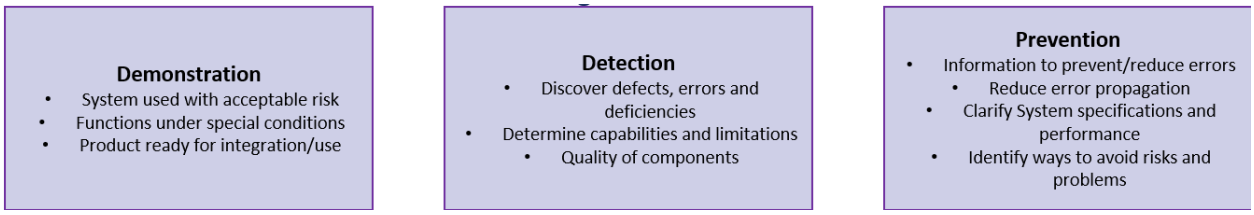
UNIT-4

- Verification: evaluate product to determine if it is built according to requirements
- Validation: evaluation a system to determine if it satisfies specified requirements

Testing Objective



Demonstration

- System can be used with acceptable risk
- Functions under special conditions
- Products are ready for integration/use

Detection

- Discover defects, errors and deficiencies

- System capabilities and limitations
- Quality of components, work products and system

Prevention

- Information to prevent/reduce number of errors
- Reduce errors that might get propagated
- System specifications and performance
- Ways to avoid risk and problems

Static Testing

- Does not include execution of code
- Checking documents, design, code (reviews, walkthroughs, inspections)
- Finds the bugs early in development
- Targets software architecture, design, database, etc.
- Occurs before validation

Dynamic Testing

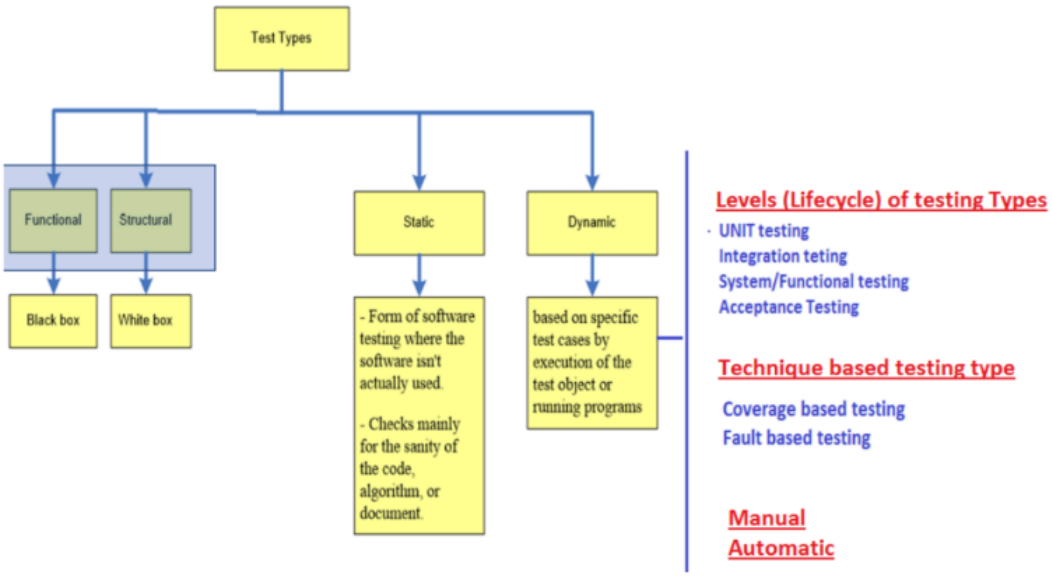
- Includes execution of code
- Validates the capabilities and features in project scope and requirements
- Typically done by testing team
- Methods: Black Box testing, White Box testing, and non-functional Testing

Terminologies

1. Defect - Deviation from the requirement
2. Bug - Coding error programmers mistake
3. Failure - Result of defect
4. Issue - Raised my end user because product doesnt meet expectation

Types of Testing

- **Functional - Black Box**
- **Structural - White Box**
- **Static**
- **Dynamic**
- **Unit**
- **Integration**
- **System**
- **Acceptance**



1) Black Box Testing

- Only considering external features
- Identifying defects in output for valid and invalid input
- Tested from the perspective of user

Advantages

- Best for larger units of code
- Early test planning

Disadvantages

- Some paths may not be tested
- Hard to direct tests to error prone code

2) White Box Testing

- Even considers internal logic
- test cases are made from the knowledge of the internal structure
- Testing from developer point of view
- need to have programming skills

Advantages

- Partitioning by execution equivalence
- Reveals hidden errors

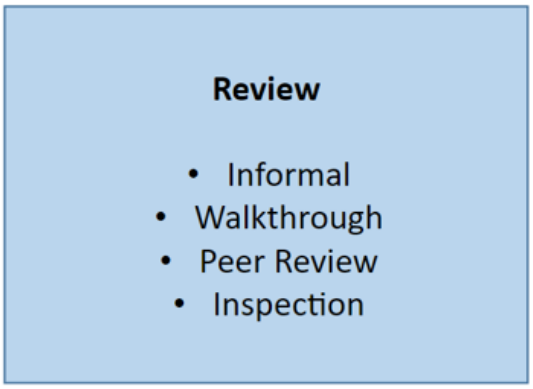
Disadvantages

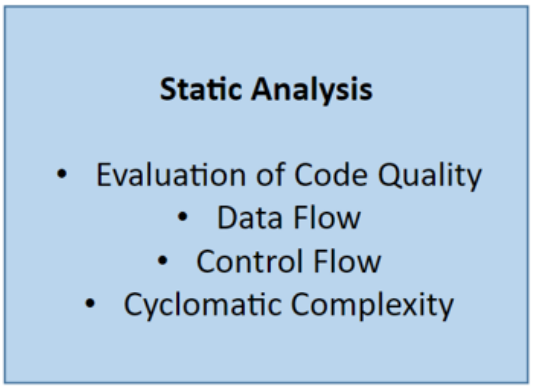
- Needs skilled testers
- Hard to test all of the code

3) Static Testing

- Check for defects in the software without executing the code

Types





Cyclomatic Complexity

- Cyclomatic complexity is the quantitative measure of the number of linearly independent paths in a code section
- It indicates the complexity of a program and is computed using the control flow graph
- **Control Flow Graph:** It is a directed graph where the nodes represent smallest group of commands and edges connect two blocks of commands

M = E – N + 2P

P = number of connected components

E = number of edges

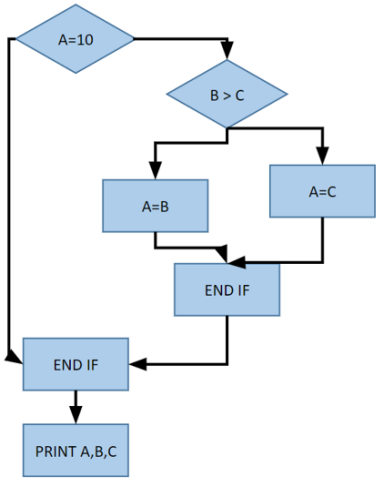
N = No of nodes

Example

Code

```
IF A = 10 THEN
    IF B > C THEN
        A = B
    ELSE
        A = C
    ENDIF
ENDIF
Print A
Print B
Print C
```

Control Flow Graph



Computing Cyclomatic Complexity

Cyclomatic Complexity = E– N + 2*P
Where, E = Number of Edges
N = Number of Nodes
P = Number of Nodes with Exit Points

Here, E = 8, N = 7, P = 1
Cyclomatic Complexity = 8 - 7 + 2
= 3

Dynamic Testing

- Dynamic testing is a type of software testing that involves executing the software code and evaluating its behavior during runtime
- Involves the execution of code for analyzing dynamic behavior
- Provide input values, observation of output values which are analyzed

Advantage

- Find difficult and complex defects

Disadvantages

- Time and Budget

Code Based Approach

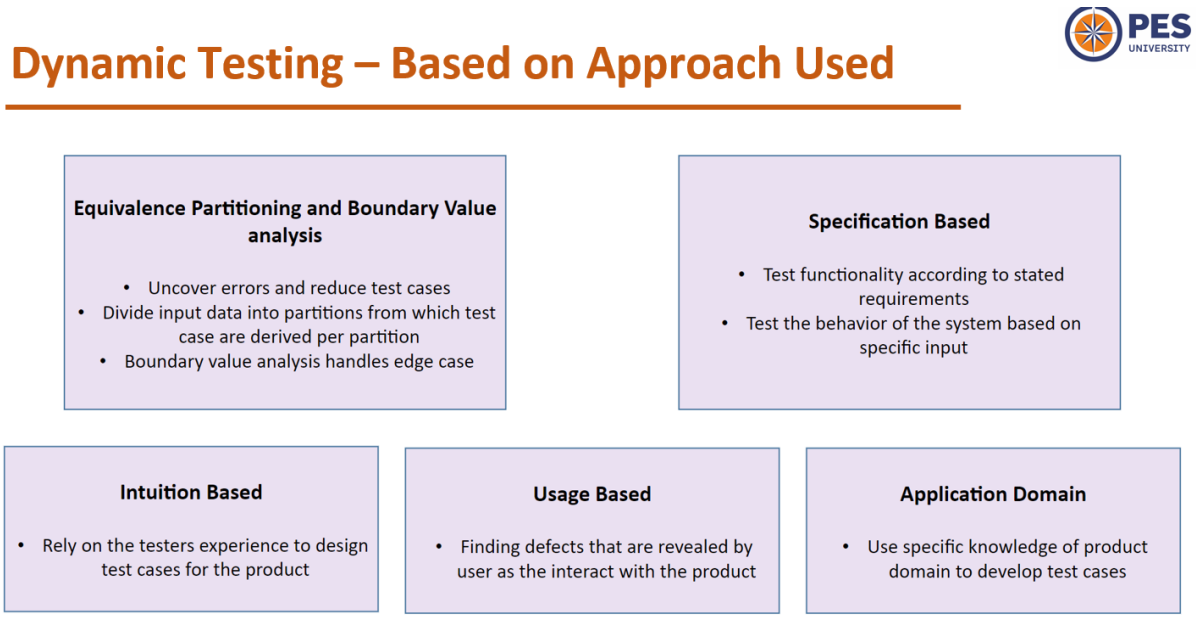
Control Flow based

- Path testing: statement, branch and condition testing
- Branch coverage: executes every path at least once
- No abnormal behaviour due to branching

- Control-flow-path: graphical representation of all paths that might be traversed during execution of the program

Data Flow Based

- Data flow refers to the movement of data within a program during its execution. It involves the transfer of values between variables and data structures as the program runs.
- Test cases are designed to validate that data is correctly input, processed, and output by the software.



Boundary Values

The idea behind this technique is that errors often occur at the extremes or boundaries of the input values rather than in the middle of the input range. By testing boundary values, testers aim to uncover potential issues that might not be apparent with typical or random input values.

1. **Test Cases:**

- Test cases are designed to include boundary values and values just beyond the boundaries.
- For example, if a system accepts input in the range of 1 to 100, boundary value analysis would involve testing with values like 0, 1, 100, 101, and other values just beyond the specified range.

2. **Objective:**
    - The primary objective of boundary analysis is to ensure that the software handles boundary values correctly and does not produce unexpected or incorrect results near the edges of the input domain.
    - It helps identify issues such as off-by-one errors, rounding errors, or other anomalies that may arise at the boundaries.

```
// Assume the below snippet of code of a printer
than can print 1 – 100 copies

copiesToPrint = input()
If (i>= i and i<=100):
    Print()
else if (i < 1):
    Output(”Need at least one copy”)
else:
    Output(“Too many copies requested. Only
    accepts up to 100”)
```

The boundary values are defined as minimum and maximum acceptable values and the values immediately before and after them, respectfully.

In our case, the boundary values to test for are 0,1,100,and 101.
Here, 1 and 100 are valid values and the program should print that many copies. However, 0 and 101 are invalid values and the program should prompt the user of the error

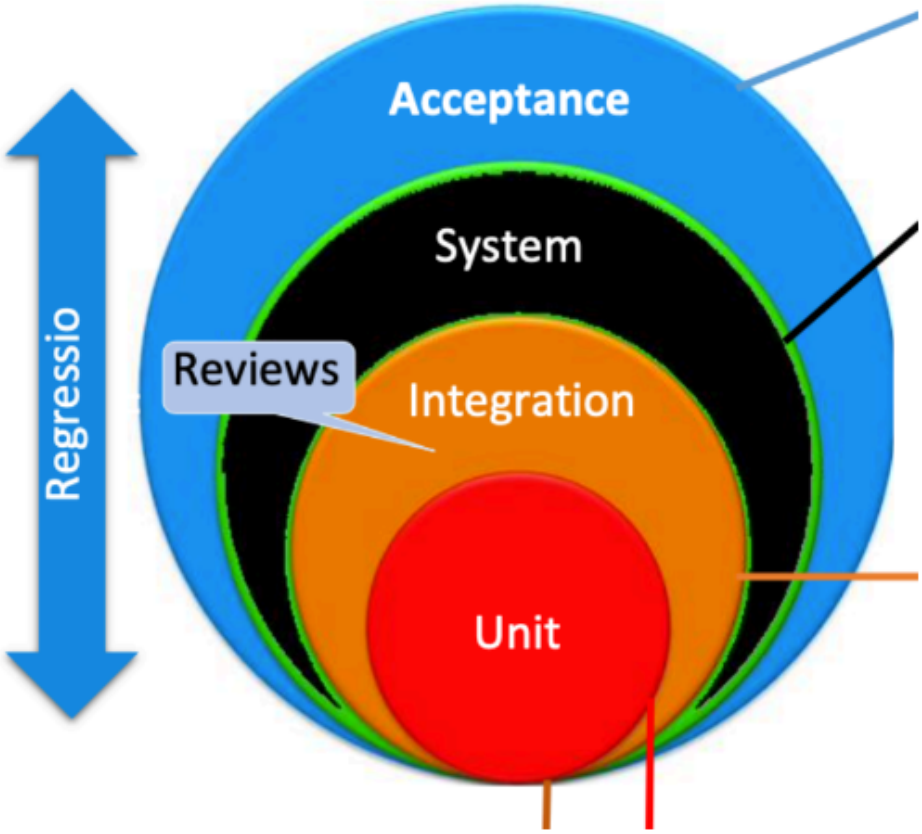
Types of testing in Dynamic Testing

Manual Testing

- Human perform the test step by step, without test scripts
- Used in complex testing where automation is very expensive
- Slow and tedious
- Hard to get test coverage
- Examples: Acceptance testing, Black Box testing, White Box testing, Unit tests

Automated Testing

- Involves test which are executed without human assistance
- Needs coding, test framework maintenance
- Fast and repeatable
- Most efficient for periodic and regular tests



Unit Testing

- Smallest executable code is tested
- Done by programmers
- All object oriented are tested here
  - Interfaces
  - Independent paths

- Boundary conditions
- Error Handling

Integration Testing

- Verify that the interactions between different components/modules of the system are working correctly.
- Identify and address issues related to the interfaces and interactions between integrated components.

Big Bang Integration Testing:

All components are integrated simultaneously, and the entire system is tested as a whole.

Iterative

- **Top-Down Integration Testing:** Testing progresses from the top of the application hierarchy down to the lower levels. Stub modules simulate the behavior of lower-level modules that are not yet integrated.
- **Bottom-Up Integration Testing:** Testing progresses from the bottom of the application hierarchy up to the higher levels. Drivers simulate the behavior of higher-level modules that are not yet integrated

System Testing

- Verify that the system meets the specified requirements.
- Ensure that the system works as expected in different environments and configurations.
- Validate that the system satisfies both functional and non-functional requirements.
- Detect defects both within “inter-assemblages” and within the system as a whole
- Processes

Test Environment Setup
Create Test Case
Create Test Data
Execute Test Case
Defect Reporting and Logging

Acceptance Testing

It primarily involves validating that the software meets the business requirements and is ready for deployment. Acceptance Testing is often the final phase of the testing process before the software is released to the end users or customers.

Key Participants:

- **End Users:** The actual users or representatives from the end-user community who will use the software in real-world scenarios.
- **Business Analysts:** Individuals who understand the business requirements and can verify if the software aligns with those requirements

Alternate Testing

Acceptance / qualification testing	Checks the system behavior against the customer's requirements, however these may have been expressed.
Installation testing	Verifies the installation in the target environment. May be identical to system testing in a new environment.
Alpha and beta testing	Before the software is released, it is sometimes given to a small, representative set of potential users for trial use, either in-house (*alpha* testing) or external (*beta* testing). ... Alpha and beta use is often uncontrolled, and is not always referred to in a test plan.
Performance testing	Aimed at verifying that the software meets the specified performance requirements (capacity and response time, for instance). A specific kind of performance testing is volume testing, in which internal program or system limitations are tried.

Additional Testing

1. **Smoke Testing:** **(Only Essesntials work)**
    - **Objective:** Ensure that the most critical functions of the software work.
    - **Purpose:** Quickly determine whether the build is stable enough for more in-depth testing.
    - **Execution:** Typically a subset of test cases covering essential functionalities.
2. **Sanity Testing:** **(Works after changes)**
    - **Objective:** Ensure specific functionalities work as intended after changes.
    - **Purpose:** Determine if the software is fit for further, more detailed testing.
    - **Scope:** Narrow and focused testing on specific areas affected by recent changes.
3. **Regression Testing:** **(Check for side effects from new changes)**
    - **Objective:** Determine if changes have unintended side effects on existing functionalities.
    - **Execution:** Re-run previously executed test cases to ensure existing features still work.
    - **Automation:** Often automated to efficiently test a large number of scenarios.
4. **Installation & Uninstallation Testing:** **(check all components are installed)**
    - **Installation Testing:** Verify proper installation of the software with all necessary components.
    - **Uninstallation Testing:** Confirm that all components are removed during the uninstallation process.
5. **Functional Testing:** **(Requirement satisfaction)**
    - **Objective:** Test features and functionalities covering various scenarios, including failure and boundary cases.

- **Scope:** Ensures the software behaves as expected according to the specified requirements.

6. **Non-Functional Testing:** **(Performance testing)**
   - **Objective:** Verify attributes such as performance, scalability, stability, and reliability.
   - **Types:** Includes performance testing, usability testing, security testing, etc.
7. **Destructive Testing:** **(Searching for points of failure)**
   - **Objective:** Test the robustness of the application by making it fail in an uncontrolled manner.
   - **Purpose:** Identify points of failure and assess how well the software handles unexpected scenarios.
8. **Software Performance Testing:**
   - **Objective:** Test responsiveness, scalability, stability, and reliability of the software product under different conditions.
   - **Types:** Includes load testing, stress testing, and scalability testing.
9. **Usability Testing:** **(Users perspective)**
   - **Objective:** Evaluate the ease of use of the system from the user's perspective.
   - **Assessment:** Considers the level of skill required to learn/use the software and the time it takes to acquire those skills.
10. **Localization Testing:**
    - **Objective:** Verify the quality of the product's localization for a specific target culture.
11. **Boundary Testing:**
    - **Objective:** Black-box testing technique using boundary values.
    - **Purpose:** Identify how the system behaves at the edge or boundary conditions.

12. **Startup/Shutdown Testing:**
    - **Shutdown Testing:** Ensures the system has not left uncleared lock files, inconsistent data, or states of tables during shutdown.
    - **Startup Testing:** Ensures the product starts in a deterministic and consistent state.
13. **Platform or Cross-Platform Tests:**
    - **Objective:** Evaluate the behavior of the application in different environments or platforms.
14. **Load Tests:**
    - **Objective:** Determine the behavior and robustness of the system under varying loads.
15. **Stress Tests:**
    - **Objective:** Push the system to maximum design load and beyond to test its limits.
16. **Security Testing:**
    - **Objective:** Uncover vulnerabilities and ensure data and resources are protected.
17. **Compliance Testing:**
    - **Objective:** Ensure compliance with internal and external standards and regulations.
18. **Recovery Tests:**
    - **Objective:** Check restart capabilities following a disaster or unanticipated shutdown.
19. **Scalability Tests:**
    - **Objective:** Determine the capability of the system to scale up or down as needed.
20. **Cloud Testing:**
    - **Objective:** Test a software application using cloud computing services.

Test Planning

Software test planning is the process of evolving a test plan which discusses what, when, how much and how has to be done to ensure quality expectations can be met

The outcome also serves as a blueprint to conduct software testing activities as a defined process

Developers, business managers and customers can understand the details of testing

Optimal amount of testing is based on

- Customer requirement
- Project schedule
- Project budget
- Product specification
- Skills and talent of test team

Test Adequacy

Criteria to determine when to stop testing or consider testing complete for that iteration

Eg: All planes test case are complete with no critically high priority issues

Testing Stratergy

Known as the test approach and defines how testing is carried out

Testing mindset or test models to be followed

- Demonstration: make sure software runs and solves problem
    - If software passes all tests, satisfies specifications
    - Developers or testers may end up testing what would succeed
    - Not advocated easily

- Evaluation: detects faults through lifecycle phases
    - Focus on analysis and review techniques to detect faults
- Destruction: try to make software fail to find faults
    - Good and effective test cases are those that find faults
    - Difficult to decide went stop testing
- Preventive: prevent faults through careful planning
    - Reviews and test driven development

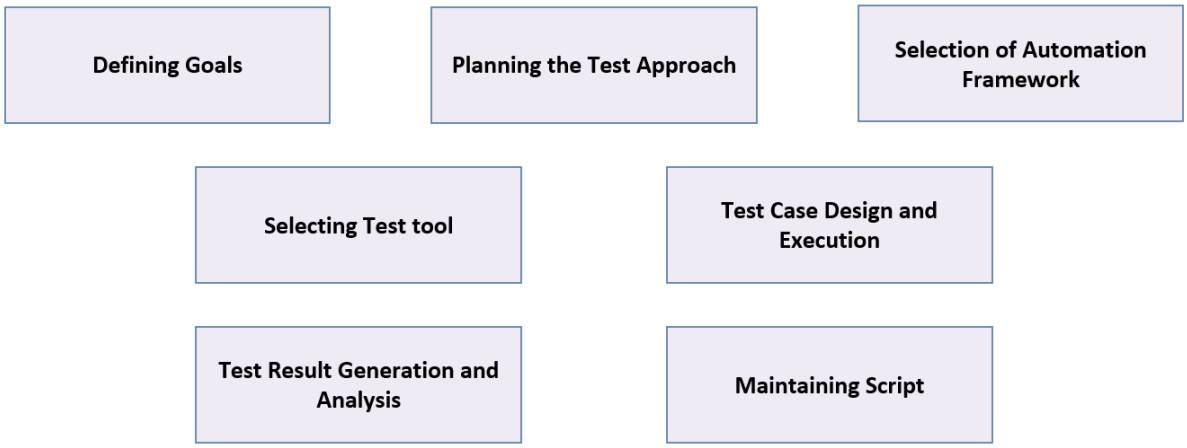
Test types will be used

- Begin by “testing-in-the-small” and move towards “testing-in-the-large”
- Top Down and Bottoms up
- Positive and Negative testing
- Dynamic and Heuristics based approach

Test environment

- setup of software and hardware for testing teams to execute
- Configured as per need of the application
- Could include test data, Database, front end, Operating System, Servers, Storage, and Network
- Correct setup ensures success of testing or result in delay, cost escalations
- Environment management involves maintenance and upkeep of test bed, monitoring and modifying components

Automation Strategy



Tools

- Testers will need to be comfortable with the tool, else it may not be effectively used
- Tools chosen needs to be balanced in terms of the features offered, ability to generate reports/data needed for different stakeholders and the ease of use.
- Cross platform support is an expectation as automated tests would/may need to run on different platforms.

Risk analysis with contingency planning

Risks in strategy could be in terms of

- Changes to the Business, Technology or competition directions
- Resources
- Quality of the software product being developed
- The test models not being able to be used
- Some type of testing chosen cannot be uses
- Test environment and its state

Test Deliverables

List of Activities and Deliverables will be identified which would need to be executed and delivered

- List of all documents, artifacts, and reports that will be produced during the testing process.
- Examples include test scripts, test reports, and traceability matrices.

Test Schedules

- Detailed timeline indicating when different testing activities will be performed.
- Identification of critical dates and milestones.
- Would include Work Breakdown Schedule and estimation using methods mentioned in project planning

Test Planning

- Identifying the number and type of servers, storage, test tools and network resources
- Identifying the number and type of people to work on the project
- Test Manager, Testers, Test Developers, Test Administrators, and SQA
- Identifying the test environment

Risk Management

- Identification of potential risks to the testing process.
- Strategies for mitigating or addressing identified risks.

Measures and Metrics

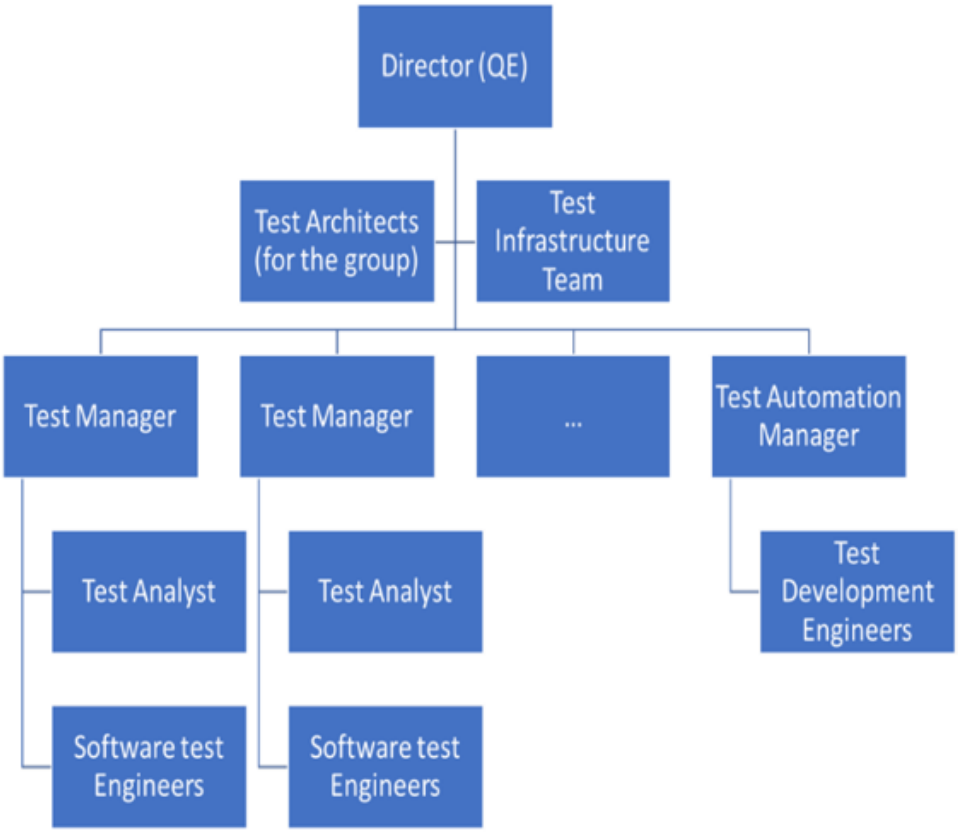
measurement in number of test cases planned and created, test cases run, amount of time in creation, execution and number of errors

- Errors are classified as critical, serious, medium or low impact
- Metrics: number of test cases executed/day or % of test cases planned, number of issues/KLoC, and so on

1. Introduction
    1. Scope
        1. In Scope
        2. Out of Scope
    2. Quality Objective
    3. Roles and Responsibilities
2. Test Methodology
    1. Overview
    2. Test Levels
    3. Bug Triage
    4. Suspension Criteria and Resumption Requirements
    5. Test Completeness
3. Test Deliverables
4. Resource & Environment Needs
    1. Testing Tools
    2. Test Environment

Testing Organization Structure

Software Testing is considered as a part of Quality Engineering



1. **Test Director**

   Provide oversight, co-ordination, strategic vision, customer and stake holder connect

2. **Test Manager**

   Prepare test strategy, plans for project, monitors and controls testing

3. **Test Infrastructure Manager**

   Manages all the infrastructure, capacity planning, maintenance, support and configurations

4. **Test Automation Manager**

   Manages the development of tools and scripts for automating

5. **Test Architect**

Designs test infrastructure, picks appropriate tools, validates test strategy

6. **Test Analyst**

   Mapping of customer environment and test conditions and features to testing conditions and documentation

7. **Software Test Engineer**

   Tests the product using appropriates testing techniques and tools

Test Process

1. **Planning and Control**

   Involves identifying requirements

   - Analyze test requirements, product architecture and Interfaces
   - Creating Test Sufficiency criteria
   - Creating Test Strategy, Planning for resources and building a schedule
   - Setup review points, status reporting mechanisms, approval boards

2. **Design**

   - Identify test conditions, design tests and test environment

3. **Implementation and Execution**

   - Develop test cases, test data, test automation
   - Execute tests, collect metrics, log results and compare with expected values
   - Track Defects, Bug Fixing

4. **Evaluate exit criteria and Reporting**

   - Evaluate test completion/stopping criteria based on application functionality
   - Eg: Test Cases, Pass Percentage, Bug rate, Deadlines, RTM

5. **Test Closure Activities**

- When testing is complete or project is cancelled
- Verify all planned deliverables
- Archive test scripts, environment and close with reports
- Perform a retrospective

Test Execution

Execute code and compare actual vs predicted values

A subset of test cases are selected based on context

- Testcases are assigned to testers for execution
- Environment is setup-configured-test data; setup steps are noted and output
  looked at
- Test are executed, results logged and status captured. Bugs logged

- If testing is blocked, worked around/resolve and continued
  • Bugs/issues are triaged and passed on
- Results reported, measurements done, metric and test results Analysed

Test Cases

Act as the starting point for text execution

Ensure proper test coverage of application under test

Writing test cases helps in thinking through the details and ensures testing is addressed from multiple angles

Typical Test Case Parameters are

- Test Case ID
- Test Scenario
- Test Case Description
- Test Setup
- Test Steps
- Prerequisite if any
- Test Parameters
- Expected Result
- Actual Result
- Comments

Value of good test cases

- **Provide following value**
- Ensures good test coverage
- Allows tester to find different ways of validating feature
- Negative test cases are also documented

- Reusable for the future

Test Case Types

Positive :- Software is doing what it supposed to be doing

Negative :- Software is not doing what it supposed to be not doing

Destructive :- Scenerios software can handle before it breaks

Test Case format

Short , strong description , concise and clear, include expected result and reusable.

Test Suites

Set of tests that helps tester execute and report
• Three states: active, in-progress and completed
• Test case can be added to multiple test suites and test plans
• Test suites created based on the cycle or based on scope
• Any type of tests

- Functional
- Non- Functional

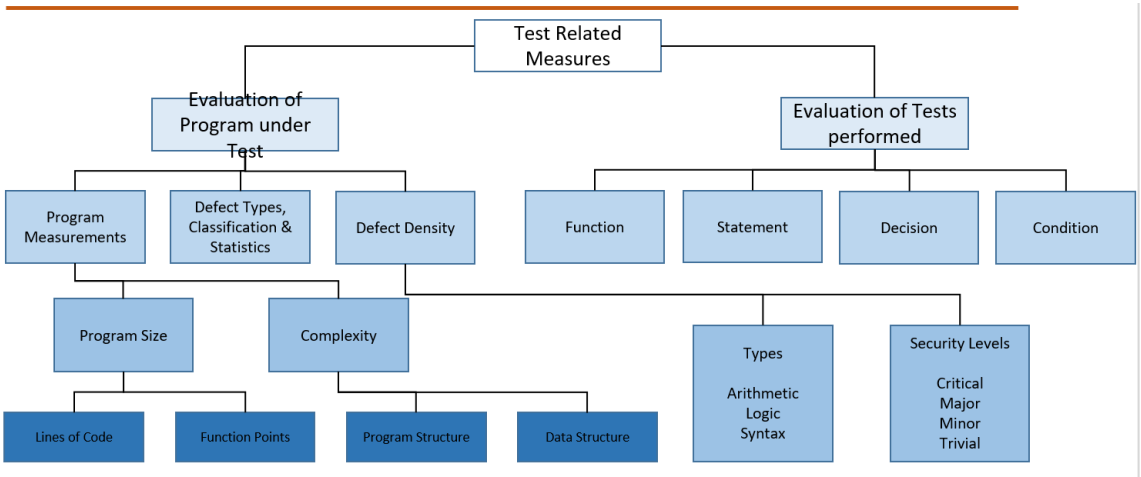
Software Test Metric

- **Software Test Metric:** Quantitative Measure of testing process indicating progress, quality, productivity and degree to which system possesses a given attribute
- Improve efficiency and effectiveness of software testing process
- Make better decisions for future testing process using reliable data

Quantitative Expressed in Values	**Understandable** Method of metric computation should be easily understood and clearly defined	**Applicability** Should be applicable in initial phase of development
Repeatable Same when measure repeatedly and consistent	**Economical** Computation should be economical	**Language Independent** Should not depend on programming language

Software Measurement

- **Measurements:** critical in performing SQA and testing
- Support increasing effectiveness of testing process
- When used as part of project planning and monitoring
- Evaluate the quality of product under test
- Helps in defect analysis
- **Measures can**
  - Represent the quality
  - Represent test coverage



Software Test Related Measures – Product Related

Metrics	Description
SLOC	Size in Lines of Code
Fault Density	Ratio of number of faults found to size of the programs
MTBF	Mean Time Between Failure is based on statistical analysis to indicate probability of failure
Failure Rate	Inverse of MTBF
Defect Distribution	% of defects attributed to a specified phase in SDLC
Defect Density of Modules	Ration of number of faults found in module to total faults found in product
Defect Leakage	Test Efficiency = [(Total number of defects found in UAT)/(Total number of defects found before UAT)] x 100

Software Reliabilty

Software Reliability is the probability of failure-free software operation for a specified period of time in a specified environment

Test Driven Development

Prevention Model

- Write the tests first, then do the design/implementation
- Could be done as part of some agile approaches like XP
- Supported by tools, e.g. Junit

Steps to do that

Add a test

- Run this and earlier tests to check if system fails
- Make a small change to make it work
- Continue incrementally running all planned tests till they run properly
- Refactor system to improve design and remove redundancies

Key issues in testing

- Test selection
- Testing for defect identification
- Objectives for a test
- Testability
- Infeasible Paths

Common ways to prioritize tests

- Test highest priority requirements first
- Test complex code first
- On basis of McCabe's cyclomatic complexity
- Test Largest modules first
- Test the most often modified modules first

Testing Tools

Selenium

Advantages of Selenium

- Open Source
- Highly Extensible
- Run tests across different browsers
- Supports various operating systems
- Supports mobile devices

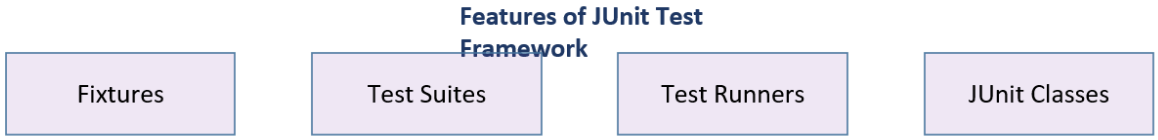
Disadvantages of Selenium

- Can only test web applications
- No built-in object repository
- Automates at a slower rate

- Data driven testing is more cumbersome
- Cannot access elements outside the web application under test
- No official User support

JUnit

- Simple framework to write repeatable test for Java Programming Language
- Eliminates need to manually verify and tall results, supporting faster product development



Advantages

- Alternate front ends to display results of tests are available
- Separate class loaders for each unit test
- Provides methods like setUp and tearDown for resource initialization
- Set of assert method to check results of tests
- Integration with tools such as Ant, Maven and IDE's such as Eclipse and Jbuilder

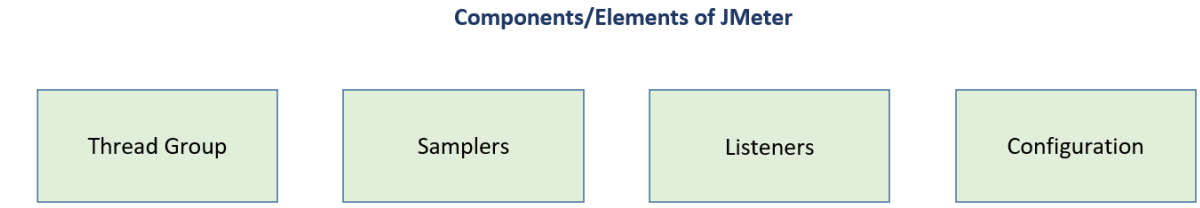
Disadvantages

- Cannot do dependency testing
- Not suitable for high level testing
- Large test suites

Cannot test various JVMs at the same time

Apache JMeter

- Apache JMeter is an open source software designed to load test functional behavior and measure performance
- Run on any environment/workstation that accepts JVM



Advantages

- Open Source
- Friendly UI
- Platform Independent
- Multi Threading Framework
- Highly Extensible
- Visualize Test result
- Unlimited testing capabilities
- Easy Installation
- Support Multi Protocol

Disadvantages

- Scripting requires some level of understanding of JMeter elements, Regular Expressions, Session Handling, etc.
- Does not have network visualization
- Single normal configuration machine is not sufficient to carry load tests
- Does not support ajax, javascript and flash

- Very limited real-time test monitoring capability

Software Maintenance

Maintenance: Sustaining process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to changed environment

- It is done to ensure software product continues to satisfy customer needs
- Committment to support system that customer invested in

Maintenance Activities

- Maintaining control over software's day to day functioning
- Maintaining control over software modifications

How to maintain?

- Understand the product which needs to be maintained by studying architecture, design, code, test cases and documentation
- Discussion with architects and developers
- Adequate and precise documentation
- Look at present and change request to identify how to satisfy task
- Look at future and analyse consequences of a solution

Factors affecting maintenance costs

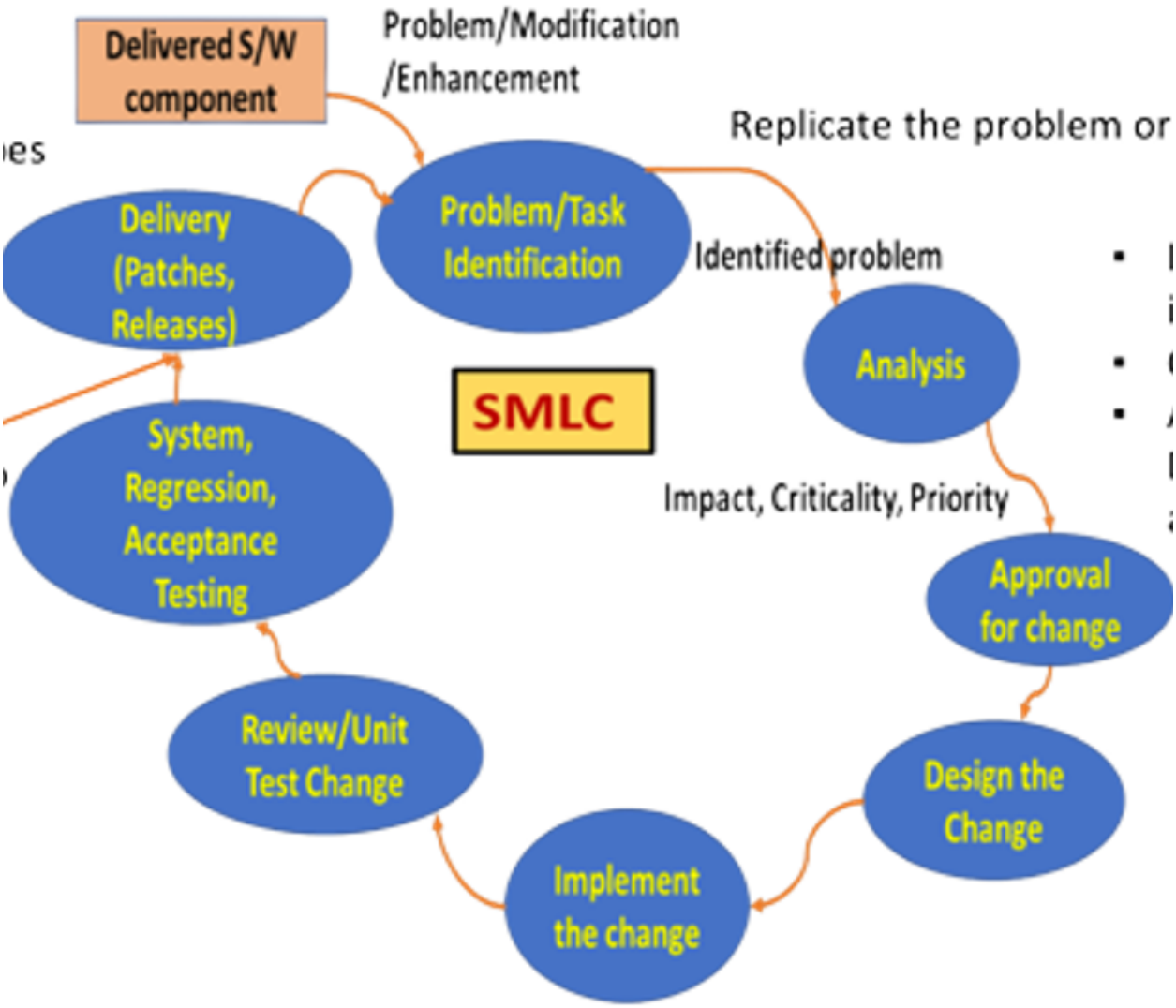
- Application type
- Software novelty
- Software maintenance staff availability
- Software lifespan

Issues with maintenance

- **Code and Documentation Quality**
- **Limited Understanding**
- **Availability of test environment to reproduce problems**
- **Impact analysis**

Cost

- Application type
- Lifespan of the system



Implementation

- Develop, Document and execute maintenance process plans including usage of tools as part of the process
- Procedures for handling user reports and modification requests, tracking of problems, responding to users

Migration

1.Develop a migration plan for the fixed/new component and execute the plan

2.This could be notifying and replacing the product and restarting

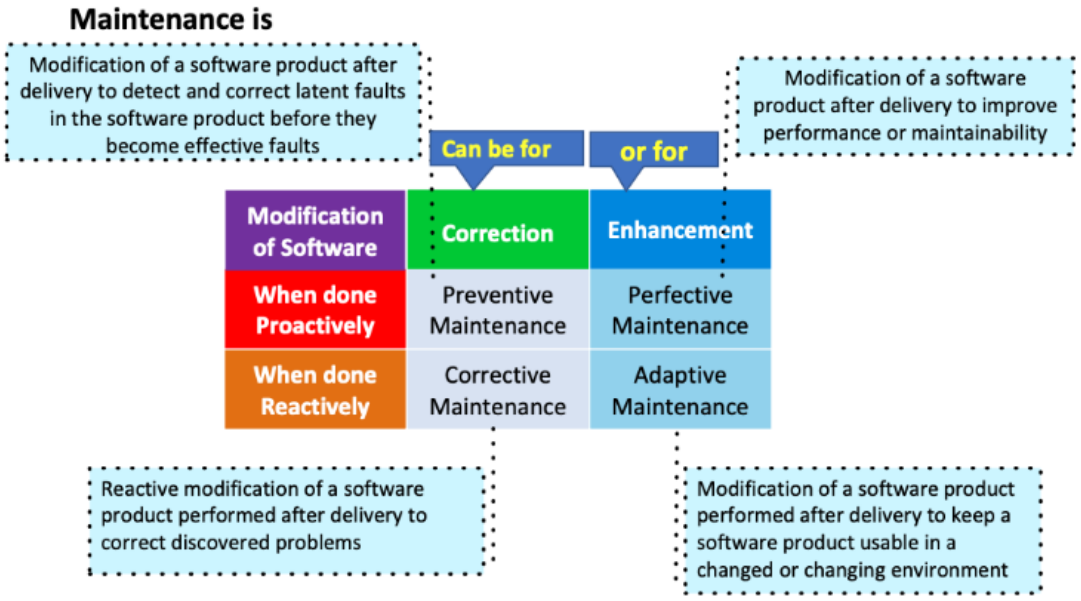
2.Applying the path with restart/no-restart based on the product and patch

3.For a complex product, elaborate set of activities

Develop a retirement plan for system including timeline, communicate and execute that plan
Retire older version and running of new in parallel

Categories of maintenance

- Preventive maintenance: done proactively for correction
  - Modification of software after delivery to detect and correct latent faults
- Perfective maintenance: done proactively for enhancement
  - Modification to improve performance or maintainability
- Corrective maintenance: done reactively for correct
  - After delivery to correct discovered problems
- Adaptive maintenance: done reactively for enhancement
  - To keep product usable in changed or changing environment



Reverse Engineering:

Passive technique to understand a piece of software prior to re-engineering

- Process of recovering specification and design information about the system from source code
- Over time, application may have been corrected, adapted and enhanced
- Application becomes complex, unstable and has unexpected and serious side effects
- Identifies components and interrelationships
- Create a representation of software different from code and documents
- Doesn’t modify product nor create a new product

Re-Engineering:

process of modifying the software to make it easier to understand, change and extend

- Improving maintainability of system at reasonable cost

- Re-Engineered code is expected to result in reduction of CPU Utilization, Readability, Usability and Performance
- Refactoring

Maintenance tips

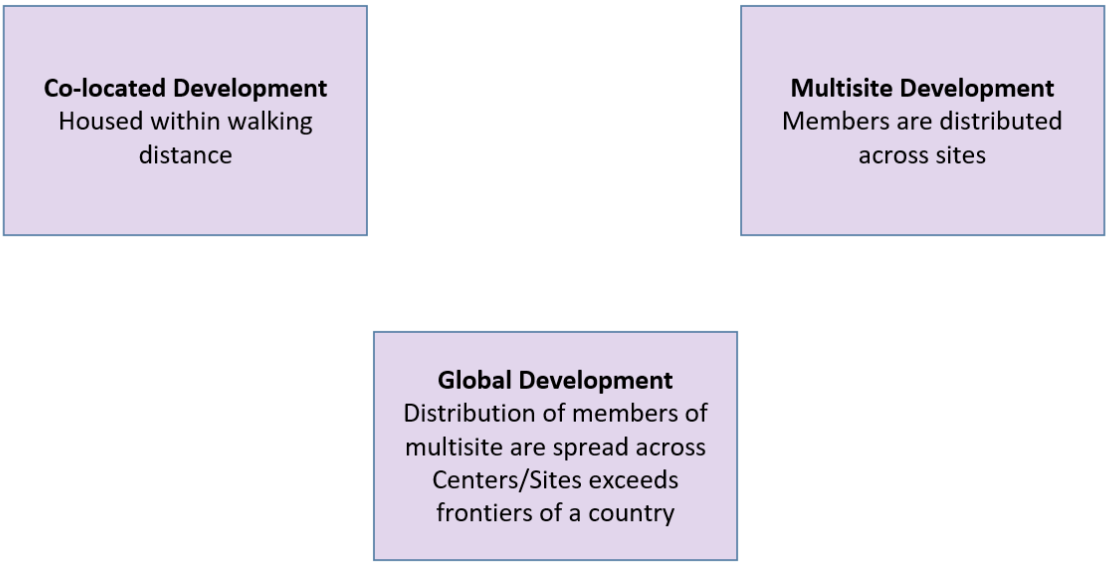
- Reproduce the problem
- check environment variables
- know error codes
- pair programming
- use logs
- testing
- one problem at a time

Global Environment

Global Software Development team

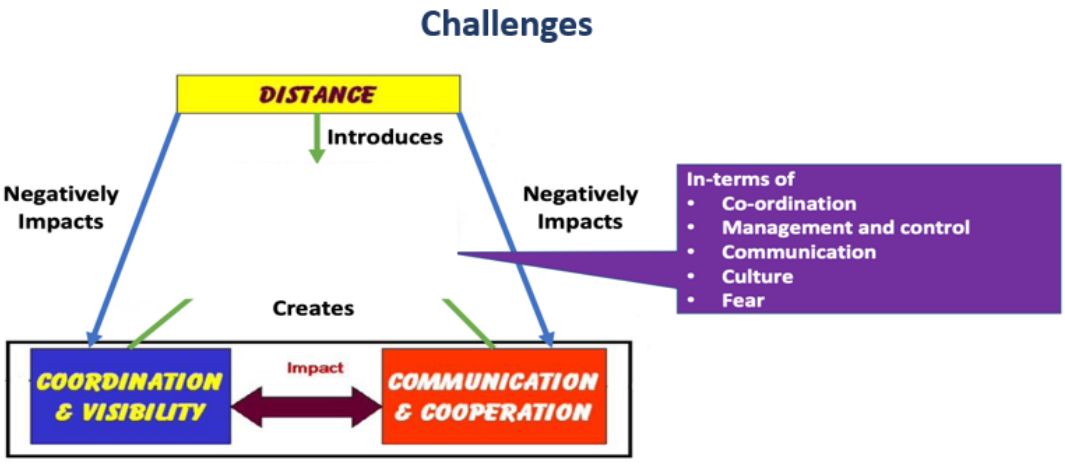
Global SD team: group of knowledge workers around the globe developing commercially viable software

- Use virtual teams
- Same goals and objectives



Types of Development

- Collocated: Housed within walking distance (same floor/building)
- Multisite: members of the team are distributed across multiple sites
- Global: distribution of multisite teams are spread across countries



Rationale behind GSD

- Cost savings by cost arbitrage
- Use time difference to extend number of productive hours

- Larger and global pool of developers
- Locating developers closer to markets and customers
- Use advantage of diversity of stakeholders knowledge and experience

Geographical Distance Challenges

Communication
- Effective information exchange (Less informal exchange, different languages, different domain knowledge)
- Build a team (Cohesiveness, removing "them and us", trust)

Coordination
- Task awareness (Shared mental model)
- Sense of urgency (Perception)

Control
- Accurate Status information (Tracking, stop blaming)
- Uniform Process (different tools and techniques)

Cultural
- Social background (directed, people orientation to task orientation)
- Attitude to authority, national culture, cultural distance
- Projection (different tools and techniques)

Overcoming Geographical Distance Challenges

- Processes and Practices
  - Processes
- Collaboration Models
  - Body leasing or team leasing
  - Team extension/augmentation
  - Project teams
- Teaming models
  - Assembly line
  - Construction (architect + skilled craftsmen + workers)
  - Engineering teams
  - No oversight
  - Practices
- Communication practices

- Project management practices
  - Common ground

Hacking

Hacking	Software engineering
Solves a Problem using non standard approaches or by exploiting weakness	Takes a possible problem and solves it within given criteria (fixed budget, performance benchmarks)
Attracted by novelty and challenges; Weakly motivated by Money and Positions	Could be more creative or original but within constraints of the problem
Tries to stretch what's possible or exists; Once functional and successful, moves on	Picks a problem and fits it within constraints
Often pokes around for a solution; Not easy to backtrack	Craft a solution by understanding why and consider best practices

- White hat hackers
  - Improve security of organisations systems by finding vulnerable flaws to prevent cybercrimes by black hats
- Black hat hackers
  - Subvert technology for stealing something valuable or other malicious reasons like disrupting reputation, corporate espionage and nation-state hacking

Ethics

Ethics: moral principles that govern behaviour or conduction of activity

Ethics - Introduction

- As computer scientists and software engineers you will be faced with making difficult decisions
  - **Personal ethics** (Integrity, Objectivity, Professional Competence, Confidentiality)
  - **Professional ethics**
    - Reuse of source code
    - Reuse of copyrighted materials
    - Enforcing strength of passwords
    - Knowledge of flaw or bug
    - Design of potentially dangerous software feature
    - Discovery of security flaw
- Often, there are clear cut laws to dictate the decision

Personal ethics: set of values or viewpoints an individual uses to influence or guide his/her personal/social/professional behaviour

- Values are reinforced and strengthened from childhood
- Dependent in context of individual

Business ethics: set of values a business and "individuals when doing business" use to influence/guide their behaviour and actions

- Focus on behaviours that pertains to work/business environment
- Business must be profitable but not overtly profitable

Professional ethics: cover behaviour and judgments of a person while performing his profession

Golden rules

- Treat others as you want to be treated
- Do not pirate software

Principle of beneficence

- Do what is right and good
- Prioritising doing good

Principle of least harm

situations where no choice appears beneficial

Principle of respect for autonomy

allow people to make decisions without inference; Autonomy is the source of all obligations

Justice ethical principle

decision is fair to all parties involved

Ethics

- **Public:** software engineers act consistently with public interest
- **Client and Employer:** Act in a manner that is in best interests of their client or employer and consistent with public interest
- **Product:** Products and Related modifications meet highest professional standards
- **Judgement:** maintain integrity and independence in professional judgement
- **Management:** subscribe to and promote an ethical approach to management
- **Profession:** advance the integrity and reputation of the profession consistent with public interest
- **Colleagues:** fair and supportive of their colleagues
- **Self:** lifelong learning and promote ethical approach

Intellectual Property [IP]

- Intellectual Property: property that includes tangible creations of human intellect
- Types: patents, copyrights, trademarks, trade secrets
- Patents: focuses on methods to do something (utility patents), look-and-feel (design patents)
- Copyrights: focus on expressions of implementations to protect original work if authorship
- Important to engineers and companies
- Provides protection of investment
- Patent/copyright infringement can be costly
- IP is an asset
- Patents have value and can be traded between companies

- Patents: focus on method to do something (Utility patent) or look-and-feel (design patents)
- Copyright: expression or implementations (books, songs, source code) to protect original works and authorship

IT System Management

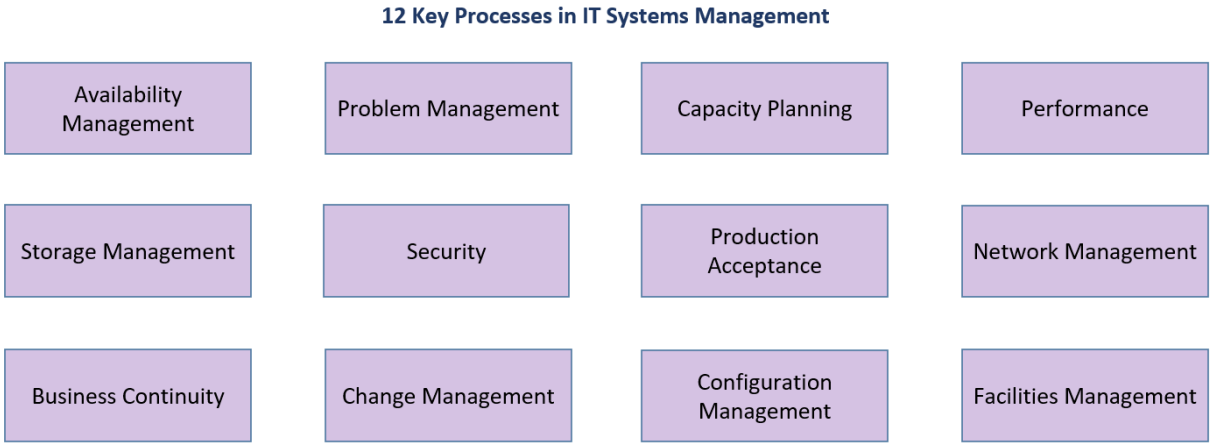
IT Systems Management:

- how processes and service are administered to ensure that IT infrastructure provides stable and responsive IT environment
- Management of processes associated with IT infrastructure to deliver the right set of services at right quality at competitive costs

IT Operations:

- support activities that the IT department needs to support within a large organization
- These applications impact the business and are offering IT services are deployed by IT operations or infrastructure management teams

- Routine and day-to-day operational tasks and maintenance, performing data back-ups, restoring system after service outage/update and would also look at organizations disaster recovery plans
- Configuring and tuning servers and other configurable infrastructure components to optimize their performance
- Allocating IT resources where they are needed to promote effective service delivery
- Monitoring and measuring performance of IT infrastructure



IT Services

refers to the application of business and technical expertise, and offering services which enable organizations in creation, management, optimization, or to provide access to information and business processes

ITSM Processes

Sl. No	ITSM Process	Question	Characteristics
1	Availability Management	Timely Recovery and approaches to reduce the frequency and duration of outages	▪ Managing expectation of the expected/agreed/guaranteed levels of functioning of the Services and Systems in a cost-effective, consistent and timely manner ▪ Uptime-Availability, Non-responsiveness-Downtime ▪ Achieved by 7 R's 1. Redundancy 2. Reputation 3. Reliability 4. Repairability 5. Recoverability 6. Responsiveness 7. Robustness
2	Performance/Tuning	How do we maximize throughput and minimize response times	▪ Maximize throughput ▪ Minimize response times ▪ Workload keeps changing and hence a challenge ▪ Performance management & tuning in an IT infrastructure can happen Servers, Disk Storage, Databases, Networks
3	Production Acceptance	How do we ensure the applications are deployed consistently and successfully	▪ Production acceptance is a methodology and the process for consistently and successfully deploying application systems into a production environment, regardless of the platform where it sets a positive impression for the Application. ▪ It is also important to maintain integrity of the environment post the deployment.

Sl. No	ITSM Process	Questions	Characteristics
4	Change Management	How do we consistently manage Change	▪ Change is the modifications to and in the IT environment, which can impact the stability and responsiveness of the IT infrastructure environment. ▪ Change is done for adding value (responding to change, managing risks or for resource optimization) or it could be to fix things with goal of reducing disruption and enhancing effectiveness (bringing in stability and responsiveness) ▪ Change Management involves controlling change (Request, Prioritize, approve) and Change co-ordination (Collaboration, Schedule, Communicate and implement).
5	Problem Management	Large number of complex systems and diverse services. How do we manage support. Service Request, Incident and Problem	▪ Process used to identify, log, track, resolve, and analyze problems impacting the IT services. ▪ A call initiated by a customer as an incident, gets analyzed and logged as a problem or resolved as a Service request. ▪ Service desk -advantages and disadvantages of a segregated or an integrated Service desk.

Sl. No	ITSM Process	Questions	Characteristics
6	Storage Management & Network Management	Storing information, Performance, Capacity .. Integrity, Reliability and Recovery	▪ Approaches to ensure performance, Integrity, reliability & recoverability of storage data ▪ Approaches for maximizing reliability and utilization of the network
7	Configuration Management	Information on configurations	▪ Focused on accurately documenting the interrelationships of varying versions of infrastructure hardware and software.
8	Capacity Management	How do we scale on the capacity	▪ Predicts when, how much, and what type of additional resources would be needed, to support accurately forecasted workloads of Services
9	Strategic Security Management	Strategic perspective for Security	▪ Intended to set up security controls to safeguard the availability, integrity, and confidentiality of designated data & programs against unauthorized access, modification, or destruction ▪ Achieved typically through Security testing, managing security incidents, security review etc.

Sl. No	ITSM Process	Questions	Characteristics
10	Business Continuity Process	Differentiating Business continuity and DR. Cost as a consideration	▪ Business continuity is a methodology to ensure the continuous operation of critical business systems in the event of widespread or localized disasters to an infrastructure environment. ▪ Business continuity management includes identification of the disasters (as a risk), contingency planning (mitigation for the risk) for the Business to get back functionally or for Business recovery and plans for getting back to a functional state till the Disaster recovery can happen.
11	Facilities Management	Differentiator and Impact	▪ Focused on ensuring that the facilities support the IT infrastructure in terms of expectations. Power, Air Conditioning, Humidity, Physical Security, Culture etc.

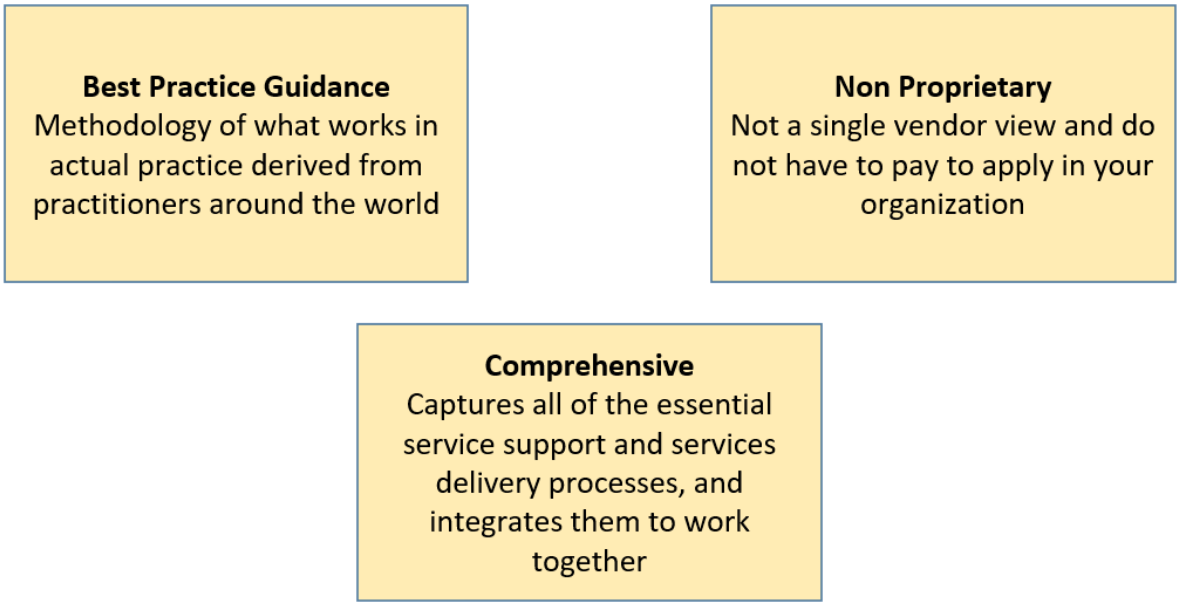
ITSM Framework

- Number of ITSM frameworks which businesses can use. Some frameworks are targeted to at specific industries
- **Popular Frameworks**
- **ITIL (V4):** framework of best practices for delivering IT services
- **Business Process Framework:** designed for telecommunications service providers
- **COBIT (Control Objectives for Information and Related Technologies):** IT governance framework
- **FitSM:** simplified, streamlined service management framework aligned with ISO/IEC 20000
- **ISO/IEC 20000:** considered the international standard for IT service management and delivery

ITIL – IT Infrastructure Library

- ITIL is a framework or set of IT Service management best practice processes that **align IT services with business needs**
- Approaches that have worked on some scenarios, for selection, planning, delivery, maintenance and overall lifecycle of IT services

- It helps businesses manage risk, strengthen customer relations, establish cost effective practices and build stable IT environment that allows for growth, scale and change



ITIL – 3: IT Service Lifecycle

1.**Service Strategy:** Understands organizational objectives and customer needs and provides strategic guidance for investments in services

a.Includes service value definition, business case development, service assets, market analysis and service provider types

2.**Service Design:** turns strategy into a plan for delivering business objectives and technology

3.**Service Transition:** develops and improves capabilities for introducing new services into supported environments

a.Relates to delivery of services required by business into live/operational use

4.**Service Operation:** Manages services in supported environments. Provide best practice for achieving delivery of agreed levels of service

5.**Continual Service Improvement:** incremental and largescale improvements to services

ITIL V4 - Holistic Service Management

ITIL V4 facilitates value to customers and stakeholder by looking at service management in 4 dimensions

a.**Organizations and People:** organization needs culture that supports objectives and the right level of capacity and competency among workforce

b.**Information and Technology:** includes the information, knowledge and technologies required for management of services

c.**Partners and Suppliers:** organizations relationships with other businesses in design, deployment, delivery, support and continual improvement

d.**Value Streams and Processes:** various parts of the organization work in integrated and coordinated way is important to enable value creation

- Expands from processes to practices by factoring in culture, technology, information and data management
- 34 management practices; various types of guidance, such as key terms and concepts, success factors, key activities, information objects, etc.
- **Plan**
- **Design and Transition**
- **Improve**
- **Obtain/Build**
- **Engage**
- **Deliver and support**

Aspect	ITIL 3	ITIL 4
Approach	IT service lifecycle approach	Holistic service management approach
Core Structure	Service strategy, design, transition, operation,	Service value system with a service value chain

Aspect	ITIL 3	ITIL 4
	continual service improvement	
Focus Areas	Emphasis on service lifecycle stages	Four dimensions: organizations, information and technology, partners and suppliers, value streams and processes
Practices	Emphasizes processes	Expands from processes to practices (34 management practices)
Service Value System	Not explicitly defined	Defined around the service value chain, including six key activities: Plan, Improve, Engage, Design and Transition, Obtain/Build, Deliver and Support
Integration of Modern Concepts	Not specified	Includes factors like culture, technology, information, and data management
Flexibility	Follows a structured lifecycle approach	Offers a flexible operating model for creation, delivery, and continual improvement
Terminology Changes	Uses service strategy, design, transition, operation, and continual service improvement	Introduces new terminology such as service value system, value chain, and management practices
Number of Management Practices	Not specified	Includes 34 management practices

What is DevOps?

Combination of cultural philosophies, practices and tools that increase the organisations ability to deliver (Deploy and support) apps and services at high velocity

- Repetitive manual labor (error prone) is removed whenever possible

- Blend of development and operations to represent collaborative/shared approach to tasks perform by application development (SDLC) and IT Operations
- DevOps in a sense is an extension of the agile way of doing things where you can deliver functionality incrementally and at a faster rate
- Deployment systems are maintained by DevOps engineers
- Operational philosophy that promotes better communications and knowledge

Terminologies part of devops

SDLC (Waterfall, Agile)

Operations Methodologies (ITSM, ITIL)

Source Code Management

Terminologies associated with development, release and deployment

Version control

- Version Control System to record changes to files stored in system
- Developers make changes individually or within groups and save these changes through commits/revisions
- Ability to commit, compare, merge and restore versions
- Richer cooperation and collaboration with minimum risk

Devops Pipeline

- Collaboration
- Affinity
- Tools
- Scaling

Continuous integration

- Continuously merge code written by developers with mainline/ master branch
- Developers take snap shot, make changes, check-in and merge
- If merge conflicts are found, they are resolved and then merged
- If not frequently integrated, results in big bang integration

  • Risk of intricate integration conflicts
  • Incompatibilities leads to failure of build
  • Issues in time, effort and cost overruns project
- Executable built is run with some sanity tests to ensure total compilation is successful

Continuous delivery

- Frequent shipping of code to given environment
- Integrated bits are ready to be deployed; supports release of new changes quickly in a sustainable way
- Triggered by last step in Continuous Integration
- Drives business outcomes like deploy on demand
  • Supports faster time to market and higher quality

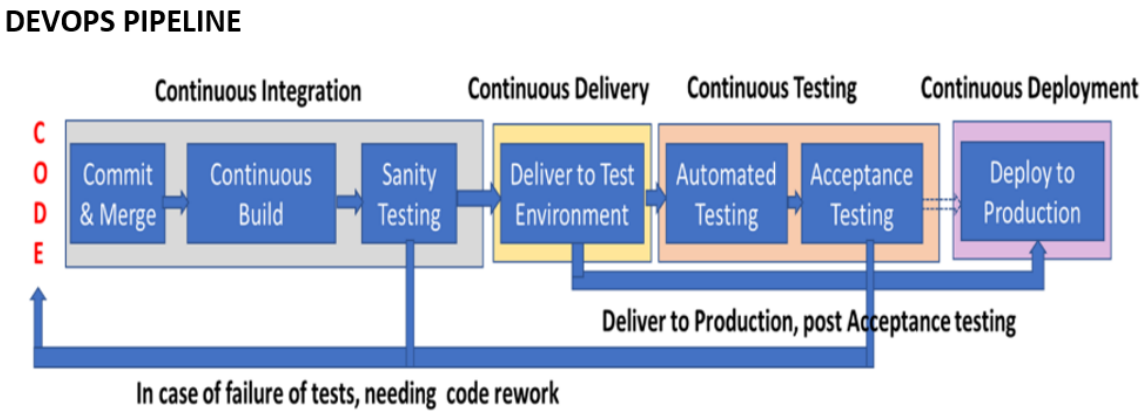
Continuous testing

- Executing predominantly automated tests to validate code to ensure the app works as envisaged and is free of bugs/defects
- Designed to execute with minimum wait times and provide early feedback and support detection of risks that are most critical
- Post Continuous Integration and could be executed daily or whenever app is updated
- Entry criteria for testing by QE: successful completion of unit and integration tests
- Quality Assurance environment involves
  - Static code analysis

- Validation of functional and non-functional requirements
- Security
- Performance tests
- Acceptance tests

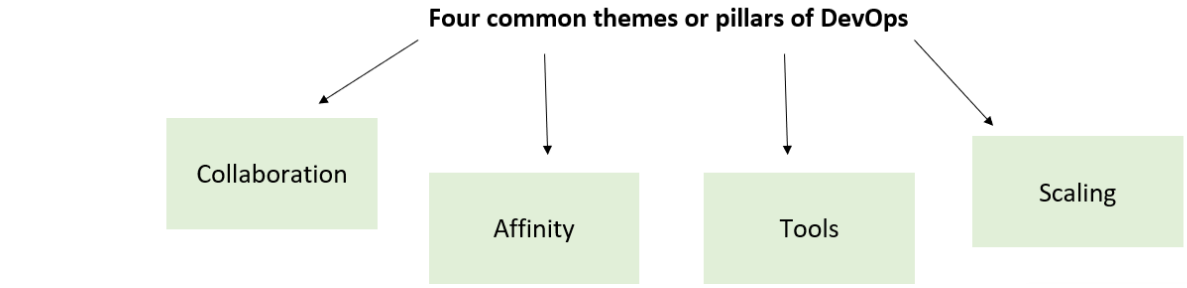
Continuous deployment

- Deploy automated testing validated product
  • Immediately and autonomously
- Used in highly mature devops teams
- Most of the times, validated integrated components are batched together and then deployed to customer environment
- Moves away from approach of moving code and testing
- Allows engineering organisations to focus on core business needs instead of infrastructure overhead



Pillars of Dev Ops

Recall the 4 pillars of DevOps



Affinity

- Strong relationship between individuals, teams and departments along with collaborative relationships to ensure developers and operations work together
  - Teams with strong affinity
- Affinity: measure of strength of relationship between individuals, teams, business units and even companies
- Building affinity: building inter-team relationships by navigating differing goals or metrics while keeping shared organisation goals, fostering empathy and learning
- Measure by
  - Shared time
  - Intensity of relationships
  - Reciprocity of shared stories
  - Reciprocity of support
- Achieved by having
  - Shared values
  - Consistent team culture
  - Team cohesion

Tools

• Acts as accelerator, driving changes based on culture and direction
• Ease of use impacts acceptance and proliferation of aspects of culture as they become common language between teams
• Tools beneficial to subset of teams/lack of tools gets in the way of working well as individuals/teams

• If cost of collaboration is high, not investing in tools raises the cost
• Eg: system provisioning tools, build tools, automation tools, testing tools, monitoring tools, log extraction tools, deployment tools

Enhance effectiveness of devops

Scaling

• Focus on processes and pivots that organisations must adopt throughout lifecycles
• Considers how other pillars can be applied as organisations grow and mature or shrink
• Different considerations (technical and cultural) for words at different scales
• Could be for organisation, infra, teams (hiring, retention, outsourcing), complexity and workload

Collaboration

- Working towards specific outcome through interactions and input and support of multiple people and groups
- Cooperation of software development and operations team is essential
- Teams that don’t work well face challenges while working individually and at the inter-team level
- Effective collaboration involves
  • Communication
  • Equal participation
- Theory of mind: others have different perspectives
- Different aspects of effective working
  • Shared goal
  • Empathy for collaborative culture through effective communication
  • Teams are effective when based on relationships of trust, empathy and reciprocity
- Empathy: different perspectives based on backgrounds
- People have different goals and needs

- Different cognitive styles
  - Hierarchies are less emphasised

DevOps	Agile
Culture and approach which looks to remove silos of development activities of building and operations activities of deployment, support and upkeep. Focuses on increasing velocity of deployment by devops pillars	Methodology/iterative SDLC which supports changes, ensures collab between developers and stakeholders, reduces planning overhead and delivers periodically
Focuses on constant testing and delivery	Focuses on constant changes
Target is to give end-to-end business solutions and fast delivery	Target is software development
Focuses on operational and business readiness	Focuses on functional and non-functional readiness

Jenkins

Jenkins is an open-source Java tool designed for continuous integration and delivery. It provides a platform with plugins for integrating and automating various aspects of the software development lifecycle.

Key Features:

1. **Test Report Generation:** Jenkins can generate test reports, helping developers understand the status and results of their tests.
2. **Version Control System (VCS) Integration:** Jenkins seamlessly integrates with different Version Control Systems, allowing developers to commit and manage their source code efficiently.
3. **Artifact Repository Integration:** It can push artifacts (compiled and packaged code) to repositories, making it easy to manage and share different versions of software.
4. **Deployment:** Jenkins supports direct deployment to production or test environments, streamlining the deployment process.
5. **Build Status Notification:** Stakeholders are notified of the build status, whether it was successful (green) or failed (red), enabling quick response to issues.

6. **Plugin Architecture:** Jenkins has a plugin system that allows integration at various stages, enabling customization and extension of its functionality.
7. **User Interface (UI) Customization:** Jenkins supports UI customization, allowing users to tailor the interface according to their preferences.
8. **Jenkins Pipeline:** It introduces a pipeline concept where each stage is represented by a cell, and columns display the names of jobs running at each stage. Green indicates success, while red indicates failure.

Continuous Integration (CI):
Jenkins supports continuous integration by automating the following steps:

- Developers frequently commit source code to a shared repository.
- Jenkins builds the repository for every commit, facilitating early detection of issues.
- Supports integration with various tools and deployment on test servers.

Continuous Testing:
Jenkins promotes continuous testing, enabling a higher velocity of deployment. It schedules and executes tests, displaying relevant statistics such as the number of tests, time taken, failures, error messages, and test trends.