SOFTWARE TESTING

INTRODUCTION TO SOFTWARE TESTING

OBJECTIVES

01
Introduction to Software Testing

02
Importance of Software Testing

03
Verification and Validation

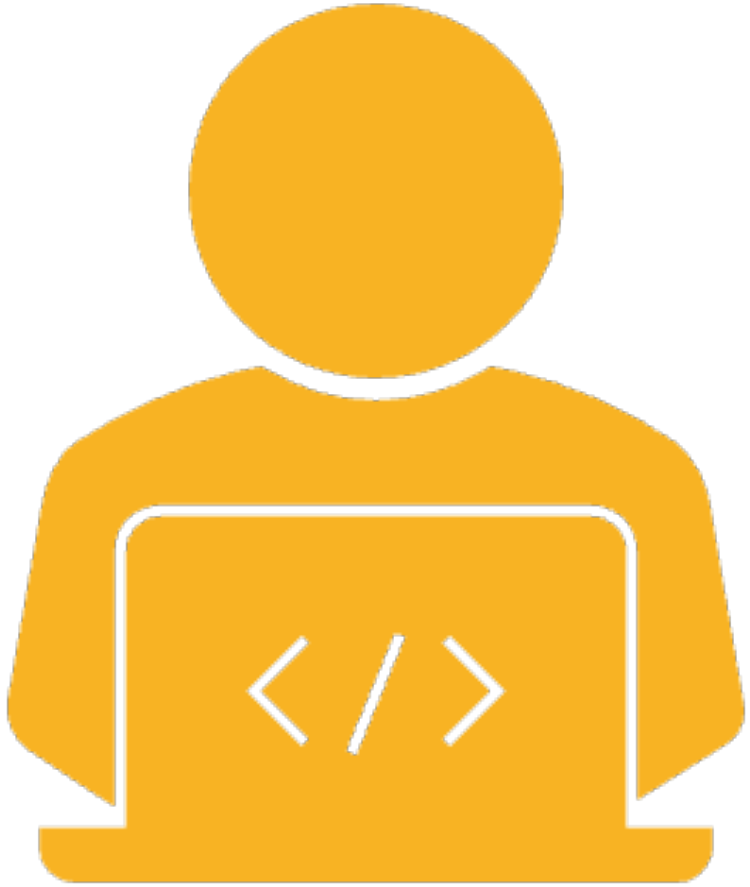
04
Software Test Lifecycle

05
Testing Process

GOAL OF TESTING

- The goal of a software tester is:
  - Ensure software meets the requirement
  - To find bugs
  - To find them as early as possible
  - Make sure they get fixed

- Testing is a means to:
  - Provide confidence and minimize the risk



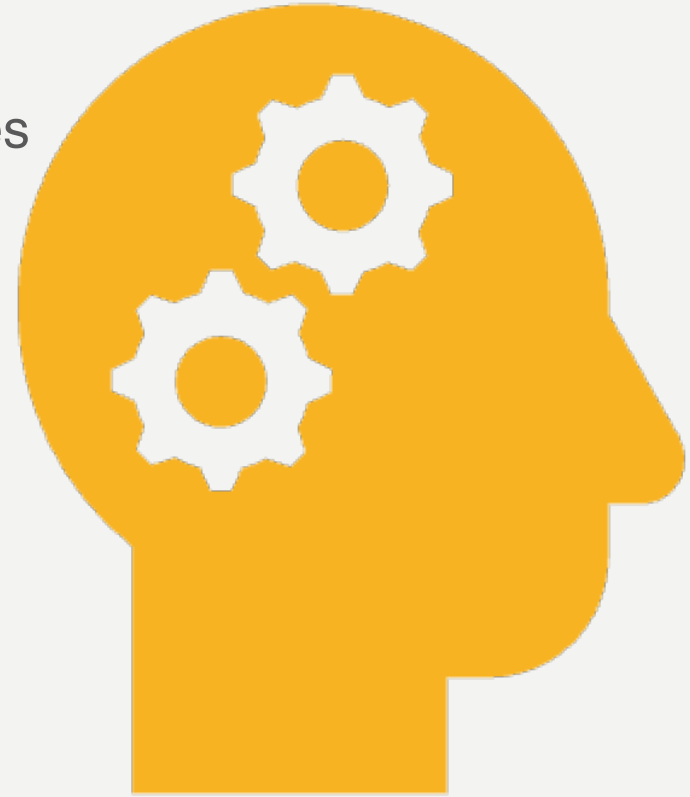
WHY IS TESTING NEEDED

- Complexity of software makes it error prone
- Untested or weakly tested software can cause losses in real life
- Software testing may be required for compliance with contractual or legal requirements
- Software bugs can cause harm to a person, to the environment or to a Company
- Testing should provide sufficient information to the stake holders for decision making regarding release of the software

WHAT MAKES A GOOD TESTER

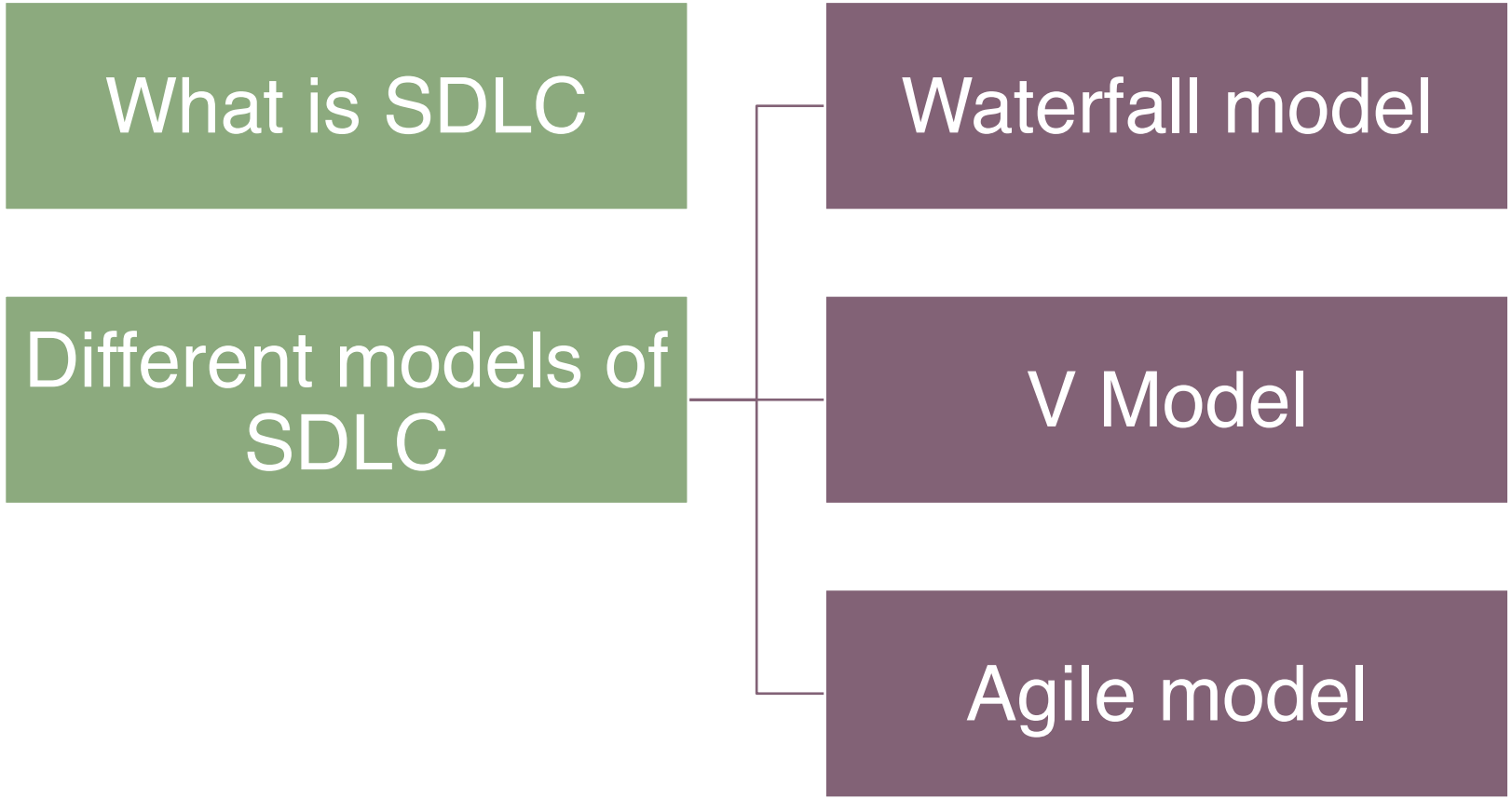
The tester’s mindset must have the below mentioned attributes

- Keen Observation
- Questioning Skills
- “Never give-up” attitude
- Interpersonal Skills



SDLC MODELS

AGENDA

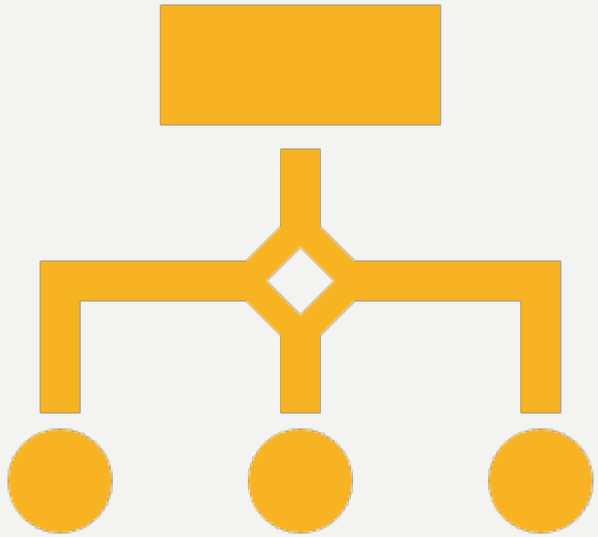

What is SDLC
Different models of SDLC
Waterfall model
V Model
Agile model

SDLC

A software project goes through certain defined stages which together are known as Software Development Life Cycle (SDLC)

Following are the phases that are part of SDLC viz;

- Project Planning
- Requirements Analysis
- Design
- Coding (Implementation)
- Verification (Integration & Testing)
- Maintenance



SDLC Models

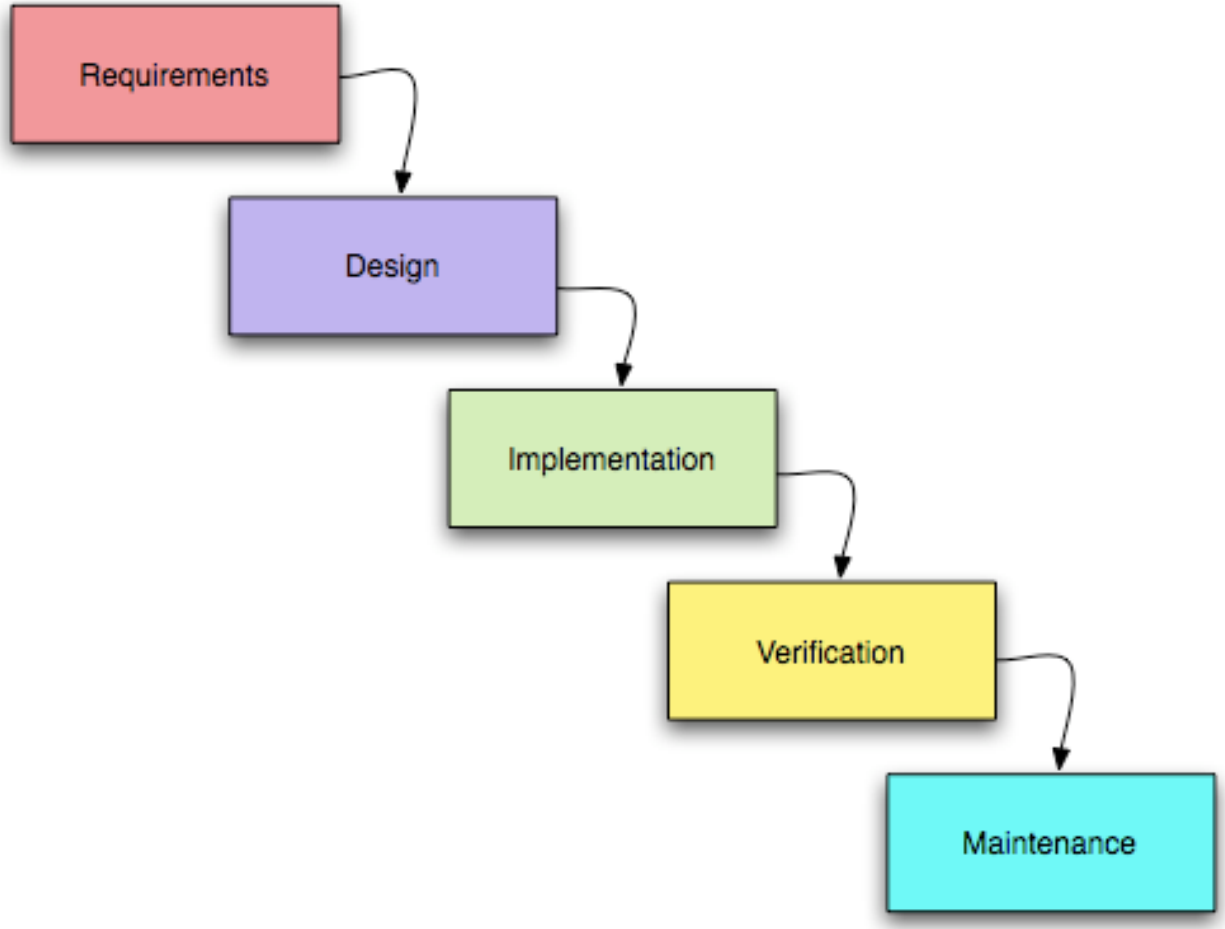
Most widely used SDLC models are as below:

- Waterfall Model
- V Model
- Spiral model
- Increment Model
- Agile

Waterfall Model

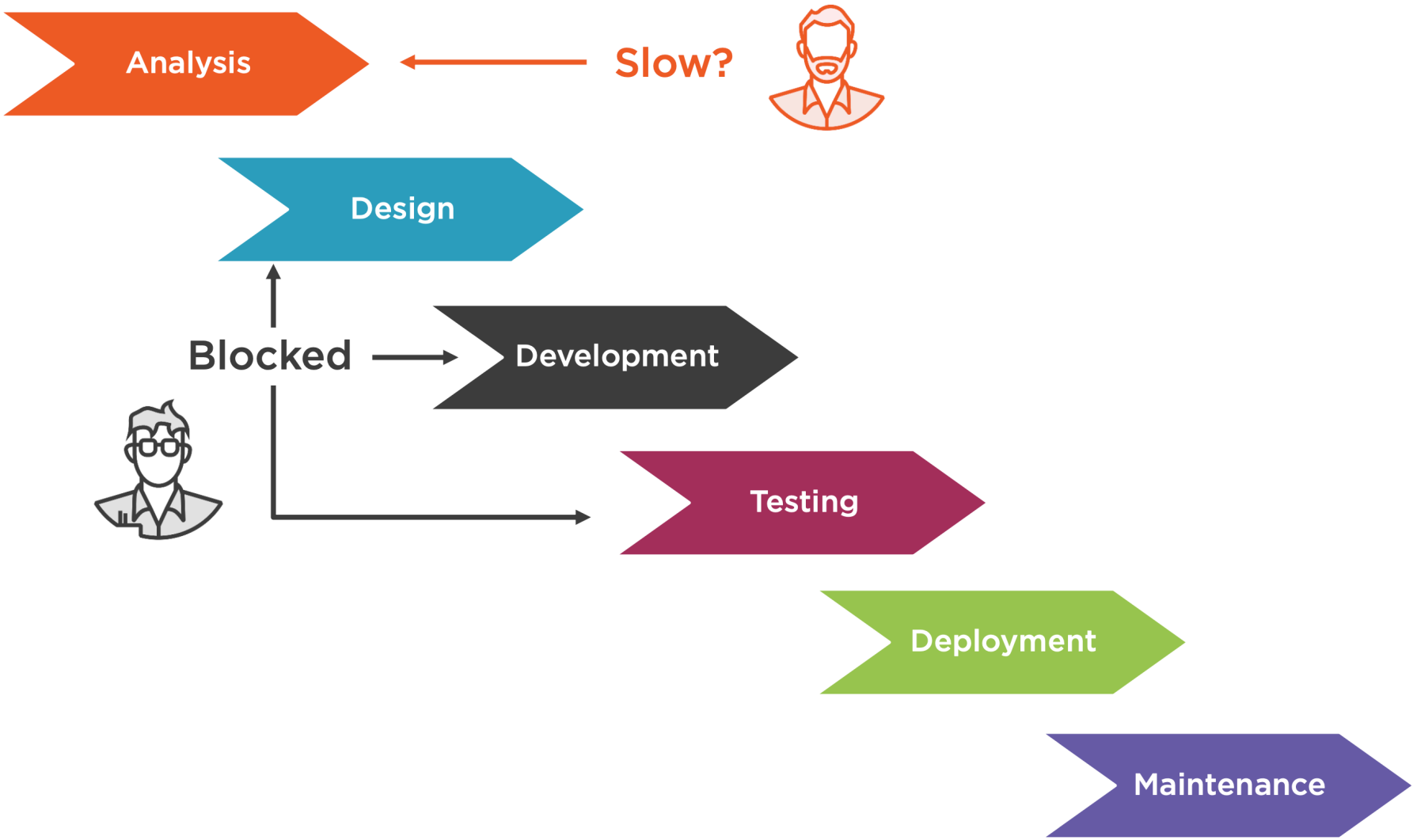
- It is a linear and sequential approach to software design
- The name of this method comes from its biggest draw back. Once water has passed over the edge of the waterfall it cannot go back, similarly, once a stage has been completed there is no going back to it
- The linearity of waterfall method is both its weaknesses and also its strengths.

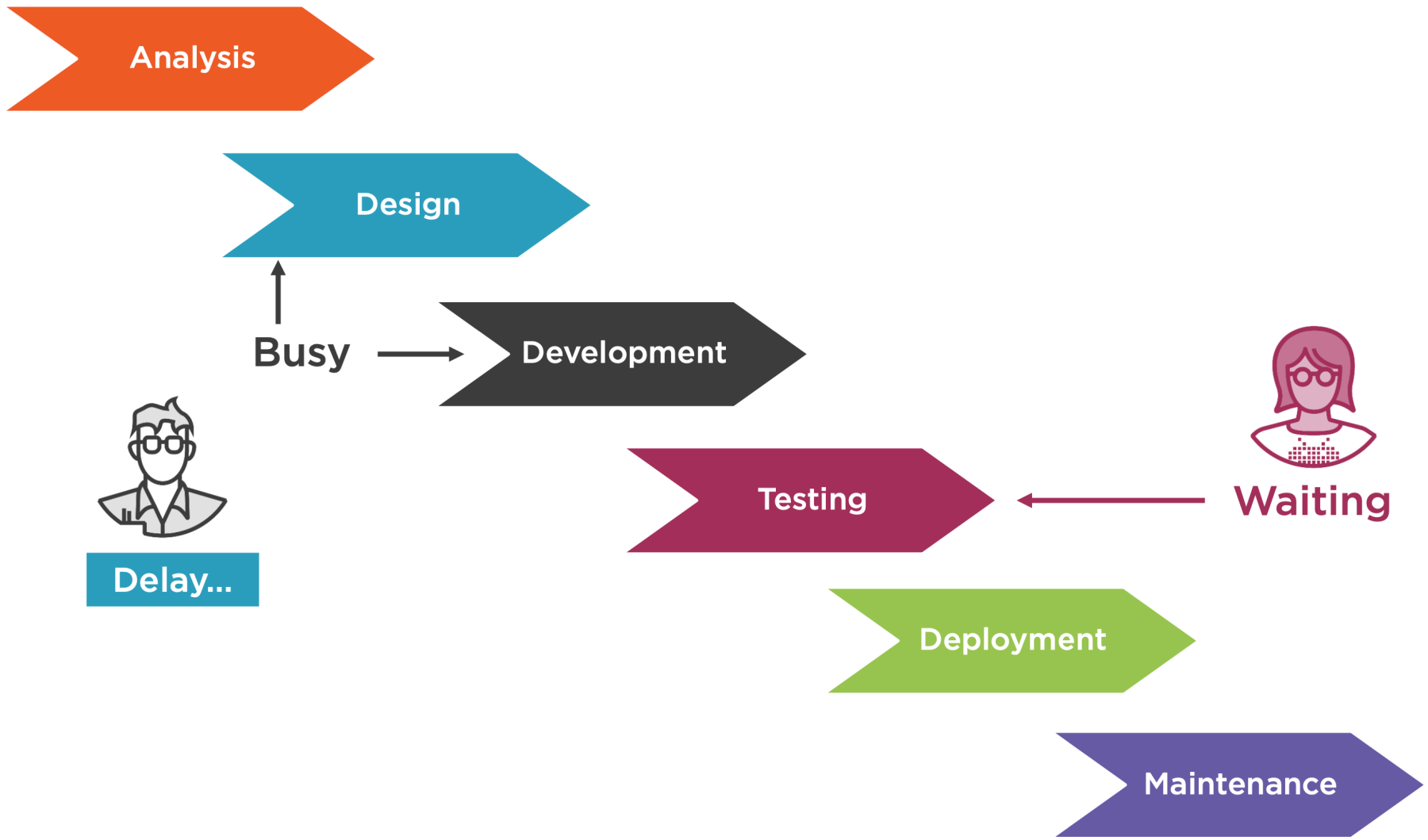
WATERFALL MODEL



Advantages of waterfall model

- Each stage can be designated to a separate team allowing for greater control on the project.
- The whole process can have deadlines set for each stage which should, in theory, make for the project being delivered on time.
- Waterfall discourages revisiting and revising any prior phase once it's complete.
- This model works well only for smaller projects where requirements are very well understood


Analysis
Slow?
Design
Blocked
Development
Testing
Deployment
Maintenance


Analysis
Design
Busy
Development
Testing
Waiting
Delay...
Deployment
Maintenance

Disadvantages of waterfall model

- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage
- No working software is produced until late during the life cycle
- High amounts of risk and uncertainty

WHEN TO USE THE WATERFALL MODEL

Requirements are very well known, clear and fixed

Technology is understood

Resources with required expertise are available

The project is small in scale

V MODEL

‘V’ stands for the terms Verification and Validation.

This model developed as a result of the evolution of software testing.

Testing techniques and various kinds of testing got its due focus that led waterfall model to evolve into V-model.

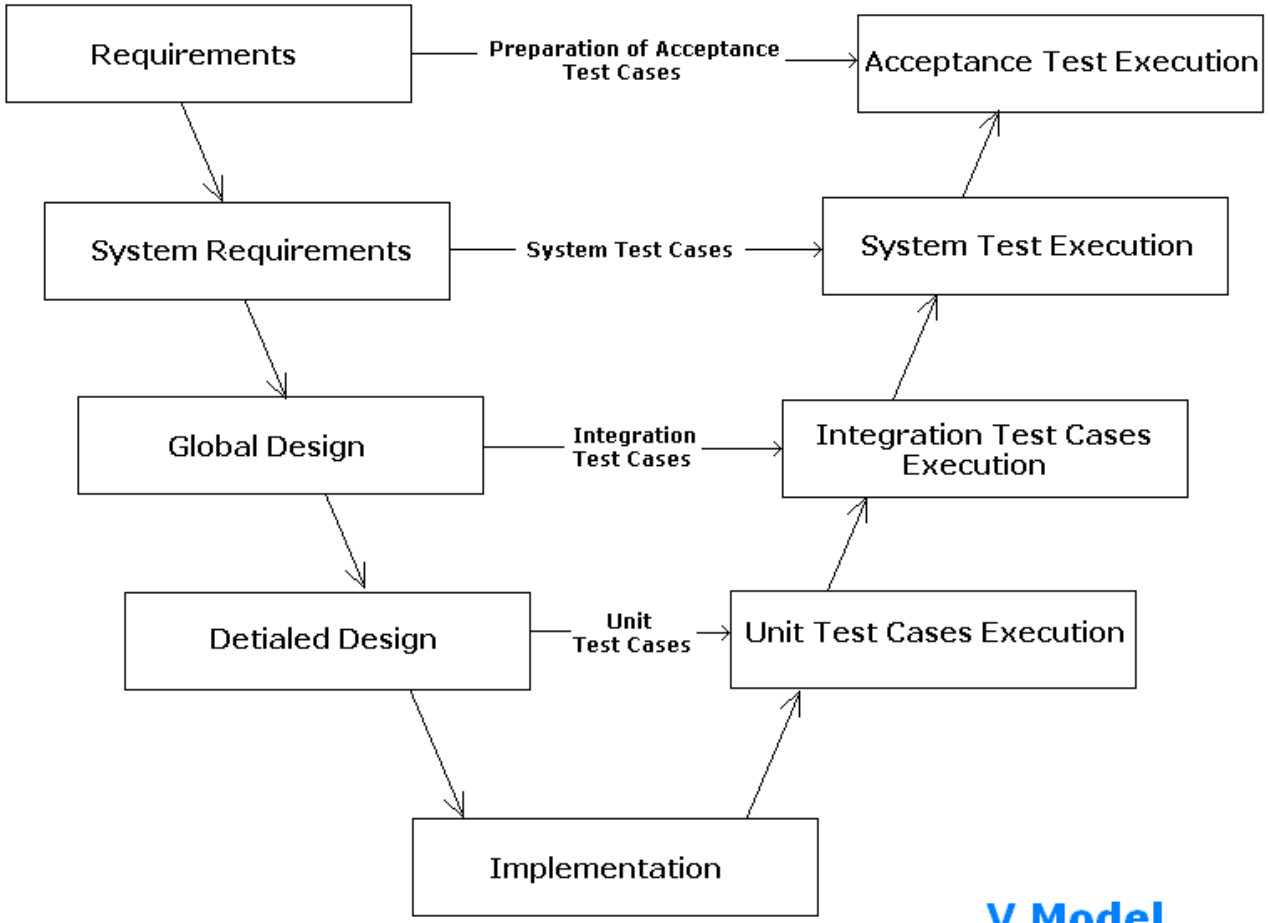
V MODEL

Testing of the product is planned in parallel with a corresponding phase of development

Each phase must be completed before the next phase begins.

Verification activities during requirements and design phase will produce deliverable like SRS and design documents.

V MODEL



V Model

V MODEL

Validation activities for each of the different type of test execution requires corresponding test cases to be prepared.

These test cases are prepared by taking help of corresponding requirements and design documents produced during verification activities.

This is what is represented in the 'V-Model' diagram as a input link between the descending form of verification activities and ascending form of corresponding validation activities.

ADVANTAGES OF V-MODEL

Testing activities like planning, test designing happens well before coding. Hence higher chance of success over the waterfall model

Proactive defect tracking – that is defects are found at early stage.

Avoids the downward flow of the defects.

Works well for small projects where requirements are easily understood.

DISADVANTAGES OF V-MODEL

Software is developed during the implementation phase, so no early prototypes of the software are produced.

If any changes happen in midway, then the test documents along with requirement documents has to be updated.

When to use the V-model

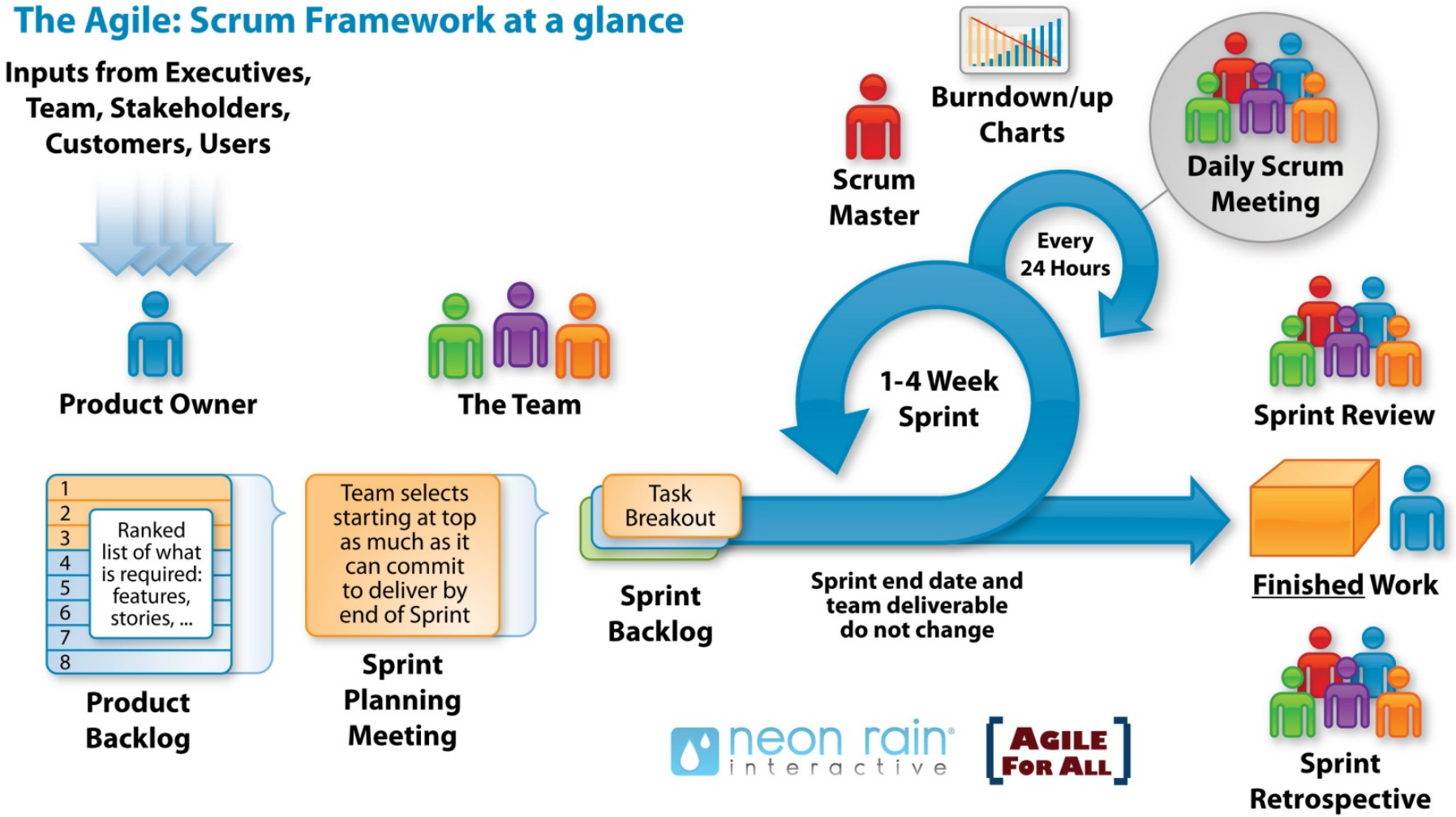
- For small to medium sized projects where requirements are clearly defined and fixed.
- Since, no prototypes are produced, there is a very high risk involved in meeting customer expectations.

Agile Model

- It is based on iterative and incremental development
- Requirements and solutions evolve through collaboration between self-organizing, cross-functional teams
- Agile methods work in cycles, typically of a week or a month, and at the end of each cycle, the priorities of the project are re-evaluated.

Some of the well-known agile software development methods include:

- Scrum
- Extreme Programming (XP)
- Agile Unified Process (AUP)
- Feature Driven Development (FDD)
- Kanban


The Agile: Scrum Framework at a glance
Inputs from Executives, Team, Stakeholders, Customers, Users
Product Owner
The Team
Scrum Master
Burndown/up Charts
Daily Scrum Meeting
Every 24 Hours
1-4 Week Sprint
Sprint Review
1
2
3
4
5
6
7
8
Ranked list of what is required: features, stories, ...
Product Backlog
Team selects starting at top as much as it can commit to deliver by end of Sprint
Sprint Planning Meeting
Task Breakout
Sprint Backlog
Sprint end date and team deliverable do not change
Finished Work
Sprint Retrospective
neon rain interactive
AGILE FOR ALL

Agile Model - Scrum

The four main principles of the Agile methods are:

1. Individuals and interactions over processes and tools

2. Working software over comprehensive documentation

3. Customer collaboration over contract negotiation

4. Responding to change over following a plan

Scrum roles

Major roles in Scrum development process are:

- Product Owner
- Team
- Scrum Master
- Stakeholders
  (Customers, end-users, vendors)

Product Owner

- Represents the stakeholders and is the voice of the customer.
- He or she is accountable for ensuring that the team delivers value to the business.
- This role should not be combined with that of the Scrum Master
- Ranks and prioritizes deliverable items

Team

- Responsible for delivering potentially shippable product increments at the end of each Sprint.
- Individuals with cross-functional skills who do the actual work; viz; analyze, design, develop, test, technical communication, document, etc.
- Is self-organizing
- Size is of 7 to 10 members

Scrum Master

- Accountable for removing impediments to the ability of the team to deliver the sprint goal or deliverable
- He or she is not the team leader
- Enforces compliance of rules of Scrum, often chairs key meetings, and challenges the team to improve

Stakeholders

- They are the people who enable the project and for whom the project produces the agreed-upon benefits
- They may be involved in the Scrum process during the Sprint Review

Sprint

- A "time-boxed" effort, i.e. it is restricted to a specific duration
- Each sprint is preceded by a Sprint planning meeting, where the tasks for the sprint are identified and an estimated commitment for the sprint goal is made
- Sprint is followed by a review or retrospective meeting, where the progress is reviewed and lessons for the next sprint are identified

Product Backlog

- An ordered list of requirements that is maintained for a product
- It consists of features, bug fixes, non-functional requirements, etc. - whatever needs to be done in order to successfully deliver a working software system
- Product Owner is ultimately responsible for ordering the items

Sprint Backlog

- The list of work the development team must address during the next sprint
- Tasks on the sprint backlog are never assigned; rather, tasks are signed up for by the team members as needed during the daily scrum according to the set priority and the Development Team member skills
- This promotes self-organization of the Development Team and developer buy-in

Agile Model - Scrum Process

- The list of Sprint backlog is filled based on the team's Capacity of how much they can complete in the next Sprint.
- This promotes self-organization of the Development Team and developer buy-in
- Once a Sprint's Product Backlog is committed, no additional functionality can be added to the Sprint except by the team

Agile Model - Scrum Process

- Once a Sprint has been delivered, the Product Backlog is analyzed and re-prioritized, if necessary, and the next set of functionality is selected for the next Sprint

Burn down chart

- A chart showing remaining work in the sprint backlog is prepared and is publicly displayed

- It is updated every day

Advantages of Agile model

- Customer satisfaction by rapid, continuous delivery of useful software
- Working software is delivered frequently (in weeks rather than months)
- Close co-operation between business and developers
- Regular adaptation to changing circumstances

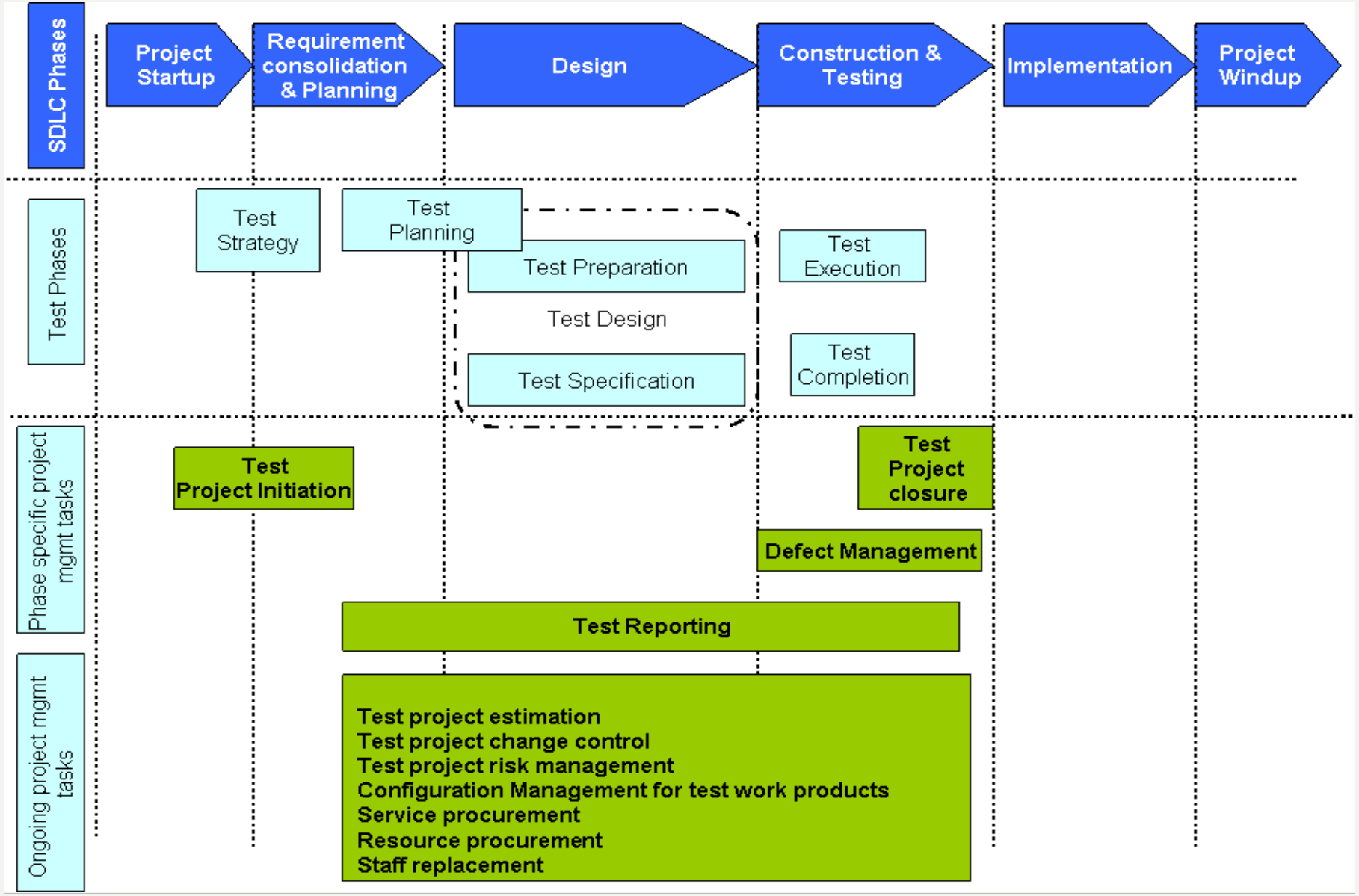
Disadvantages of Agile model

- For large projects, it is difficult to assess the effort required at the beginning of the software development life cycle
- Lack of emphasis on necessary designing and documentation, may have adverse impact later
- Without larger group of experienced resources in the team, it is difficult to take quick decisions as there is less time to do research

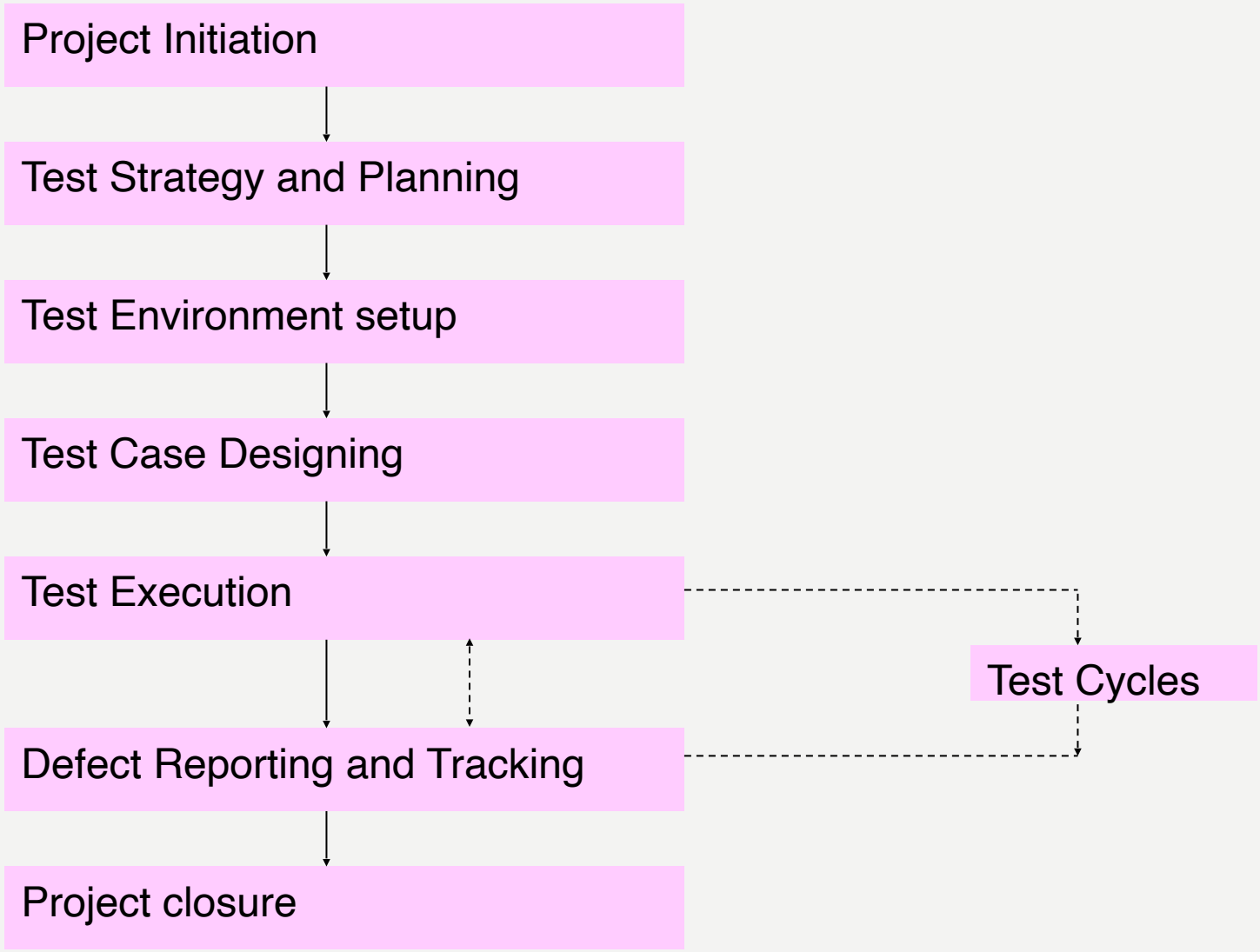
When to use Agile model

- Scenarios where end users needs are ever changing in a dynamic business and requires frequent customer feedback to develop solution

SDLC AND STLC



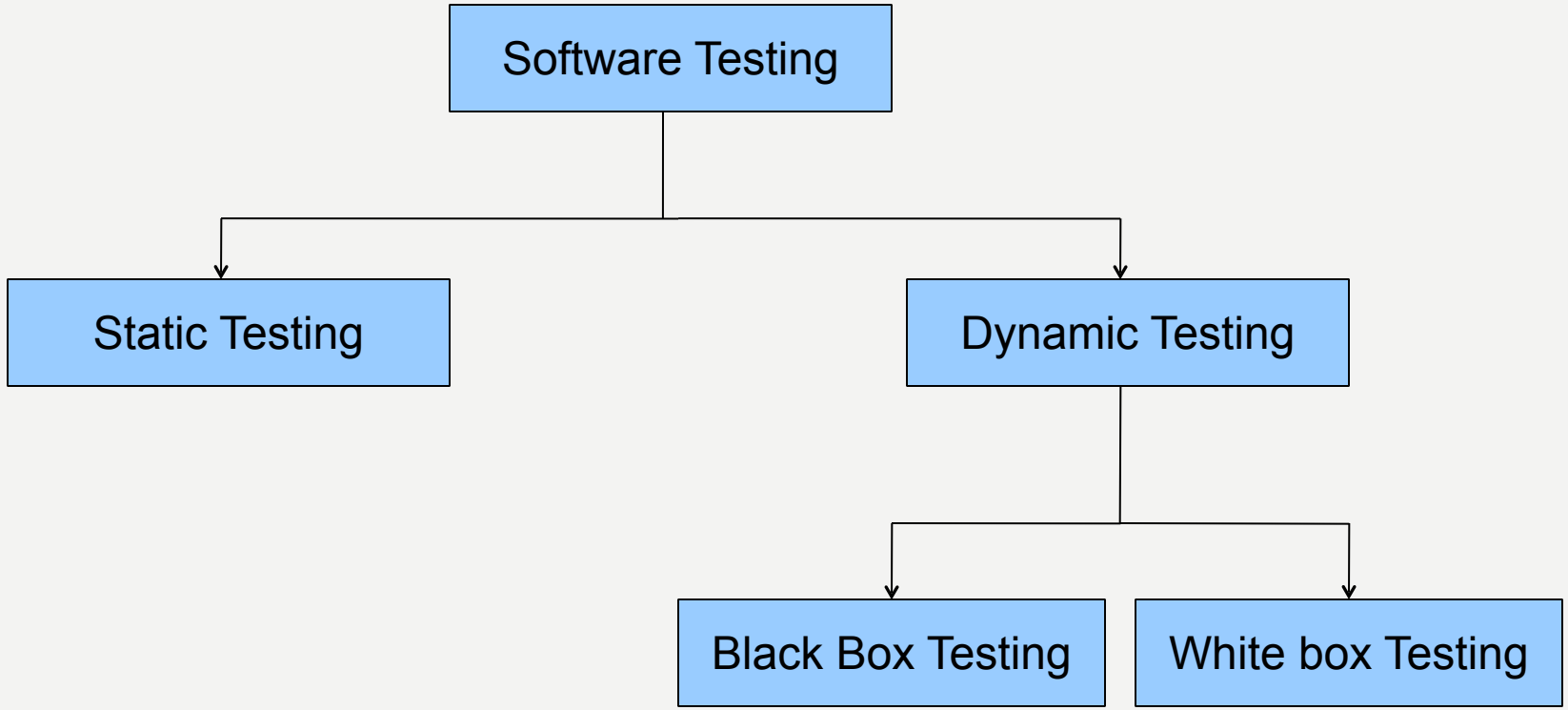
THE TESTING PROCESS



Discuss on Agile Process:: Grooming, Sprint Planning, Standup Call

CATEGORIES OF TESTING

BROAD CATEGORIES OF TESTING



DYNAMIC TESTING TECHNIQUES

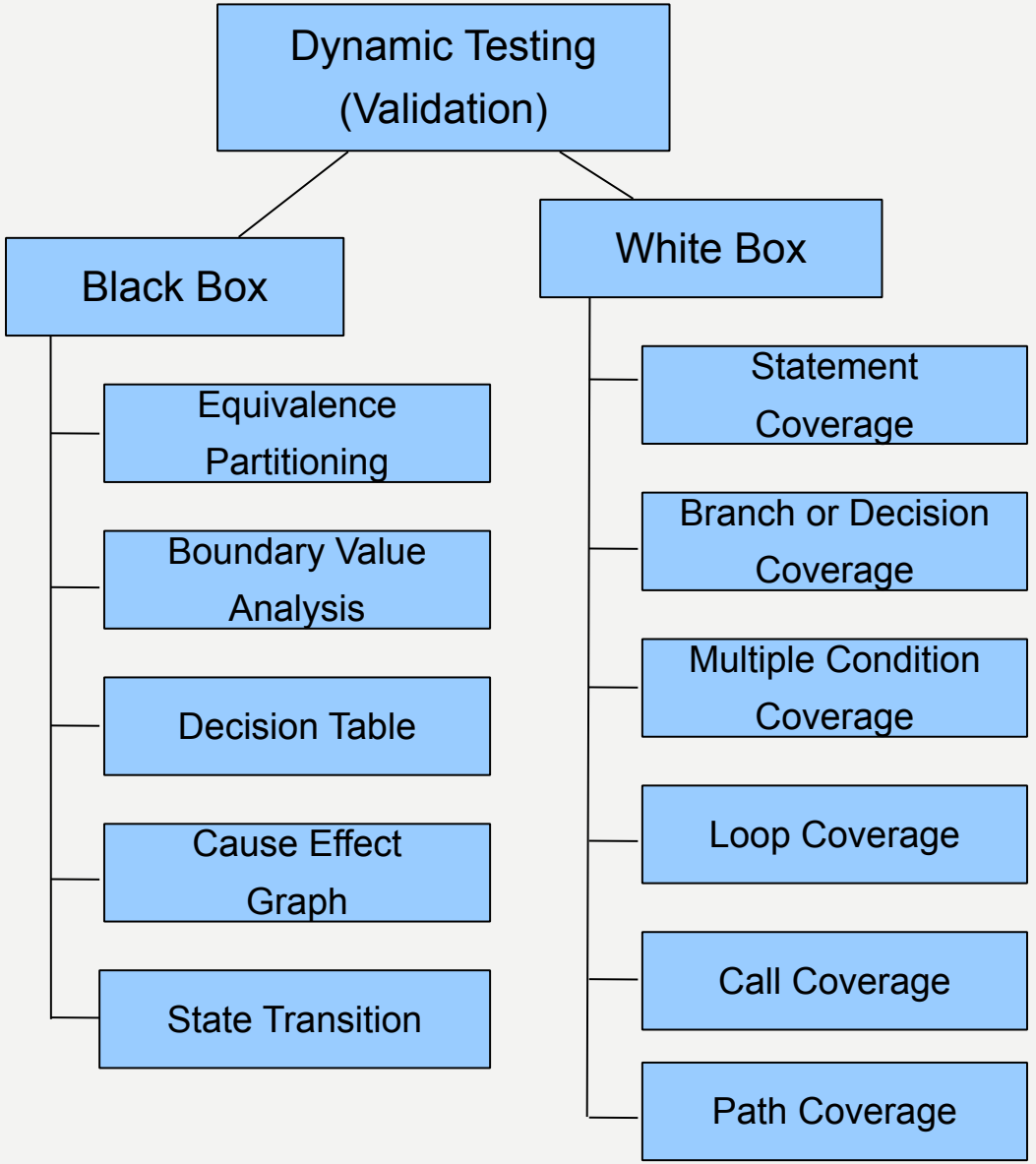
- White Box Testing

Tests at micro level of the programs that test each and every implemented functional task, ensuring that all code options are exercised. Requires knowledge of the internal code.

- Black Box Testing

Testing that focuses solely on the outputs generated in response to selected inputs and execution conditions. Requirements are the only test basis and knowledge of the internal code is not required.

DYNAMIC TESTING TECHNIQUES



BLACK BOX TECHNIQUE : EQUIVALENCE PARTITIONING

- Partitioning the input domain of a program into a finite number of classes set to identify a minimal set of well selected test cases to represent these classes.

- There are two types of input equivalence classes, valid and invalid.

- Equivalence class testing can significantly reduce the number of test cases that must be created and executed

Illustration for EP

EP may be best explained with an example of a function which has the pass parameter "month" of a date. The valid range for the month is 1 to 12, standing for January to December. This valid range is called a partition. In this example there are two further partitions of invalid ranges.

The first invalid partition would be <= 0 and the second invalid partition would be >= 13.

BLACK BOX TECHNIQUE : BOUNDARY VALUE ANALYSIS

- A selection technique in which test data are chosen to lie along "boundaries“ of the input domain [or output range] classes, data structures, procedure parameters, etc.

- Choices often include maximum, minimum, and trivial values

Focus on the boundaries of the input

If input condition specifies a range bounded by a certain values, say, a and b, then test cases should include

- The values for a and b
- The values just above and just below a and b

If an input condition specifies any number of values, test cases should be

- the minimum and maximum numbers,
- the values just above and just below the minimum and maximum values

Illustration for BVA

If the same example of a function which has the pass parameter "month" of a date

Valid Class is 1<= month <=12

Invalid Class 1 is month <1

Invalid Class 2 is month >12

When compared to EP, which says select any test case within a range and any on either side of it , in BVA the_emphasis is on the ‘edges’

1 and 12 for the ‘edges’ of the Valid class

7(a middle value) for the Valid Class

0 and 13 for the Invalid class

Decision Tables

Record the inputs and expected outputs to test with

The tabular form is quick to scan and comprehend

Automated testing tools often support this format for tests

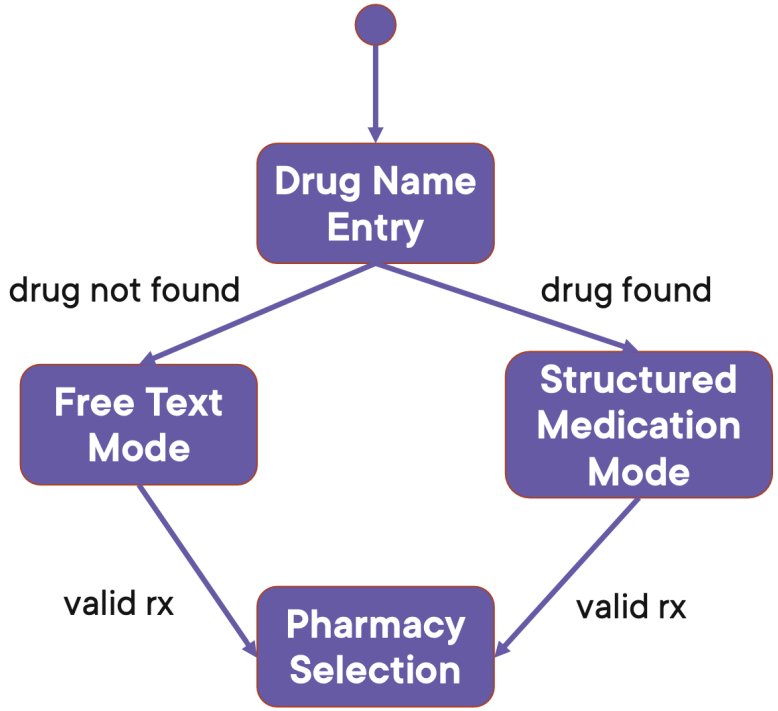
Medication	Duration	Prescribe button enabled?	Duration exceeded message?
Aspirin	32 days	True	False
Hydrocodone	30 days	True	False
Hydrocodone	31 days	False	True

State Transition Testing

Another type of input-output testing

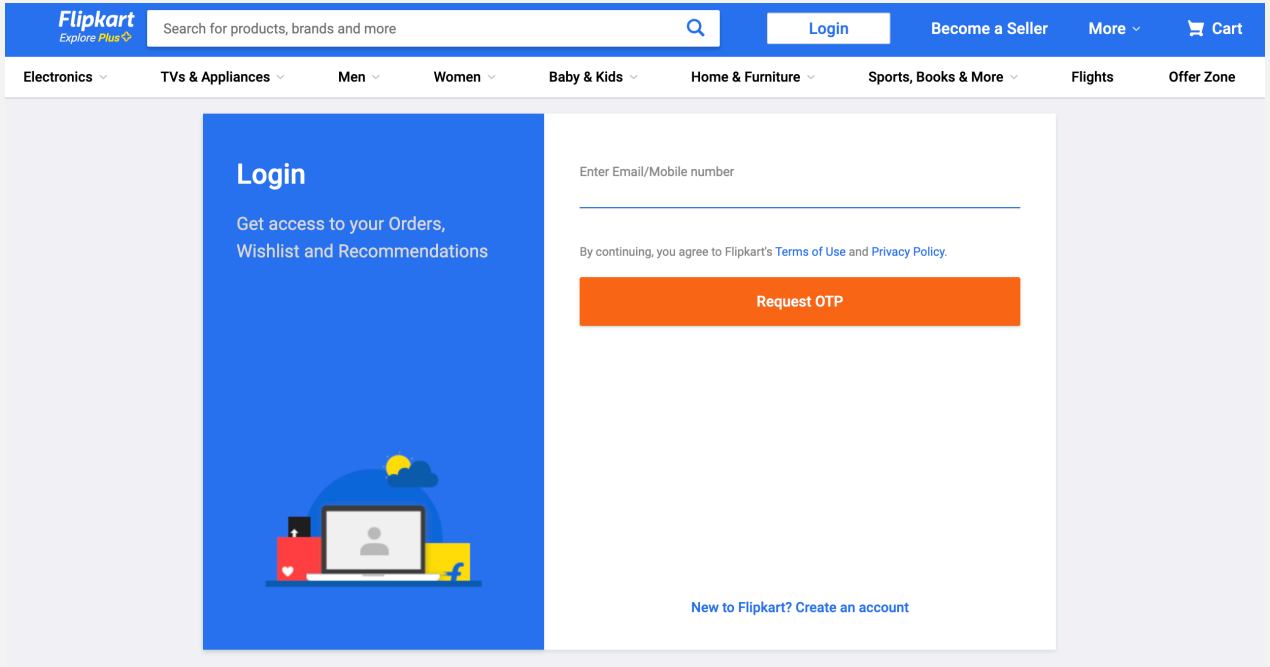
Consider what states the system or part of the system can be in

State transition diagrams can be a valuable aid to testing



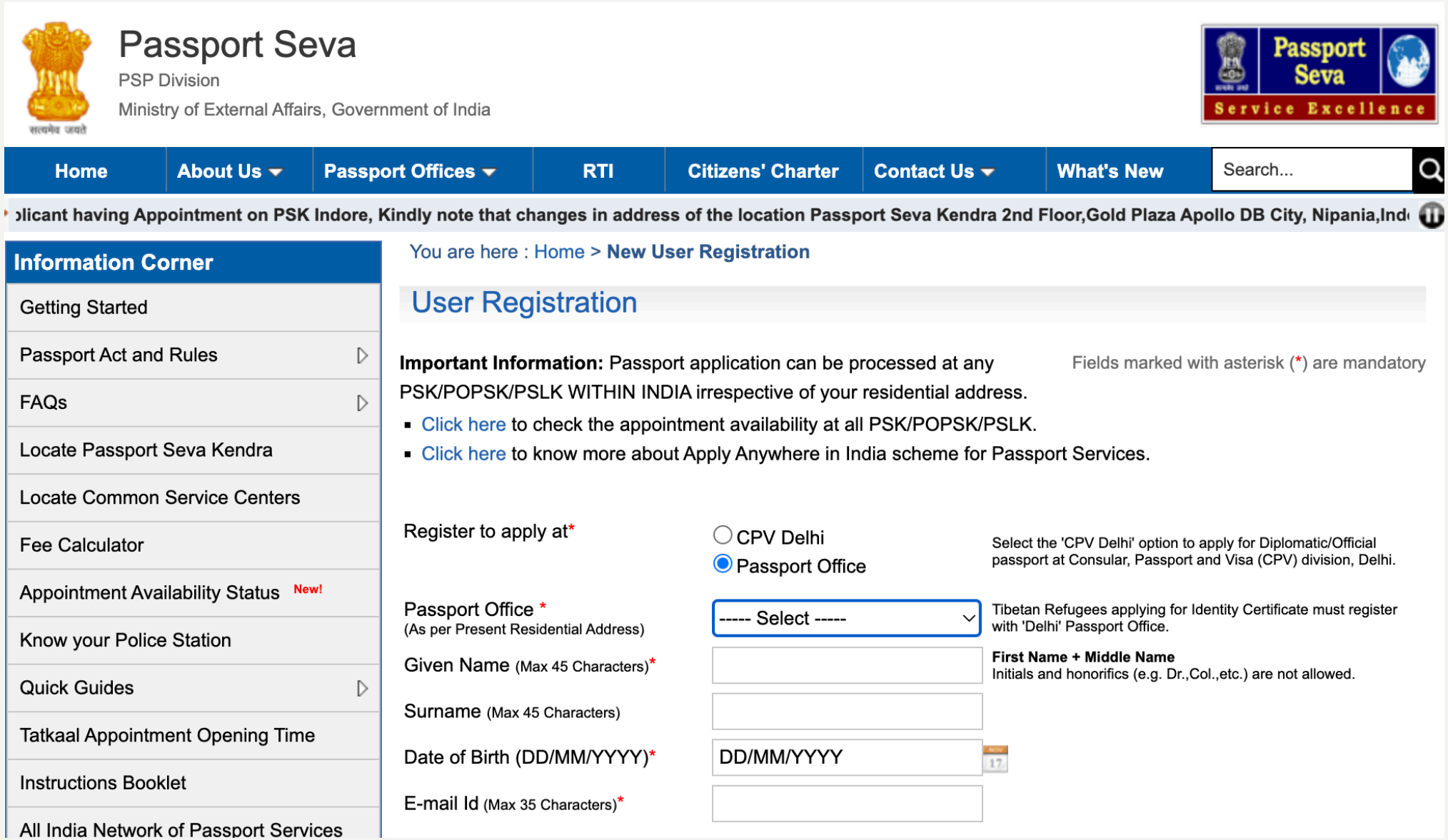
Equivalence Partition Test ?

BVA ?



Equivalence Partition Test ?

BVA ?



White Box Technique: Statement Coverage

- Testing to satisfy the criterion that each statement in a program to be executed at least once during program testing. Coverage is 100 percentage when a set of test cases causes every program statement to be executed at least once.

- The chief disadvantage of statement coverage is that it is insensitive to some control structures.

Example

```
1   int select ( int a[], int n, int x)
2   {
3       int i = 0;
4       while ( i < n && a[i] < x )
5       {
6           if (a[i] < 0)
7              a[i] = - a[i];
8              i++;
9       }
10      return 1;
11  }
```

One test case n=1, a[0]=-7, x=9 covers everything ,

Flow 1 - > 2 - > 3 - > 4 - > 5 - > 6 - > 7 - > 8 - > 9 - > 10 - > 11

White Box Technique: Branch or Decision Coverage

A test coverage criteria which requires that for each decision point or each possible branch be executed at least once

Example

```
1    int select ( int a[], int n, int x)
2    {
3        int i = 0;
4        while ( i < n && a[i] < x )
5        {
6            if (a[i] < 0)
7              a[i] = - a[i];
8              i++;
9        }
10       return 1;
11   }
```

Test Data

Branch coverage	i	n	x	a[i]	Branch Outcome
while (i < n && a[i] < x)	0	1	9	-7	True
	0	1	7	9	False

Flow A : 1 - > 2 - > 3 - > 4 - > 5 - > 6 - > 7 - > 8 - > 9 - > 10 - > 11

Flow B : 1 - > 2 - > 3 - > 4 - > 10 - > 11

White Box Technique: Multiple Condition Coverage

- A test coverage criteria which requires enough test cases such that all possible combinations of condition outcomes in each decision, and all points of entry, are invoked at least once.

- A large number of test cases may be required for full multiple condition coverage

Example

```
1    int select ( int a[], int n, int x)
2    {
3        int i = 0;
4        while ( i < n && a[i] < x )
5        {
6            if (a[i] < 0)
7                a[i] = - a[i];
8                i++;
9        }
10       return 1;
11   }
```

Test Data

Multiple Condition Coverage	i	n	a[i]	x	Outcome
while (i < n && a[i] < x)	0	4	-10	10	True
	0	4	10	10	False
	0	-4	-10	10	False
	0	-4	10	10	False
if (a[i] < 0)	-	-	-10	-	True
	-	-	10	-	False

White Box Technique: Loop Coverage

A test coverage criteria which checks whether loop body executed zero times, exactly once or more than once

Example

```
main ( )
{
  int i, n, a[10],x;
  printf (“Enter the values”);
  scanf (“%d %d %d %d”, &i, &n, &a[i], &x);
   while ( i < n && a[i] < x )
   {
      if (a[i] < 0)
      a[i] = - a[i];
      i++;
    }
    printf (“%d” , a[i] );
}
```

Test Data

Loop Coverage	i	n	x	a[i]	loop called
while (i < n && a[i] < x)	0	4	5	10	0 times
	3	4	5	-10	1 time
	1	4	5	-10	3 times

White Box Technique: Call Coverage

- A test coverage criteria which checks whether function called zero times, exactly once or more than once

Example

```
main ( )
{
int a, b, i ;
printf (“Enter the value of a, b, i”);
scanf (“ %d %d %d “, &a ,&b, &i);
if ( i  < 10 )
{
sample ( a, b);
i = i + 1;
}
}
sample ( int x , int y )
{
If ( x > 10 )
    x = x + y ; break ;
if  ( y > 10 )
    y = y + x ; break ;
}
```

Test Data

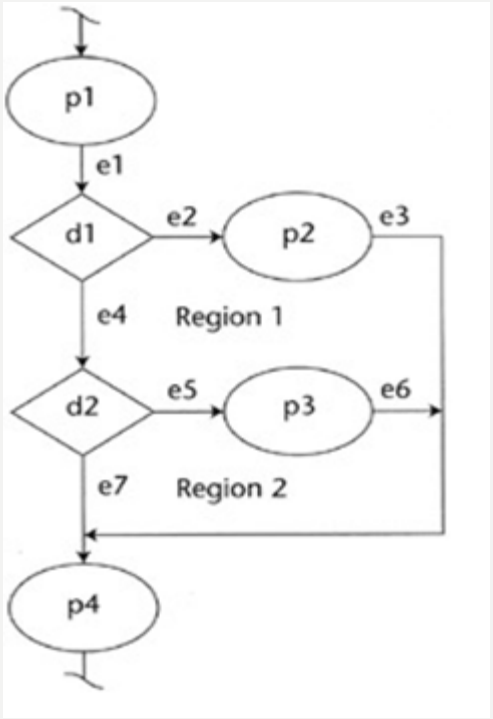
Call Coverage	a	b	i	function called
sample (int x, int y)	2	4	10	0 times
	2	4	9	1 time
	2	4	7	3 times

White Box Technique: Path Coverage

- Testing to satisfy coverage criteria that each logical path through the program be tested. Often paths through the program are grouped into a finite set of classes. One path from each class is then tested

- General coverage requires executing all paths, number of paths may be infinite if there are loops

White Box Technique: Path Coverage continued…



Example

Linear Independent Paths

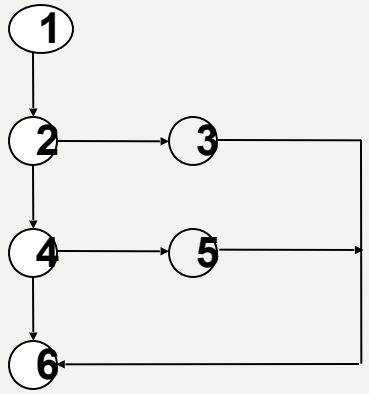
Path 1 -> p1 – d1 – d2 – p4

Path 2 -> p1 – d1 – p2 – p4

Path 3 -> p1 – d1 – d2 – p3 – p4

Sample Program

```
1     sample ( int x , int y )
      {
2           If ( x > 10 )
3                 x = x + y ; break ;
4           if  ( y > 10 )
5                 y = y + x ; break ;
      }
6     printf (“%d %d”, x , y);
```



What is Exploratory Testing?

An approach to testing that allows testers to evolve and vary what they test based on current objectives

A response to scripted testing

Allows the tester to intuit and explore tangents during the test – to use their brain

Encompasses multiple activities and techniques

Learning

Test Design

Execution and Interpretation

LEVELS OF TESTING

There are broadly four levels of Testing done in any Testing Project

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

The first 2 levels concentrate mainly on the “Functional aspects”.
During System Testing level, the functionality of the application is tested first, and then the non-functional aspects mainly the “Performance” of the application can be tested.

Acceptance Testing is the last level of testing in the SDLC

Apart from the ‘Performance’ there are other non-functional aspects that can be tested in the application as per the requirements

Common Objectives

- Reducing risk
- Verifying functional and non-functional behaviors
- Build confidence in the system
- Find defects
- Prevent defects from escaping to higher levels

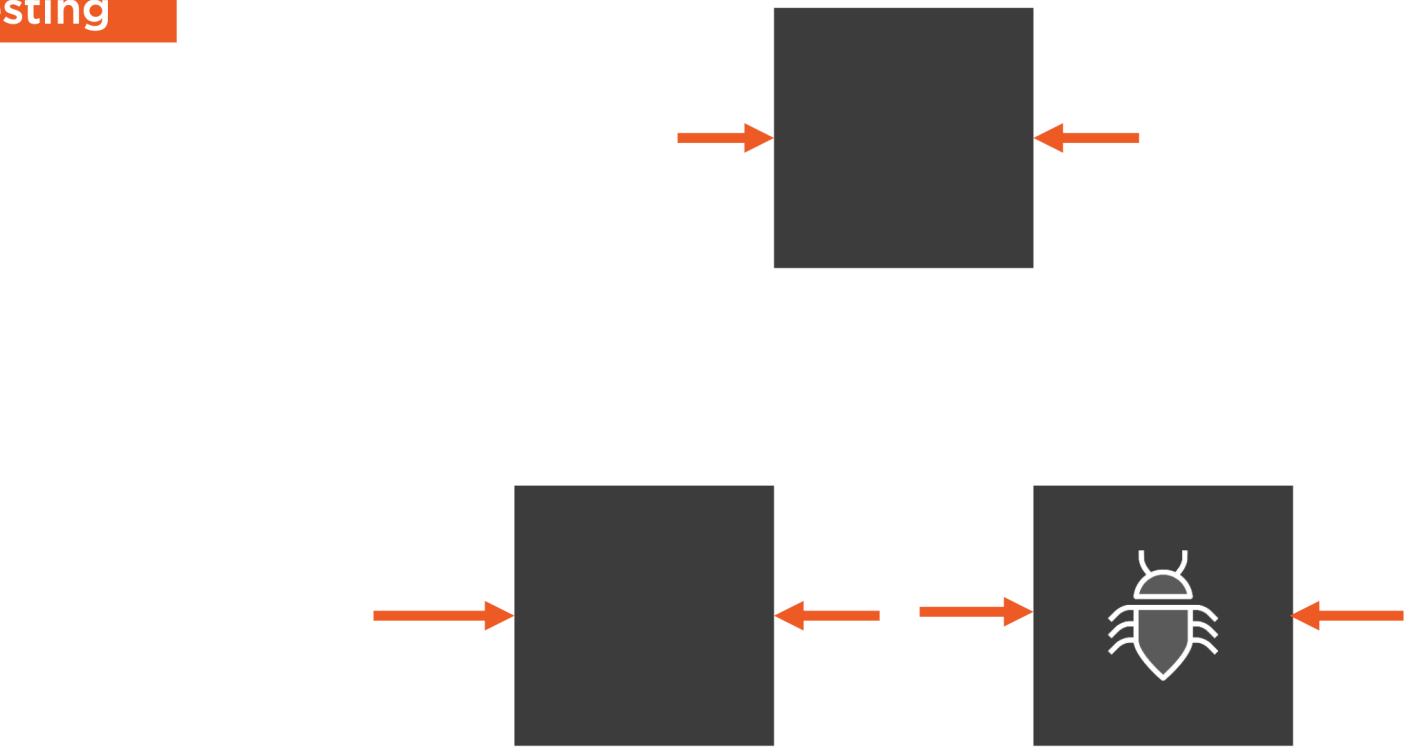
UNIT TESTING

- To test a unit of code (program or set of programs) using Unit Test Specifications, after coding is completed. Involves the basic testing of a piece of code, the size of which is often undefined in practice, although it is usually a function or a subroutine

• Example

– Testing of a cobol program in the reservation system that calculates the price for the ticket requested based on the inputs supplied to the program from the calling cobol program.

Component Testing

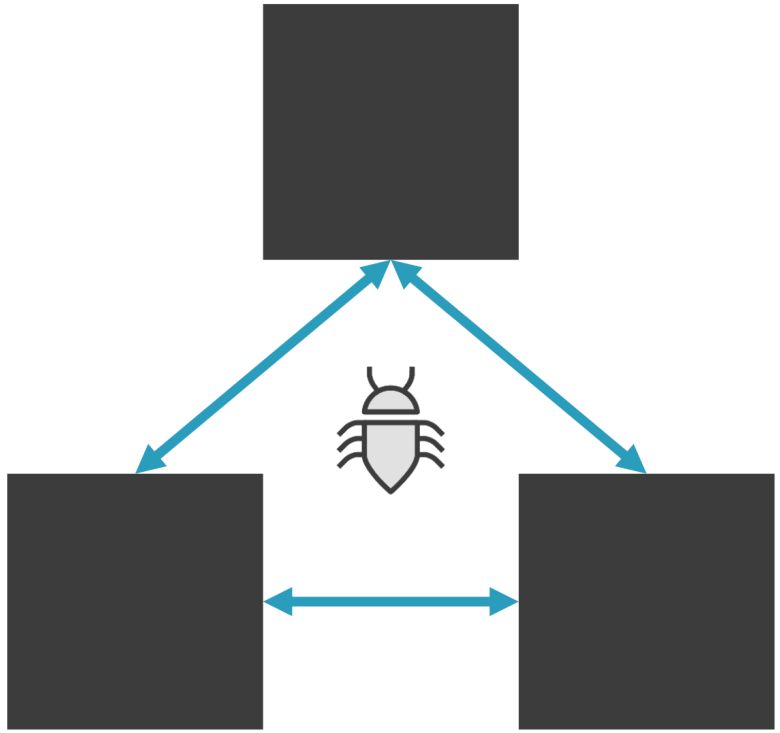


INTEGRATION TESTING

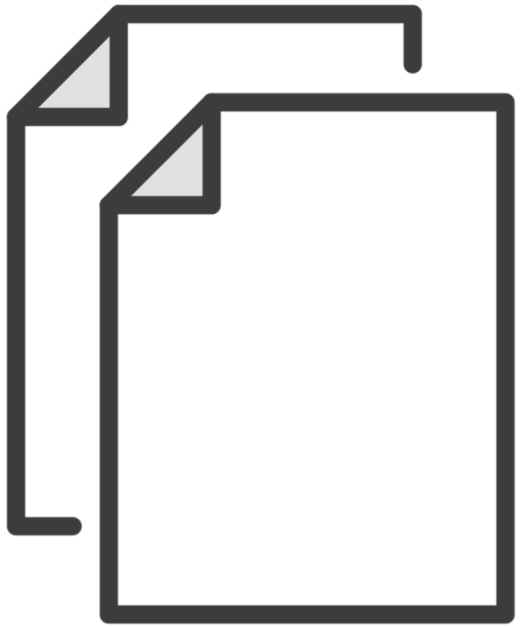
- The process of testing interfaces and data flows between the programs within a sub system,and between the sub-systems within a system.

- Integration testing means that the tester must look for bugs in the relationship and the interfaces between pairs of components and groups of components under test.

Integration Testing



Integration Testing



Test objects:

- Subsystems
- Databases
- Infrastructure components
- Microservices

Test basis:

- Technical spec
- Design documents
- Sequence diagrams
- Public interface definitions

SYSTEM TESTING

- It is a test, executed by the developer or independent test team in a laboratory environment that should demonstrate that the developed system or subsystems meet the requirements set in the functional and quality specifications.
- The process of proving that the system meets its stated design specifications (design documents) w.r.t criteria such as recoverability, maintainability and security.

• Example

– Comprehensive black box testing of railway reservation system with transactions initiated and validations performed on databases and reports generated after the completion of the transactions.

System Test Level

Full integration testing of all modules

System Testing

Your starting point is typically not within the system, but rather outside

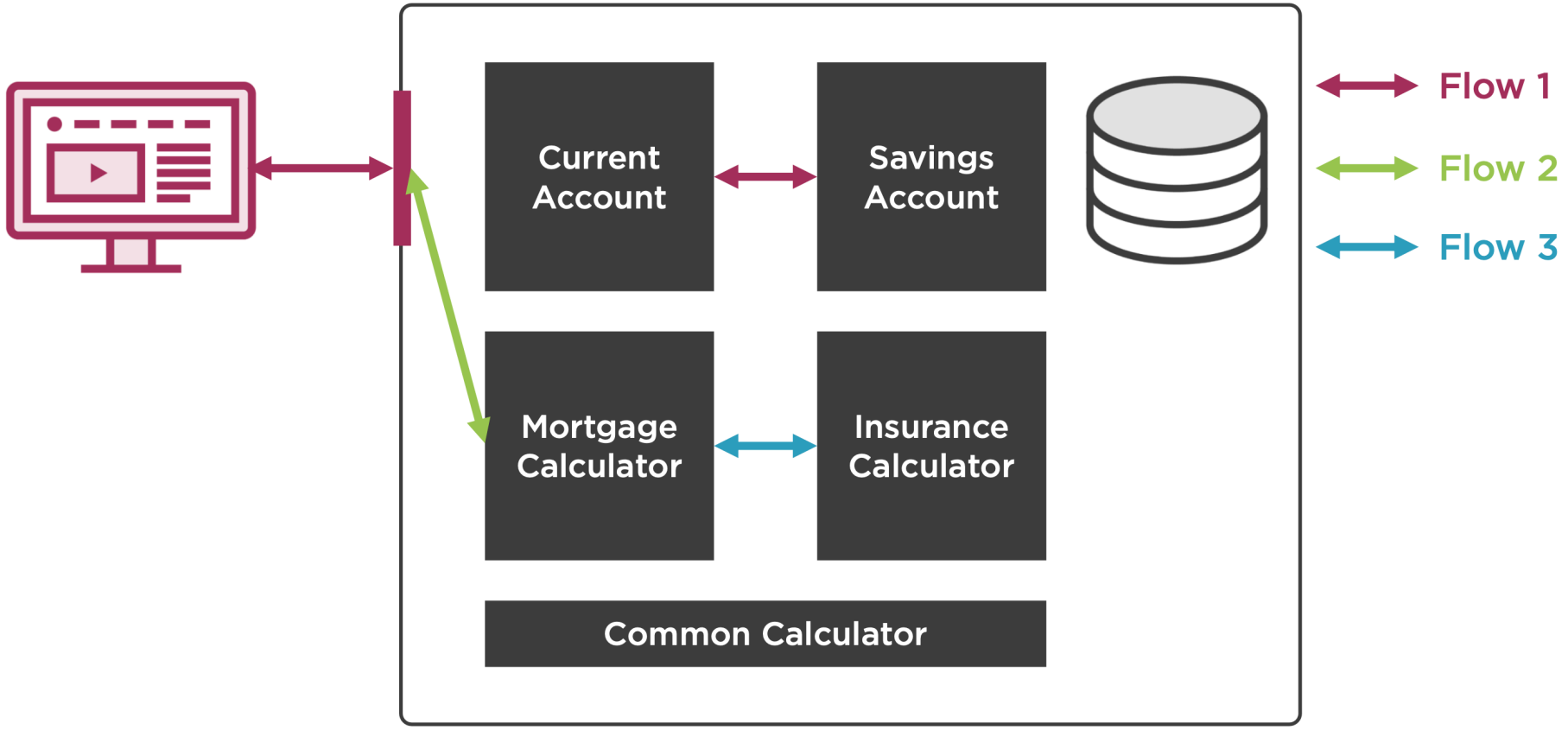
Considers system paths and flows

- End-to-end (e2e)
- Front-to-back (f2b)

External behavior vs. inner structure

Car Integration Testing

Car System Testing


Current Account
Savings Account
Mortgage Calculator
Insurance Calculator
Common Calculator
Flow 1
Flow 2
Flow 3

ACCEPTANCE TESTING

It is a test, executed by the user(s) and system manager(s) in an environment simulating the operational environment to the greatest possible extent, that should demonstrate that the developed system meets the functional and quality requirements.

Acceptance Testing

UAT: User Acceptance Testing

OAT: Operational Acceptance Testing

- Backup and restore
- Installing, uninstalling, upgrading
- Disaster recovery
- Data load and migration
- Performance and load testing

Acceptance Testing

Categories: Alpha and Beta testing

Beta testing: for Commercial Off-the-shelf (COTS) software

Both carried out by independent testers or potential customers

Acceptance Testing

Alpha testing happens at the developer's site

Beta testing happens at the site of the customer

The point of Beta testing is to use the infrastructure, both hardware and software, of the end users.

It works on my machine!

FUNCTIONAL TESTING

NON-FUNCTIONAL & OTHER TYPES OF TESTING

- Performance testing
- Volume testing
- Load testing
- Limit testing
- Stress testing
- Disaster Testing
- Recovery testing
- Security testing
- Reliability testing
- Installation Testing
- Usability Testing
- Accessibility Testing
- Regression testing

TESTING ARTIFACTS AND TEST DESIGN

OBJECTIVES

- Importance of Documentation
- Testing Artifacts
- Introduction to Test Design
- Test Environment and Test Data

TESTING DOCUMENTATION

- Integral part of testing
- Important for conducting the test and reuse of the test program during maintenance
- Test documentation should start in the requirements phase and continue throughout the life cycle of the project
- Structured documentation is easier to update and reuse
- Good Documentation makes Testing “tester” independent
- Every stage of Testing Lifecycle should be documented correctly.

TESTING ARTIFACTS

The important artifacts in Software Testing are:

- Test Strategy
- Test Plan
- Test Design Specification
- Test Case Specification
- Test Execution Log
- Test Defect Report
- Test Summary Report

Testing Artifacts Contd...

- Activities carried out at the Test Strategy level are as given below.
    1. Define test objectives, scope of testing, testing phases and activities
    2. Define testing Approach, Identifying high risk items.
    3. Test Environment and Test Data set up
    4. Define test dependencies, Test assumptions, test completion and user acceptance criteria
    5. Establish configuration standards and defect management

- The schedule of the project will be documented in the test plan. The test strategy details with application specific data will also be present in Test plan.

- Test Design specification gives a high level test conditions and Test Case specification details how the identified test conditions are tested.

- Test Log and Test Defect Report are prepared during Test Execution Phase.

- Test summary report contains summary of product tested ,summary of results, summary of activities ,Sign off documents etc.

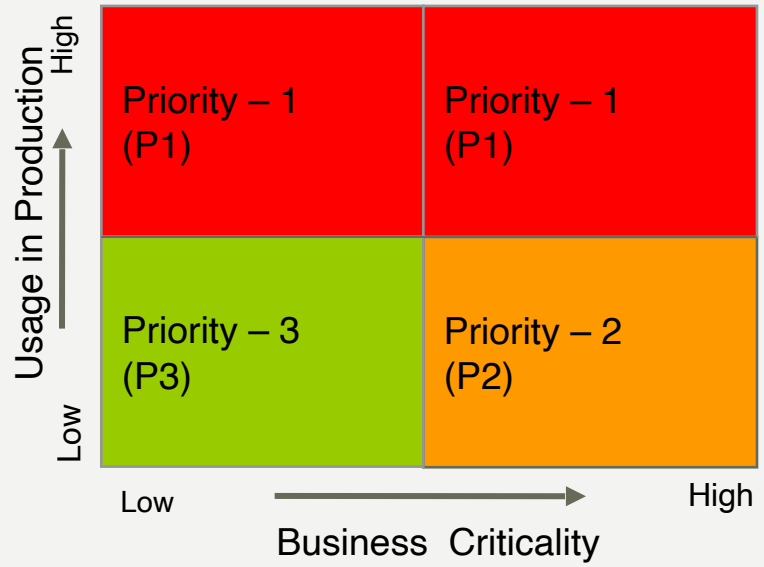
TEST DESIGN

- The test objectives established in the test plan should be decomposed into individual test cases. Begin the process with identifying high-level test conditions. These are decomposed into lower and lower objectives until functional and structural test objectives are defined individually.

- The main steps involved in Test Design phase are listed below.

1. Identify test resources.
2. Identify conditions to be tested – Test Scenario preparation.
3. Prioritize the test conditions.
4. Determine the Expected Results for each scenario.
5. Create test cases for the approved test scenarios.

TEST PRIORITIZATION



Usage in Production	
High	Functionality is used extensively in production
Medium	Functionality is used moderately in production
Low	Functionality is rarely used in production

Business Criticality Defined based on : Financial & Consumer Impact, Potential Regulatory impact,	
High	If the vulnerability could affect critical business process
Medium	If the vulnerability could affect supporting activities of Business process
Low	If the vulnerability could affect individuals and activities that are not necessarily related to business

WHAT IS A TEST CASE?

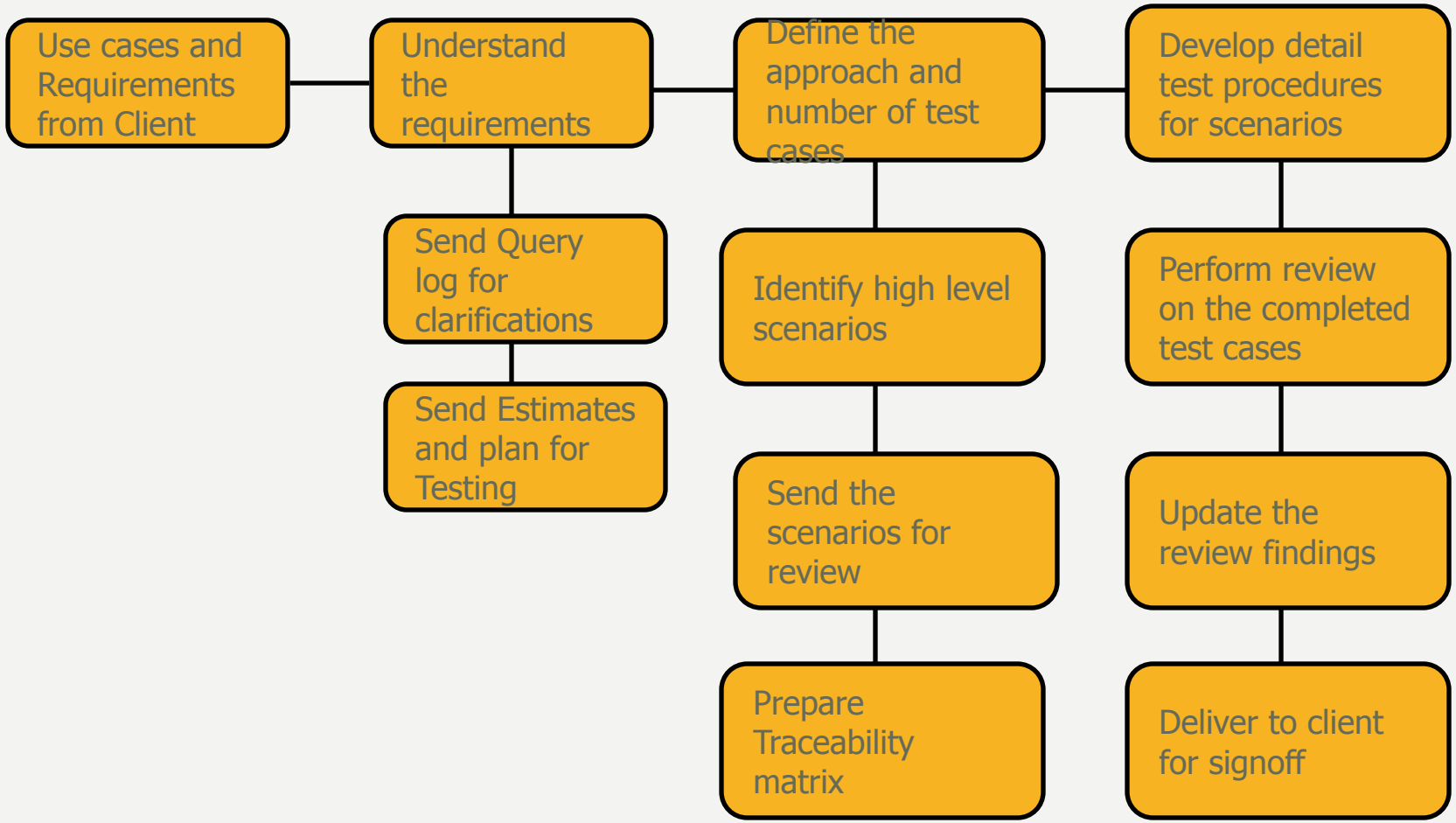
- It is the smallest unit of Testing
- A test case is a detailed procedure that fully tests a feature or an aspect of a feature. Whereas the test plan describes what to test, a test case describes how to perform a particular test.
- A test case has components that describes an input, action or event and an expected response, to determine if a feature of an application is working correctly.
- Test cases must be written by a team member who thoroughly understands the function being tested.

Language usage in Test Case writing

- Use Simple and Easy-to-Understand language
- Use Active voice while writing test cases For eg.
    - Click on OK button
    - Navigate to the account Summary page.
- Use words like “Verify” for starting any sentence in Test Case description For eg.
    - Verify whether the account information gets displayed on clicking the “Account Summary” menu
- Use words like “Must” and use Future Tense for Expected Results. For eg.
    - The application must display the account information screen

- The test case should have Test Case ID, Test Case Description, Test Steps, Test Step description, Expected Results, Priority and Requirement ID. Pre-requisites and Test Data need to be updated

Test Case Development:



TEST ENVIRONMENT AND TEST DATA

Test Environment

- A dedicated **Test environment** allows failures to be consistently and accurately recreated and ensures that the software under test will behave the same way in Production.
- To ensure successful software deployment, it is crucial to have a stable environment dedicated to testing.

Test Data

- Test Data means the data that is required for testing an application .
  - ➢Entry data
    - Data that are entered into the system as input to execute a test case.
  - ➢Background data
    - Data that are setup in the Backend to execute a test case, like say in a test database

Test Execution and Defect management

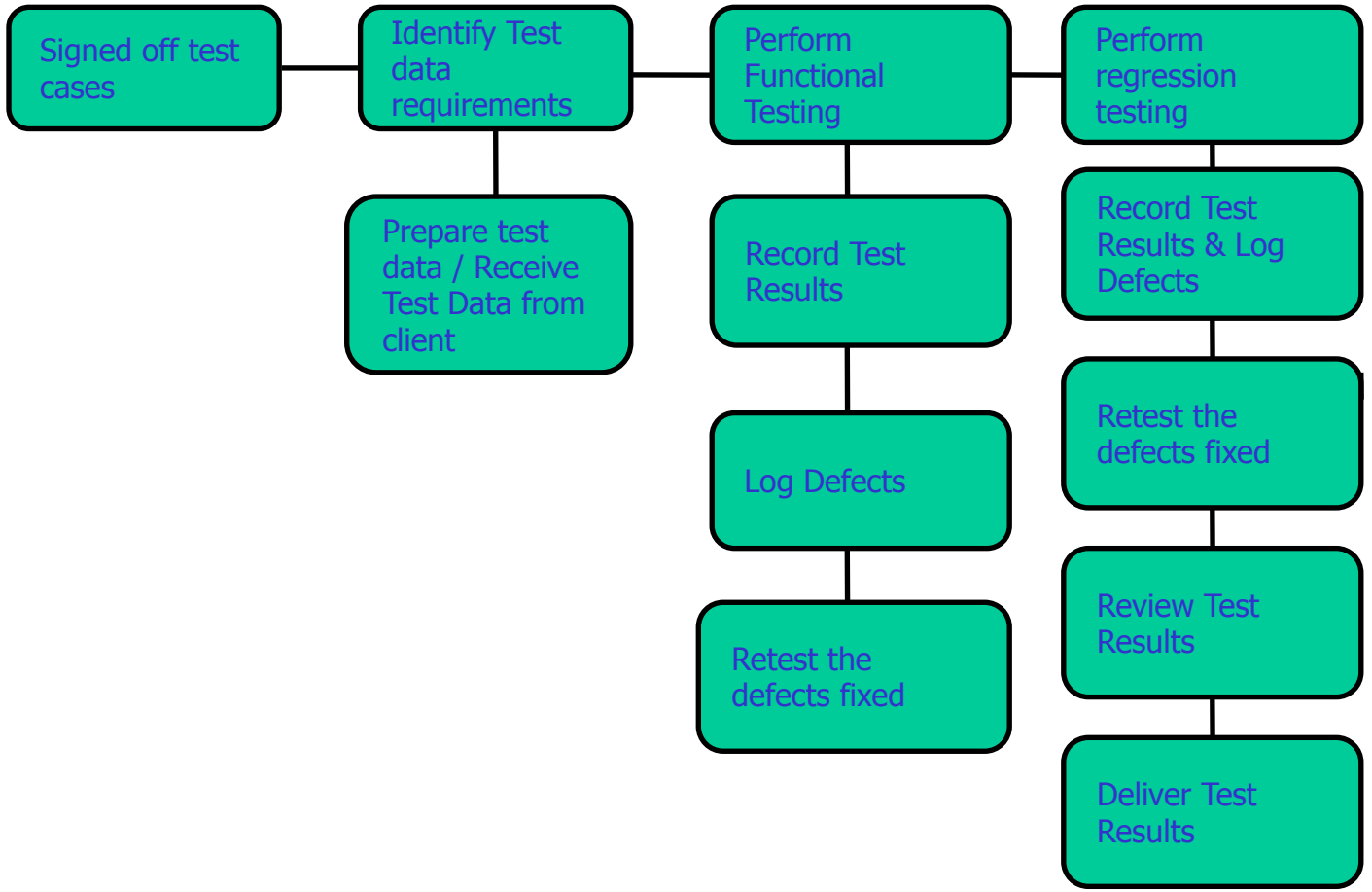
Objectives

- Introduction to Test Execution
- Checklist of Test Execution
- Defect management
- Defect Classification

THE TEST EXECUTION PHASE

- Execute test scripts - manual or automated
- Capture Test results
- Defect reporting and tracking
- Compile Test reports
- Causal Analysis
- Re-test

Test Execution Phase



CHECKLIST OF TEST EXECUTION

- Stable Test Environment
- Up to date Software upgrades if required
- Complete Test Cases & Test Data
- Problems reported from previous cycles must be resolved
- Test Execution plan must be ready

Test Case Writing Demo

- **Test Case ID**:
- **Test Summary**:
- **Test Steps:**
- **Prerequisites**:
- **Browser**:
- **Test Data:**
- **Expected/Intended Results:**

Test Case Writing Exercise

Write Test Cases of Text Message of WhatsApp application

Write Test Cases of Video Call of WhatsApp application

Write Test Cases of attachment feature of Gmail

Write Test Cases of COD feature of Amazon

Write Test Cases of UPI feature of Paytm

DEFECT MANAGEMENT AND REPORTING

CAUSES OF SOFTWARE DEFECTS

An error or mistake that human being make, produces defects (fault ,bug) in a software or in a code or in a system or in a document.

When a defect in the code is executed, the system will fail to do what it is intended to do Defects may result in failures, but not all defects lead to failures.

Failures can also happen because of changing hardware conditions.

Defect may occur because of the following reasons like

- Fallible nature of human beings
- Time pressure
- Complex code
- Complexity of infrastructure
- Changed technologies
- Many system interactions etc.

DEFECT

• A software bug or defect is an error, flaw, mistake, failure, or fault in a computer program that prevents it from behaving as intended.

• Defects are the gaps between the expected behavior and the actual behavior of the application or system.

• Defects arise from the verification and validation of the programming products like SRS, Test Specifications & Code.

• Defect can arise in any of the Software Test Life Cycle Phase. Defects arising from each phase should be logged and tracked to closure.

Defect Classification- Severity and Priority

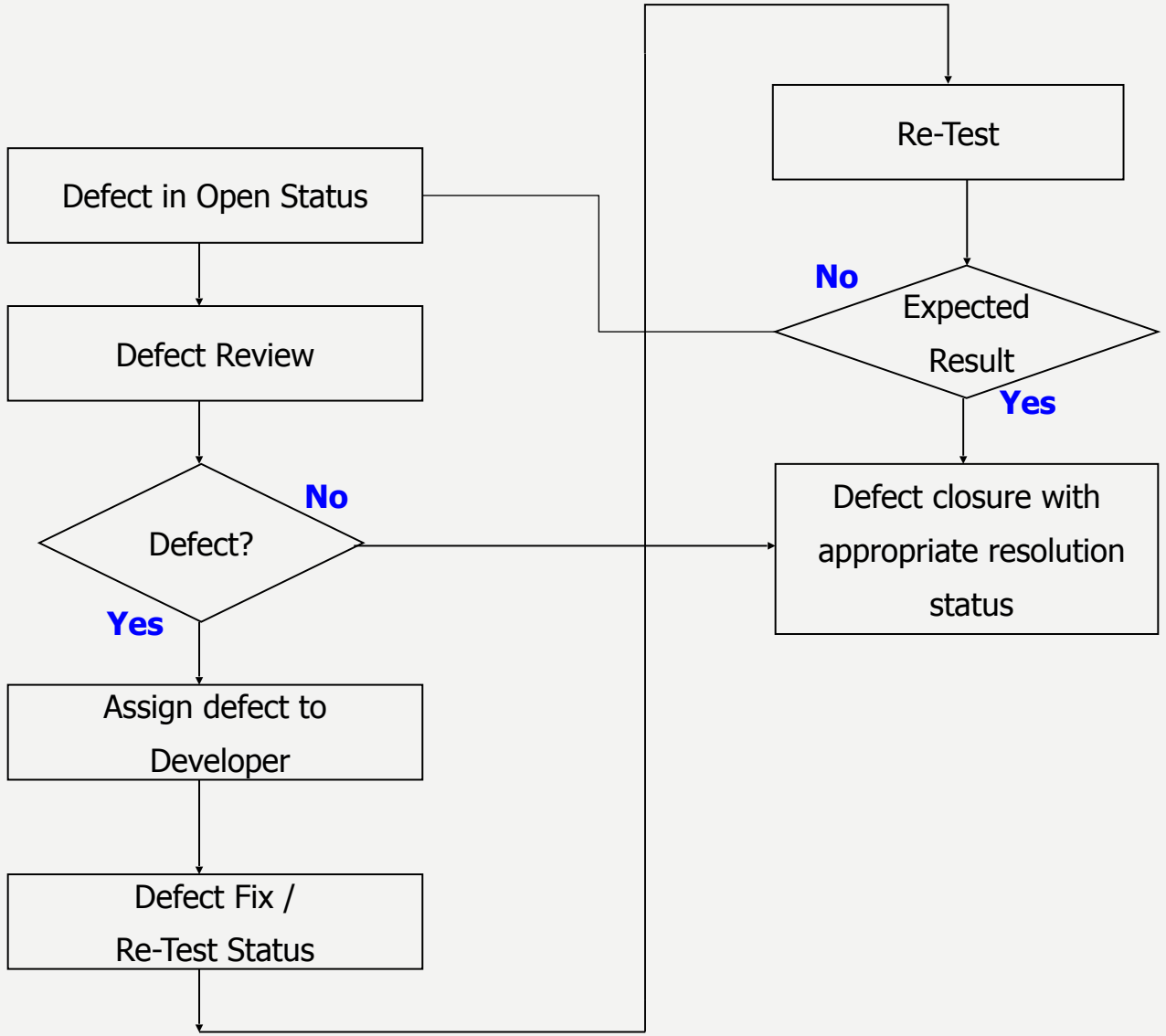
All the defects which are raised would be assigned a severity level and a Priority.

Severity	Description
Severity 1	Defects that impact areas critical to the business and which stop the testing of substantial areas of the system under testing.
Severity 2	Defects that are serious problem whose impact is confined to a functional area and for which there is a work round
Severity 3	Minor problem in a single screen or related to a single function for which there is a workaround
Severity 4	Nice to have/suggestion

Priority	Description
High	The defect needs to be fixed as soon as possible. Further testing might not be possible or meaningful unless this is fixed.
Medium	The defect needs to be fixed at the earliest opportunity available. The defect might be an obstacle for testing a section of the system. However, the testing of the rest of the system is not severely affected by this defect.
Low	The defect might be affecting only a minor section of the application and/or overall testing is not severely affected by this.

* The nomenclature of severity level and Priority may be different in different Defect/ Test management tools

DEFECT LIFE CYCLE FLOW



DEFECT LIFE CYCLE DEFECT STATUS

Defect Status	Responsibility	Inference
Open	Testing team	The defect is created for the first time and submitted to the development team
Fixed	Development team	After acknowledging the defect, the development team fixes it and sends it back to the testing team
Closed	Testing team	Upon verification, the testing team finds the defect to be fixed and marks it as Closed
Reopened	Testing team	Upon verification, the testing team finds the defects to be still exists and so reopens it to the development team
Withdrawn	Testing team	When the testing team accepts the defect to be an invalid defect
Invalid	Development team	When the development team does not accept it to be a valid defect
Deferred	Development team	When the development team decides to defer or forward the defect to be fixed in future version
Duplicate	Development team	When the development team feels that the same error condition was reported through a different test condition.

DEFECT MANAGEMENT

It is a common practice for software to be released with known bugs that are considered non-critical.

Most big software projects maintain a list of "known bugs". This list informs users about bugs that are not fixed in the current release, or not fixed at all, and often a workaround is offered additionally.

There are various reasons for such a list:

- The developers often don't have time to fix all non-severe bugs.
- The bug could be fixed in a new version or patch that is not yet released.
- The changes to the code required to fix the bug would be large and would bring with them the chance of introducing other bugs into the system.

DEFECT REPORTING CHARACTERISTICS

Defects/Incidents are logged in a tool or a defect log sheet maintained at project level

At a minimum a defect Log could include,

- Defect ID
- Descriptive defect name and type
- Source of defect - test case or other source
- Defect severity
- Defect priority
- Defect status (e.g., open, fixed, closed, user error, design, and so on) – more robust tools provide a status history for the defect
- Date and time tracking for either the most recent status change, or for each change in the status history
- Detailed description, including the steps necessary to reproduce the defect
- Component or program where defect was found
- Screen prints, logs, etc., that will aid the developer in the resolution process
- Stage of origination
- Person assigned to research and correct the defect

- Effective defect reports will:
  - ✓ Reduce the number of defects returned from development
  - ✓ Improve the speed of getting defect fixes
  - ✓ Improve the credibility of test
  - ✓ Enhance teamwork between test and development

OBJECTIVE: The objective is not to write the perfect defect report, but to write an effective defect report that conveys the proper message, gets the job done, and simplifies the process for everyone.

EFFECTIVE DEFECT REPORTING NEEDS

BEST PRACTICE DEFECT REPORTING

- Key points to ensure effectiveness of reporting:
  - **Condense** - The comments must be clear and brief
  - **Accurate** - Is it a defect or could it be user error, misunderstanding, etc.?
  - **Neutralize** - Just the facts. No humour No emotion.
  - **Precise** - Explicitly, what is the problem?
  - **Isolate** - What has been done to isolate the problem?
  - **Generalize** - What has been done to understand how general the problem is?
  - **Re-create** - What are the essentials in triggering/re-creating this problem? (environment, steps, conditions)
  - **Impact** - What is the impact to the customer? What is the impact to test?
  - **Debug** - What does development need to make it easier to debug? (traces, dumps, logs, immediate access, etc.)
  - **Evidence** - What documentation will prove the existence of the error?

- It is not just good technical writing skills that leads to effective defect reports.
- It is more important to make sure that you have asked and answered the right questions.
- It is key to make sure that you have covered the essential items that will be of most benefit to the intended audience of the defect report.

Defect Writing Exercise

Write a defect related to the Text Message of WhatsApp application

Write a defect related to the Video Call of WhatsApp application

Write a defect related to the attachment feature of Gmail

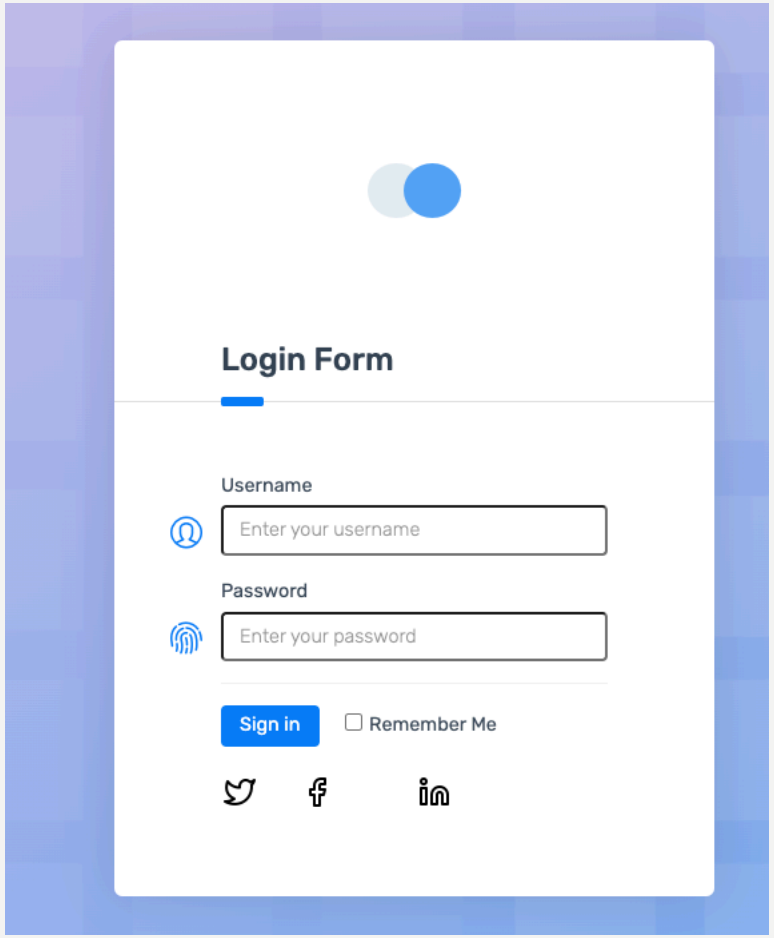
Write a defect related to the COD feature of Amazon

Write a defect related to the UPI feature of Paytm

https://www.saucedemo.com/

Accepted usernames are:	Password for all users:
standard_user	secret_sauce
locked_out_user	
problem_user	
performance_glitch_user	
error_user	
visual_user	

https://demo.applitools.com/#



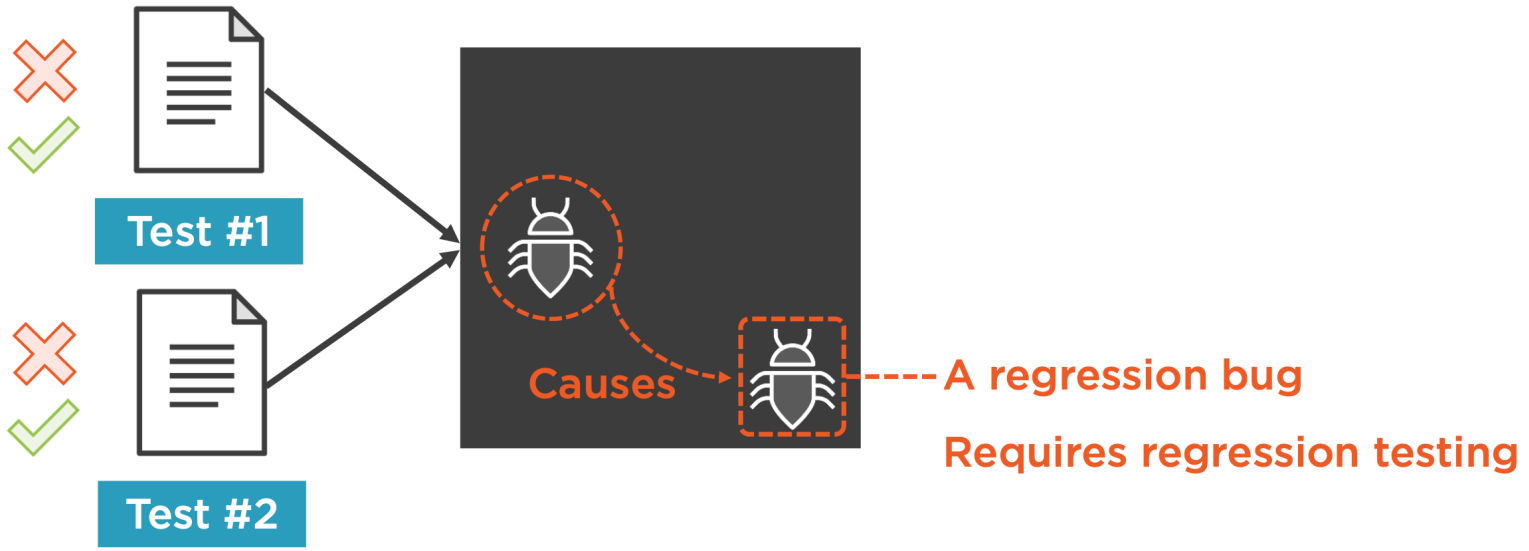
Change-related Testing

Confirmation Testing

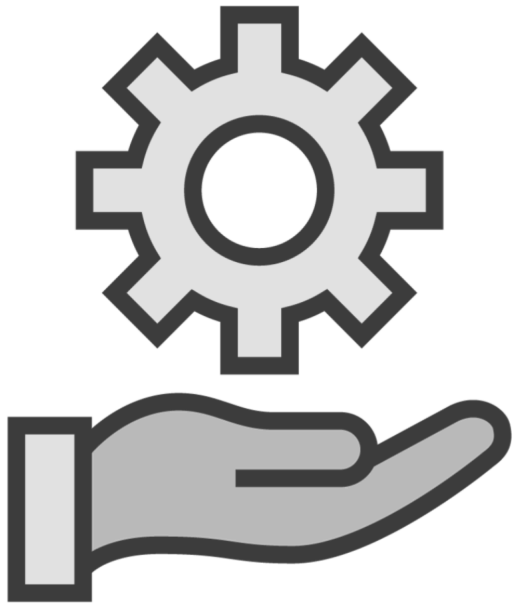
Was the bug really fixed?

Regression Testing

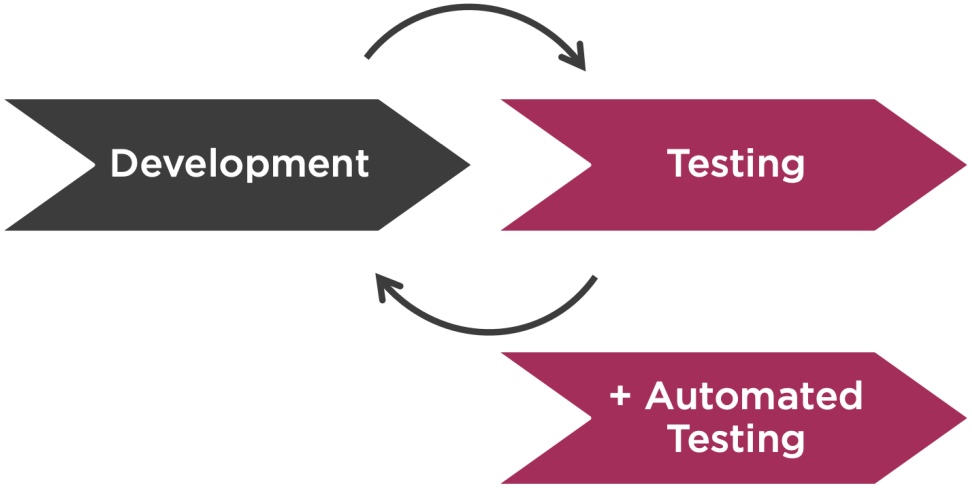
Did the fix (or a change) break anything else?


Test #1
Test #2
Causes
A regression bug
Requires regression testing

Change Types



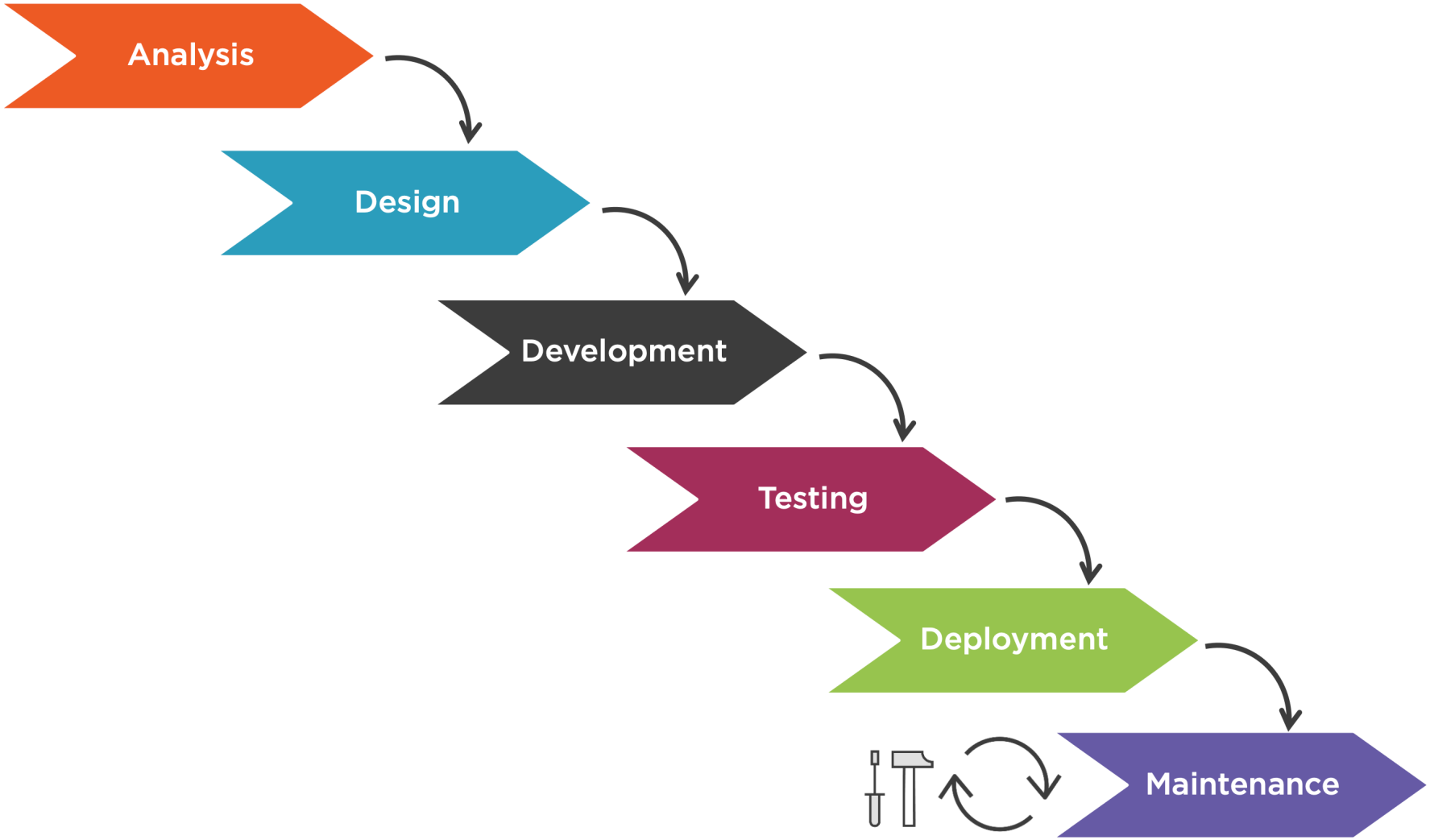
- **Bug fix**
- **New features**
- **Change to existing features**
- **Configuration and environmental changes**



- Runs x100+ faster
- Prevents testing from becoming a bottleneck

Exploring Maintenance Testing

Systems need to be maintained because changes are inevitable.


Analysis
Design
Development
Testing
Deployment
Maintenance

Planned

- Software enhancements
- Operational and environmental upgrades
  - Example: SQL Server migration to a newer version
- Retirement
  - SW not fit for purpose anymore
  - Rewriting the SW using newer tech

Unplanned

- Bugs or failures in production
- A “hotfix” required

Impact Analysis

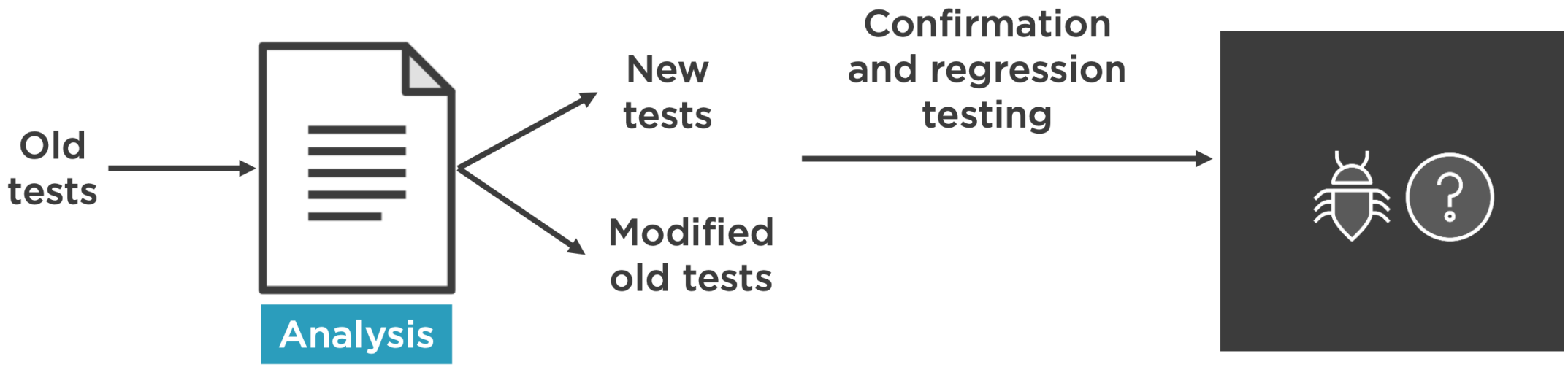
Evaluates the changes made to identify consequences and potential side-effects

What has changed?

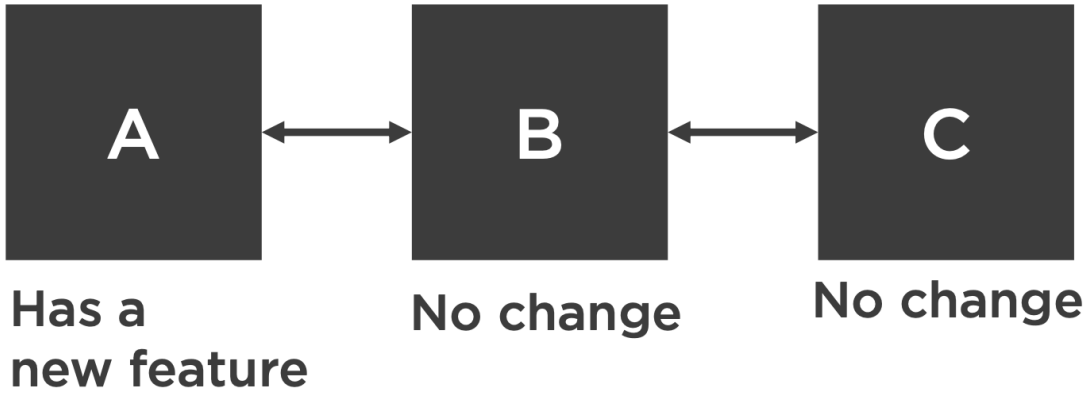
Where in the system?

Which parts are definitely affected?

Which parts are likely to be affected?


Old tests
Analysis
New tests
Modified old tests
Confirmation and regression testing

I need to test this, but I only have 2 days...



Prioritize:

1) Test Module A (the new feature)
2) In the past 2 months:
   - Module B had 10 bugs ← Focus here!
   - Module C had 2 bugs
3) Test Module C if you have spare time

Test Summary Report: Release Note

The final deliverable from a Testing team, that summarizes the Following

- *Test Requirements*
- *Features tested*
- *Metrics*
- *Issues & Resolutions*
- *Defect Log*

JIRA Test Management Tool Demo

NON-FUNCTIONAL SYSTEM TESTING

NON-FUNCTIONAL SYSTEM TESTING

- Non-Functional Testing is defined as a type of Software testing to check non-functional aspects (performance, usability, reliability, etc.) of a software system.
- It is designed to test the readiness of a system as per nonfunctional aspects which are never addressed by functional testing.
- Non-functional testing is equally important as functional testing and affects client satisfaction.

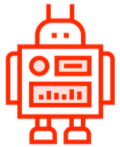
Non-functional System Characteristic Examples

Performance

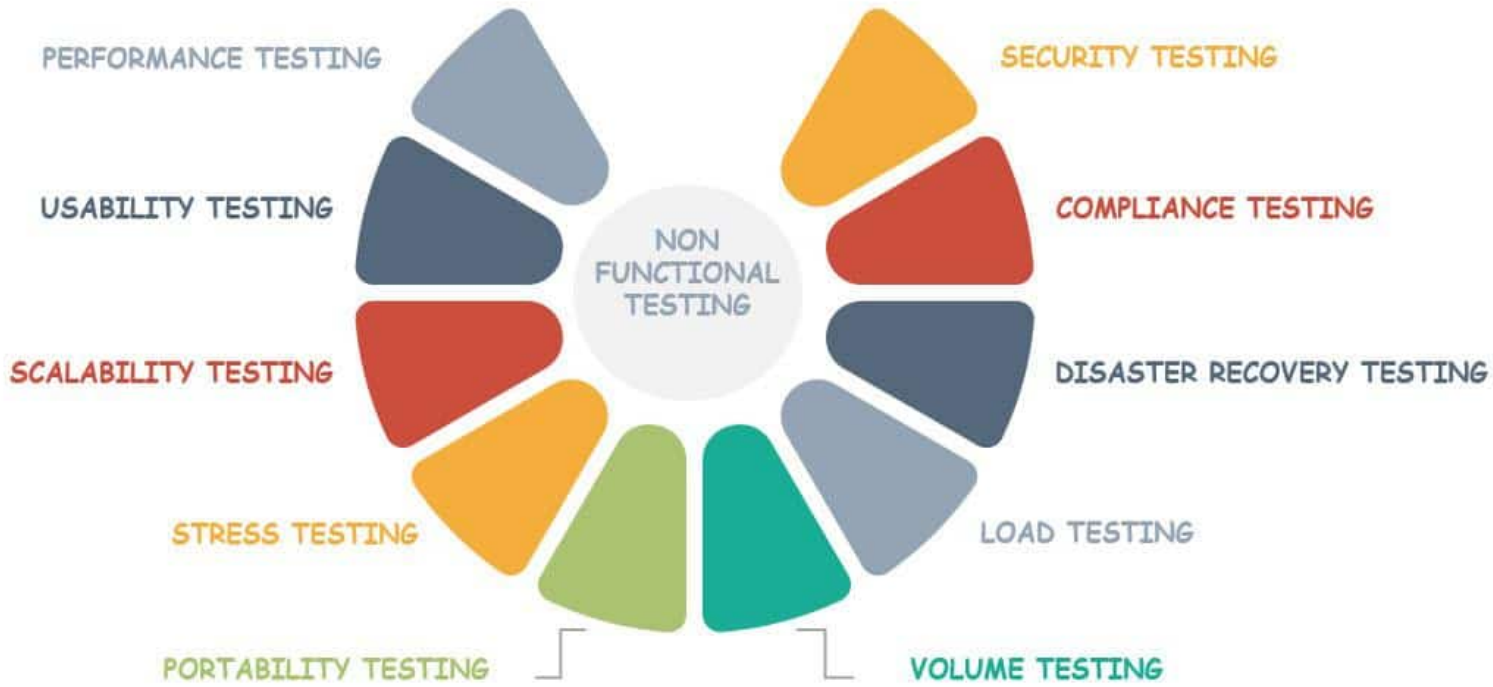
Security

Reliability

Maintainability



Usability



NON FUNCTIONAL TESTING

Performance Testing

Performance

- Typically how long it takes to complete a user scenario and how responsive the application is to a user

Load and Stress

- Simulate a heavy load on a system
- Determine how it performs under an expected high load (load), and the upper limits before it becomes unusable (stress)

Scalability

- Examine the ability of the system to scale up or out

Highly specialized tools that can simulate traffic and measure responsiveness are used

NON-FUNCTIONAL SYSTEM TESTING

- Volume Testing:
  - Evaluates the behavior of the software when a large volume of data is involved.
- Key points are:
  - When the software is subject to large amounts of data, checks the limit where the software fails.
  - Maximum database size is created and multiple clients query the database or create a larger report.
- Example– If the application is processing the database to create a report, a volume test would be to use a large result set and check if the report is printed correctly.

NON-FUNCTIONAL SYSTEM TESTING

- Usability Testing:
  - Evaluates the system for human use or checks if it is fit for use.
- Key points are:
  - Is the output correct and meaningful and is it the same as which was expected as per the business?
  - Are the errors diagnosed correctly?
  - Is the GUI correct and consistent with the standard?
  - Is the application easy for use?

NON-FUNCTIONAL SYSTEM TESTING

- Compatibility Testing:
- Evaluates that the application is compatible with other hardware /software with minimum and maximum configuration.
- Key points are:
  - Test each hardware with minimum and maximum configuration.
  - Test with different browsers.
  - Test cases are the same as those that were executed during functional testing.
  - In case the number of hardware and software are too many, then we can use Orthogonal Array Testing (OAT) techniques to arrive at the test cases to have maximum coverage.

NON-FUNCTIONAL SYSTEM TESTING

- Recovery Testing:
  - Evaluates that the application terminates gracefully in case of any failure and the data is recovered appropriately from any hardware and software failures.
- The tests are not limited to the below points:
  - Power interruption, to the client while doing CURD activities.
  - Invalid database-pointers and keys.
  - Database process is aborted or prematurely terminated.
  - Database pointers, fields and keys are corrupted manually and directly within the database.
  - Physically disconnect the communication, power turn off, turn down the routers and network servers.

ACCESSIBILITY TESTING

- Check usability by people with disabilities
  - Blind and low vision, deaf, color-blind, ...
- Use accessibility guidelines
  - Direct usability testing with all relevant groups is usually impractical; checking compliance to guidelines is practical and often reveals problems
- Example: W3C Web Content Accessibility Guidelines
  - Parts can be checked automatically
  - but manual check is still required
    - e.g., is the “alt” tag of the image meaningful?

INSTALLATION TESTING

- Before the testing
  - Configure the system
  - Attach proper number and kind of devices
  - Establish communication with other system
- The testing
  - Regression tests: to verify that the system has been installed properly and works

UI TESTING ("ACCEPTANCE")

- Automated UI testing ("automation")
  - Scripts and such that use your app and look for failures
  - A black-box system test
- Manual tests
  - Human beings click through predetermined paths
  - Need to write down the specific tests each time
- Ad-hoc tests
  - Human beings are "turned loose" on the app to see if they can break it

USABILITY TEST

- A usable product
  - is quickly learned
  - allows users to work efficiently
  - is pleasant to use
- Objective criteria
  - Time and number of operations to perform a task
  - Frequency of user error
- Plus overall, subjective satisfaction

Security Testing

Penetration Testing

- Simulated attack on a system to discover exploitable vulnerabilities

Component Vulnerability Analysis

- Scan for known vulnerabilities in open source components used by the application

API Security Testing

- Test the security of the business API for an application

Static Analysis

- Inspect code of the application for insecure practices

DIFFERENCE BETWEEN FUNCTIONAL AND NON-FUNCTIONAL TESTING

Features	*Functional Testing*	*Non-Functional Testing*
Objective	Functional testing describes the behavior of the software system.	Non-Functional testing describes the performance or usability of the software system.
Focus Area	It is based on the requirements of the business or the client.	It depends on the expectations of the end-user.
What to test	Functional testing tests the functionality of the software and helps in describing what the system should do.	Non-Functional testing tests the performance of the software. It helps in describing how the system should work.
Execution	It is done before Non Functional tests.	It is done after the Functional tests.
Requirement	Defining functional requirements is not difficult in Functional Testing.	For Non-Functional testing, it is difficult to define the requirements.
Testing Types	·Unit testing ·Smoke testing ·Integration Testing ·Regression testing ·User Acceptance Testing	·Performance Testing ·Usability Testing ·Scalability testing ·Stress Testing ·Portability Testing ·Volume Testing ·Load Testing ·Disaster Recover Testing ·Compliance Testing ·Localization and Internationalization Testing

NON-FUNCTIONAL SYSTEM TESTING TOOLS

- There are several tools available in the market for Performance (Load & Stress) testing.
- Few of them are listed below:
  - JMeter
  - Loadster
  - Loadrunner
  - Loadstorm
  - Neoload
  - Forecast
  - Load Complete
  - Webserver Stress Tool
  - WebLoad Professional
  - Loadtracer
  - vPerformer

Thank You

APPENDIX

EFFECTIVE DEFECT REPORT

If there is a defect in the software, then that should be proved through a well documented, detailed description of the steps to reproduce the defects and in some cases screen shot of the defect when it occurred. A good defect not only

helps the developers to quickly identify the problem but also help them to prioritize which defects to fix first.

- A good defect report is :
  - ✓ A proof for a defect discovered.
  - ✓ Determines the “Quality” of the tester.
  - ✓ Increases the probability of more defects getting fixed.
  - ✓ A critical work product which helps in process improvement and risk management.

- Characteristics of a good defect report are that it’s right to the point, lists all appropriate information, and easy to understand.

- It is easier and logical to follow a pattern rather that write unorganized and confusing reports.

DEFECT TRACKING & REPORTING

Defect tracking is the process of finding defects in a product (by inspection, testing or recording feedback from customers), and making new versions of the product that fix the defects.

Defects are useful

- In correcting the software
- To predict the quality of the software
- To gather statistics used to develop defect expectations in future applications
- To take preventive actions like training, establishing methods & procedures etc
- To reduce CoQ (CoQ = Preventive + Appraisal + Failure)
- To improve the software development process.

Stakeholders should be informed about

- Test progress
- Software quality

Defect tracking systems are computer database systems that store defects and help people to manage them. Some third-party defect management tools available are as follows:

- HP Quality Center
- Bugzilla (Opensource)
- JIRA

PERFORMANCE TESTING

- Stress Testing
  - Checks if the system can respond to many simultaneous requests (maximum # of users, peak demands)
- Volume testing
  - Test what happens if large amounts of data are handled
- Configuration testing
  - Test the various software and hardware configurations
- Compatibility test
  - Test backward compatibility with existing systems
- Security testing
  - Try to violate security requirements

PERFORMANCE TESTING

- Timing testing
  - Evaluate response times and time to perform a function
- Environmental test
  - Test tolerances for heat, humidity, motion, portability
- Quality testing
  - Test reliability, maintainability & availability of the system
- Recovery testing
  - Tests system's response to presence of errors or loss of data.
- Human factors testing
  - Tests user interface with user

NON-FUNCTIONAL SYSTEM TESTING

- Performance Testing:
  - Evaluates the overall performance of the system.
- Key elements are as follows:
  - Validates that the system meets the expected response time.
  - Evaluates that the significant elements of the application meet the desired response time.
  - It can also be conducted as a part of integration testing and system testing.

NON-FUNCTIONAL SYSTEM TESTING

- Load Testing:
  - Evaluates whether the system's performance is as expected under normal and expected conditions.
- Key points are:
  - Validates that the system performs as expected when concurrent users access the application and get the expected response time.
  - This test is repeated with multiple users to get the response time and throughput.
  - At the time of testing, the database should be realistic.
  - The test should be conducted on a dedicated server which stimulates the actual environment.
- How many hits/requests should the system be able to handle?
- What should be its performance under these circumstances?

NON-FUNCTIONAL SYSTEM TESTING

- Stress Testing:
  - Evaluates whether the system's performance is as expected when it is low on resources.
- Key points are:
  - Test on low memory or low disc space on clients/servers that reveal the defects which cannot be found under normal conditions.
  - Multiple users perform the same transactions on the same data.
  - Multiple clients are connected to the servers with different workloads.
  - Reduce the Think Time to "Zero" to stress the servers to their maximum stress.
- Think Time: Just like the time interval between typing your user and password.

STRESS TESTING

- Often requires extensive simulation of the execution environment
  - With systematic variation: What happens when we push the parameters? What if the number of users or requests is 10 times more, or 1000 times more?
- Often requires more resources (human and machine) than typical test cases
  - Separate from regular feature tests
  - Run less often, with more manual control
  - Diagnose deviations from expectation
    - Which may include difficult debugging of latent faults!