RTPNOTES.vercel.app

Software-Testing-Module-1-Important-Topics-PYQs

For more notes visit

https://rtpnotes.vercel.app

RTPNOTES.vercel.app

Coverage

  - a) Functional Coverage:
  - b) Statement Coverage:
  - c) Branch Coverage:
  - d) Conditional Coverage:
- 9. Write the positive and negative testcases for an ATM machine.
  - 1. Test Case: Valid Card Insertion
  - 2. Test Case: Correct PIN Entry
  - 3. Test Case: Check Account Balance
  - 4. Test Case: Withdraw Cash
  - 5. Test Case: Transfer Money
  - 6. Test Case: Print Receipt
  - 7. Test Case: ATM Machine Out of Service
- 10. Explain the five levels of testing goals based on test process maturity
  - Level 1: Initialization
  - Level 2: Definition
  - Level 3: Integration
  - Level 4: Measurement and Management
  - Level 5: Optimization
- 11. Explain the different types of system testing
  - 1. Functional Testing
  - 2. Performance Testing
  - 3. Security Testing
  - 4. Compatibility Testing
  - 5. Usability Testing
  - 6. Regression Testing
  - 7. Acceptance Testing
- 12. Design six test cases for a program which checks whether a number between 1 and 100 is prime or not
  - Test Case 1: Test a Prime Number (Basic Prime)
  - Test Case 2: Test a Non-Prime Number (Composite Number)
  - Test Case 3: Test the Number 1 (Edge Case)
  - Test Case 4: Test the Number 100 (Upper Bound)

RTPNOTES.vercel.app

1. State the four objectives of testing and define Test Case

Objectives of Testing

1. **Find Defects Before They Cause Problems**
   - The main goal is to **catch errors** in the software before it goes live.
   - Think of it like checking your homework for mistakes before submitting it to avoid bad grades!
2. **Improve Software Quality**
   - After finding errors, **fix them** and **test again** to ensure the software meets the required quality standards.
   - This step is like **editing** your work after finding typos to make it better.
3. **Efficient Testing within Budget and Time Limits**
   - Testing needs to be **effective** (find errors) and **efficient** (do it quickly and within budget).
   - Imagine trying to finish a project before a deadline, making sure you do a good job but without spending too much time or money.
4. **Record Errors for Future Prevention**
   - Keep a **record of errors** to learn from them and avoid making the same mistakes in future software projects.
   - It's like writing down all the things you got wrong on your tests so you can study harder next time and avoid them.

What is a Test Case?

- A **Test Case** is a set of **conditions** and **steps** used to check if a software system works correctly.
- It includes:
  1. **Inputs** (What you enter into the system)
  2. **Actions** (What the system should do)
  3. **Expected Results** (What the system should return or do)

RTPNOTES.vercel.app

2. Explain Regression testing at different software testing levels

Regression Testing is a critical process in software testing that ensures recent code changes or additions do not negatively impact the existing functionality of the software. It's about verifying that modifications, bug fixes, or new features don't break or affect the already tested features of the system. Let's look at how regression testing is applied at different levels of software testing.

1. Unit Testing Level

- **Purpose**: At the unit testing level, regression testing ensures that changes to individual units of the code (like functions or methods) do not break the behavior of the unit.
- **How it Works**: When a developer fixes a bug or modifies a function, the previously written unit tests for that function or method are re-run. If the new changes cause any failures in the existing tests, it indicates that the changes have impacted the unit's functionality.
- **Example**: If you fix a bug in a "Login" function, regression testing will re-run unit tests that validate correct login functionality and ensure that the fix didn't break the login feature.

2. Integration Testing Level

- **Purpose**: At the integration testing level, regression testing ensures that changes in individual units (like added features or fixed defects) don't disrupt how different units or modules of the software interact.
- **How it Works**: When a bug is fixed or a new feature is added, previously executed integration tests are re-run to ensure that the interaction between integrated components still works as expected.
- **Example**: If a new "Payment Gateway" feature is added, regression testing would check whether this new feature still allows the rest of the system (like order processing and inventory management) to work properly when integrated.

3. System Testing Level

- **Purpose**: Regression testing at the system testing level ensures that the system as a whole remains intact after changes.
- **How it Works**: After the system undergoes changes (new features, fixes, or improvements), full system tests are run to verify that the new code hasn't broken any other part of the

RTPNOTES.vercel.app

system. The test focuses on the complete product, validating both functional and non-functional aspects.

- **Example**: After adding a new "Search" feature to the system, regression testing ensures that the system-wide processes, like login, checkout, and reporting, still function correctly.

4. Acceptance Testing Level

- **Purpose**: At the acceptance testing level, regression testing ensures that the software still meets the business requirements and that no new issues are introduced in the final product.
- **How it Works**: Regression testing during acceptance tests involves verifying that the newly developed or modified features meet the expected user requirements while ensuring that previously validated features still behave as expected.
- **Example**: If new performance enhancements are made, regression testing checks whether the performance improvements haven’t caused any issues with core features like the user dashboard or data entry forms.

3. Differentiate between Verification and Validation

Verification and Validation are two fundamental aspects of software quality assurance. They ensure that the software is both well-built and meets user expectations.

What is Verification?

- **Definition**: Verification is the process of evaluating software to ensure that the products of a given development phase satisfy the conditions or requirements imposed at the start of that phase. It ensures that the software is built correctly according to the specifications and design.
- **Characteristics of Verification**:
  - **Static Process**: No actual execution of the code is involved.
  - **Focuses on Documentation and Design**: Verifying whether the software is being developed according to the correct specifications, design, and architecture.
  - **Objective Process**: It is relatively objective and involves checking whether the system is engineered correctly and is free of errors in terms of design and code
- **Methods of Verification**:
  - **Static Testing**

RTPNOTES.vercel.app

- **Walkthrough**
- **Inspection**
- **Review**

What is Validation?

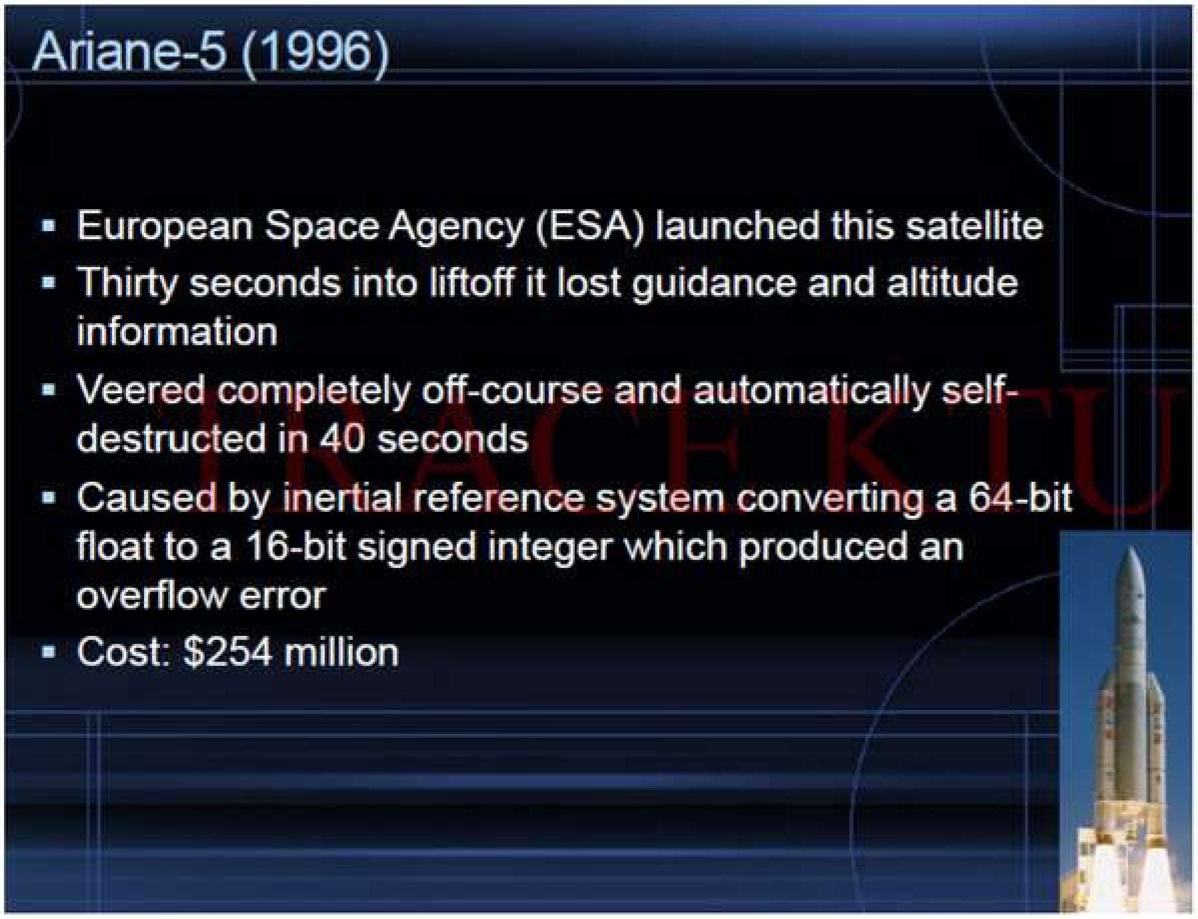
- **Definition**: Validation is the process of evaluating the software during or at the end of the development process to ensure it meets the specified requirements. It ensures that the software satisfies customer expectations and is fit for its intended purpose.
- **Characteristics of Validation**:
  - **Dynamic Process**: It involves the execution of the code to verify whether the software works as expected.
  - **Focuses on the Final Product**: Validation checks if the software meets the functional requirements and user needs.
  - **Subjective Process**: It is more subjective, focusing on whether the product meets the customer's expectations and real-world use cases.
- **Methods of Validation**:
  - **Dynamic Testing** (e.g., Functional Testing)
  - **End-User Testing** (e.g., User Acceptance Testing)
- **Example**: Running the actual application with test data to see if it meets customer requirements, such as confirming that the user can log in successfully using valid credentials.

Aspect	Verification	Validation
Nature	Static process (no code execution)	Dynamic process (involves code execution)
Focus	Ensures the software is built correctly according to specs	Ensures the software meets customer expectations
Methods	Walkthroughs, reviews, inspections, desk-checking	Black-box testing, gray-box testing, white-box testing
Involvement	Human-based checking of design and documentation	Involves actual testing and interaction with the software
Primary Goal	Check if the software conforms to specifications	Check if the software meets customer requirements and needs
Level	Low-level exercise (focused on code, design, and architecture)	High-level exercise (focused on actual product usability)

RTPNOTES.vercel.app

Aspect	Verification	Validation
Performed By	QA team, developers, designers	Testing team, sometimes with involvement from end users
Error Detection	Can catch errors related to design and implementation	Can catch errors related to user expectations and real-world functionality
When Performed	Before validation (early in the development process)	After verification (later in the development process)

4. List out any 3 popular software bugs

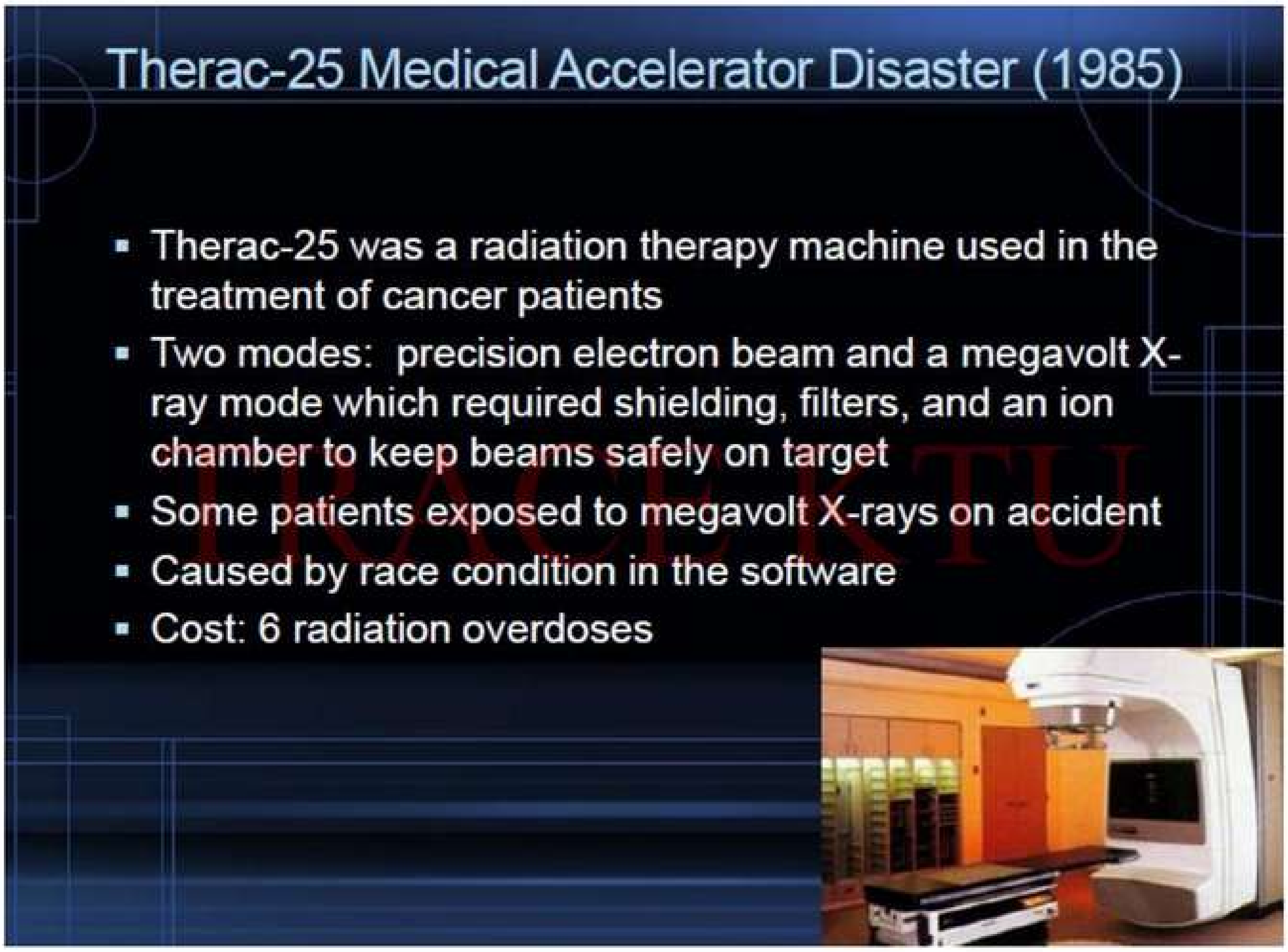


Ariane-5 (1996)

- European Space Agency (ESA) launched this satellite
- Thirty seconds into liftoff it lost guidance and altitude information
- Veered completely off-course and automatically self-destructed in 40 seconds
- Caused by inertial reference system converting a 64-bit float to a 16-bit signed integer which produced an overflow error
- Cost: $254 million

TRACE KTU

RTPNOTES.vercel.app

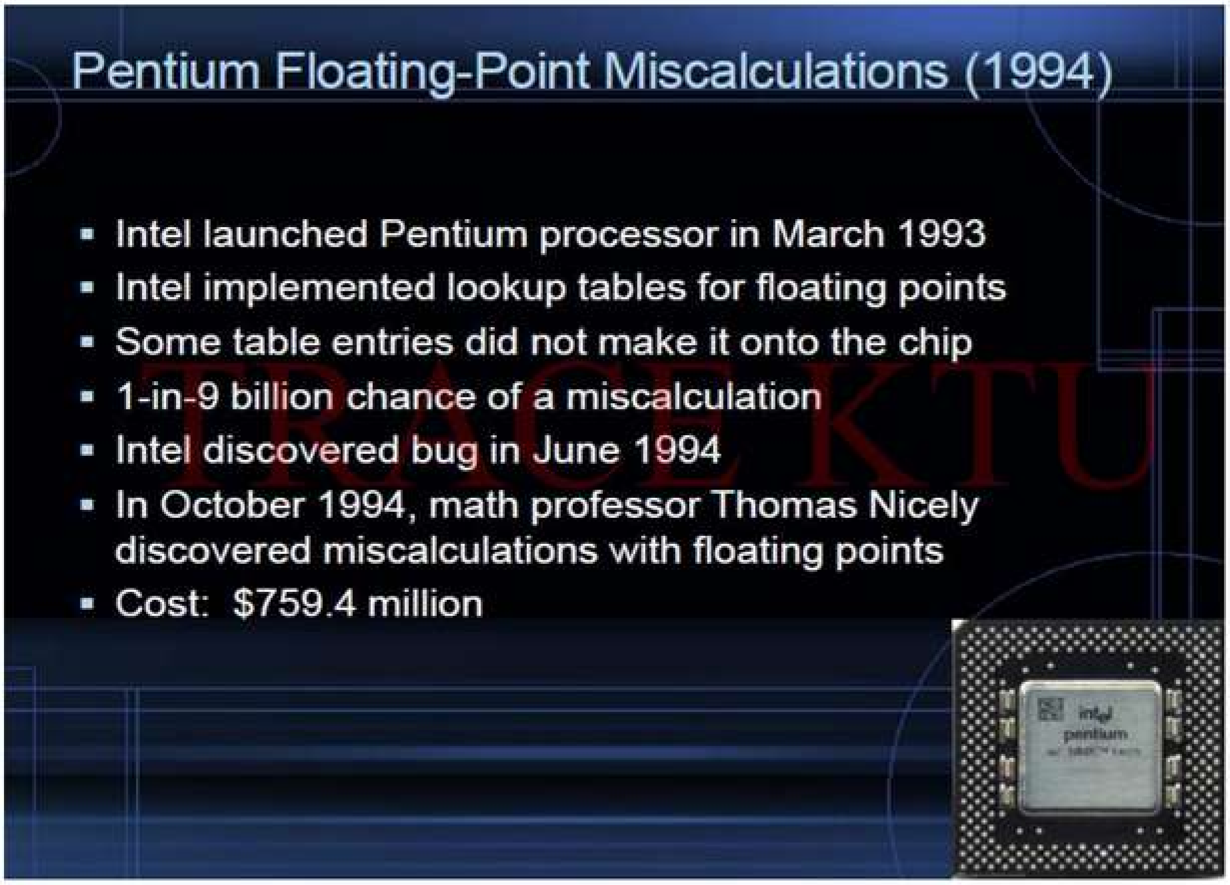


Therac-25 Medical Accelerator Disaster (1985)

- Therac-25 was a radiation therapy machine used in the treatment of cancer patients
- Two modes: precision electron beam and a megavolt X-ray mode which required shielding, filters, and an ion chamber to keep beams safely on target
- Some patients exposed to megavolt X-rays on accident
- Caused by race condition in the software
- Cost: 6 radiation overdoses

TRACE KTU

RTPNOTES.vercel.app



Pentium Floating-Point Miscalculations (1994)

- Intel launched Pentium processor in March 1993
- Intel implemented lookup tables for floating points
- Some table entries did not make it onto the chip
- 1-in-9 billion chance of a miscalculation
- Intel discovered bug in June 1994
- In October 1994, math professor Thomas Nicely discovered miscalculations with floating points
- Cost: $759.4 million

TRACE KTU

5. Define the terms Failure, Error, Fault and defect in software testing literature

1. **Failure**:
   - **Definition**: A **failure** happens when the software does not perform as expected or does not meet the requirements. It's when the software behaves incorrectly or produces incorrect results during execution.
   - **Example**: If a website crashes when a user tries to log in, that’s a failure.
2. **Error**:
   - **Definition**: An **error** is a mistake made by a developer while writing the code or during design. It's a human action that leads to incorrect results or behavior.
   - **Example**: If a developer writes the wrong condition in an if-statement, that’s an error in the code.
3. **Fault**:

RTPNOTES.vercel.app

- **Definition**: A **fault** (sometimes called a bug) is a defect in the software, caused by an error in the code, that has the potential to cause a failure. A fault is a problem in the software's design or coding.
- **Example**: If the code has a logical flaw that could cause incorrect output, it’s a fault.

4. **Defect**:
   - **Definition**: A **defect** is any issue or flaw in the software that results in failure or incorrect behavior. It’s a problem that needs fixing, whether it’s caused by a fault or another issue in the software.
   - **Example**: If a user enters an incorrect password and the system doesn’t show an error message, that's a defect in how the system handles invalid input.

In simple terms:

- **Error** is a mistake made by the developer.
- **Fault** is the bug caused by that error in the code.
- **Defect** is a general term for any issue that could cause problems in the software.
- **Failure** is the result or outcome when the software behaves incorrectly.

---

6. Discuss various types of testing methods with examples a) Blackbox testing, b) Whitebox testing, c) Graybox testing

a) Black-box Testing

Definition:
Black-box testing is a testing method where the tester focuses on testing the functionality of the software without knowing its internal workings. It’s concerned with **inputs and outputs** and ensures the software behaves as expected from the user’s perspective.

Key Characteristics:

- No knowledge of internal code or structure is required.
- Focuses on functional requirements and features of the software.
- Often used for higher-level testing (like system testing, acceptance testing).

Examples:

RTPNOTES.vercel.app

1. **Functional Testing**: Testing whether a login form works as expected (e.g., entering a valid username and password leads to a successful login).
2. **User Interface Testing**: Checking if the buttons, text fields, and forms on a website or app behave correctly when clicked or filled in.
3. **Regression Testing**: Ensuring that the new features or changes to the software don't break existing functionality.

Example Scenario:
Imagine you are testing a calculator app. You might test if:

- **5 + 3 = 8**.
- **4 ÷ 2 = 2**. You don't need to know how the calculator app performs the calculation inside (no knowledge of code); you just focus on if it produces the correct results for different inputs.

b) White-box Testing

Definition:
White-box testing, also known as **clear-box** or **structural testing**, is a testing method where the tester has full knowledge of the internal workings, code, and structure of the application.

Key Characteristics:

- Requires knowledge of the internal code or system architecture.
- Focuses on code correctness, logic, paths, and branches.
- It's typically used by developers for unit testing and integration testing.

Examples:

1. **Unit Testing**: Testing individual components or functions of a program (e.g., checking if a function that calculates interest is working correctly by using sample inputs).
2. **Code Coverage Testing**: Ensuring that every line of code, branch, condition, and path in a function or method has been tested at least once.
3. **Path Testing**: Evaluating all possible execution paths in the code to ensure that no path produces incorrect results.

Example Scenario:
Consider a function that checks if a number is even. As part of white-box testing, the tester will:

- Check if the condition `if (number % 2 == 0)` is properly executed.

RTPNOTES.vercel.app

- Verify if all possible branches are covered, like testing both even and odd numbers.

c) Gray-box Testing

Definition:
Gray-box testing is a hybrid testing method that combines aspects of both black-box and white-box testing. The tester has partial knowledge of the internal workings of the application, which helps design test cases based on the software's architecture while still focusing on the user perspective.

Key Characteristics:

- Combines the tester’s knowledge of the code structure with the ability to test functionality.
- Often used in integration testing, penetration testing, and security testing.
- Typically performed by testers with some understanding of the application’s internals but who also simulate the end-user’s perspective.

Examples:

1. **Penetration Testing**: A security expert who knows some internal workings of the software tests for vulnerabilities or potential security flaws by attacking the system like an outside hacker.
2. **Integration Testing**: Testing the interfaces and interaction between different modules with some knowledge of the system’s architecture.
3. **Database Testing**: Checking whether the backend database is being accessed and modified properly when a user interacts with the frontend, based on knowledge of the database structure.

Example Scenario:
Imagine you are testing an online shopping application. In gray-box testing, you might:

- Know how the database stores user orders and check if the system properly updates the database when a user places an order.
- Test the user interface (black-box) to check if the user can correctly place an order, but you also know the backend (white-box) and check if the order details are correctly passed to the database.

RTPNOTES.vercel.app

7. Explain coverage criteria for testing and identify the characteristics of a good coverage criteria.

Coverage criteria are guidelines or measures used to assess whether a test case or a suite of test cases adequately covers the software under test. The main goal of coverage criteria is to ensure that the tests cover a sufficient range of conditions, inputs, and program structures to detect defects effectively. The more comprehensive the coverage, the higher the likelihood of discovering defects in the software.

Characteristics of a Good Coverage Criterion

A good **coverage criterion** should have the following characteristics to ensure the effectiveness of the testing process:

1. **Comprehensiveness**:
   A good coverage criterion should ensure that all aspects of the system (e.g., code, requirements, functions, user behavior) are tested. The criterion should help identify areas of the software that are not covered by the test cases.
2. **Effectiveness**:
   The criterion should effectively uncover defects in the system. A coverage criterion is effective if it helps the tester identify parts of the system where bugs are likely to occur, especially in critical or complex parts of the code.
3. **Simplicity**:
   While comprehensive, the criterion should not be too complex to implement. A simple criterion is easier to apply and understand, making it practical for real-world testing environments.
4. **Relevance to Test Objectives**:
   The criterion should be aligned with the goals of testing. For example, if the goal is to ensure that all business requirements are satisfied, then **requirements coverage** will be the most relevant. If the goal is to test the reliability of the code, then **code coverage** is more appropriate.
5. **Achievability**:
   A good coverage criterion should be realistic and achievable within the project constraints, such as time, budget, and available resources. Testing for 100% coverage might not always be feasible, so it’s important to set practical goals.
6. **Minimizing Redundancy**:
   The criterion should help eliminate redundant test cases while ensuring that all important

aspects of the software are tested. For instance, testing the same functionality under the same conditions repeatedly does not add value and should be avoided

8. Explain the following code fragment based on the following coverage criteria a) Functional Coverage b) Statement Coverage c)Branch Coverage d)Conditional Coverage

```
int foo(int x,int y){
    int z=0;
    if (x>0) && (y>0)){
        z=x;
    }
    return z;
}
```

a) Functional Coverage:

Functional Coverage ensures that all the functional aspects or features of the program have been tested. In this case, the function `foo()` takes two inputs, `x` and `y`, and returns an integer `z`. To achieve functional coverage, we would need to ensure that the function has been tested for all possible input combinations to verify its behavior.

The function's behavior depends on the values of `x` and `y`. There are two possible functional outcomes:

1. If both `x` and `y` are greater than 0, `z` will be set to the value of `x`.
2. If either `x` or `y` is not greater than 0, `z` remains 0.

Test cases for functional coverage:

- Test case 1: `x = 1, y = 1` → The condition `(x > 0) && (y > 0)` is true, so `z` will be set to `1`.
- Test case 2: `x = -1, y = 1` → The condition `(x > 0) && (y > 0)` is false, so `z` remains `0`.
- Test case 3: `x = 1, y = -1` → The condition `(x > 0) && (y > 0)` is false, so `z` remains `0`.

RTPNOTES.vercel.app

- Test case 4: `x = -1, y = -1` → The condition `(x > 0) && (y > 0)` is false, so `z` remains `0`.

Functional coverage requires testing these different input combinations to ensure the function behaves as expected.

b) Statement Coverage:

Statement Coverage (also known as line coverage) ensures that every line of code in the program has been executed at least once during testing.

The function has the following statements:

1. `int z = 0;` - Initialization of `z`.
2. `if ((x > 0) && (y > 0)) { z = x; }` - Conditional check and assignment to `z`.
3. `return z;` - Return statement.

To achieve **100% statement coverage**, we need to ensure that both the true and false branches of the `if` statement are tested. Specifically:

- Test case 1: `x = 1, y = 1` → This will cover the `if` statement and set `z = x`.
- Test case 2: `x = -1, y = 1` → This will cover the `else` part (the `if` condition fails), and `z` remains `0`.

With these two test cases, all the statements will be executed at least once

c) Branch Coverage:

Branch Coverage ensures that every branch (decision point) in the code is executed at least once.

The critical branch here is the conditional statement `if ((x > 0) && (y > 0))`. The condition has two branches:

1. The **true branch**: If `(x > 0) && (y > 0)` is true, then `z = x`.
2. The **false branch**: If `(x > 0) && (y > 0)` is false, then `z` remains `0`.

To achieve **100% branch coverage**, we need to test both the true and false branches. The two test cases mentioned for statement coverage also fulfill the branch coverage requirement:

- Test case 1: `x = 1, y = 1` → This covers the **true branch** where `z = x`.

RTPNOTES.vercel.app

- Test case 2: `x = -1, y = 1` or `x = 1, y = -1` → This covers the **false branch**, where `z` remains `0`.

Thus, we achieve 100% branch coverage with these two test cases.

d) Conditional Coverage:

Conditional Coverage ensures that each individual condition in a compound condition has been tested for both true and false outcomes.

The compound condition in the `if` statement is `(x > 0) && (y > 0)`, which consists of two conditions:

1. `x > 0`
2. `y > 0`

To achieve **100% conditional coverage**, we need to test both conditions for both their true and false values. This requires testing:

1. `x > 0` to be true and `y > 0` to be true.
2. `x > 0` to be true and `y > 0` to be false.
3. `x > 0` to be false and `y > 0` to be true.
4. `x > 0` to be false and `y > 0` to be false.

The test cases to achieve conditional coverage:

- Test case 1: `x = 1, y = 1` → Both conditions are true, so the `if` condition is true, and `z = x`.
- Test case 2: `x = -1, y = 1` → The first condition (`x > 0`) is false, while the second condition (`y > 0`) is true, so the `if` condition is false.
- Test case 3: `x = 1, y = -1` → The first condition (`x > 0`) is true, while the second condition (`y > 0`) is false, so the `if` condition is false.
- Test case 4: `x = -1, y = -1` → Both conditions are false, so the `if` condition is false.

By using these four test cases, we cover all possible true/false outcomes for each individual condition in the `if` statement.

RTPNOTES.vercel.app

Coverage Type	Test Case(s)	Description
Functional Coverage	`x = 1, y = 1` (true), `x = -1, y = 1` (false)	Ensure all functional behaviors are tested (inputs and outputs).
Statement Coverage	`x = 1, y = 1`, `x = -1, y = 1`	Ensure every line of code is executed.
Branch Coverage	`x = 1, y = 1`, `x = -1, y = 1`	Ensure all branches of the `if` condition are tested.
Conditional Coverage	`x = 1, y = 1`, `x = -1, y = 1`, `x = 1, y = -1`, `x = -1, y = -1`	Ensure each individual condition is tested for both true and false outcomes.

9. Write the positive and negative testcases for an ATM machine.

1. Test Case: Valid Card Insertion

- **Positive Test Case**:
  - **Test Scenario**: Insert a valid ATM card.
  - **Steps**:
    1. Insert a valid ATM card with a correct PIN.
    2. Ensure that the machine reads the card.
    3. The ATM prompts for PIN entry.
  - **Expected Result**: The card is read successfully, and the system prompts the user for PIN entry.
- **Negative Test Case**:
  - **Test Scenario**: Insert an expired ATM card.
  - **Steps**:
    1. Insert an expired ATM card.
    2. Wait for the system response.
  - **Expected Result**: The ATM rejects the expired card and displays an error message, such as "Card expired."

2. Test Case: Correct PIN Entry

RTPNOTES.vercel.app

- **Positive Test Case**:
  - **Test Scenario**: Enter a valid PIN after inserting a valid card.
  - **Steps**:
    1. Insert a valid ATM card.
    2. Enter the correct PIN.
    3. Wait for the ATM to authenticate the PIN.
  - **Expected Result**: The ATM authenticates the PIN, and the user is granted access to their account options.
- **Negative Test Case**:
  - **Test Scenario**: Enter an incorrect PIN multiple times.
  - **Steps**:
    1. Insert a valid ATM card.
    2. Enter an incorrect PIN (e.g., 3 times).
    3. Wait for the system response.
  - **Expected Result**: The ATM locks the account after a predefined number of incorrect PIN attempts and displays an error message like "Account locked due to multiple failed attempts."

3. Test Case: Check Account Balance

- **Positive Test Case**:
  - **Test Scenario**: Check the balance of a valid account.
  - **Steps**:
    1. Insert a valid ATM card and enter the correct PIN.
    2. Select the "Check Balance" option.
    3. Wait for the ATM to display the balance.
  - **Expected Result**: The ATM displays the correct account balance.
- **Negative Test Case**:
  - **Test Scenario**: Check balance with insufficient funds (if the balance is zero or negative).
  - **Steps**:
    1. Insert a valid ATM card and enter the correct PIN.
    2. Select the "Check Balance" option for an account with no funds.

RTPNOTES.vercel.app

- **Expected Result**: The ATM displays a message like "Insufficient funds" or "Zero balance.

4. Test Case: Withdraw Cash

- **Positive Test Case**:
  - **Test Scenario**: Withdraw a valid amount of cash from an account with sufficient balance.
  - **Steps**:
    1. Insert a valid ATM card and enter the correct PIN.
    2. Select the "Withdraw" option and enter the desired amount (e.g., $100).
    3. Wait for the ATM to dispense the cash
  - **Expected Result**: The ATM dispenses the correct amount of cash, and the balance is updated accordingly.
- **Negative Test Case**:
  - **Test Scenario**: Attempt to withdraw an amount exceeding the available balance.
  - **Steps**:
    1. Insert a valid ATM card and enter the correct PIN.
    2. Select the "Withdraw" option and enter an amount larger than the available balance (e.g., $500 when the balance is $200).
  - **Expected Result**: The ATM displays an error message like "Insufficient funds" and does not dispense any cash.

5. Test Case: Transfer Money

- **Positive Test Case**:
  - **Test Scenario**: Transfer money between two valid accounts.
  - **Steps**:
    1. Insert a valid ATM card and enter the correct PIN.
    2. Select the "Transfer" option.
    3. Enter the recipient account number, amount to transfer (e.g., $50), and confirm the details.
    4. Wait for the ATM to process the transaction.
  - **Expected Result**: The ATM successfully transfers the money, and both accounts are updated accordingly.

RTPNOTES.vercel.app

- **Negative Test Case**:
  - **Test Scenario**: Attempt to transfer money with an invalid account number.
  - **Steps**:
    1. Insert a valid ATM card and enter the correct PIN.
    2. Select the "Transfer" option.
    3. Enter an invalid recipient account number or one that does not exist.
  - **Expected Result**: The ATM displays an error message like "Invalid account number" and does not process the transfer.

6. Test Case: Print Receipt

- **Positive Test Case**:
  - **Test Scenario**: Request for a receipt after performing a transaction.
  - **Steps**:
    1. Insert a valid ATM card and enter the correct PIN.
    2. Perform a withdrawal or transfer.
    3. Select the option to print a receipt.
  - **Expected Result**: The ATM prints a receipt with the transaction details (e.g., withdrawal amount, balance after transaction, etc.).
- **Negative Test Case**:
  - **Test Scenario**: Attempt to print a receipt when the printer is out of paper or not functioning.
  - **Steps**:
    1. Insert a valid ATM card and enter the correct PIN.
    2. Perform a withdrawal or transfer.
    3. Attempt to print a receipt when the ATM printer is out of paper.
  - **Expected Result**: The ATM displays a message like "Receipt not available due to printer error."

7. Test Case: ATM Machine Out of Service

- **Positive Test Case**:
  - **Test Scenario**: Check for a properly functioning ATM machine.
  - **Steps**:
    1. Insert a valid ATM card.

RTPNOTES.vercel.app

2. Enter the correct PIN.
3. Perform any transaction like balance check or withdrawal.

- **Expected Result**: The ATM performs the transaction without issues.

- **Negative Test Case**:
  - **Test Scenario**: Attempt a transaction when the ATM machine is out of service.
  - **Steps**:
    1. Insert a valid ATM card.
    2. Enter the correct PIN.
    3. Attempt a withdrawal or transfer.
  - **Expected Result**: The ATM displays a message like "ATM is currently out of service" and does not process any transactions.

Test Case Type	Positive Test Case	Negative Test Case
Card Insertion	Valid card inserted	Expired or invalid card inserted
PIN Entry	Correct PIN entered	Incorrect PIN entered multiple times
Balance Check	Balance checked for a valid account	Checking balance on an account with zero balance
Cash Withdrawal	Valid amount withdrawn	Attempt to withdraw more than available balance
Money Transfer	Successful transfer between valid accounts	Invalid account number for transfer
Receipt	Receipt printed after transaction	Printer error or no paper for receipt
ATM Status	ATM machine operational	ATM out of service
Network Connectivity	Good network connectivity for transaction	Network error preventing transaction completion

10. Explain the five levels of testing goals based on test process maturity

The **Test Maturity Model (TMM)** helps organizations assess and improve their software testing processes. It consists of **five levels**, each focusing on a different stage of testing maturity. These levels help identify how advanced the testing process is and guide improvements to make the testing more effective and efficient.

RTPNOTES.vercel.app

Level 1: Initialization

- **What it means**: At this level, testing is unstructured and informal.
- **Key points**:
  - No formal testing processes are in place.
  - Testing is done **ad-hoc** (randomly without any planned process).
  - There is no formal quality check before releasing the software.
- **Example**: Imagine testing the software randomly without planning or following any process, just to make sure it works at a basic level.

Level 2: Definition

- **What it means**: This level introduces more structure and planning into the testing process.
- **Key points**:
  - **Requirements** are clearly defined.
  - **Test strategies, test plans, and test cases** are created.
  - Test cases are executed based on requirements to check the software.
- **Example**: Before starting testing, a detailed plan is created about what to test, how to test it, and what results are expected.

Level 3: Integration

- **What it means**: At this stage, testing is integrated with the software development process.
- **Key points**:
  - Testing procedures are **integrated with the software development lifecycle (SDLC)**.
  - Testing is done independently **after the development phase**.
  - The main goal is to **identify and manage risks**.
- **Example**: Testing is done after each development cycle, and any issues found during development are addressed in testing to prevent them from affecting the software.

Level 4: Measurement and Management

- **What it means**: This level focuses on managing and measuring the effectiveness of the testing process.
- **Key points**:
  - Testing becomes a part of the **entire software lifecycle**.

RTPNOTES.vercel.app

    - Reviews of **requirements, design documents, and code** are done regularly.
    - **Unit testing** and **integration testing** are performed as part of coding.
    - Testing activities are **measured** and tracked.
- **Example**: You measure how effective the testing process is, track progress, and make decisions based on data. For example, you might track how many bugs are found or how long testing takes.

Level 5: Optimization

- **What it means**: This is the highest level, where testing processes are continuously improved and optimized.
- **Key points**:
    - Testing processes are **optimized** for better efficiency.
    - Measures are taken to **prevent defects** in the future.
    - The focus is on **improving testing** by using advanced tools and techniques.
- **Example**: You might start automating tests, use better tools, and implement practices to prevent defects from happening again. The goal is to make the testing process as efficient as possible.

11. Explain the different types of system testing

1. Functional Testing

- **What it is**: Functional testing checks if the system's features work correctly, just like they were designed to.
- **Example**: If you have a login page, functional testing will check if entering the correct username and password allows you to log in.

2. Performance Testing

- **What it is**: Performance testing measures how well the system works under different conditions. This could include checking how fast it runs or how much traffic it can handle.
- **Example**: If many people are using an online store at once, performance testing checks if the site still loads quickly or if it crashes.

RTPNOTES.vercel.app

3. Security Testing

- **What it is**: Security testing ensures the system is safe from threats. It checks if sensitive data (like passwords or payment info) is protected from unauthorized access or attacks.
- **Example**: Security testing checks if hackers can easily break into the system or access confidential information.

4. Compatibility Testing

- **What it is**: Compatibility testing checks if the system works well on different devices, operating systems, browsers, or networks.
- **Example**: The app should work the same way on both Android and iOS phones, and it should run well on Chrome, Firefox, or Safari browsers.

5. Usability Testing

- **What it is**: Usability testing checks how easy and user-friendly the system is for people to use.
- **Example**: If you're using a mobile app, usability testing ensures that it's easy to navigate, understand, and complete tasks (like placing an order).

6. Regression Testing

- **What it is**: Regression testing ensures that new changes, features, or bug fixes don't break any existing features of the system.
- **Example**: After adding a new payment feature to an app, regression testing checks if the existing features, like login or account settings, still work.

7. Acceptance Testing

- **What it is**: Acceptance testing checks if the system meets the business and customer requirements before it's released for use.
- **Example**: Before launching an e-commerce website, acceptance testing ensures that it meets all the customer's requirements, like smooth checkout, product display, and payment integration.

RTPNOTES.vercel.app

12. Design six test cases for a program which checks whether a number between 1 and 100 is prime or not

Test Case 1: Test a Prime Number (Basic Prime)

- **Input**: 7
- **Expected Output**: Prime
- **Reason**: 7 is a prime number because it is only divisible by 1 and itself.

Test Case 2: Test a Non-Prime Number (Composite Number)

- **Input**: 9
- **Expected Output**: Not Prime
- **Reason**: 9 is a composite number because it is divisible by 1, 3, and 9.

Test Case 3: Test the Number 1 (Edge Case)

- **Input**: 1
- **Expected Output**: Not Prime
- **Reason**: 1 is not considered a prime number as prime numbers must be greater than 1.

Test Case 4: Test the Number 100 (Upper Bound)

- **Input**: 100
- **Expected Output**: Not Prime
- **Reason**: 100 is divisible by multiple numbers (1, 2, 4, 5, 10, 20, 25, 50, 100), so it's not a prime number.

Test Case 5: Test a Larger Prime Number

- **Input**: 97
- **Expected Output**: Prime
- **Reason**: 97 is a prime number because it has no divisors other than 1 and itself.

Test Case 6: Test a Negative Number (Invalid Input)

- **Input**: -5
- **Expected Output**: Invalid input

RTPNOTES.vercel.app

- **Reason**: Negative numbers cannot be prime, as prime numbers are defined only for positive integers greater than 1.