, which is usually a smart option only when there's little that can be done or when the likelihood and impact are low.
8. Logistics or product quality problems that block tests: These can be miti gated through careful planning, good defect triage and management, and robust test design.
9. Test items that won't install in the test environment: These can be mitigated through smoke (or acceptance) testing prior to starting test phases or as part of a nightly build or continuous integration. Having a defined uninstall process is a good contingency plan.
10. Excessive change to the product that invalidates test results or requires updates to test cases, expected results and environments: These can be mit igated through good change-control processes, robust test design and light weight test documentation. When severe incidents occur, transference of the risk by escalation to management is often in order.
11. Insufficient or unrealistic test environments that yield misleading results: One option is to transfer the risks to management by explaining the limits on test results obtained in limited environments. Mitigation - sometimes com plete alleviation - can be achieved by outsourcing tests such as performance tests that are particularly sensitive to proper test environments.
12. Organizational issues such as shortages of people, skills or training, problems with communicating and responding to test results, bad expectations of what testing can achieve and complexity of the project team or organization.
13. Supplier issues such as problems with underlying platforms or hardware, failure to consider testing issues in the contract or failure to properly respond to the issues when they arise.
14. Technical problems related to ambiguous, conflicting or unprioritized requirements, an excessively large number of requirements given other project constraints, high system complexity and quality problems with the design, the code or the tests.

**4.2.28 What is Defect**

Main purpose of testing is to detect defects / bugs in the application before it is released into production.

1. Nonconformance to requirements or functional / program specification is a Defect
2. A problem that causes a program to produce invalid output or to crash (lock up). The problem is either insufficient logic or erroneous logic. For example, a program can crash if there are not enough validity checks performed on the input or on the calculations themselves e.g. the computer attempts to divide by zero.
3. A program with bad logic may produce bad output without crashing, which is the reason extensive testing is required. For example, if the program is supposed to add an amount, but subtracts it instead, this leads to bad output results.
4. A Software Defect / Bug is a condition in a software product which does not meet a software requirement (as stated in the requirement specifications) or end-user expectations (which may not be specified but are reasonable). In other words, a defect is an error in coding or logic that causes a program to malfunction or to produce incorrect/unexpected results.
5. A program that contains a large number of bugs is said to be buggy.
6. Reports detailing bugs in software are known as bug reports. (Defect Report)
7. Applications for tracking bugs are known as bug tracking tools.
8. The process of finding the cause of bugs is known as debugging

**4.2.29 Defect Priority & Severity**

It is the extent to which the defect can affect the software. In other words it defines the impact that a given defect has on the system. For example: If an application or web page crashes when a remote link is clicked, in this case clicking the remote link by an user is rare but the impact of application crashing is severe. Here the severity is high but priority is low.

Severity can be of following types:

1. Critical: The defect that results in the termination of the complete system or one or more component of the system and causes extensive corruption of the data. The failed function is unusable and there is no acceptable alternative method to achieve the required results then the severity will be stated as critical.
2. Major: The defect that results in the termination of the complete system or one or more component of the system and causes extensive corruption of the data. The failed function is unusable but there exists an acceptable alternative method to achieve the required results then the severity will be stated as major
3. Moderate: The defect that does not result in the termination, but causes the system to produce incorrect, incomplete or inconsistent results then the severity will be stated as moderate.
4. Minor: The defect that does not result in the termination and does not damage the usability of the system and the desired results can be easily obtained by working around the defects then the severity is stated as minor.
5. Cosmetic: The defect that is related to the enhancement of the system where the changes are related to the look and feel of the application then the severity is stated as cosmetic (Trivial).

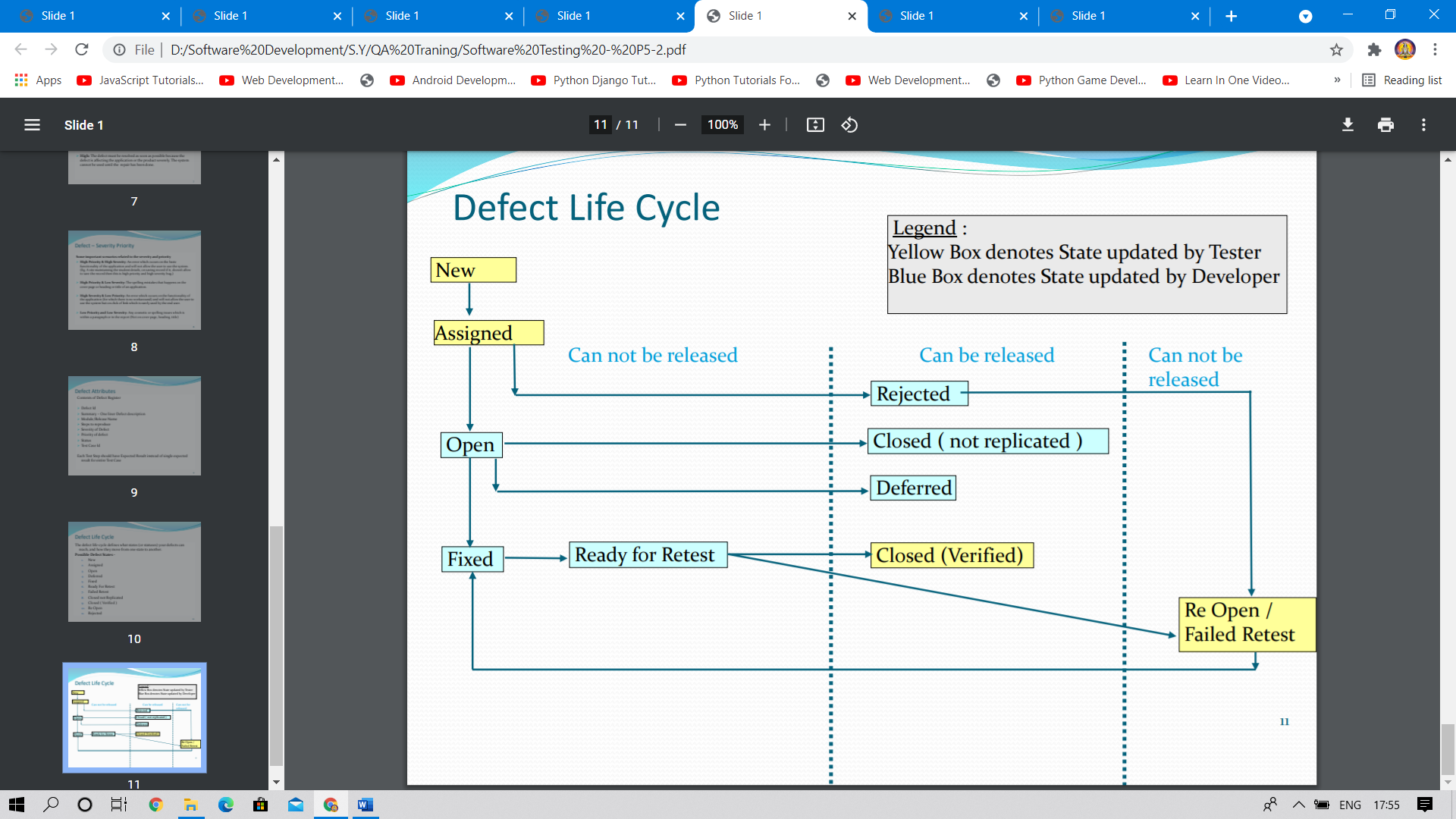
Priority can be of following types:

1. Low: The defect is an irritant which should be repaired, but repair can be deferred until after more serious defect have been fixed.
2. Medium: The defect should be resolved in the normal course of development activities. It can wait until a new build or version is created.
3. High: The defect must be resolved as soon as possible because the defect is affecting the application or the product severely. The system cannot be used until the repair has been done.

Some important scenarios related to the severity and priority

1. High Priority & High Severity: An error which occurs on the basic functionality of the application and will not allow the user to use the system. (Eg. A site maintaining the student details, on saving record if it, doesn’t allow to save the record then this is high priority and high severity bug.)
2. High Priority & Low Severity: The spelling mistakes that happens on the cover page or heading or title of an application.
3. High Severity & Low Priority: An error which occurs on the functionality of the application (for which there is no workaround) and will not allow the user to use the system but on click of link which is rarely used by the end user.
4. Low Priority and Low Severity: Any cosmetic or spelling issues which is within a paragraph or in the report (Not on cover page, heading, title)

**4.2.30 Defect Life Cycle**



The defect life-cycle defines what states (or statuses) your defects can reach, and how they move from one state to another.

Possible Defect States –

1. New
2. Assigned
3. Open
4. Deferred
5. Fixed
6. Ready For Retest
7. Failed Retest
8. Closed not Replicated
9. Closed (Verified)
10. Re Open
11. Rejected

**4.2.31 Attributes of Defect**

Contents of Defect Register

1. Defect Id
2. Summary – One liner Defect description
3. Module/Release Name
4. Steps to reproduce
5. Severity of Defect
6. Priority of defect
7. Status
8. Test Case Id

Each Test Step should have Expected Result instead of single expected result for entire Test Case

**Subsection 4.3: Results / observations / work / experience / Practical’s done as per the Job Role QP in the OJT Company.**

The following Practical’s were completed successfully during the OJT.

1. ORANGE-HRM Testing Project
2. Functional Requirement Analysis
3. Bug Reporting
4. Individual Test Scenarios on ATM Machine
5. Test Cases Writing
6. Live Testing on the HR Management Application
7. Reporting Team Members on Email

**Subsection 4.4: What challenges were experienced during OJT?**

The following minor challenges were faced during the OJT:

1. As a test lead team management was difficult.
2. There were some difficulties to meet time limit given by customer due to team members various reasons.
3. **Conclusion**

The Student was able to acquire new skills within the specified job role of QA Engineer in a timely and structured manner.

The Following Objectives were completed in the training: Contribute to quality assurance projects, manage work to meet requirements, work effectively with team members, provide data/information in standard formats.

1. **Bibliography**
2. www.javapoint.com
3. www.ibm.com
4. www.guru99.com
5. https://[opensource-demo.orangehrmlive.com/](https://opensource-demo.orangehrmlive.com/)
6. [www.wikipedia.com](http://www.wikipedia.com)
7. [www.softwaretestingmaterial.com](http://www.softwaretestingmaterial.com)