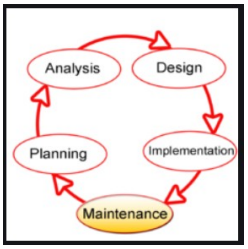
What is SDLC?

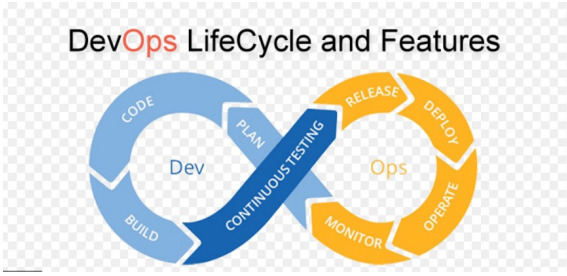
• The systems development life cycle (SDLC), also referred to as the Software development life-cycle, is a term used in information and software engineering to describe a process for planning, creating, and deploying an information system.



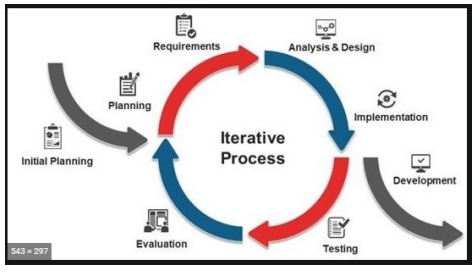
systems testing,

6 Basic SDLC Methodologies

1. **Waterfall**: It is widely considered the **oldest** of the structured SDLC methodologies. It's also a very straightforward approach:
   1. **finish one phase, then move on to the next**. No going back.
   2. **Each stage relies on information from the previous stage** and has its own project plan.
2. **Agile**: The Agile model has been around for about a decade. But lately, it has become a **major driving force behind software development** in many organizations.
   1. Some businesses value the Agile methodology so much that they are now applying it to other types of projects, including **non-tech initiatives**
3. **DevOps**:The DevOps methodology is the **newcomer** to the SDLC scene.
   1. It emerged from two trends:
      1. The **application of Agile and Lean practices to operations work**
      2. The **general shift in business toward seeing the value** of collaboration between development and operations staff at all stages of the SDLC process
   2. The high pace of code builds per day delivering stability and reliability.
   3. The DevOps model is to:
      1. Development writes the code
      2. Development Automatically deploys the code into the Automated Test environment
      3. Test team execute Automated Tests
      4. Deploys into the production environment



4. **Lean**: The Lean model for software development is inspired by l**ean manufacturing practices and principles.**
   1. The seven Lean principles are: • Eliminate waste • Amplify learning • Decide as late possible • Deliver as fast as possible • Empower the team • Build integrity in • and see the whole
5. **Iterative**: The Iterative model is repetition incarnate. Instead of starting with fully known requirements, project teams implement a set of software requirements, **then test, evaluate and pinpoint further requirements.** "A new version of the software is produced with each phase, or iteration. Rinse and repeat until the complete system is ready"



6. **Spiral**: One of the most flexible SDLC methodologies, **the Spiral model takes a cue from the Iterative model and its repetition;** the project passes through four phases: • Planning • Risk Analysis • Engineering • Evaluation. over and over in a "spiral" until completed, allowing for multiple rounds of refinement. **Spiral Model is a combination of a waterfall model and iterative model.**

The Definition of the Business Requirements

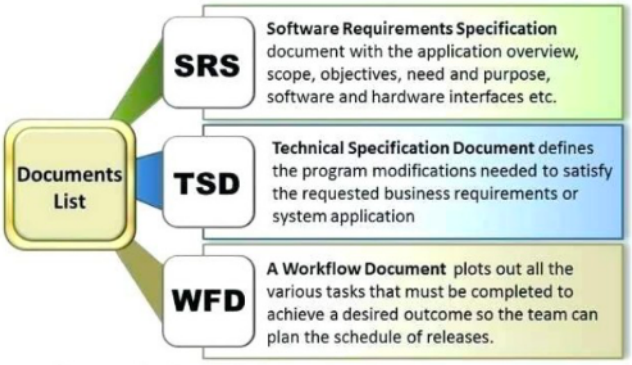
- **Business requirements** describe the **characteristics of the proposed system** from the viewpoint of end user like a Concept of Operations.

- **Business Requirements** define what the system should do from the Business (end user) perspective.
- For Example: As a Product Owner, I would like to let our customers know about our promotions once they purchase any of our B2B Services.

The Definition of the Functional Requirements

- A functional requirement, in software and systems engineering, **is a declaration of the intended function of a system and its components.**
- Based on **functional requirements, an engineer determines the behavior** (output) that a device or software is expected to exhibit in the case of a certain **input**.
- Functional Requirements **describe how the features** of a product must behave.
- A **Developer can implement the functional requirements** to enable the user to accomplish his/her tasks.
- Example: After the user confirms the purchase of the eService, prompt the user with a list of top 10 of our products in an alphabetical order.

Technical Specification Document (TSD)



•A **technical specification document** defines the requirements for a project, product, or system.
•A **specification** is the information on **technical** design, development, and procedures related to the requirements it outlines.
•It describes the System Architecture, System Design Considerations Inputs/Outputs, Database Design, APIs, Communications Protocols, Security Architecture, Data Formatting, Choice of the Technology

Example of System Architecture Diagram

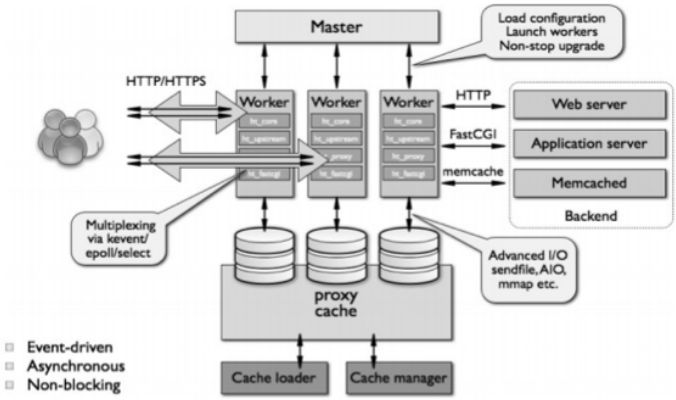
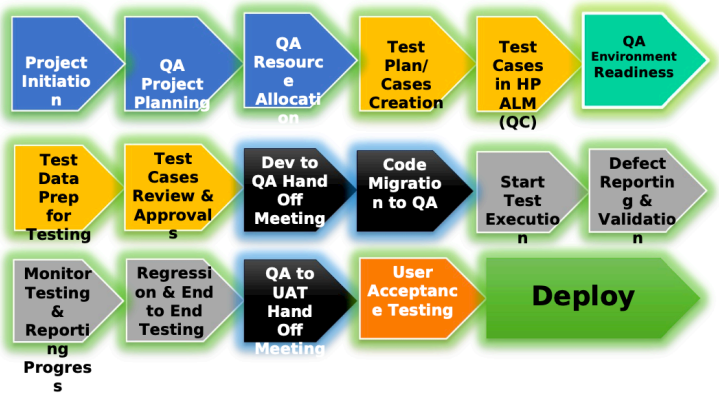
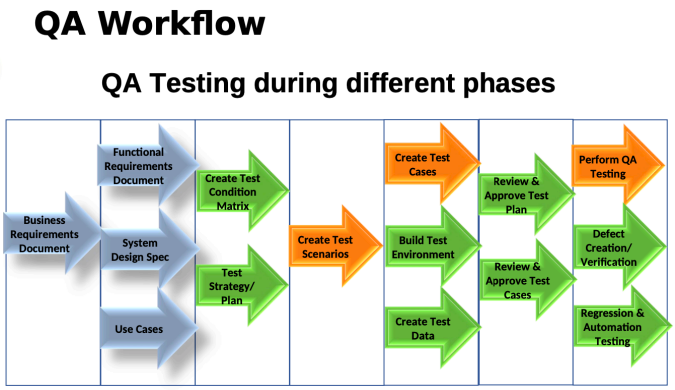


Figure 4: Hardware architecture

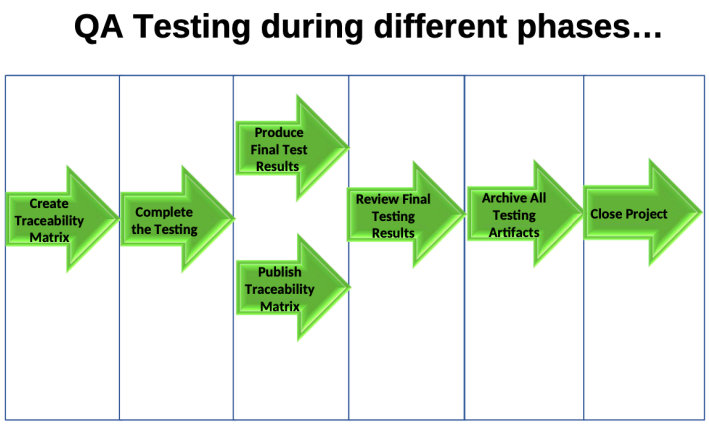
Table 4: Systems descriptions

A Simplistic QA Workflow (Waterfall)





Test Creation/Project Closure



Waterfall Methodology Stages

Sample Phases:

1. **Initiation**
2. **Requirements gathering**
3. **Design**
4. **Implementation (Construction)**
5. **QA Testing**
6. **Production Readiness**
7. **Pilot**
8. **Deployment**
9. **Maintenance**

SDLC– “Waterfall” Methodology

What is a Test Plan or a Test Strategy Document?

- **Test Plan is a dynamic document**. The success of a testing project depends upon a well-written Test Plan document that is current at all times.
- Test Plan is more or less like a **blueprint of how the testing activity is going** to take place in a project.
- The Test Plan is shared with the Business Analysts, Project Managers, Dev team and the other teams. This helps to enhance the level of transparency of the QA team's work to the external teams.
- It is documented by the **QA lead** based on the inputs from the QA team members.
- The **more detailed and comprehensive** the plan is, the **more successful** will be the testing activity.

The Test Plan Sections

- **Test Plan/ Test Strategy** is a document that outlines the: Scope of testing, Testing Schedules, Roles and Responsibilities, Testing Phases, Features to be Tested (in scope), Features not to be tested (out of scope), QA Entrance Criteria, QA Exist Criteria, Dependencies, Risks, Defect Severities/Processes
- Developed by the Lead QA, reviewed by all members on the project team. Approved by the managers.

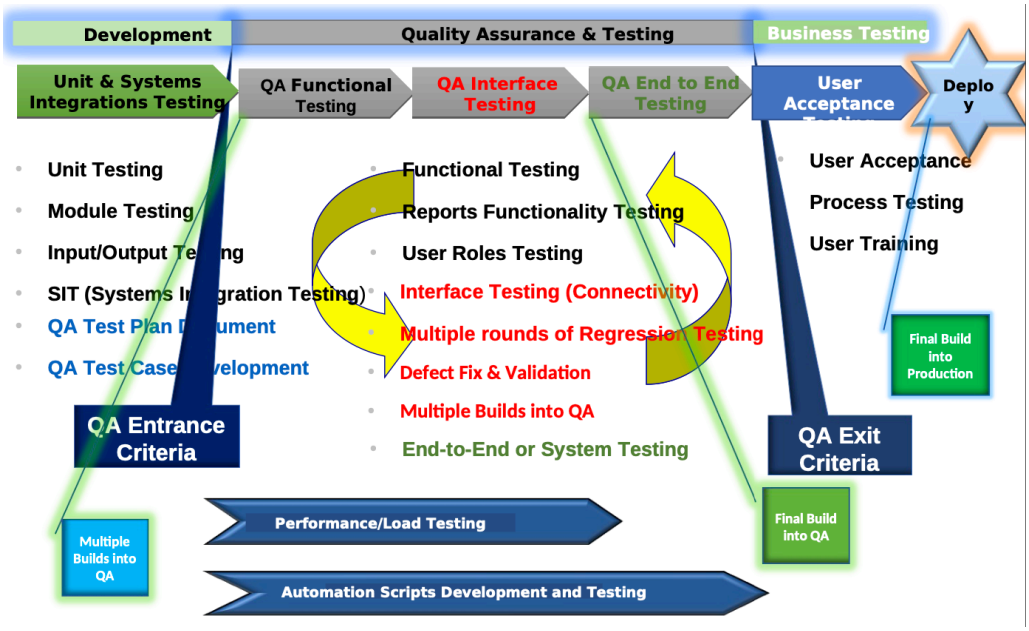
Testing Phases

1. **Unit Testing:** Testing performed at the module level in the Development environment by the developers
2. **Systems Integration Testing (SIT):** Interface testing performed in the Development environment by the Business Systems Analysts and the developers
3. **QA Functional Testing:** QA testing of the application functionality
4. **QA Interface Testing:** Testing of the interfaces using the applications as the end points
5. **QA End to End Testing:** Complete functional and Interface testing using business processes
6. **User Acceptance Testing (UAT) :** User Acceptance Testing using business processes (performed by the UAT team)

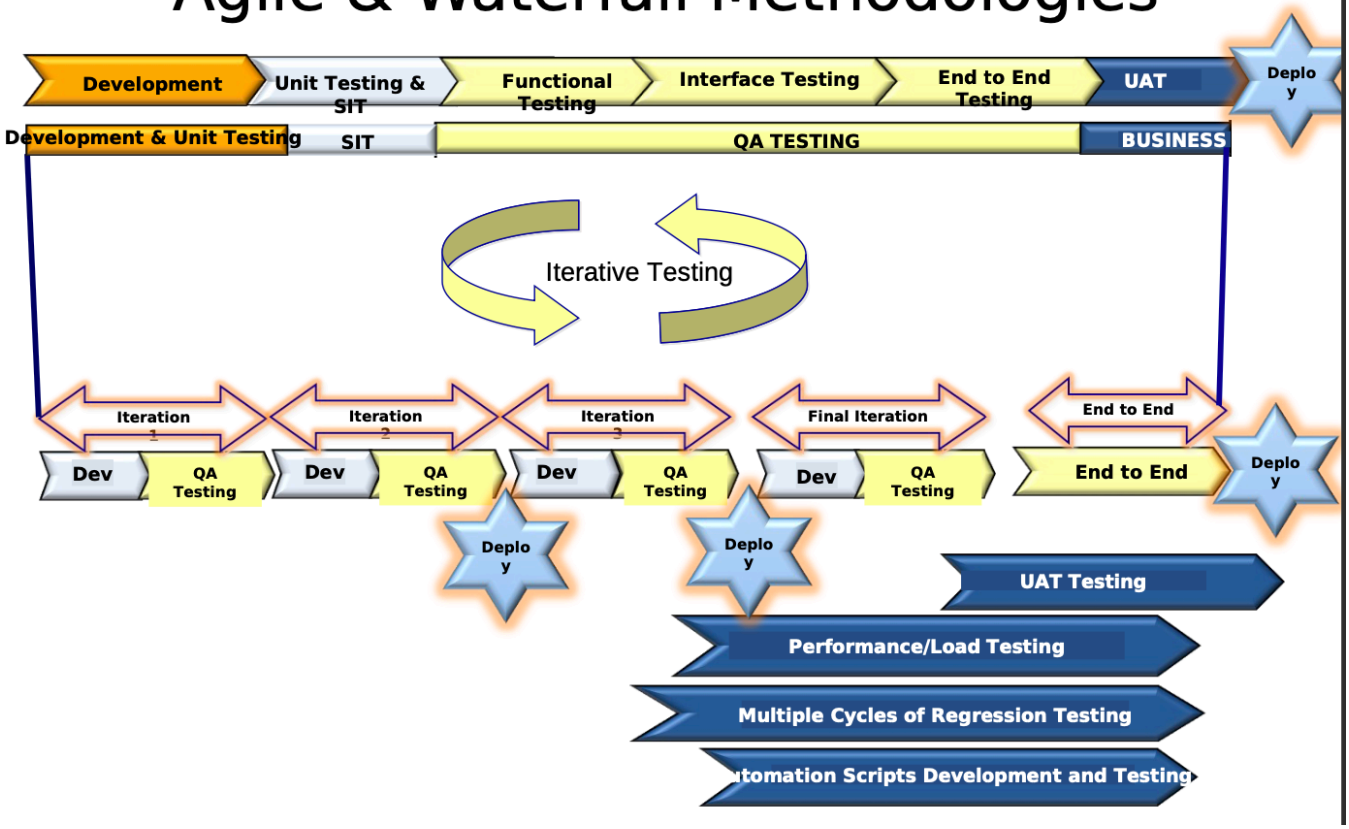
More Testing Related Definitions...

- **Smoke Testing** is a type of software **testing** that comprises of a non-exhaustive set of **tests** that aim at ensuring that the most important functions work. A.K.A "**Build Verification Testing**".
  - The term '**smoke testing**' is from a similar type of hardware **testing**, in which the device passes the **test** if it **does** not catch fire (or smoke) the first time it is turned on.
- **REGRESSION TESING** is to ensures any changes made to a **Build** did not negatively impact any of the functionality of the site or an application.
  - It is a much **deeper level testing,** and it is usually primed for test automation.
- **A Build** is a versioned release of a software that has been built and is being delivered to QA for testing. All Builds have specific release numbers for identification purposes **(e.g., Release 5.1)**

Waterfall Testing Methodology



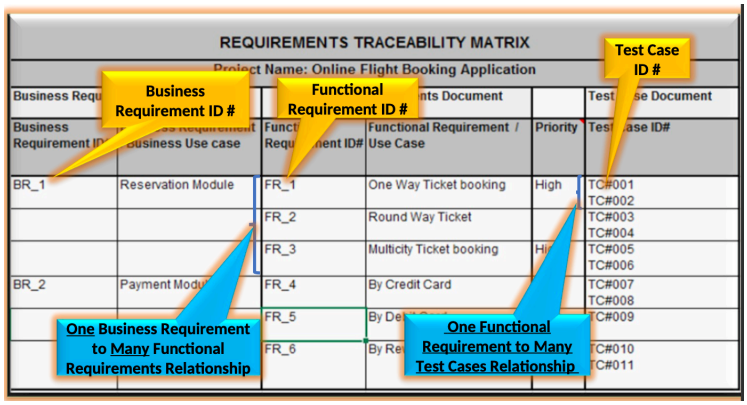
Agile & Waterfall Methodologies



Traceability Matrix

- **Purpose**: The purpose of the Requirements Traceability Matrix is to ensure that all requirements defined for a system are tested in the test protocols.
- **Why is it Important?**

1. Ensure that All of the requirements have been mapped to the test cases for a complete testing coverage.
2. When requirements change midway through a project, a traceability matrix allows you to identify all of the impacted workflows, test cases, training materials, software code, etc.
3. What does it map?: Business requirements -> functional requirements -> test cases -> defects



RACI chart

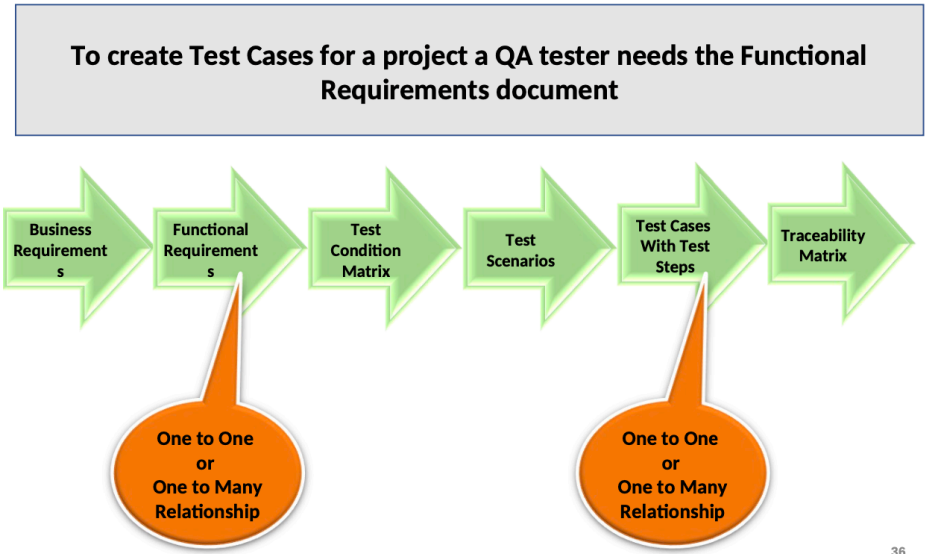
- **RACI chart** shows the roles of the resources on a project. It is a Project Management task:
  - **R = Responsible -** Those **who do the work to complete** the task. There is at least one role with a participation type of responsible, although others can be delegated to assist in the work required.
  - **A = Accountable -** The one ultimately **accountable for the correct and thorough completion** of the deliverable or task. An accountable must sign off (approve) work that responsible provides.
  - **C = Consulted -** Those whose opinions are **sought, typically subject matter experts (SME)**; and with whom there is two-way communication.

- **I = Informed -** Those who **are kept up-to-date on progress**, often only on completion of the task or deliverable; and with whom there is just one-way communication.

Pros and Cons of Waterfall Methodology

- **Pros**:
  - Everyone **understands the objectives** through the technical documentation.
  - Timelines are more achievable due to the "phased" development.
  - **Costs can be estimated** more accurately once the requirements have been defined.
  - The **testing is easier** due to the creation of the various documentation.
  - The outcome is very clear.
  - The documentation **drives the testing**, and the planning can be done more accurately.
- **Cons**:
  - No flexibility. Everything is locked down.
  - Difficult to make changes to the requirements.
  - Longer delivery time.
  - **Dependency** of each phase on another phase.
  - The user's **involvement is very minimal** in the process.
  - The **users' needs** may have changed since the beginning of the project.

A Summary of the Testing Artifacts



What is a Test Management Tool?

- Test Management Tools are used to **store information on how testing is to be done**, plan testing activities and **report the status of quality assurance activities**.
- The tools have different approaches to testing and thus have different sets of features.
- A Test Management tool for testing is **Mandatory**.
- Two Examples:
  - Quality Center – ALM
  - Jira
  - TestRail
- **Why Needed?**
  - To create and store the **Business and Functional Requirements**.
  - To create and store the **test cases** based on the requirements.
  - Allows mapping of the **Requirements to Test Cases** (Traceability Matrix).

- Allows the **execution** of the manual or automated test cases.
- Allows the **storage** of the test results/test snapshots.
- **Create reports** on the testing progress.
- **Management** of the Defect Workflow.
- **To store** the Testing Artifacts (Actual Results, Screen Shots, Defects and etc.)

What is a defect?

- A **defect** is an **error in coding or logic** that causes a program to malfunction or to produce incorrect/unexpected results.
- Defects are found during different **cycles of testing**, especially in the QA cycle.
- **A Defect can be found in many different situations:**
  - When the Actual Results deviate from the Expected Results of a test case.
  - When an error message in a program or a system is displayed on the screen.
  - When the performance of a function is not within the SLA (Service Level Agreement)
  - A **"crash"** during the execution of a function.
- **Defect Life Cycle and the Defect Fields**
  - **Defect ID** – The unique identification number
  - **Reported Date** – The Date when the bug is reported
  - **Reported By** – The details of the tester who reported the bug like Name and ID
  - **Status** – The Status of the defect like New, Assigned, Open, Retest, Verification, Closed, Failed, Deferred, etc.
  - **Version Found In**– The product version of the application in which the defect is found.
- **Defect Fields...**
  - **Defect Description** – The abstract of the issue. This includes the detailed steps of the issue with the screenshots attached so that developers can recreate it.
  - **Fixed by** – The details of the developer who fixed it like Name and ID
  - **Date Closed** – The Date when the bug is closed
  - **Severity –** Shows the impact of the defect or bug in the software application. Critical, Major or Minor.
  - **Priority –** The order of fixing the defect can be made. High, Medium and Low.
- **Defect Severity vs. Defect Priority**
  - **Defect Severity** is the degree of impact that a **defect** has on the system; whereas, **Bug Priority** is the order of **severity** which has impacted the system.
  - The QA Tester may set up the Defect Impact/Severity: • Critical • Major • Minor • Low
  - The Product Manager may set up the Defect Priority: • Severe • High • Medium • Low

Status	Description
New	This defect has been discovered and reported
Assigned	This defect has been **Assigned** to a development resource for investigation
Dev Fixed	A development resource has solved the issue but the defect is not yet associated with a build
Dev Rejected	The Developer has rejected the defect because either he/she did not understand the steps and requires more information, or was not able to reproduce the problem.
Ready For QA	The fix for this defect has been introduced into a build and is pending verification by QA tester.
Closed	This defect has been **Closed** by QA to remove it from process consideration (The Closed Reason must be filled out)
Closed Reason: Deferred Documentation Duplicate Not a Bug QA Verified	The reason for the defect closure. Mandatory field
Failed QA Verify	The fix for this defect has been tested but has **Failed Verification**--it is still a defect.