Software Test Design Techniques

PMIT 6111: ST & QA

Software Requirements

- A **requirement** is a feature that the system must have or a constraint that it must satisfy to be accepted by the client.
- **Requirements engineering** aims at defining the requirements of the system under construction.
- Requires collaboration of people with different backgrounds
  - Users with application domain knowledge
  - Developer with solution domain knowledge (design knowledge, implementation knowledge)
- Bridging the gap between user and developer:
  - Scenarios: Example of the use of the system in terms of a series of interactions with between the user and the system
  - Use cases: Abstraction that describes a class of scenarios
  - Recall the user's stories in XP

Types of requirement

- User requirements
  - Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.
- System requirements
  - A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

User and system requirements

User Requirement Definition

1. The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System Requirements Specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost, and the prescribing clinics shall be generated.

1.2 The system shall automatically generate the report for printing after 17.30 on the last working day of the month.

1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed, and the total cost of the prescribed drugs.

1.4 If drugs are available in different dose units (e.g., 10 mg, 20 mg) separate reports shall be created for each dose unit.

1.5 Access to all cost reports shall be restricted to authorized users listed on a management access control list.

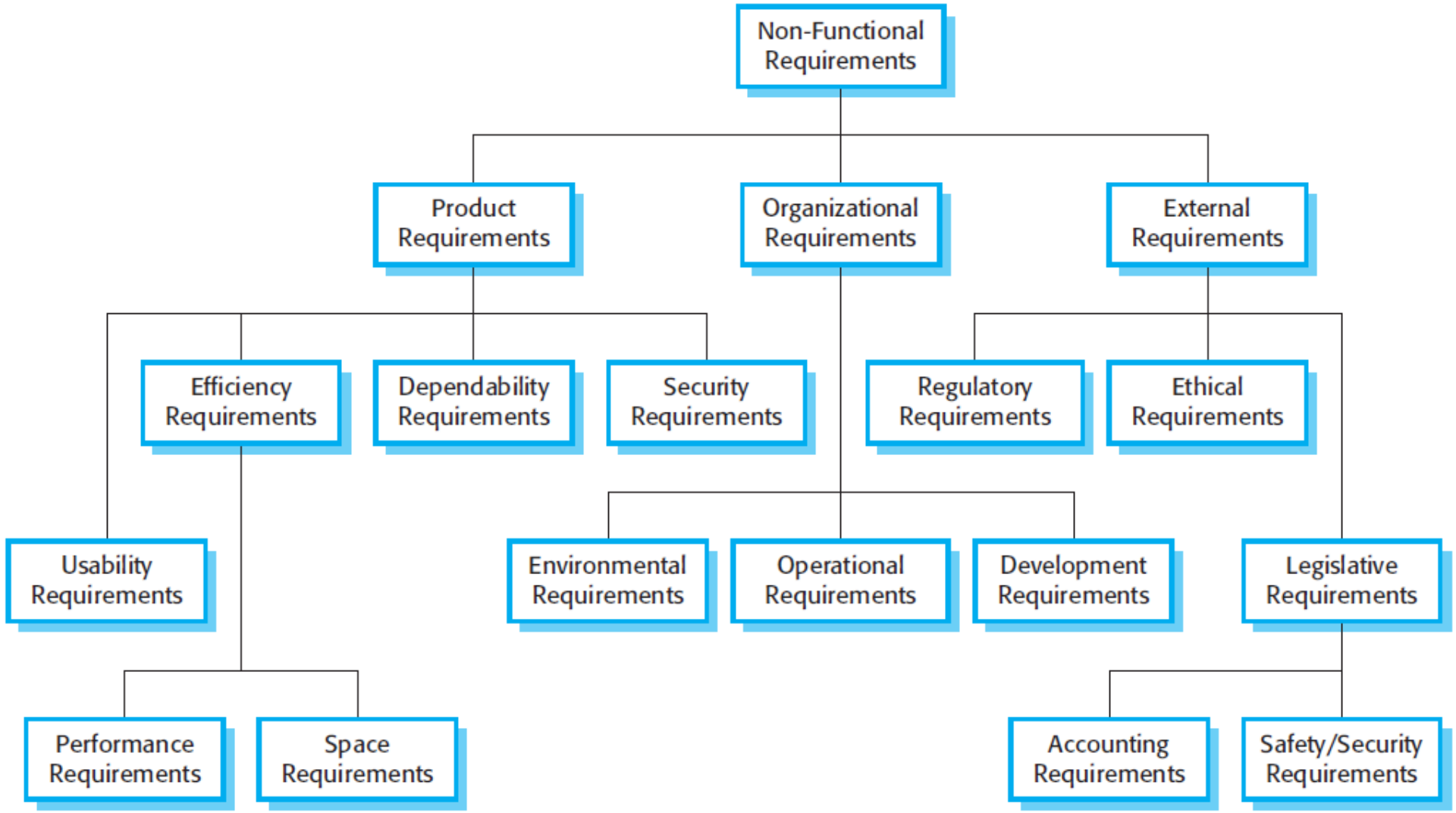
Functional and non-functional requirements

- Functional requirements
  - Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
  - May state what the system should not do.
- Non-functional requirements
  - Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
  - Often apply to the system as a whole rather than individual features or services.
- Domain requirements
  - Constraints on the system from the domain of operation

Functional requirements for the MHC-PMS

- A user shall be able to search the appointments lists for all clinics.
- The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day.
- Each staff member using the system shall be uniquely identified by his or her 8-digit employee number.

Types of nonfunctional requirement



Examples of nonfunctional requirements in the MHC-PMS

Product requirement
The MHC-PMS shall be available to all clinics during normal working hours (Mon–Fri, 08.30–17.30). Downtime within normal working hours shall not exceed five seconds in any one day.

Organizational requirement
Users of the MHC-PMS system shall authenticate themselves using their health authority identity card.

External requirement
The system shall implement patient privacy provisions as set out in HStan-03-2006-priv.

Sample of a Test Case

- **Title**: Login Page – Authenticate Successfully on gmail.com
- **Description:** A registered user should be able to successfully login at gmail.com.
  - Precondition: the user must already be registered with an email address and password.
  - Assumption: a supported browser is being used.
- **Test Steps:**
  - Navigate to gmail.com
  - In the ’email’ field, enter the email of the registered user.
  - Click the ‘Next’ button.
  - Enter the password of the registered user
  - Click ‘Sign In’
- **Expected Result:** A page displaying the gmail user’s inbox should load, showing any new message at the top of the page.

Test Development Process

- Test development process includes:
  - Test Analysis
    - Defining the approach
    - Identifying the right techniques
    - Identify associated risks
  - Test Design
    - Test design involves the creation of test cases and test data.
  - Test Implementation
    - Test implementation is about defining test procedures to group the test cases in an appropriate way. This is to execute and specify the sequential steps to be performed to run the test. Some test cases may need to be run in a specific sequence. Writing the test procedure is another opportunity to prioritize the tests.
    - Test implementation also involves preparing test execution schedule.

Equivalence Partitioning

- Equivalence partitioning is a method of deriving test cases when there is a large number of input data ranges.
- Equivalence partitioning helps to cut down exponentially on the number of cases required to test system. It is an attempt to get good test coverage, to find more errors with less number of test cases.
- In this method, classes of input conditions called equivalence classes are identified, with same kind of processing. Therefore, it leads to the generation of the same output.
- The implementation of equivalence partitioning includes:
  - Examining input and output
  - Dividing them into equivalence classes based on the behavior.
  - Inputs can be valid or positive, and invalid or negative.

Guidelines for Identifying Equivalence Classes

- As per requirement if input is numeric value within a range of values, identify one valid class inputs, which are within the valid range and two invalid equivalence class's inputs. The invalid classes can be both lower and higher than the expected input.
  - If a system accepts 'date of birth' as input and gives 'age' as output, the classes can be identified as follows:
    - First valid input class: Date is greater than zero. This will generate a valid output which is greater than zero.
    - 1st invalid class: Zero.
    - Second invalid class: Date is less than zero. For this invalid input, output class will generate an error message.
  - Hence, in this case, the valid input class is Date of Birth greater than zero.
  - An invalid class is Date of Birth less than zero. Any value within the same class gives output value of age.
  - The output value of age should be greater than zero if the class is greater than zero. If the class is less than zero, then, the output value should be an error message.

Equivalence Partitioning – Example

- Requirement: Employee ID should be 6 digits.
- System inputs have to be partitioned into ‘equivalence class.’ Input is a 6-digit integer, which can be from 100,000 to 999,999.
- Valid and Invalid equivalence partitions:
  - Valid: 100000 to 999999
  - Invalid: < 100000
  - Invalid: > 999999
- To have a robust test and reasonable coverage:
  - One or two values from each equivalence class should be selected;
  - The result or output for each type of input should be determined; and
  - The test cases should be defined accordingly.

Boundary Value Analysis

- Boundary Value Analysis or BVA is a black-box test design technique in which test cases are designed based on testing the boundaries between partitions for both valid and invalid boundaries.
- Boundary value is an input or output value, which is on the edge of an equivalence partition or at the smallest incremental distance on either side of an edge.

BVA – Example 1

- A program accepts a number in the range, -100 to +100. Here, the three sets of valid equivalent partitions are:
- First class: Negative range: -100 to -1
  - For the lower boundary, the values that need to be checked include -101, -100, -99,-2, and -1.
  - For the upper boundary, the values that need to be checked include -1, 0, and +1.
- Second class: Zero
- Third class: Positive range: Between 1 and 100
  - For the positive range, the lower boundary is 1, and the upper boundary is 100. Therefore, the values that need to be checked include 0, 1, 2, 99, 100, and 101.
- In this example, note that there is some duplication in values for the test data. When the duplicate inputs are removed, then, the final input conditions are: -101, -100, -99, -1, 0, 1, 99, 100, and 101.

BVA – Example 2

- Test cases, at the boundary of each identified class using equivalence partitions technique, can be chosen as given below.

Category	Equivalence Class	Boundary Values
Valid	100000 to 999999	100000, 100001, and 999999
Invalid	< 100000	000000, and 99999
Invalid	> 999999	999998, and 1000000

Decision Table

- A decision table displays a combination of inputs and/or stimuli, which is termed as ‘causes,’ with their associated outputs and/or actions, which are termed as ‘effects.’
- Permutations and combinations of these inputs are used to design test cases and are called decision table. This technique is also referred to as '***cause-effect***' table as there is an associated logical diagramming technique called ***'cause-effect graphing***,' which is sometimes used to help derive the decision table.
- Decision tables aid the systematic selection of effective test cases and can have the beneficial effect of finding problems and ambiguities in the specification. It is a technique that works well with equivalence partitioning. The combination of conditions explored may be that of equivalence partitions.

Decision Table Example

	Conditions/ Courses of Action	Rules					
		1	2	3	4	5	6
Condition Stubs	Employee type	S	H	S	H	S	H
	Hours worked	<40	<40	40	40	>40	>40
Action Stubs	Pay base salary	X		X		X	
	Calculate hourly wage		X		X		X
	Calculate overtime						X
	Produce Absence Report		X				

State Transition Diagrams – Key Terms

- Event: Input action that may cause a transition is called an Event.
- State: A State is a condition in which a system waits for events.
- Transition: Change from one state to another as a result of an event is called Transition.
- Action: An operation initiated by a transition is called Action.
- State Diagram: State diagram depicts a state that a component or a system can assume and show the events or circumstances that cause the system to change from one state to another.
- Behavior: Behavior is the sequence of messages or events that an object accepts.
- Object: An object is a substance, which is in a particular state and has a behavior.

State Transition Testing

- State transition testing is a black box test design technique in which test cases are designed to execute valid and invalid state transitions.
- Consider the switch is in ‘off’ state. An event occurs when the switch is on, and the action of this event is ‘light is in on state.’ Moving the switch from off state to on state is a transition.
- Here is the state transition table for this example.

	Step 1	Step 2	Step 3
Current State	OFF	ON	OFF
Input	Switch ON	Switch OFF	Switch ON
Output	Light ON	Light OFF	Light ON
Finish State	ON	OFF	ON

Steps of State Transition Testing

- Consider the current state and provide input to see the output and finish state for 3 different steps.
- Step 1
  - In step 1, the current state is OFF, and input is Switch ON. In this case, the output is Light ON and finish state is ON.
- Step 2
  - In step 2, the current state is ON, and input is Switch OFF. In this case, the output is Light OFF and finish state is OFF.
- Step 3
  - In step 3, the current state is OFF, and input is Switch ON. In this case, the output is Light ON and finish state is ON.

State Transition Testing – Example

- Given below is an example of State Transition Testing. Test Steps test a scenario where a user tries to log in to another session while being logged into another session already.

	Step 1	Step 2	Step 3	Step 4
Current State	Not Signed In	Logged In	Logged In	Logged In
Input	User Name and PWD	Account Number	User Name and PWD in a new session	Click Sign Out
Output	Message and section Change	Report	Error Message	Sign Out Message
Finish State	Logged In	Logged In	Logged In	Signed Out

Steps in State Transition Testing – Example

- Step 1: In step one, the current state of the User is ‘Not signed in.’
  - When the user provides the input of a valid User Name and password and clicks Submit, a message gets displayed, and the section changes. The final state of step one is ‘Logged in.’
- Step 2: In step two, the current state of the User is ‘Logged In.’
  - When the user provides the input of a valid Account name, the user gets a report as Output. The final state of step two is ‘still Logged in.’
- Step 3: In step three, the current state is, ‘the User continues to be Logged In.’
  - However, the User tries to login into another session at the same time. When the user provides the same User Name and password in another session, an error message is displayed that the user is already logged in. The final state of step three is ‘Logged in the first session.’
- Step 4: In step four, the current state is ‘the User remains Logged In.’
  - When the user clicks Signs out, a message is displayed that the user has signed out and the section changes. The final state of this step is ‘Logged Out.’

Use Case Testing – Key Terms

- Scenarios: Interactions arranged sequentially are called Scenarios.
- Test Case: Each scenario instance is called Test Case.
- Use Case: Collection of possible scenarios performed by the User on the system is called Use Case.
- Use Case Precondition: Conditions to be met for the successful completion of use case is called Use Case Pre-condition.
- Use Case Postcondition: The final state of the system after the completion of use case is called Use Case Post-condition.

Use Case Testing

- A use case is a description of a particular use of the system by a user. Each use case describes the user interactions with the system, to achieve a specific task.

- Use case testing is a technique that helps to identify test cases that exercise the system on a transaction by transaction basis from start to finish. It helps in designing acceptance testing and identifying integration defects and defects from common real-life scenarios.

Use Case Testing – Example

- For example, an online bill payment that has three main use cases. They are:
  - Login
  - Online Payment
  - Logout
- These use cases define all the conditions regarding the requirement for the logical part of the system or application. These use cases are inter-related. Before making an online payment or logging out of the application, the user needs to log into the application.
- Test cases are derived from use cases to represent end-to-end transactions in the system

Coverage of Structure-Based Testing Techniques

- A **Coverage** is defined as the number of items covered in testing divided by the total number of items. Coverage, therefore, defines the extent of code that has been tested out of the total code in the system.
  - Similarly, **requirements coverage** mean the extent of requirements tested out of the total requirements in the system.
- Coverage items can be a statement, branch, condition, multiple conditions, or a component. The objective of testing is to achieve maximum code coverage as even a small amount of untested code can result in defects when the system goes live.
- **Coverage techniques** measure a single dimension of a multi-dimensional concept. Two different test cases may achieve the same coverage.
  - However, the input data of one test case may be able to find an error, and the input data of another test might not be able to find the error.

Structure-Based Testing Techniques and Coverage Types

- ***Statement Coverage :*** Coverage is some statements executed divided by a total number of statements.
- ***Decision Coverage:*** Decision Coverage is, the number of decision outcomes achieved divided by a total number of decision outcomes. The goal of testing should be to ensure all branches and statements have been tested at least once.
- ***Condition Coverage:*** Most of the outcomes of test cases designed as per the white box test design technique are derived by following the condition coverage method. By following this method, the test cases are designed in a manner that the condition outcomes (the evaluation of a condition as 'true' or 'false') are executed automatically.
- ***Multiple Condition Coverage:*** Every combination of 'true' or 'false' for the conditions related to a decision have to be tested in this technique

- Though these techniques are mostly used at a component level, these can also be applied to integration and system testing levels.

Structure-Based Testing Techniques Examples

- Example 1
  - Look at the piece of code below and understand the percentage of test coverage using three techniques which we have just now discussed.

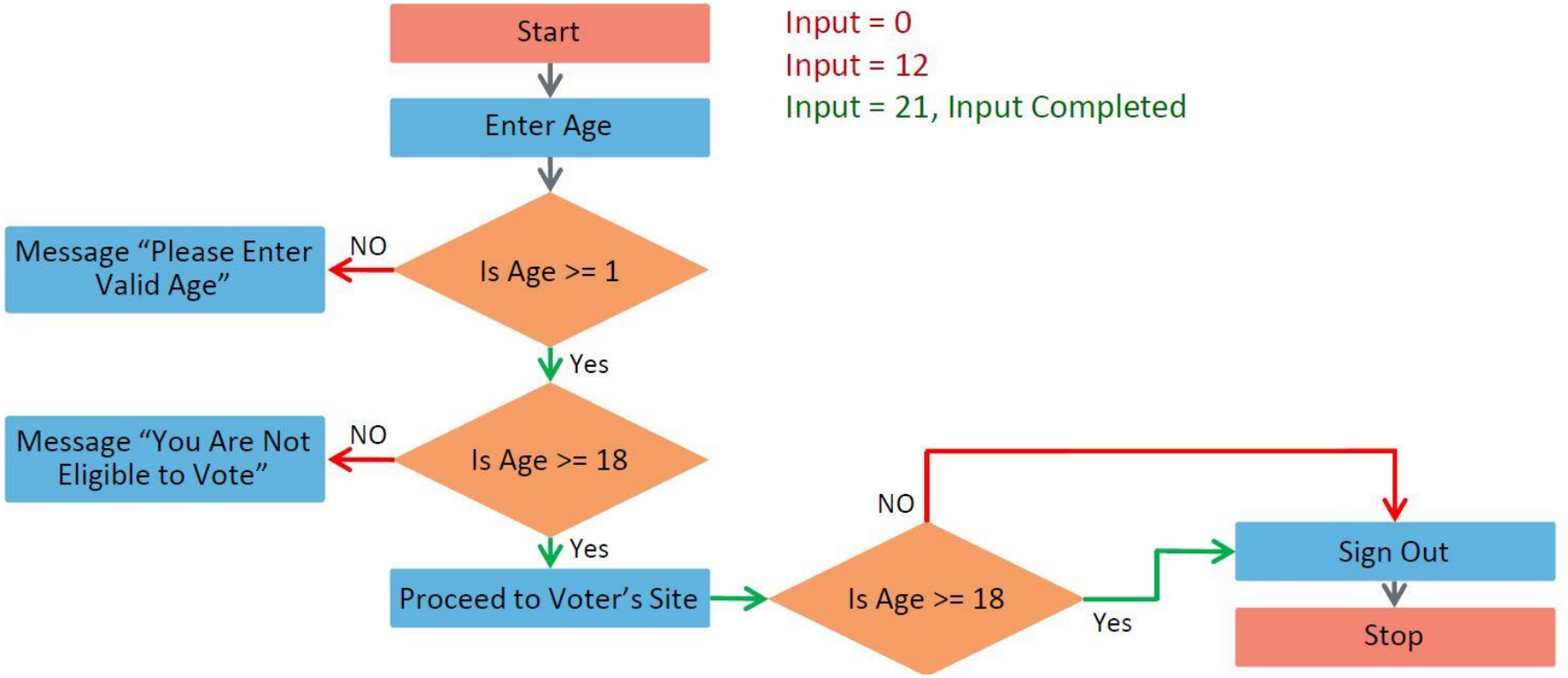
```
Var Ave Bal
If ave bal < 5000
{ msg = “Insufficient monthly balance”}
else If ave bal >= 5000
{
msg = “Thank you for maintaining proper monthly balance”
If ave bal >= 50000
{ msg = “ you have earned over “ + ave bal / 50000 * 100 + “credit points”}
end if
}
end if
```

If **ave bal = 4500** then the input value covers 33% of statements, 50% of decisions, and 33% of branches.

If **ave bal = 55000** then the input value covers covers 88% of statement, 50% of decision and 66% of branch.

Structure-Based Testing Techniques Examples..

- Example 2
  - Consider the example given below of a small application which decides whether an applicant is eligible to vote or not.



Structure-Based Testing Techniques Examples..

- Example 2
  - Different inputs to the program, execute different decisions and branches.
  - An invalid input of 0, fails the condition that age is greater than or equal to 1. Hence, an error message is displayed, and the user is prompted to enter a valid age.
  - The second input of 12, passes the first condition that age is greater than or equal to 1 but fails the second condition that age is greater than or equal to 18. Hence, a message is displayed to the user as ‘You are not eligible to vote.’
  - The third input of 21, passes the first and the second condition. Hence, the user is directed to the Voters site.
  - At this stage, the system has a condition to check whether the user has completed the input correctly or not. It is the input of 21, along with the completed input to the system in the last stage that accomplishes the goal of the system.
  - Different inputs execute different pieces of the code. Hence, Testers should build test cases using different input classes to ensure maximum statement, decision, and branch coverage.

Experience-Based Testing Techniques – Types

- Exploratory Testing
  - In Exploratory Testing, tests are derived based on Tester's experience and intuition. It is a hands-on approach in which Testers are involved in minimum planning and maximum test execution.
  - The test design and test execution activities are performed simultaneously and most of the times with no formal or limited documentation, test conditions, cases, or scripts.
  - For example, the tester decides to use BVA and tests the most important boundary values without writing them. Some notes will be written during the Exploratory Testing session, for the later production of a report.

Experience-Based Testing Techniques – Types

- Fault Attack/ Error Guessing:
  - Error guessing is a test design technique where the experience of the Tester is used to identify defects in the component or system. It is a widely practiced technique. The Tester is encouraged to think of situations in which the software may not be able to cope.
  - For example, division by zero, blank input, empty files, and incorrect data.
- Random Testing
  - In Random testing, Tester chooses already defined test cases and executes them. The tester takes the components randomly and starts testing its functionality.
  - Random testing is also known as Monkey Testing as the application is not tested in sequence. It can also be categorized as black box testing technique.

Experience-Based Techniques vs. Other techniques

	Experience-Based	Behavior-Based and Structural based
Formality	This is more informal. The process is dependent on the tester's skills and experience.	The formal process of creation of test design, test execution, and defect logging.
Effort	Needs less effort.	Needs more effort during Test Design and Test Execution.
Effectiveness	This is very effective when requirements are not clearly documented or understood. Coverage is not measured. Hence, completeness of testing cannot be measured accurately.	This is very effective when requirements are clearly documented and understood. Level of coverage provided by these techniques improves effectiveness.

Choosing Test Techniques

- Choosing the appropriate testing techniques is based on some factors such as
  - Development life cycle
  - Use case models
  - Type of system
  - Level and type of risk
  - Test objective
  - Time and budget
  - Tester’s experience on the type of defects found in similar systems.

Source & References

- Ian Sommerville, “software Engineering”, 9th edition, Chapter 4: Requirements Engineering.
- Software Test Design Techniques – Static and Dynamic Testing, By Arvind Rongala, Manager, Business Development and Marketing, Invensis Technologies; https://www.invensis.net/blog/it/software-test-design-techniques-static-and-dynamic-testing/
- Test Design Techniques CTFL Tutorial, https://www.simplilearn.com/test-design-techniques-ctfl-tutorial-video