Software Quality

(Preparation for block sessions)

1# Block Session – 1:

Model Based Testing

# Definition: Model-based testing is an approach for generating test artefacts based on models.

Key objectives of software testing:

- Evaluate artifacts: Assess quality of requirements, user stories, design, and code.
- Verify requirements: Ensure all specified requirements are met.
- Validate completeness: Confirm the software works as expected by users and stakeholders.
- Build confidence: Demonstrate the quality level of the software.
- Detect issues: Find errors and deficiencies in the software.
- Reduce risks: Minimize the chance of undetected errors in production.
- Ensure compliance: Demonstrate adherence to contractual, legal or regulatory standards.

# What is ISTQB?

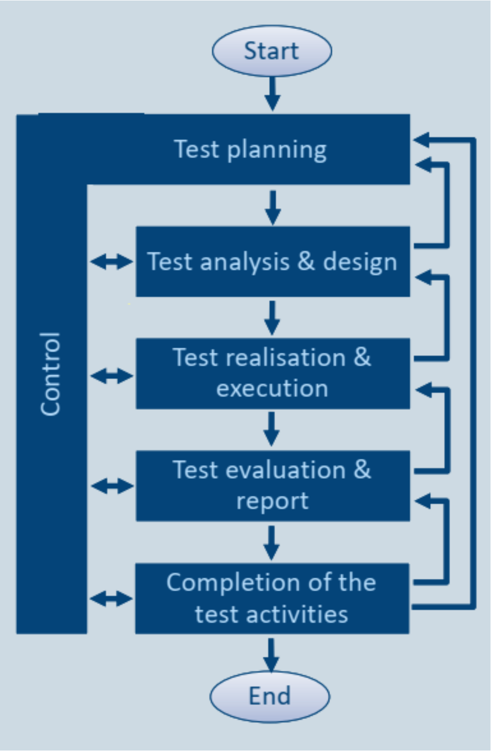
ISTQB (International Software Testing Qualifications Board) is a global, non-profit organization that sets the standard for software testing certifications. Its mission is to advance the profession by:

- **Developing globally recognized certifications:** Ensuring consistent competency among testers.
- **Promoting the value of software testing:** Emphasizing its importance for quality software.
- **Supporting professional development:** Offering resources and networking opportunities.
- **Facilitating collaboration:** Connecting testers worldwide through events and forums.

The fundamental test process according to ISTQB (International Software Testing Qualifications Board) consists of the following key activities:

1.Test Planning:

● Define the test objectives, scope, and approach.
● Identify the test items and prioritize them.
● Determine the test environment requirements.
● Prepare the test schedule and estimate the required resources.
● Define the test deliverables and exit criteria.
● Example: In planning the testing of a web application, the test plan may include defining the scope of the testing, such as testing the login functionality, user registration, and profile management. The plan would also specify the approach, resources required (e.g., test environment, tools), and the schedule of the testing activities.



2.Test Analysis & Design:

- Identify and specify the test conditions based on the requirements and specifications.
- Design test cases and test scenarios that cover the identified test conditions.
- Define the test data and test environment setup required for executing the test cases.
- Determine the expected results for each test case.
- Select and prioritize the test cases to be executed.
- Example: For a mobile banking application, during the analysis and design phase, the tester identifies test conditions like account creation, funds transfer, and balance inquiry. Based on these conditions, test cases are designed to cover various scenarios, such as valid and invalid inputs, boundary values, and error handling.

3.Test Realization & Execution:

- Implement the test cases and execute them according to the defined test procedures.
- Record the actual results and compare them with the expected results.
- Log any defects or issues encountered during the test execution.
- Retest the fixed defects and validate the corrections.
- Conduct regression testing to ensure that changes or fixes did not introduce new defects.
- Monitor and control the test execution progress.
- Example: In executing the test cases for an e-commerce website, the tester runs the test cases to validate different functionalities like adding items to the cart, proceeding to checkout, and making a payment. The tester records the actual results, compares them with the expected results, and logs any defects encountered during the execution.

4.Test Evaluation & Reporting:

- Evaluate the test results and compare them against the defined exit criteria.
- Analyze the defects and their impact on the system.
- Prepare test summary reports and defect reports.
- Provide feedback to the stakeholders on the quality of the system under test.
- Make recommendations for further actions, such as additional testing or system improvements.
- Example: After conducting performance testing for a web application, the tester evaluates the test results and compares them against the defined performance criteria, such as response time and resource utilization. The tester prepares a test summary report, highlighting the performance issues identified and their impact on the system.

5.Completion of Test Activities:

- Review and finalize the test documentation, including test cases, test scripts, and test data.

● Gather lessons learned from the testing process and document them for future projects.
● Archive the test artifacts and ensure their traceability for future reference.
● Conduct a test closure meeting to discuss the overall testing process and outcomes.

● Example: At the end of the testing process, the tester reviews and finalizes the test documentation, ensuring that all test cases, scripts, and data are properly documented. Lessons learned from the testing process, such as challenges faced and best practices, are documented for future reference. The test artifacts, including test plans and reports, are archived for future audits or reference.

These activities are iterative and can overlap throughout the software development lifecycle, depending on the chosen development methodology. The test process aims to ensure that the software meets the specified requirements, functions as expected, and is of high quality.

# Software Quality According to ISO 9126 / 25000

ISO standards provide frameworks for evaluating software quality:

- **ISO 9126**: Focuses on product quality with six key characteristics:
    - **Functionality**: Does the software meet user needs?
        - **Example**: A finance application correctly calculates interest and displays accurate account balances.
    - **Reliability**: Does the software perform under specified conditions?
        - **Example**: An online banking system remains operational and responsive even under peak load conditions.
    - **Usability**: Is the software easy to use?
        - **Example**: A mobile app has an intuitive interface that users can navigate without a manual.
    - **Efficiency**: Does the software use resources optimally?
        - **Example**: A photo editing tool processes images quickly without consuming excessive memory.
    - **Maintainability**: Can the software be easily modified?
        - **Example**: Modular code structure allows developers to update individual components without affecting the whole system.
    - **Portability**: Can the software operate in different environments?
        - **Example**: A web application runs seamlessly across various browsers and operating systems.
- **ISO/IEC 25000**: Extends ISO 9126 with a more detailed set of standards covering product quality, data quality, and quality in use.
    - **Example**: A comprehensive quality model evaluates a CRM system's functionality, performance, security, and user satisfaction.

→ Functional Quality Characteristics:

1. Accuracy

- Ensures the software delivers correct or agreed-upon results

- Critical for reliability and user trust

2. Suitability

- Measures how well the software fulfills specified tasks
- Ensures the product meets user needs and expectations

3. Security

- Focuses on preventing unauthorized access to programs and data
- Essential for protecting sensitive information and maintaining user trust

4. Interoperability

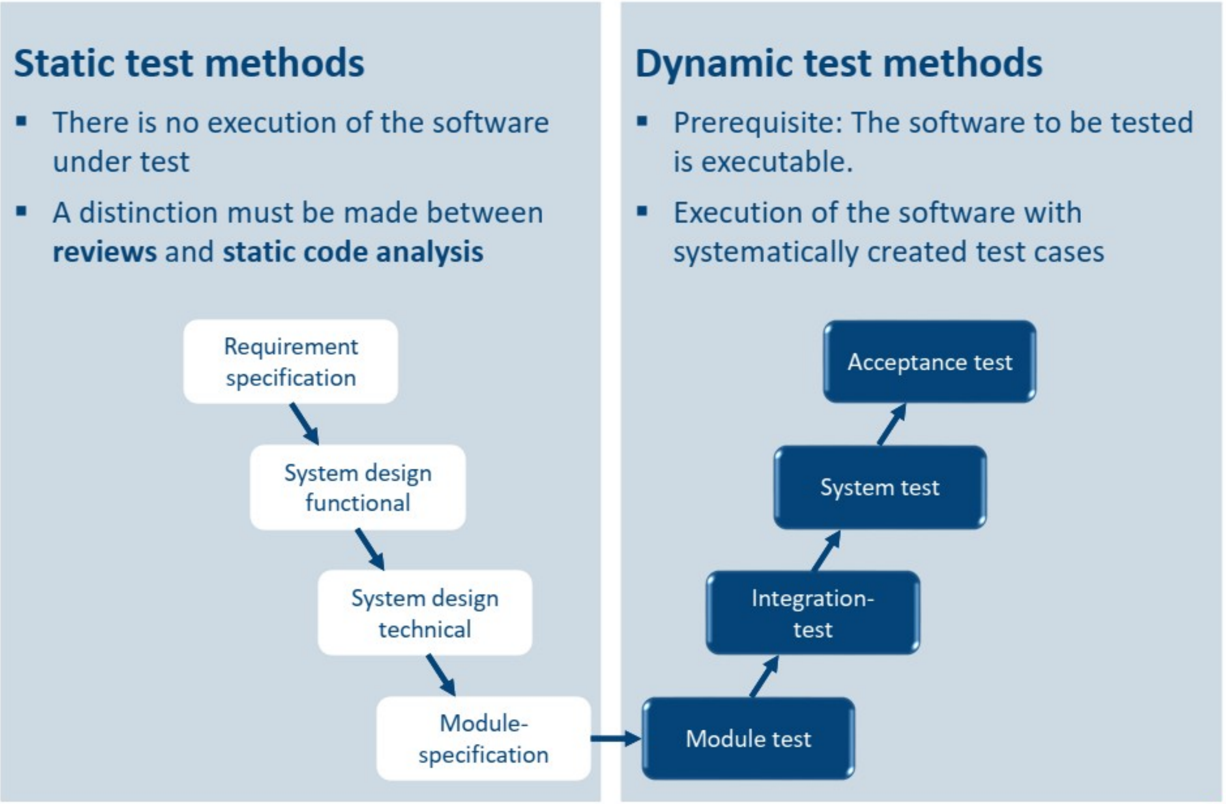
- Assesses the software's ability to interact with other systems
- Important for integration in complex IT environments

5. Compliance

- Ensure compliance with relevant standards, conventions or legal requirements
- Critical for regulatory compliance and industry acceptance

These characteristics are typically verified through functional testing, which aims to assess the software's behavior against specified requirements and expected outcomes.

# Functional testing and software life cycle:



Static test methods

- There is no execution of the software under test
- A distinction must be made between **reviews** and **static code analysis**

Requirement specification

System design functional

System design technical

Module-specification

Dynamic test methods

- Prerequisite: The software to be tested is executable.
- Execution of the software with systematically created test cases

Module test

Integration-test

System test

Acceptance test

# Design Techniques for dynamic testing:

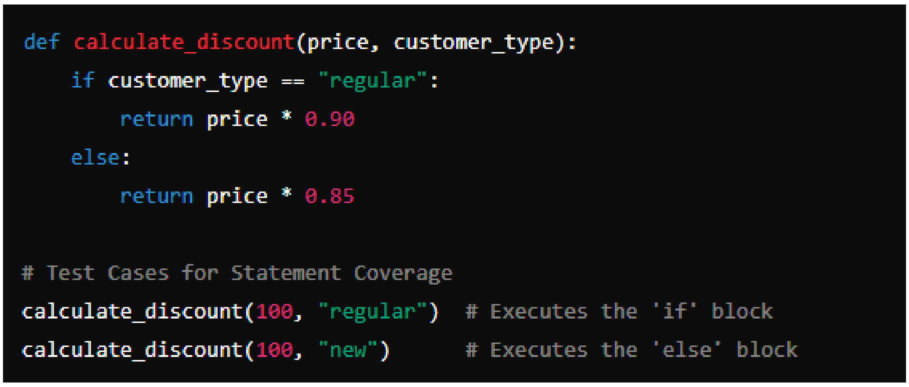
Design techniques for dynamic tests are crucial for ensuring that the software behaves as expected under various conditions. These techniques can be broadly categorized into two main approaches: structure-oriented (white-box) techniques and specification-based (black-box) techniques.

1. Structure-Oriented (White-Box) Techniques

Definition: Structure-oriented techniques focus on the internal structure of the software. These techniques involve testing the internal paths, logic, and structures of the code.

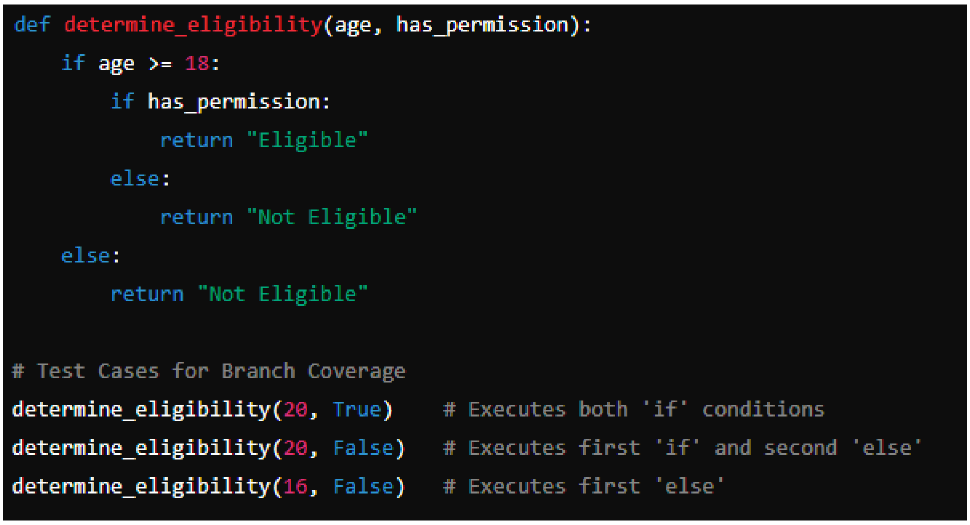
Types and Examples:

- **Statement Coverage**:
  - **Definition**: Ensures that every statement in the code is executed at least once.
  - **Example**: If a function contains an `if-else` statement, tests are designed to ensure both the `if` and the `else` blocks are executed.



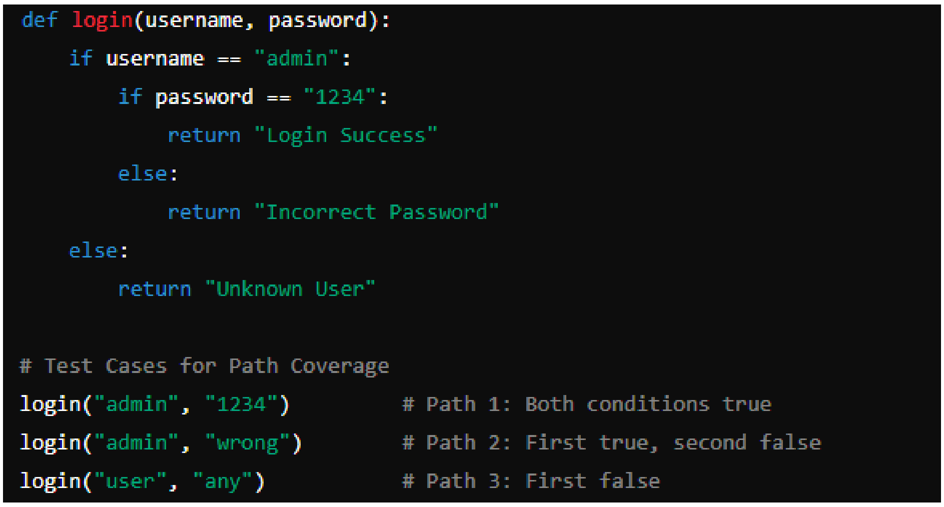
```
def calculate_discount(price, customer_type):
    if customer_type == "regular":
        return price * 0.90
    else:
        return price * 0.85

# Test Cases for Statement Coverage
calculate_discount(100, "regular")  # Executes the 'if' block
calculate_discount(100, "new")      # Executes the 'else' block
```

- **Branch Coverage**:
  - **Definition**: Ensures that each possible branch (true/false) of every decision point is executed.
  - **Example**: For a function with multiple conditional statements, tests cover all branches.



```
def determine_eligibility(age, has_permission):
    if age >= 18:
        if has_permission:
            return "Eligible"
        else:
            return "Not Eligible"
    else:
        return "Not Eligible"

# Test Cases for Branch Coverage
determine_eligibility(20, True)    # Executes both 'if' conditions
determine_eligibility(20, False)   # Executes first 'if' and second 'else'
determine_eligibility(16, False)   # Executes first 'else'
```

- **Path Coverage**:
  - **Definition**: Ensures that all possible paths through the code are executed.
  - **Example**: For nested conditional statements, tests are designed to cover all paths from entry to exit.



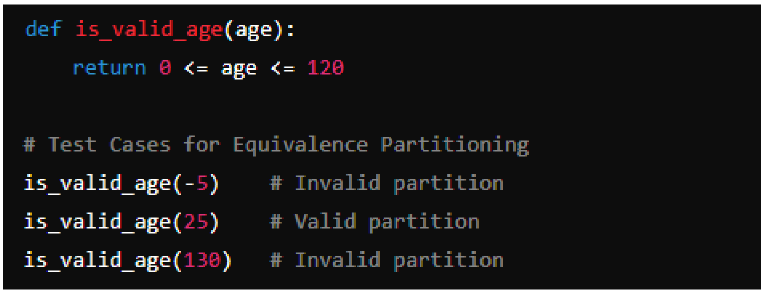
```python
def login(username, password):
    if username == "admin":
        if password == "1234":
            return "Login Success"
        else:
            return "Incorrect Password"
    else:
        return "Unknown User"

# Test Cases for Path Coverage
login("admin", "1234")         # Path 1: Both conditions true
login("admin", "wrong")        # Path 2: First true, second false
login("user", "any")           # Path 3: First false
```

2. Specification-Based (Black-Box) Techniques

Definition: Specification-based techniques focus on the external behavior of the software, ignoring internal structures. These techniques involve designing tests based on the specifications and requirements of the software.

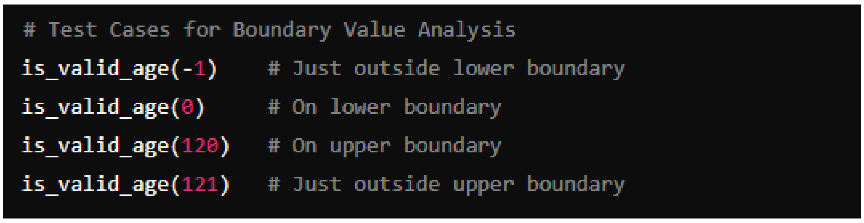
Types and Examples:

- **Equivalence Partitioning**:
  - **Definition**: Divides input data into partitions of equivalent data from which test cases can be derived.
  - **Example**: For a function that accepts ages from 0 to 120, partitions might be <0, 0-120, >120.



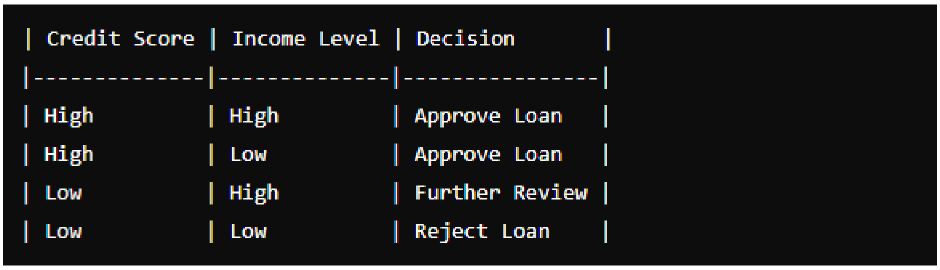
```python
def is_valid_age(age):
    return 0 <= age <= 120

# Test Cases for Equivalence Partitioning
is_valid_age(-5)    # Invalid partition
is_valid_age(25)    # Valid partition
is_valid_age(130)   # Invalid partition
```

- **Boundary Value Analysis**:
  - **Definition**: Focuses on the boundaries between partitions. Tests are designed at the edges of these partitions.
  - **Example**: For the same age function, tests at boundaries like -1, 0, 120, 121.

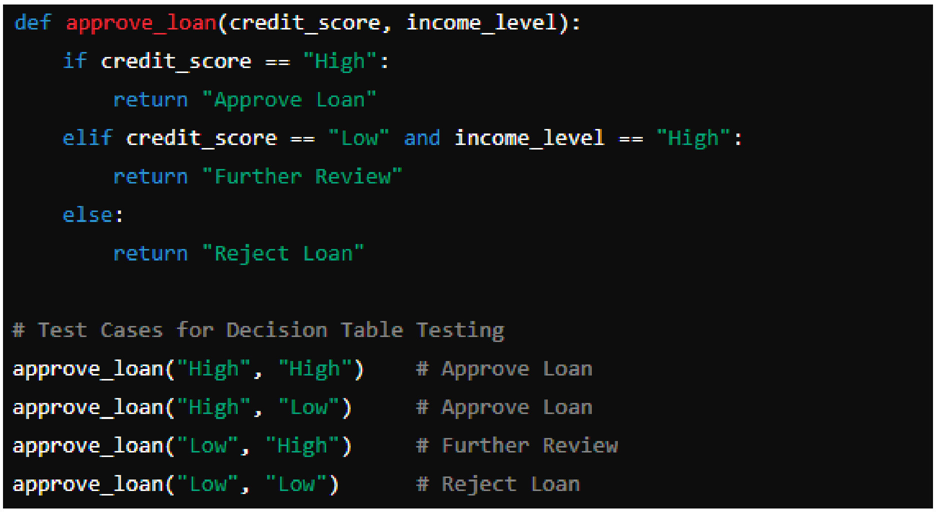


```python
# Test Cases for Boundary Value Analysis
is_valid_age(-1)    # Just outside lower boundary
is_valid_age(0)     # On lower boundary
is_valid_age(120)   # On upper boundary
is_valid_age(121)   # Just outside upper boundary
```

- **Decision Table Testing**:
    - **Definition**: Uses decision tables to represent combinations of inputs and their corresponding outputs.
    - **Example**: For a loan approval system, inputs could be credit score and income, with decisions based on combinations of these inputs.

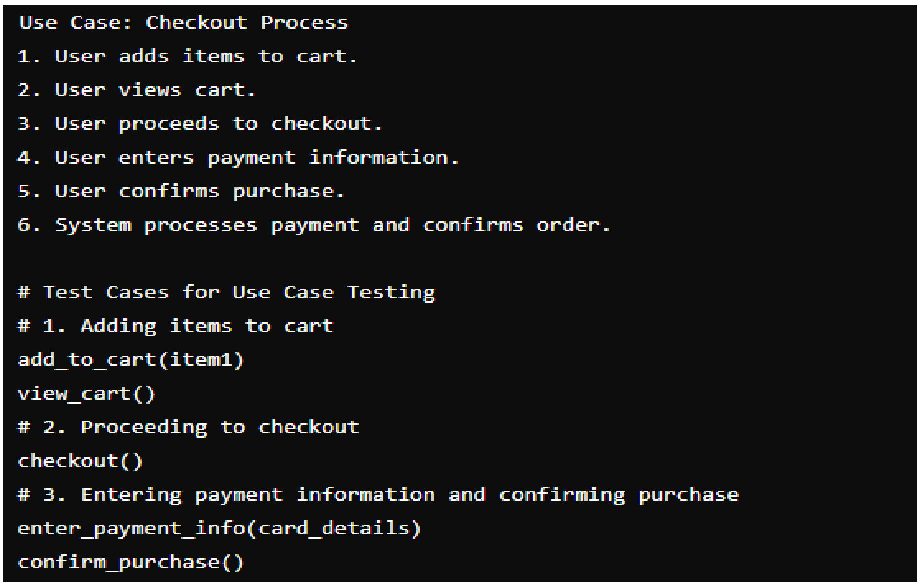


```
| Credit Score | Income Level | Decision       |
|--------------|--------------|----------------|
| High         | High         | Approve Loan   |
| High         | Low          | Approve Loan   |
| Low          | High         | Further Review |
| Low          | Low          | Reject Loan    |
```



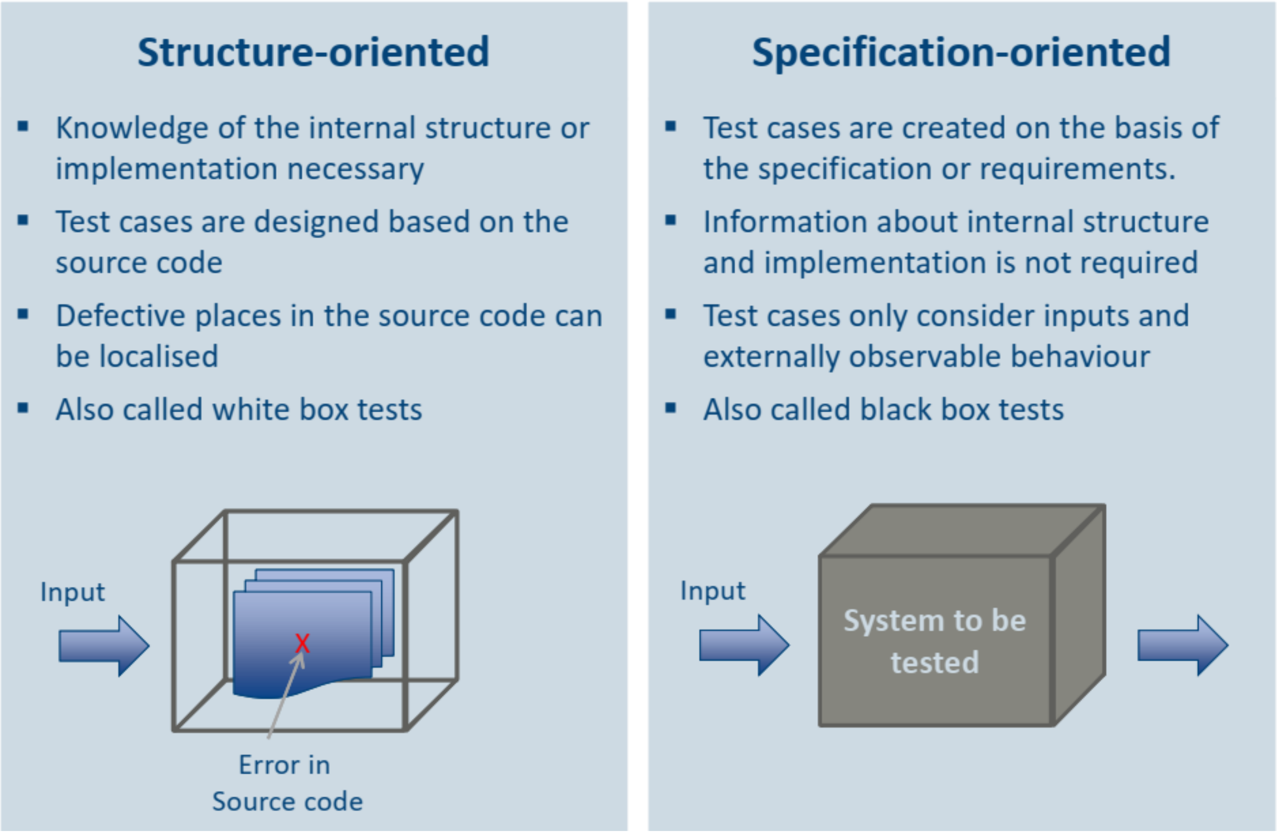
```python
def approve_loan(credit_score, income_level):
    if credit_score == "High":
        return "Approve Loan"
    elif credit_score == "Low" and income_level == "High":
        return "Further Review"
    else:
        return "Reject Loan"

# Test Cases for Decision Table Testing
approve_loan("High", "High")    # Approve Loan
approve_loan("High", "Low")     # Approve Loan
approve_loan("Low", "High")     # Further Review
approve_loan("Low", "Low")      # Reject Loan
```

- **Use Case Testing**:
    - **Definition**: Involves creating test cases based on use cases, which describe interactions between the user and the system.
    - **Example**: For an online shopping system, a use case might describe the process of adding items to a cart and checking out.



```
Use Case: Checkout Process
1. User adds items to cart.
2. User views cart.
3. User proceeds to checkout.
4. User enters payment information.
5. User confirms purchase.
6. System processes payment and confirms order.

# Test Cases for Use Case Testing
# 1. Adding items to cart
add_to_cart(item1)
view_cart()
# 2. Proceeding to checkout
checkout()
# 3. Entering payment information and confirming purchase
enter_payment_info(card_details)
confirm_purchase()
```

# Basic difference between structure oriented (white box testing) and specification based testing (black box testing):



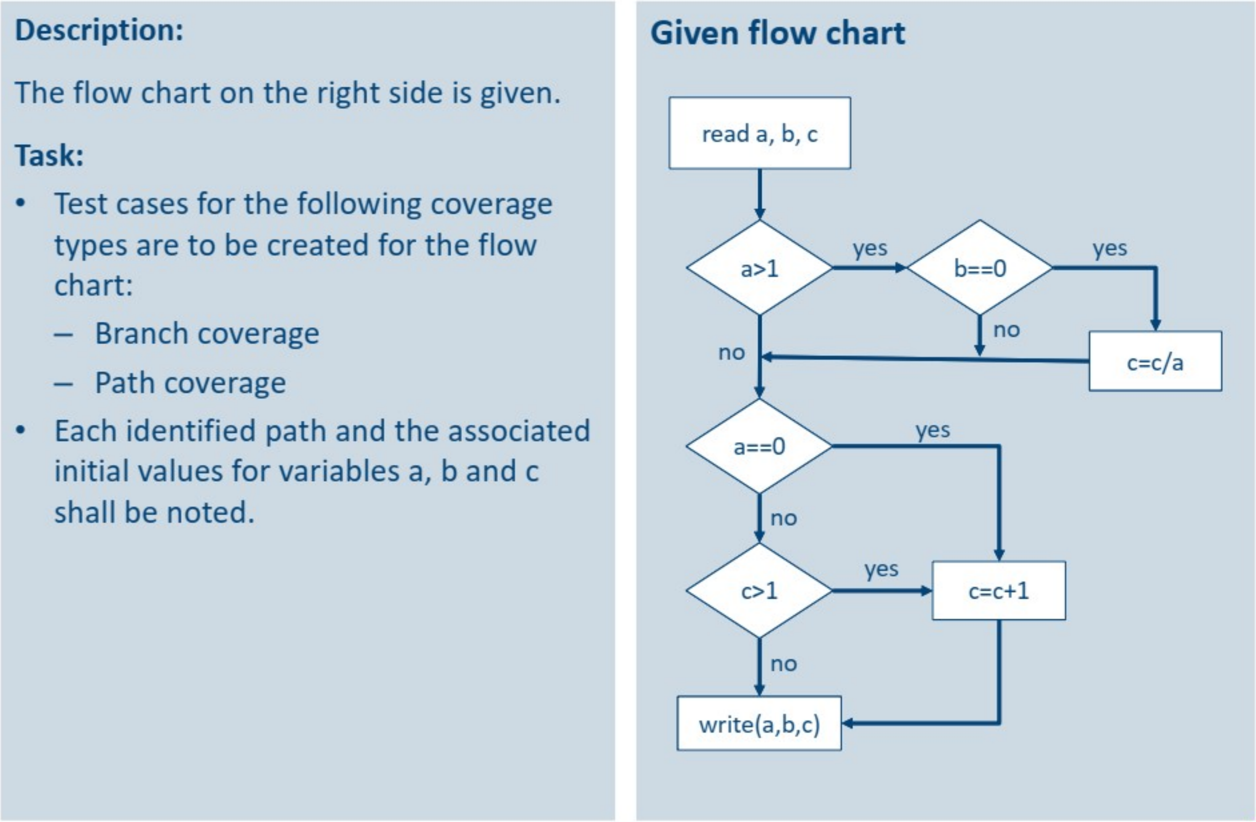
Structure-oriented

- Knowledge of the internal structure or implementation necessary
- Test cases are designed based on the source code
- Defective places in the source code can be localised
- Also called white box tests

Specification-oriented

- Test cases are created on the basis of the specification or requirements.
- Information about internal structure and implementation is not required
- Test cases only consider inputs and externally observable behaviour
- Also called black box tests

# Exercise for white box testing:



Description:

The flow chart on the right side is given.

Task:

- Test cases for the following coverage types are to be created for the flow chart:
  - Branch coverage
  - Path coverage
- Each identified path and the associated initial values for variables a, b and c shall be noted.

Given flow chart

The test cases for branch and path coverage:

1. **Test Case 1**: `a = 2, b = 0, c = 2`
   - Branches: `a > 1` (true), `b == 0` (true)
   - Path: `a > 1` (true) -> `b == 0` (true) -> `write(a, b, c)`
2. **Test Case 2**: `a = 2, b = 1, c = 2`
   - Branches: `a > 1` (true), `b == 0` (false)
   - Path: `a > 1` (true) -> `b == 0` (false) -> `write(a, b, c)`
3. **Test Case 3**: `a = 0, b = 1, c = 2`
   - Branches: `a > 1` (false), `a == 0` (true)
   - Path: `a > 1` (false) -> `a == 0` (true) -> `write(a, b, c)`
4. **Test Case 4**: `a = -1, b = 1, c = 2`
   - Branches: `a > 1` (false), `a == 0` (false), `c > 1` (true)
   - Path: `a > 1` (false) -> `a == 0` (false) -> `c > 1` (true) -> `write(a, b, c)`
5. **Test Case 5**: `a = -1, b = 1, c = 1`
   - Branches: `a > 1` (false), `a == 0` (false), `c > 1` (false)
   - Path: `a > 1` (false) -> `a == 0` (false) -> `c > 1` (false) -> `write(a, b, c)`

By executing these test cases, you will achieve both branch coverage and path coverage for the given flowchart.

# Model-Based Testing (MBT) Overview:

Model-Based Testing (MBT) uses abstract models of a system's desired behavior to automatically generate, execute, and analyze test cases.

What MBT Can Do

1. **Automate Test Case Generation**: Reduces manual effort by generating test cases from models.
2. **Improve Test Coverage**: Systematically covers various aspects of the system.
3. **Detect Inconsistencies Early**: Finds issues in requirements/design early.
4. **Enhance Maintainability**: Models update test cases automatically when the system changes.
5. **Facilitate Regression Testing**: Quickly adapts to system changes for effective regression testing.
6. **Enable Traceability**: Links requirements to test cases for easy impact analysis.

What MBT Cannot Do

1. **Ensure Model Completeness/Accuracy**: Relies on accurate, complete models.
2. **Simplify Complex Model Creation**: Requires expertise and time to create detailed models.
3. **Perform Non-Functional Testing**: Less effective for performance, usability, and security testing.

4. **Overcome Tool Limitations**: Depends on the capabilities of MBT tools.
5. **Adapt Easily to Frequent Changes**: Frequent significant changes can be challenging to model.
6. **Replace Human Insight**: Cannot replicate the intuition and creativity of exploratory testing.

Motivation: Effectiveness and Efficiency

Effectiveness

1. **Comprehensive Testing**: Ensures thorough testing and defect detection.
2. **Early Defect Detection**: Finds issues early, reducing fix costs.
3. **Improved Quality**: Higher quality test cases lead to better system quality.

Efficiency

1. **Reduced Manual Effort**: Automates test case generation and execution.
2. **Faster Regression Testing**: Quickly updates and executes test cases with system changes.
3. **Cost Savings**: Saves time and effort, reducing costs.
4. **Scalability**: Handles large, complex systems efficiently.

In summary, MBT enhances testing effectiveness and efficiency by automating test case generation and ensuring comprehensive coverage, though it requires accurate models and cannot replace human exploratory testing.

# Sequential Process Models
Sequential process models, also known as **linear models**, follow a straightforward, step-by-step approach. Each phase must be completed before the next begins. The most well-known sequential model is the **Waterfall model**.

Waterfall Model

1. **Requirements Analysis**:
    - Gather and document all system requirements.
2. **System Design**:
    - Create architecture and design specifications based on requirements.
3. **Implementation**:
    - Develop the software according to the design specifications.
4. **Integration and Testing**:
    - Integrate modules and test the entire system to find and fix defects.
5. **Deployment**:
    - Deploy the system to the production environment.
6. **Maintenance**:
    - Perform ongoing maintenance to fix any issues and make improvements.

Advantages:

- Simplicity and ease of use.
- Clear milestones and deliverables at each stage.

- Well-suited for projects with well-understood requirements.

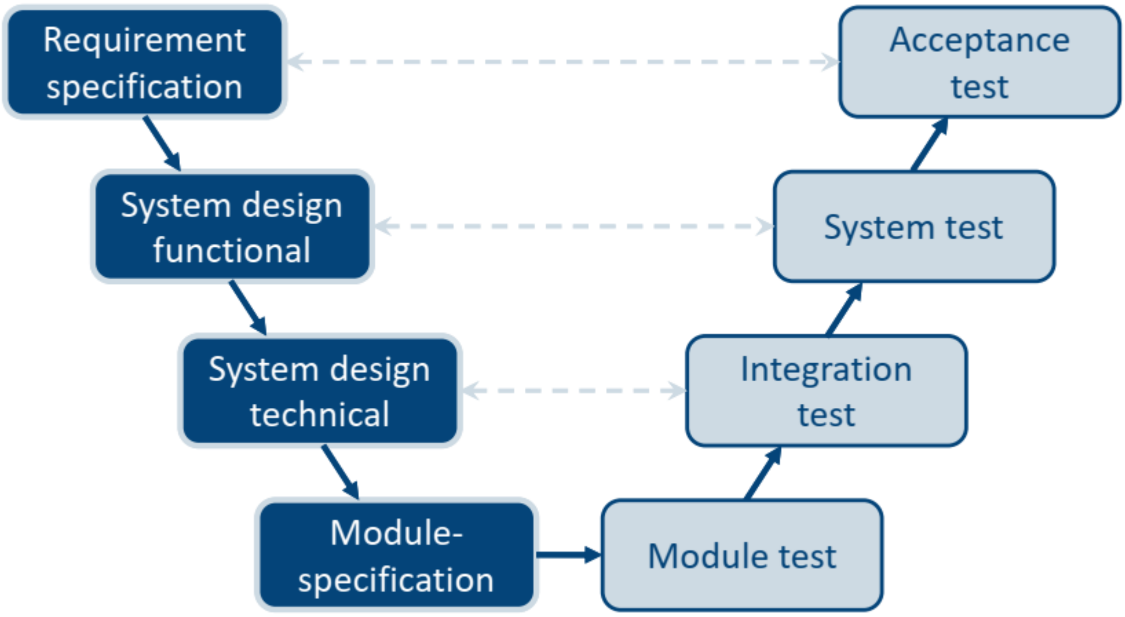
Disadvantages:

- Inflexibility in handling changes once the process has started.
- High risk of discovering defects late in the development process.
- Not suitable for complex or long-term projects where requirements may evolve.

Iterative Process Models

Iterative process models emphasize repetition and refinement. The development process is divided into iterations, each of which results in a working product increment. This allows for regular assessment and incorporation of feedback. Two common iterative models are the **V-Model** and the **Spiral Model**.

V-Model (Verification and Validation Model)



The V-Model is an extension of the Waterfall model that emphasizes verification and validation. Each development phase is associated with a corresponding testing phase.

1. **Requirements Analysis ↔ Acceptance Testing**:
    - Gather requirements and develop acceptance tests to validate them.
2. **System Design ↔ System Testing**:
    - Design the system architecture and plan system tests.
3. **Detailed Design ↔ Integration Testing**:
    - Design system modules and plan integration tests.
4. **Implementation ↔ Unit Testing**:
    - Develop code and perform unit tests on individual components.

Advantages:

- Emphasis on verification and validation improves product quality.
- Early detection of defects through corresponding test plans.
- Clear stages and deliverables, similar to the Waterfall model.

Disadvantages:

- Rigidity and inflexibility, making it less suited for projects with changing requirements.
- High risk if requirements are misunderstood early on.

# Commonalities in Sequential and Iterative Process Models in Relation to Model-Based Testing (MBT):

1. **Formalized Requirements and Design**
    - **Sequential Models**: Requirements and design phases provide inputs for MBT to create detailed test models.
    - **Iterative Models**: Continuous refinement of models with evolving requirements in each iteration.
2. **Systematic Test Case Generation**
    - **Sequential Models**: MBT ensures comprehensive test case generation from static requirements and designs.
    - **Iterative Models**: MBT adapts and updates test cases for each iteration, maintaining alignment with changes.
3. **Enhanced Test Coverage and Efficiency**
    - **Sequential Models**: Structured testing and coverage metrics from MBT ensure thorough validation.
    - **Iterative Models**: Adaptive testing and efficient regression testing with MBT ensure continuous coverage and quality.
4. **Traceability and Documentation**
    - **Sequential Models**: MBT provides clear traceability from requirements to test cases.
    - **Iterative Models**: MBT maintains traceability and version control across iterations.
5. **Early Defect Detection and Risk Management**
    - **Sequential Models**: MBT enables early defect detection through early test case generation.
    - **Iterative Models**: MBT supports risk management by focusing on high-risk areas early and providing continuous feedback.

Summary

MBT enhances both sequential and iterative models by:

- Providing systematic and automated test case generation.
- Ensuring comprehensive test coverage and efficiency.
- Maintaining traceability and thorough documentation.
- Facilitating early defect detection and effective risk management.

# Modeling Languages: Key Aspects

A modeling language defines the structure and behavior of models through its semantics and syntax, allowing for clear and consistent representation of system elements. Modelling languages can be used to represent various aspects of a system, including structural aspects, behavior, and domain-specific artifacts.

1. Definition of a Modeling Language

Semantics:

- **Meaning**: Semantics refer to the meaning of the constructs within the modelling language. It defines what each element in the model represents and ensures consistency in interpretation.
- **Examples**:
    - In UML, a "Class" represents a blueprint for objects.
    - In BPMN (Business Process Model and Notation), an "Event" represents something that happens during a process.

Syntax:

- **Rules**: Syntax refers to the rules and grammar by which models are created. It defines how elements can be combined and related.
- **Examples**:
    - In UML, a "Class Diagram" must have classes connected by relationships like associations, generalizations, or dependencies.
    - In BPMN, a "Sequence Flow" connects activities to show the order of execution.

2. Usage of Modeling Languages

Modeling languages can be applied to various aspects of systems, including structural aspects, behavior, and domain-specific artefacts.

a) Structural Aspects:

- **Purpose**: Represent the static architecture of a system, detailing components and their relationships.
- **Examples**:
    - **UML Class Diagrams**: Show the static structure of the system, classes, their attributes, operations, and relationships.
    - **ER Diagrams (Entity-Relationship Diagrams)**: Represent the data model and relationships between entities in a database.

b) Behaviour:

- **Purpose**: Represent the dynamic aspects of the system, including how it behaves in response to various inputs or events.
- **Examples**:
    - **UML State Diagrams**: Depict the states an object can be in and the transitions between these states.
    - **BPMN**: Illustrates business processes and workflows, showing how tasks are sequenced and performed.

    - **UML Sequence Diagrams**: Show how objects interact in a particular sequence of events.

c) Domain-Specific Artefacts:

- **Purpose**: Represent aspects specific to a particular domain or industry, tailored to meet specialized needs.
- **Examples**:
    - **SysML (Systems Modeling Language)**: Tailored for systems engineering, it supports the specification, analysis, design, and verification of complex systems.
    - **DSLs (Domain-Specific Languages)**: Custom languages designed for specific domains, such as SQL for database queries or VHDL for hardware description.

# Types of Model-Based Testing (MBT) Models

Model-Based Testing (MBT) employs different types of models to represent various aspects of the system under test and its environment.

1. System Model

Definition:

- A system model represents the structure, behavior, and interactions within the system under test (SUT). It provides a formal representation of the system's components, their relationships, and how they operate.

Components:

- **Structural Aspects**: Elements such as classes, objects, and components, and their relationships (e.g., class diagrams, component diagrams).
- **Behavioral Aspects**: Dynamic behavior such as interactions, state changes, and activities (e.g., sequence diagrams, state machines).

Purpose:

- To describe the system's functionality and architecture.
- To serve as a basis for generating test cases that cover different scenarios and interactions within the system.

Example:

- **UML Class Diagram**: Represents the classes involved in a banking application, such as Account, Transaction, and User, and their relationships.
- **UML Sequence Diagram**: Illustrates the interaction between a user and the banking system during a funds transfer, showing the sequence of messages exchanged.

2. Environmental Model

Definition:

- An environmental model describes the external conditions, interactions, and entities that interact with the system under test. It encompasses everything outside the system that can affect its operation.

Components:

- **External Actors**: Entities interacting with the system, such as users, external systems, and hardware.
- **Environmental Conditions**: External conditions and events that impact the system's behavior (e.g., network conditions, user inputs).

Purpose:

- To simulate real-world conditions and interactions that the system will encounter.
- To ensure that the system behaves correctly in different environmental scenarios and conditions.

Example:

- **Use Case Diagram**: Depicts external actors such as customers, administrators, and third-party services interacting with the banking application.
- **Activity Diagram**: Represents the flow of actions and decisions taken by an external user during the registration process, including the environmental conditions like network delays.

3. Test Model

Definition:

- A test model defines the test cases, test data, and test scenarios derived from the system and environmental models. It is used to automate and execute tests systematically.

Components:

- **Test Cases**: Specific conditions under which a test is executed, including inputs, expected outputs, and execution steps.
- **Test Data**: Data required to execute the test cases, including valid and invalid inputs.
- **Test Scenarios**: Comprehensive scenarios that combine multiple test cases to validate complex interactions and workflows.

Purpose:

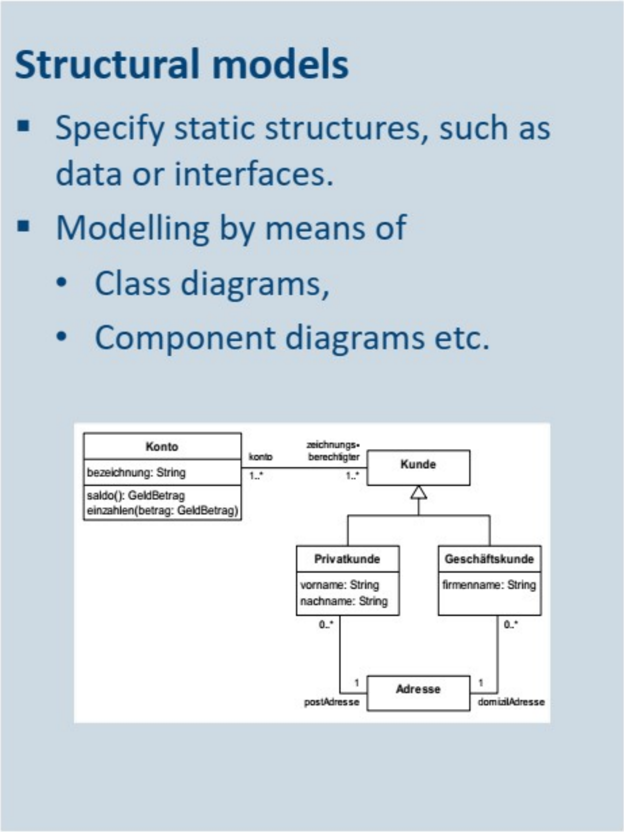
- To automate the generation and execution of test cases.
- To validate that the system meets its requirements and behaves correctly under various conditions.

Example:

- **Test Case Specification**: A document that outlines specific test cases for the banking application, such as validating successful registration and error handling for duplicate usernames.
- **Test Data**: A dataset containing sample user information, including valid and invalid usernames, passwords, and contact details, used for testing the registration process.
- **Test Script**: An automated script that performs the registration process, inputs the test data, and verifies the system's responses against expected outcomes.

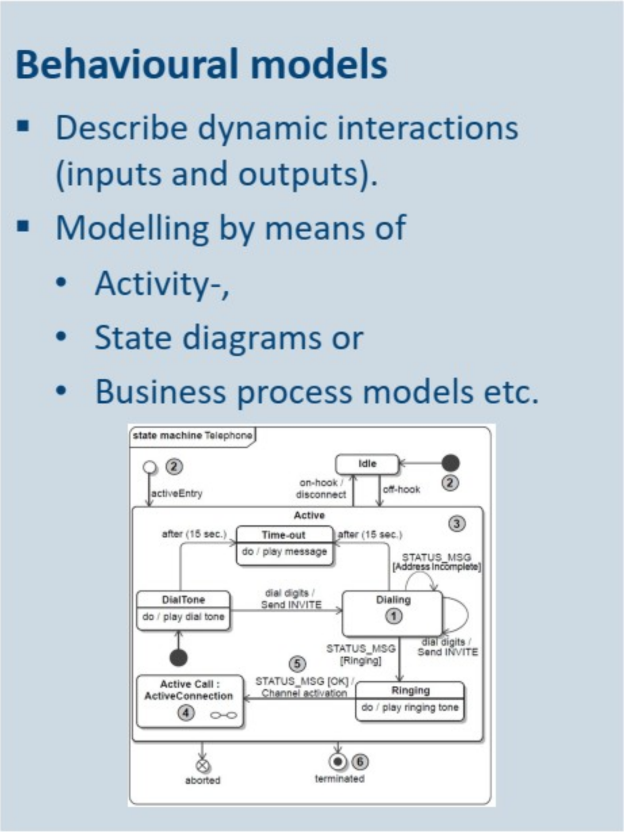
Main Focus of MBT Models:

1. **Functional Validation**
   - **Purpose**: Ensure the system meets its functional requirements.
   - **Example**: Validating user registration and transaction processing in an online banking system.
2. **Behavioral Verification**
   - **Purpose**: Verify the dynamic behavior of the system.
   - **Example**: Checking state transitions like login, session timeout, and logout.
3. **Structural Analysis**
   - **Purpose**: Analyze the system's architecture and component interactions.
   - **Example**: Validating relationships between User, Account, and Transaction entities.
4. **Environment Simulation**
   - **Purpose**: Simulate external conditions and interactions affecting the system.
   - **Example**: Testing the system under network latency and user load conditions.
5. **Test Automation**
   - **Purpose**: Automate the generation and execution of test cases.
   - **Example**: Generating automated test scripts for the registration process to validate each step.



Structural models

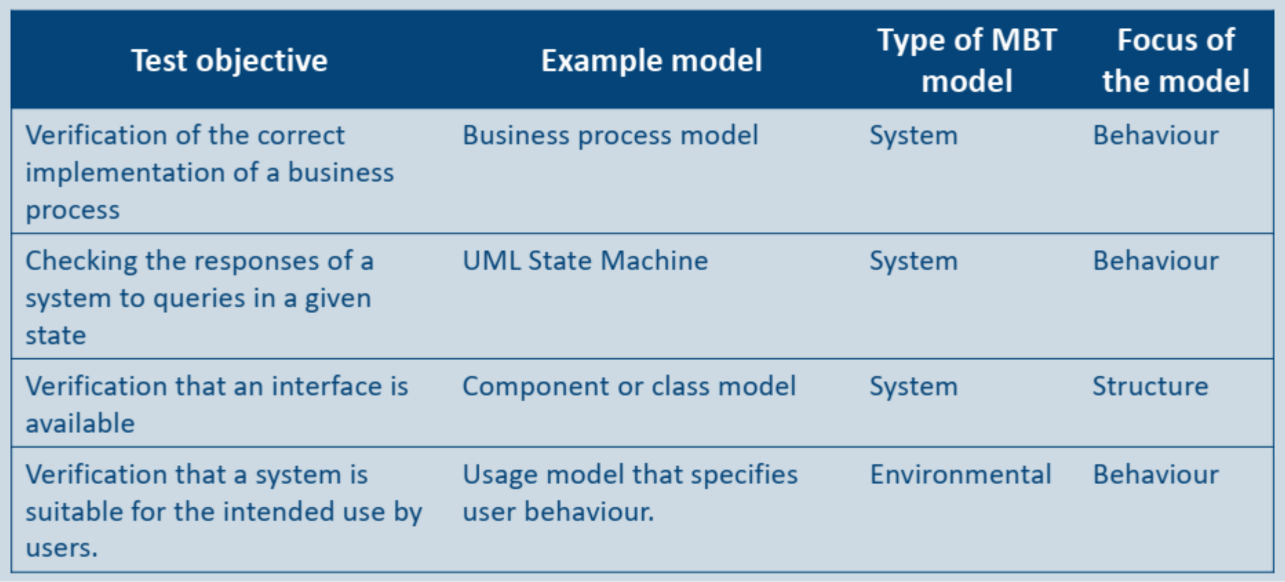
- Specify static structures, such as data or interfaces.
- Modelling by means of
  - Class diagrams,
  - Component diagrams etc.



Behavioural models

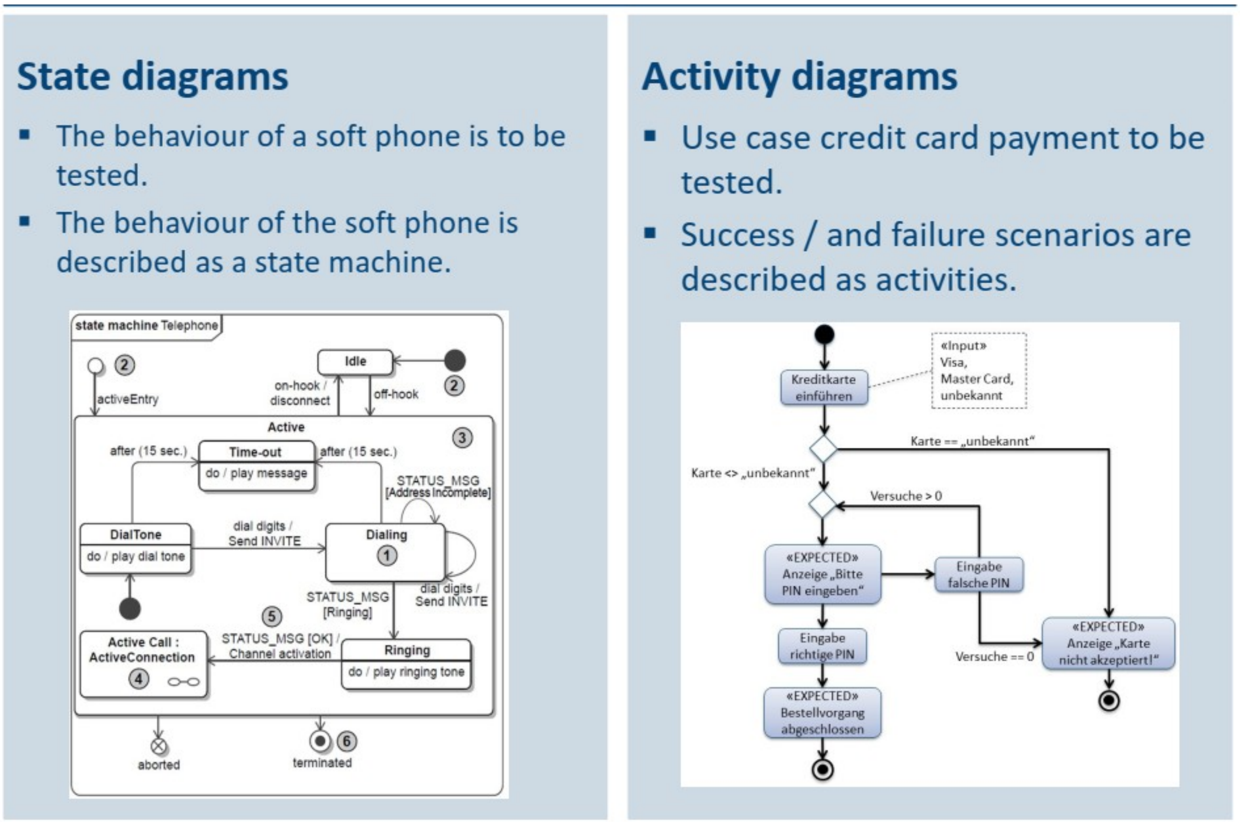
- Describe dynamic interactions (inputs and outputs).
- Modelling by means of
  - Activity-,
  - State diagrams or
  - Business process models etc.

# Relation between MBT models and Test objects:



Test objective	Example model	Type of MBT model	Focus of the model
Verification of the correct implementation of a business process	Business process model	System	Behaviour
Checking the responses of a system to queries in a given state	UML State Machine	System	Behaviour
Verification that an interface is available	Component or class model	System	Structure
Verification that a system is suitable for the intended use by users.	Usage model that specifies user behaviour.	Environmental	Behaviour

# Selection of modeling language from this two,



State diagrams

- The behaviour of a soft phone is to be tested.
- The behaviour of the soft phone is described as a state machine.

Activity diagrams

- Use case credit card payment to be tested.
- Success / and failure scenarios are described as activities.

# Modeling Languages Available for MBT

1. **Unified Modeling Language (UML)**
    - **Purpose**: Standardized language for system design.
    - **Examples**: Class diagrams, sequence diagrams, state machine diagrams, activity diagrams.

2. **Domain-Specific Languages (DSLs)**
   - **Purpose**: Tailored to specific domains for high-level abstraction.
   - **Examples**: DSLs for business process modeling in ERP systems.
3. **Specification Languages**
   - **Purpose**: Formal description of system behavior and constraints.
   - **Examples**: Z Notation, Alloy.
4. **Formal Languages**
   - **Purpose**: Mathematical notation for precise system modeling.
   - **Examples**: TLA+ (Temporal Logic of Actions), Petri Nets.
5. **Graphical Modeling Languages**
   - **Purpose**: Visual representation of system components and interactions.
   - **Examples**: BPMN (Business Process Model and Notation).

Exercises

Exercise-A:

Description:

• An input form for customer registration has an input field for a mobile phone number.
• Only numbers from the German mobile network must be entered in the input field.
• The input should be in the form xxxx-yyyyyyy, where x stands for the prefix digits and y for subscriber digits.
• Note: Mobile prefixes are in the range 015-017;
The subscriber part can consist of seven or eight digits.
• If an invalid combination of digits is entered, an error message should be displayed to the user.

Task:

• Identify all valid and invalid equivalence classes.
• The boundary values must also be analyzed in this context.

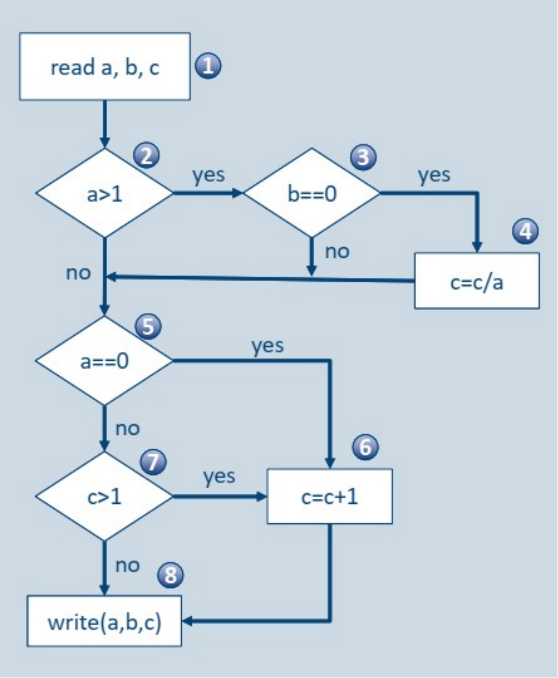
Solution:

• Equivalence classes and boundary values for prefix digits:
– Invalid numbers 1: 0000 - 0149
– Invalid numbers 2: 0180 - 9999
– Valid numbers: 0150 - 0179
– Lower boundary values: 0149 / 0150 / 0151
– Upper boundary values: 0179 / 0180 / 0181
– The number of digits could also be checked here: 0 - 3 digits
• Equivalence classes and boundary values for subscriber numbers:
– Focus is on the number of digits
– Invalid numbers 1: Numbers with 0 to 6 digits

– Invalid numbers 2: Numbers with 9 and more digits
– Valid numbers: Numbers with 7 / 8 digits
– Boundary values: 0, 1 / 6,7,8 / 8,9,10
– Note: Due to overlapping values of the boundary value analysis, a combination of boundary values and equivalency class analysis is recommended.

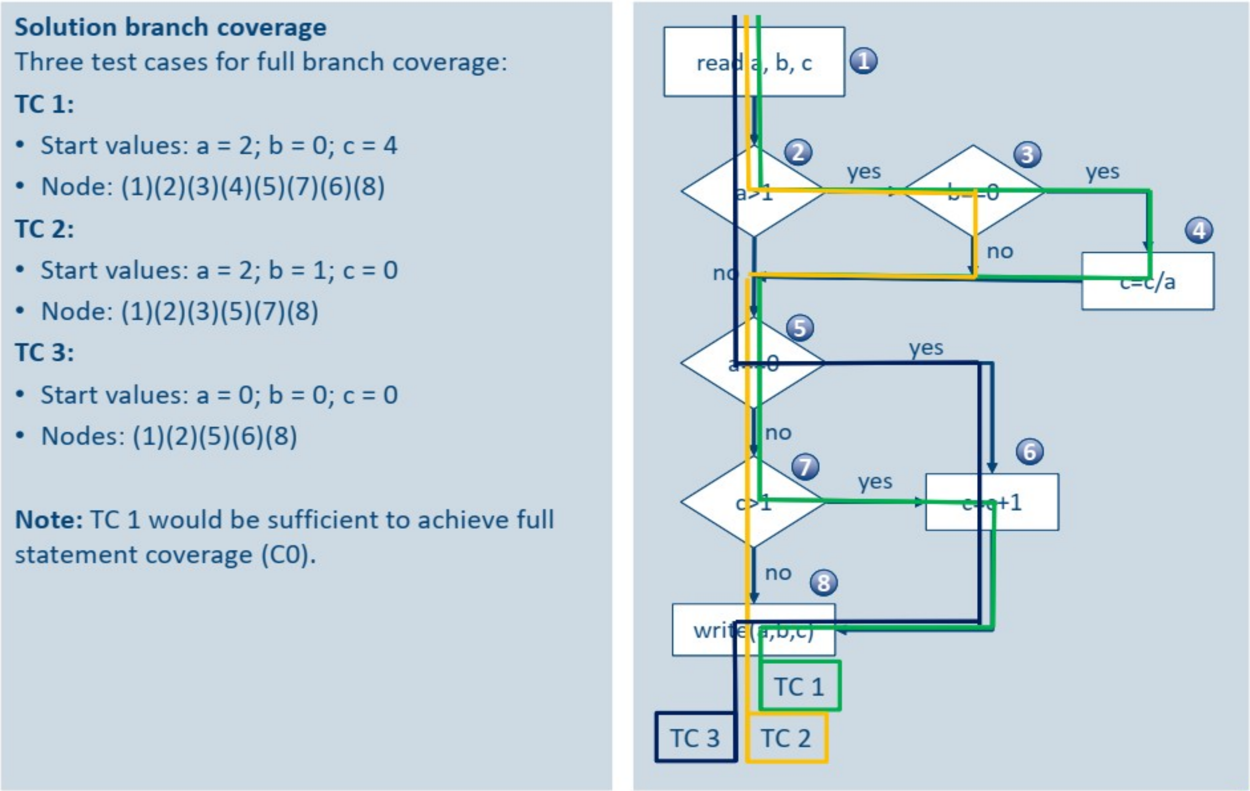
# Exercise-B:

Description: The flow diagram on the right side is given.



Task:
• Test cases for the following coverage types are to be created for the flow chart:
– Branch coverage
– Path coverage

• Each identified path and the associated initial values for variables a, b and c shall be noted.



Solution branch coverage

Three test cases for full branch coverage:

TC 1:

- Start values: a = 2; b = 0; c = 4
- Node: (1)(2)(3)(4)(5)(7)(6)(8)

TC 2:

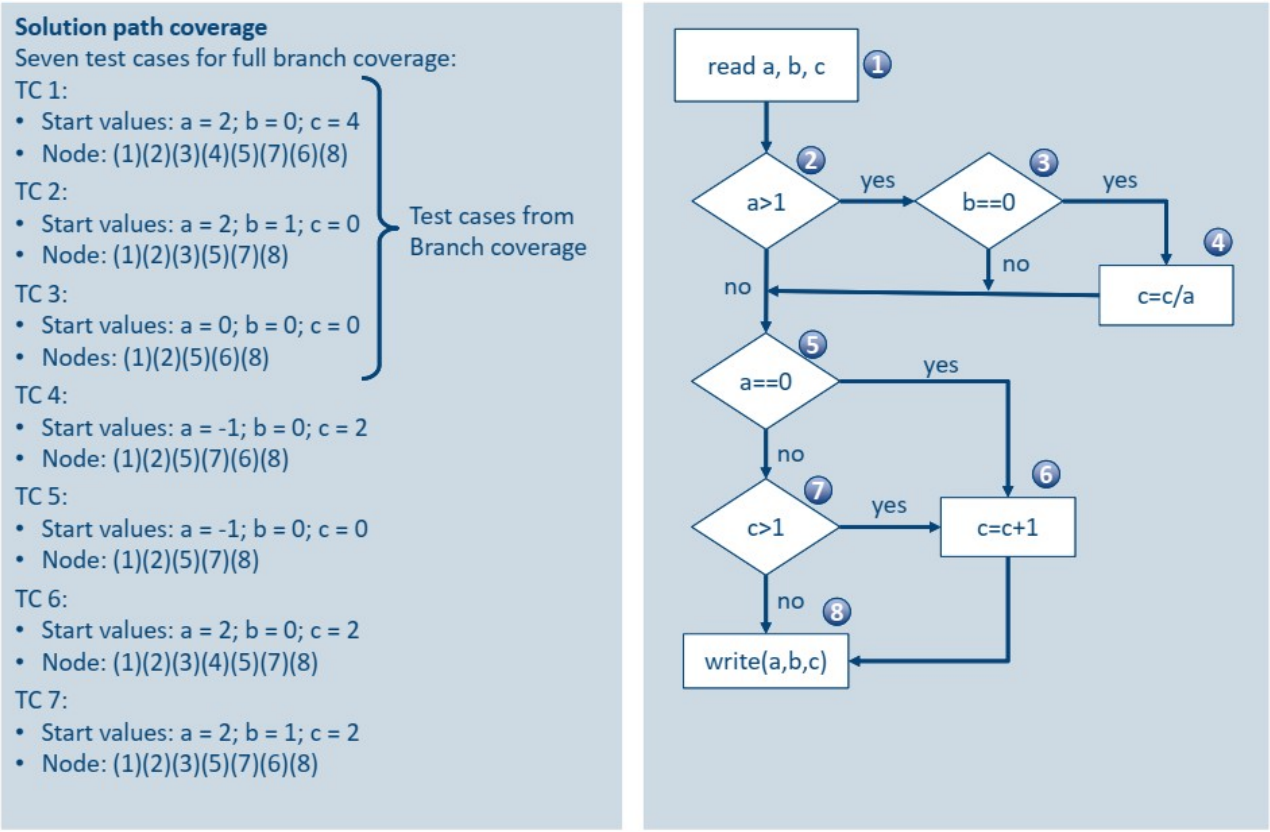
- Start values: a = 2; b = 1; c = 0
- Node: (1)(2)(3)(5)(7)(8)

TC 3:

- Start values: a = 0; b = 0; c = 0
- Nodes: (1)(2)(5)(6)(8)

Note: TC 1 would be sufficient to achieve full statement coverage (C0).

Branch Coverage



Solution path coverage

Seven test cases for full branch coverage:

TC 1:

- Start values: a = 2; b = 0; c = 4
- Node: (1)(2)(3)(4)(5)(7)(6)(8)

TC 2:

- Start values: a = 2; b = 1; c = 0
- Node: (1)(2)(3)(5)(7)(8)

TC 3:

- Start values: a = 0; b = 0; c = 0
- Nodes: (1)(2)(5)(6)(8)

Test cases from Branch coverage

TC 4:

- Start values: a = -1; b = 0; c = 2
- Node: (1)(2)(5)(7)(6)(8)

TC 5:

- Start values: a = -1; b = 0; c = 0
- Node: (1)(2)(5)(7)(8)

TC 6:

- Start values: a = 2; b = 0; c = 2
- Node: (1)(2)(3)(4)(5)(7)(8)

TC 7:

- Start values: a = 2; b = 1; c = 2
- Node: (1)(2)(3)(5)(7)(6)(8)

Path Coverage

Exercise-C:

Use Case: Registration of new customers for a web portal for online banking.

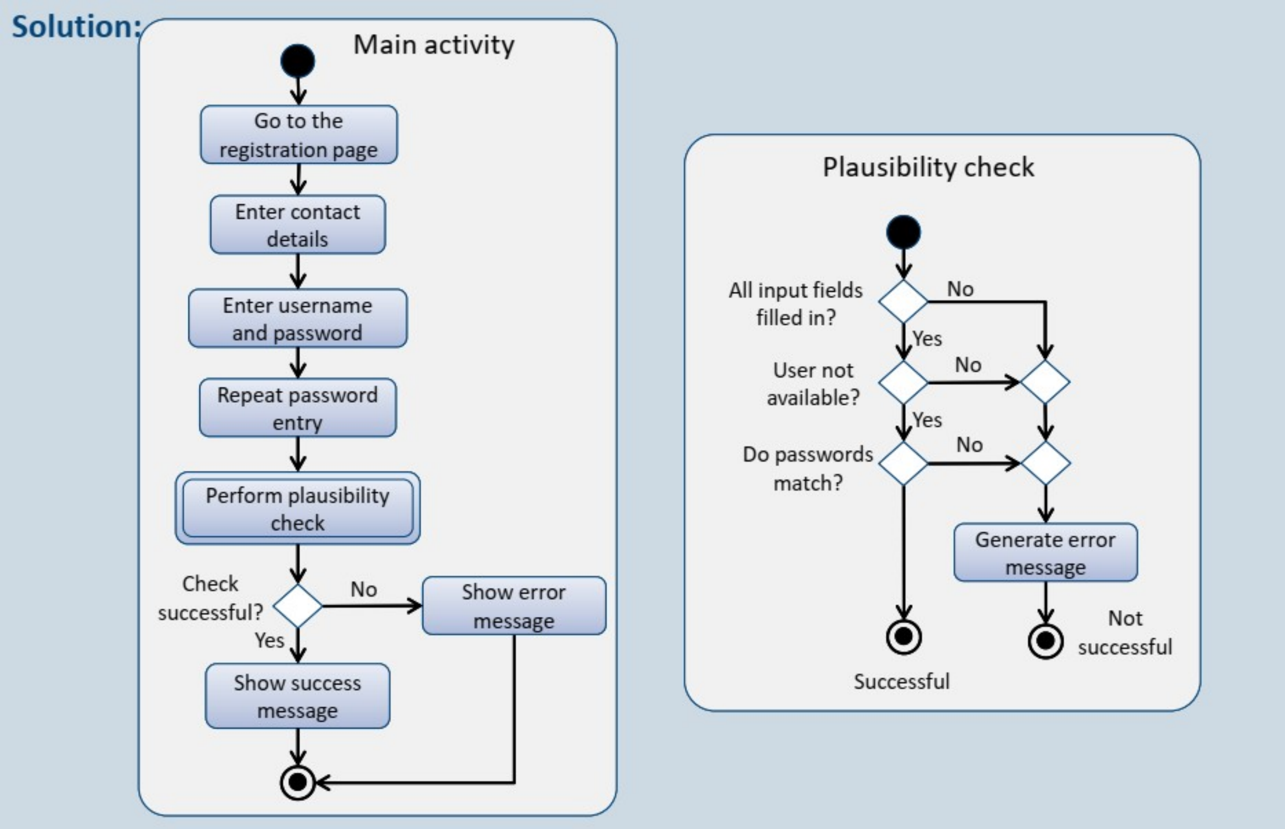
Steps to be taken:

1. Opening the registration page
2. Enter the contact details, the desired username and password.
3. The desired password must be confirmed by entering it again.
4. Complete the registration by clicking the "Register" button.
5. The system performs a plausibility check of the entered data.
6. If the check was completed successfully, a success message is displayed to the customer.
7. If the plausibility check is negative, the user is shown an appropriate error message.

Plausibility checks:

- No input field may be empty.
- No double registration may take place for an already existing user name.
- The password and its confirmation must match.

Task: An activity diagram is to be created for the given use case.



2# Block Session – 2:

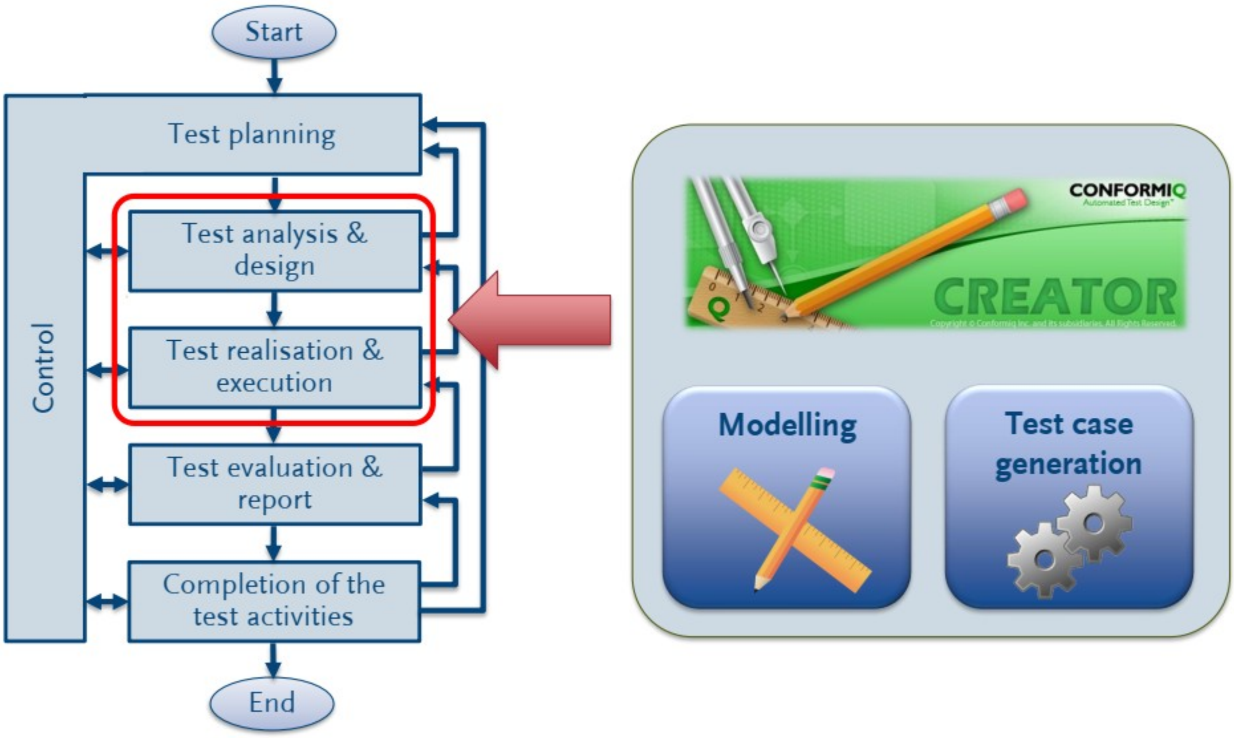
Model Based Testing

# MBT tool conformiq creator: Conformiq Creator is a model-based testing (MBT) tool that helps you design and generate automated test cases from models. It's a powerful tool for creating comprehensive and efficient test suites, especially for complex systems.

Conformiq Creator is an automated test case generation tool for MBT that:

1. Creates test cases based on visual models of system behavior
2. Provides a graphical interface for modeling application workflows
3. Automates test case creation, reducing manual effort
4. Supports both structure and activity diagrams
5. Offers a no-code environment for broader team accessibility
6. Incorporates AI to enhance test generation speed and coverage
7. Aligns with Agile methodologies
8. Exports tests in various formats for execution
9. Ensures requirements traceability

# Conformiq Creator in the MBT Test Process



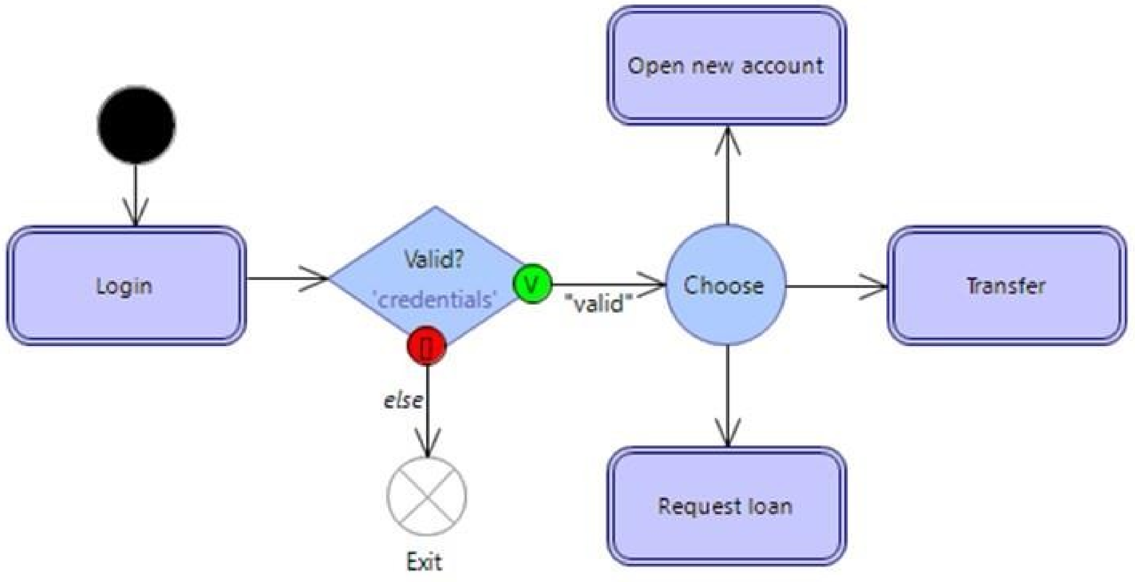
- **Phases Supported:**
  - **Test Analysis & Design:**
    - Creation of MBT models
    - Selection of test cases based on selection criteria
    - Determination of test coverage
    - Optional: Import requirements and automatically create diagrams via import

- **Test Realization & Execution:**
    - Generation of test cases
    - Review of generated test cases
    - Generation of executable test scripts
    - Optional: Generation of test descriptions

# Types of Conformiq Creator

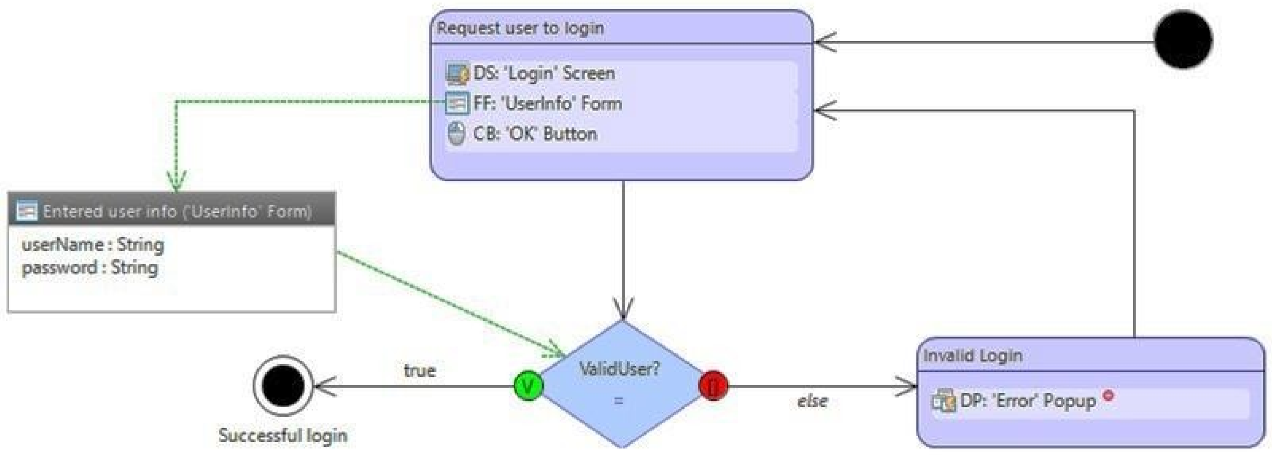
- **Business Activity Models:**

    - Used for modeling business transactions
    - High level of abstraction
    - Suitable for generating test case descriptions
    - Consist of activity diagrams with business activity nodes



Test Models:

- Models aspects of the system to be tested
- Focus on behavioral and structural aspects
- Consist of structure diagrams (SD) and activity diagrams (AD)
- Suitable for generating executable test scripts

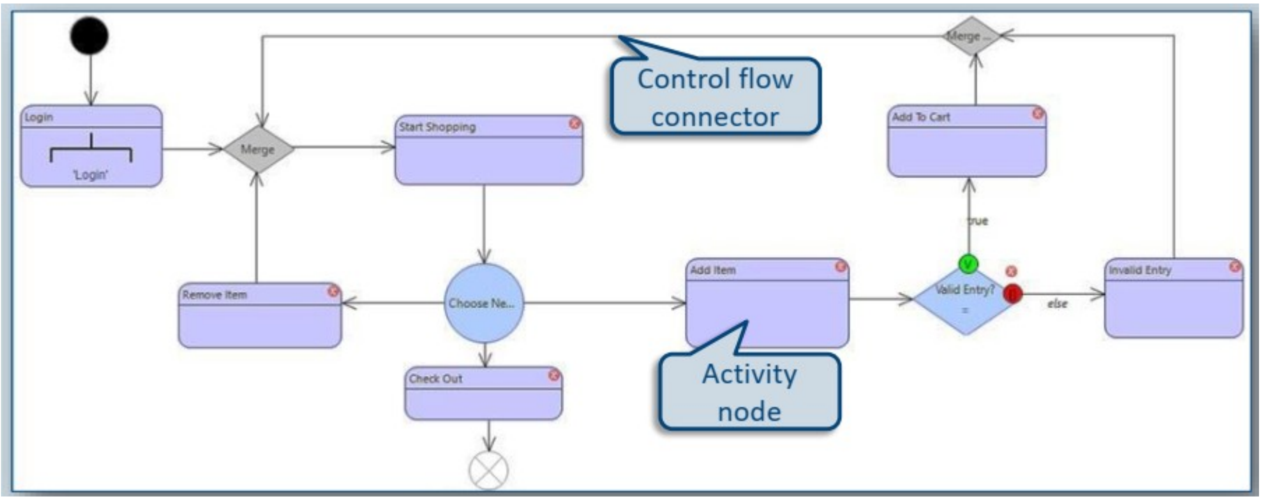


# Importable/Exportable Artefacts

- **Requirements Import:**
  - CA Agile Central
  - IBM DOORS
  - Microfocus ALM
  - Excel
  - JIRA
- **Test Management Tools:**
  - Microfocus ALM
  - Tricentis Tosca
  - CA Agile Central
- **Executable Test Scripts:**
  - Microfocus UFT
  - Selenium
  - JUnit
- **Test Case Descriptions:**
  - Gherkin
  - Excel
  - HTML
  - XML

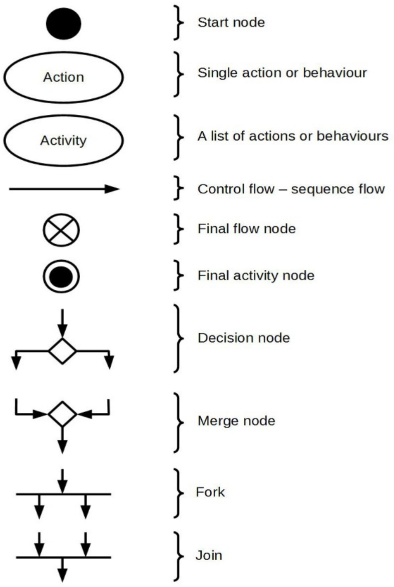
Business Activity Models

# Essential Model Elements of Activity Diagrams

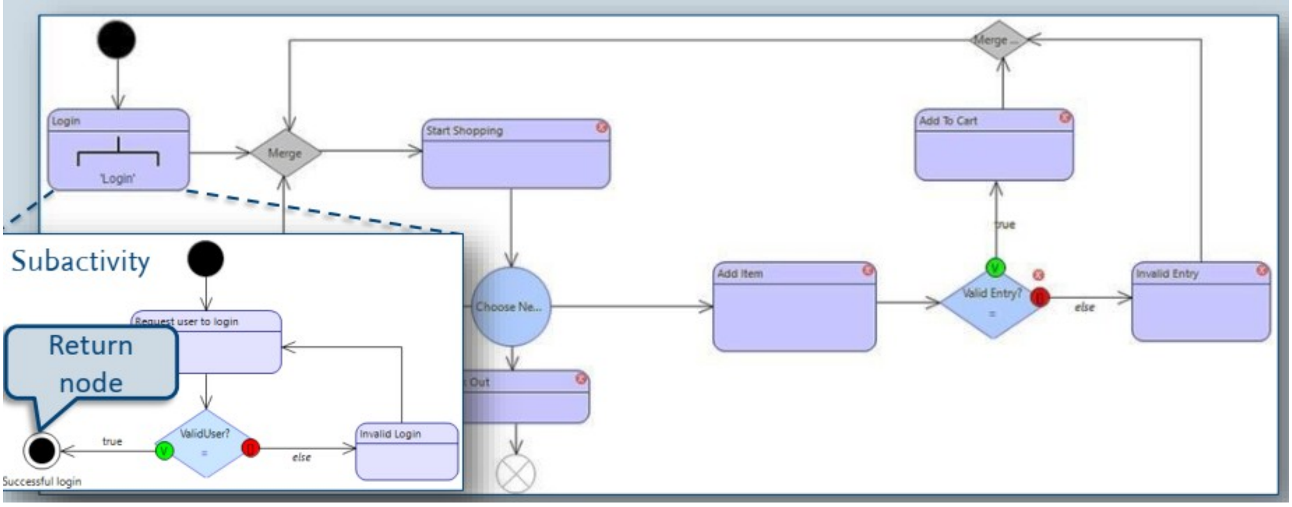


- **Control Flow Connectors:**
  - Define the sequence of an activity
  - Connect the model elements in the diagram
- **Activity Node:**
  - Contains actions for inputs and verification’s

- Only used in test models



- **Other Elements:**
  - **Initial Node:** Starting point of activities
  - **Flow Final Node:** Terminates a complete activity
  - **Hierarchical Activity Node:** Calls up a sub-activity
  - **Return Node:** Returns to the calling AD diagram
  - **Decision Node:** Splits control flow based on conditions
  - **Merge Node:** Merges several control flow branches
  - **Event Node:** Chooses control flow based on input actions
  - **Block Node:** Terminates an activity and is ignored in test case generation



Creation of Business Activity Models

- Focus on modeling business transactions with a high level of abstraction
- Suitable for generating test case descriptions
- Consist of activity diagrams with business activity nodes

Exercise:

▪ Given are the three use cases of the activity diagram exercise.
▪ An initial business activity model, which contains activity nodes but no actions, is provided.

Task:
▪ Based on the given use cases for the Parabank demo, the provided business activity model is to be completed with actions.

Solution: Let’s complete the business activity model for the Parabank demo based on the given use cases. Here’s a detailed activity diagram with actions:

Business Activity Model

1.Start

•Action: Begin the process.

2.Login as Customer

•Action:

•Enter username and password.

•Verify credentials.

•Redirect to the dashboard upon successful login.

3.Registration as New Customer

•Action:

•Enter personal information (name, address, contact details).

•Submit registration form.

•Receive confirmation email.

4.Restoration of User Data (Customer Lookup)

•Action:

•Enter customer ID or email.

•Retrieve customer data from the database.

•Display customer information.

5.End

•Action: Complete the process.

```
+-------------------------+
|          Start          |
+-----------+-------------+
            |
            v
+-----------+-------------+
|  Login as Customer      |
|  - Enter username       |
|  - Verify credentials   |
|  - Redirect to dashboard|
+-----------+-------------+
            |
            v
+-----------+-------------+
|  Registration as New    |
|  Customer               |
|  - Enter personal info  |
|  - Submit form          |
|  - Receive confirmation |
+-----------+-------------+
            |
            v
+-----------+-------------+
|  Restoration of User    |
|  Data (Customer Lookup) |
|  - Enter customer ID    |
|  - Retrieve data        |
|  - Display information  |
+-----------+-------------+
            |
            v
+-------------------------+
|           End           |
+-------------------------+
```

Test Models

Test models are essential for Model-Based Testing (MBT) as they help to systematically represent the aspects of a system that need to be tested. These models primarily focus on behavioral and structural aspects, providing a detailed and executable basis for generating test scripts.

→ **Basics of Test Models**

▪ Modeling the aspects of a system to be tested

▪ Focus of the models is on behavioral and structural aspects

▪ Models consist of structure diagrams (SD) and activity diagrams (AD)

▪ Modeling takes place in detail

▪ Suitable for generating executable test scripts

Essential Elements of SD Diagrams

- **Interface Types:**
  - Graphical User Interfaces (GUIs)
  - Messages and procedures
- **Widgets:**
  - **Container Widgets:** Tabs, Groups, Forms
  - **Action Widgets:** Buttons, Menu Bars
  - **Input Widgets:** Text Boxes, Check Boxes

Widget Properties and Data Types

- Widgets have properties like Name, Type, Default Value, and Default Status
- Data types include simple (String, Number, Boolean, Date) and complex types

Creation of Test Models

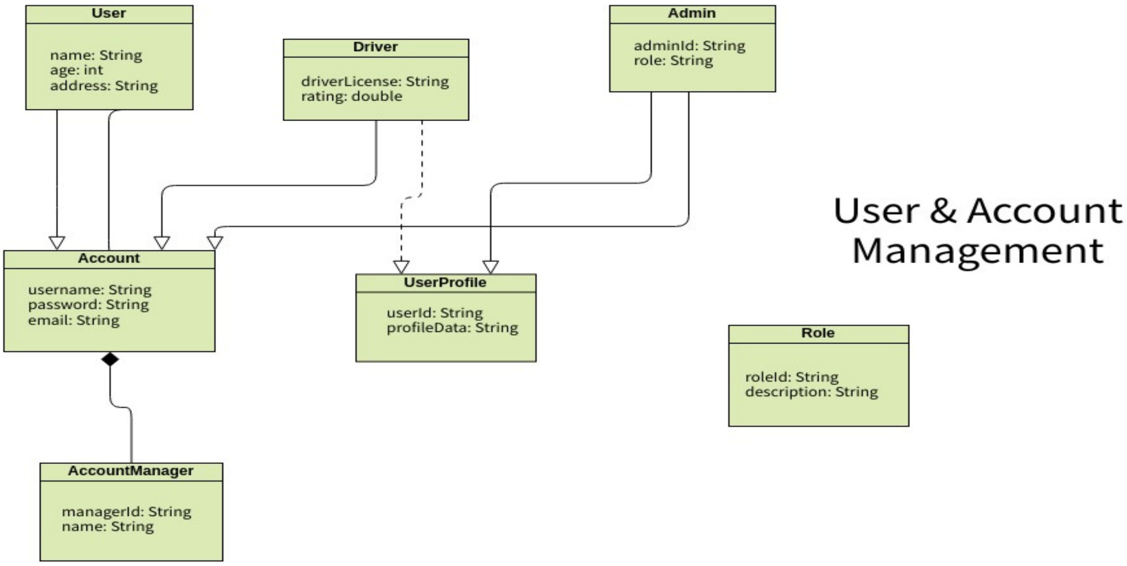
- **Structure Diagrams (SD):** Define interfaces of the system under test
- **Activity Diagrams (AD):** Behavioral modeling with activity nodes

Detailed Steps for Creating Test Models

1. Structure Diagrams (SD):

- **Define Interfaces:** Identify all interfaces (e.g., GUI elements, API endpoints) through which the system interacts with users or other systems.
- **Identify Data Types and Structures:** Define the data types (e.g., strings, integers, booleans) and structures (e.g., arrays, objects) used in the system.
- **Establish Relationships:** Specify relationships between different components and data structures (e.g., parent-child relationships in GUIs, data dependencies in APIs).

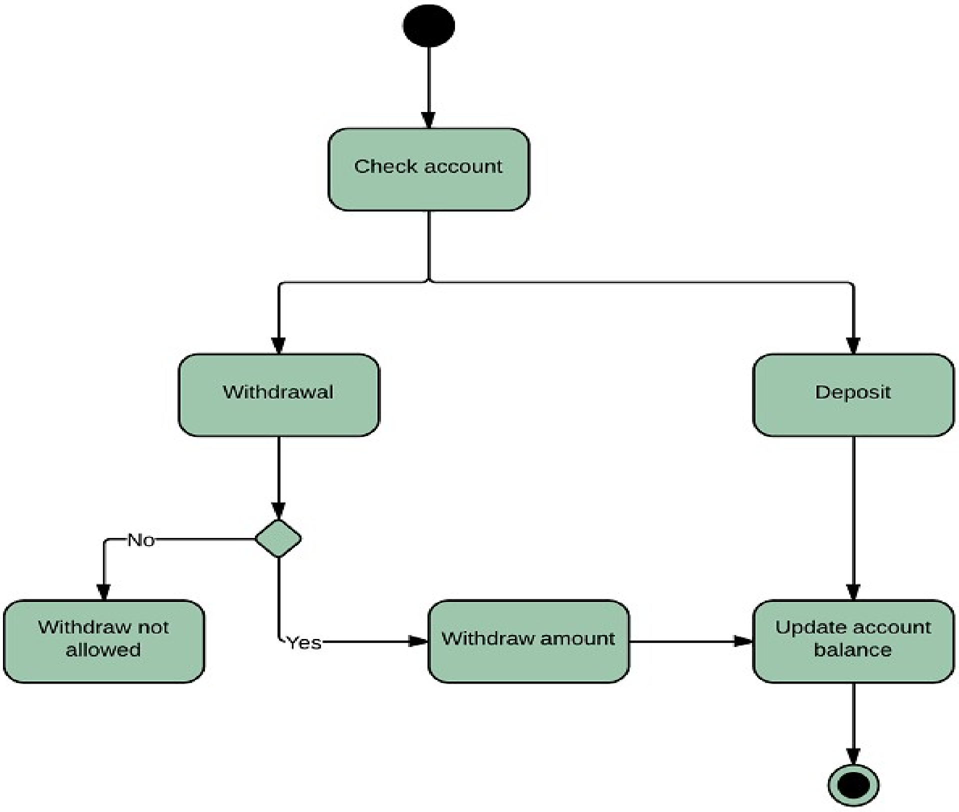
Example Structure Diagram for a Banking Application:



2. **Activity Diagrams (AD):**

- **Identify Key Activities:** Break down processes or workflows into key activities (e.g., user login, fund transfer).
- **Define Actions and Transitions:** Specify actions within each activity and transitions between activities (e.g., entering credentials, validating input).
- **Use Control Flow Elements:** Utilize decision nodes, merge nodes, and final nodes to control the flow of activities (e.g., decision node for login validity, final node for ending activity).

Example:

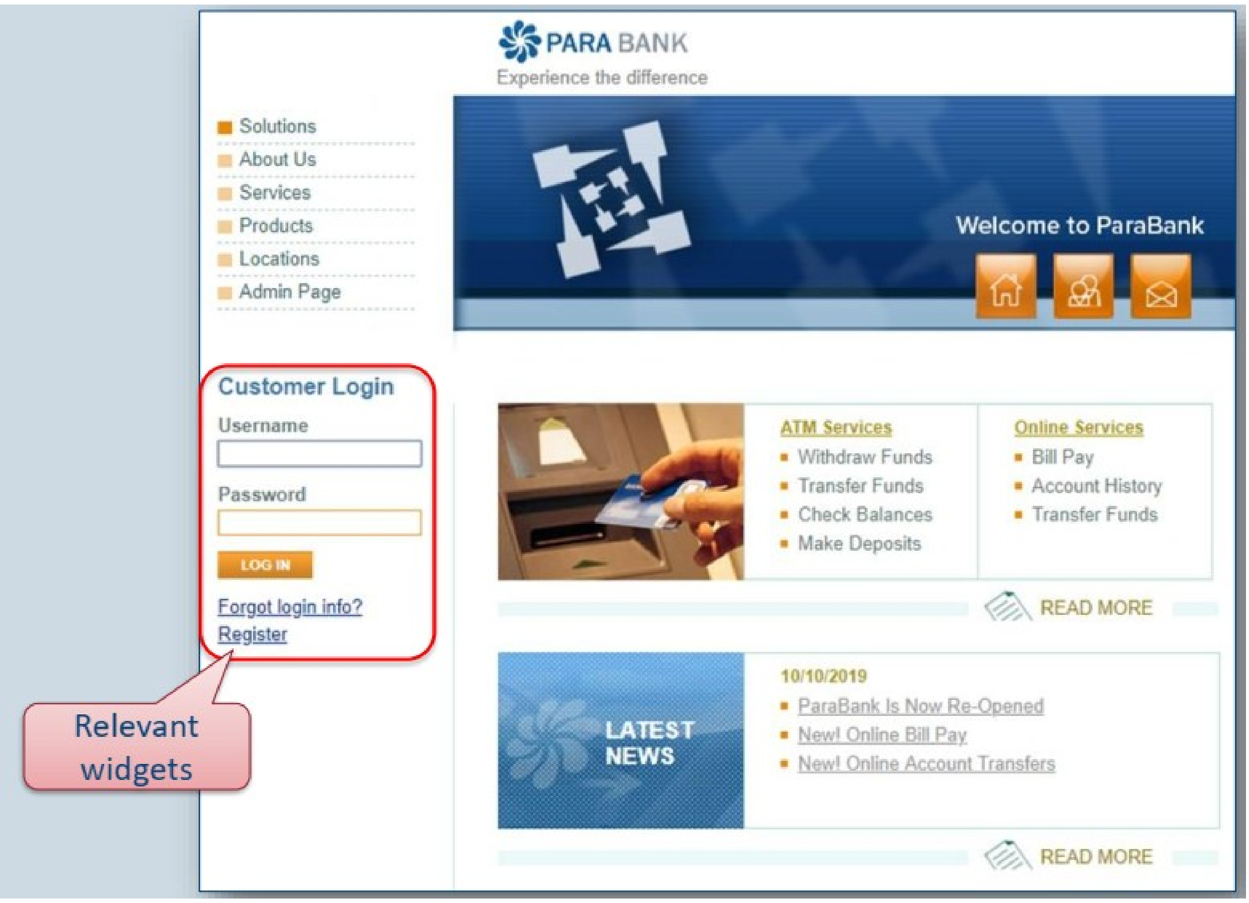


Exercise:

Description:
▪ Given are the three use cases of the activity diagram exercise.
▪ Relevant screenshots of input screens from the Parabank demo.

Task:
▪ A new Creator project is to be created.
▪ Based on the given use cases and screenshots of the input masks, a structure diagram is to be created.
▪ When creating the diagram, also specify suitable values for the properties "Default Status" and "Default Value" of the widget elements.

Solution:



Customer Login

Username

Password

Forgot login info?

Register

Relevant widgets

ATM Services

- Withdraw Funds
- Transfer Funds
- Check Balances
- Make Deposits

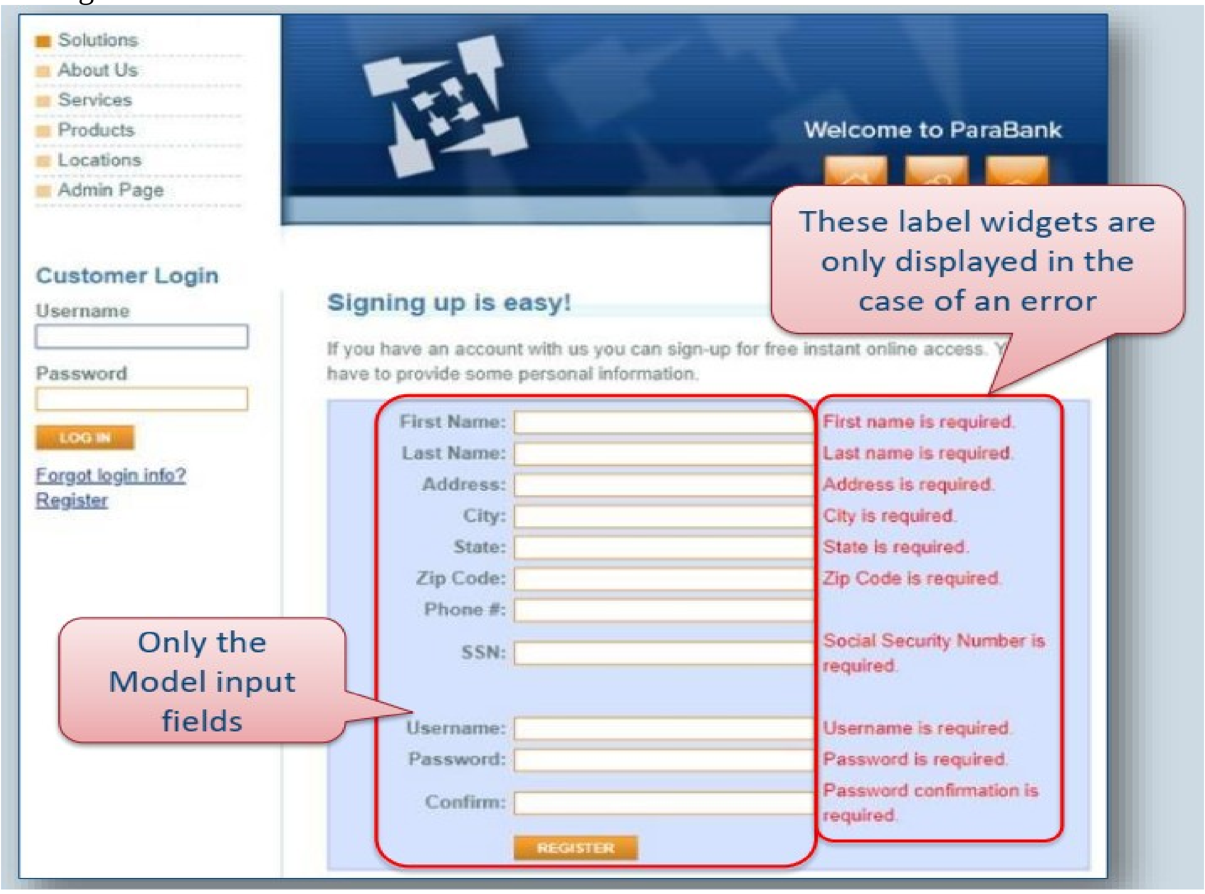
Online Services

- Bill Pay
- Account History
- Transfer Funds

10/10/2019

- ParaBank Is Now Re-Opened
- New! Online Bill Pay
- New! Online Account Transfers

Home Page



These label widgets are only displayed in the case of an error

Customer Login

Username

Password

Forgot login info?

Register

Signing up is easy!

If you have an account with us you can sign-up for free instant online access. Y have to provide some personal information.

First Name: First name is required.

Last Name: Last name is required.

Address: Address is required.

City: City is required.

State: State is required.

Zip Code: Zip Code is required.

Phone #:

SSN: Social Security Number is required.

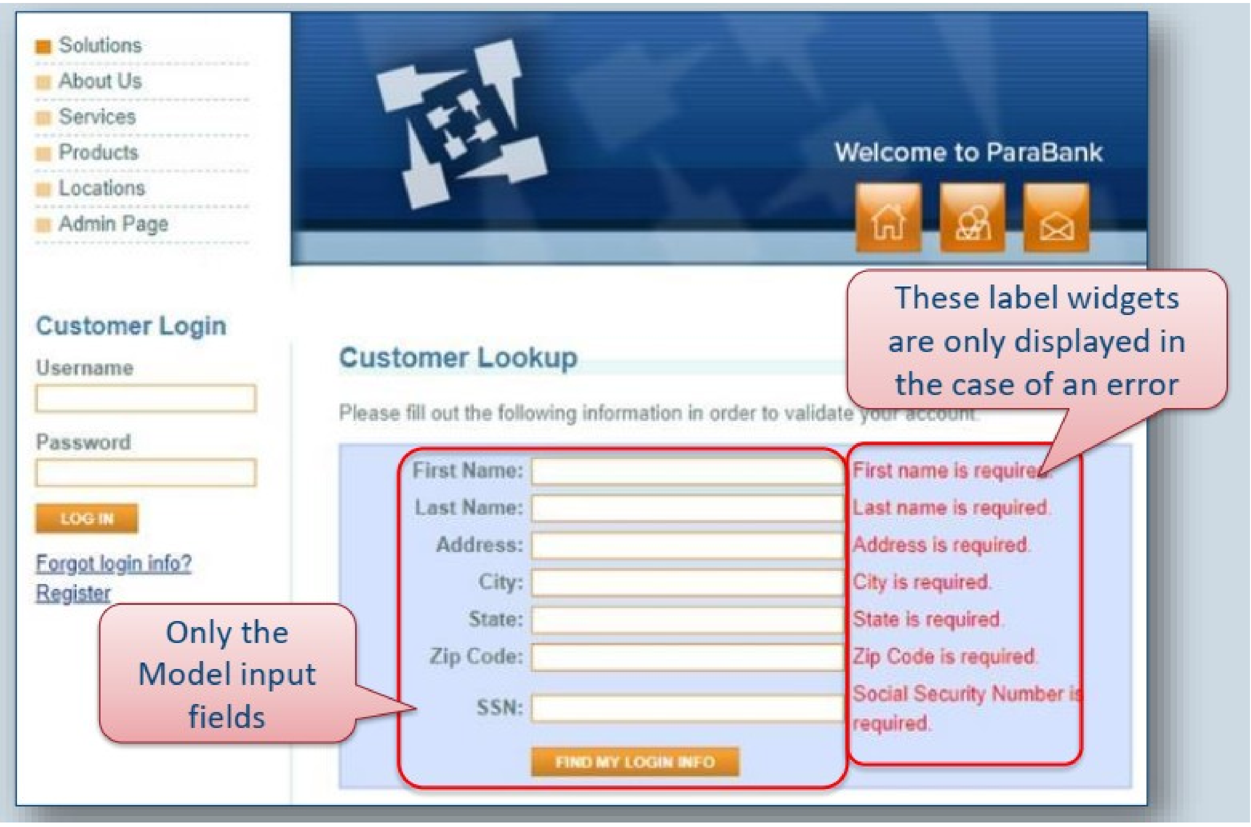
Username: Username is required.

Password: Password is required.

Confirm: Password confirmation is required.

Only the Model input fields

Registration Page



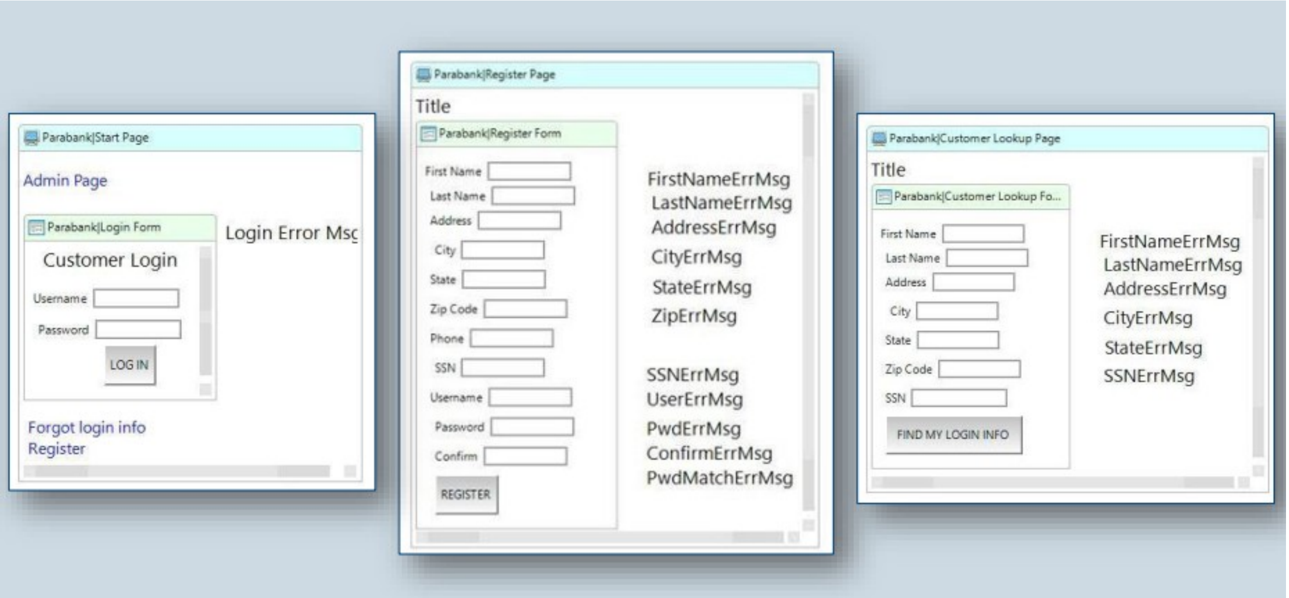
Please fill out the following information in order to validate your account.

First name is required.
Last name is required.
Address is required.
City is required.
State is required.
Zip Code is required.
Social Security Number is required.

These label widgets are only displayed in the case of an error

Only the Model input fields

Customer Lookup



Possible Structure

Tutorial

Activity Diagrams – Exercise

Description: Given the following use cases for the parabank demo:

a) Registration as a new customer
b) Restoration of user data (customer lookup)
c) Login as customer

Task:

▪ Based on the given use cases, a test model is to be created with the Conformiq Creator.
▪ Use Case (c) is in the main activity diagram and the two Use Cases (a) and (b) are to be created as sub-activities.
▪ The use cases are only to be modelled at a high level. => Activity nodes do not contain actions.
▪ For activity nodes and decision nodes, "speaking" names should be used in accordance with the use cases.

Solution: Activity Diagram in Conformiq Creator

Main Activity Diagram: Login as Customer

1. **Start Node**
2. **Activity Node:** "Enter Username and Password"
3. **Decision Node:** "Check Login Plausibility"
    - **Condition 1:** "Successful Plausibility Check"
        - **Activity Node:** "Display Customer Page"
    - **Condition 2:** "Failed Plausibility Check"
        - **Activity Node:** "Display Error Message"
4. **End Node**

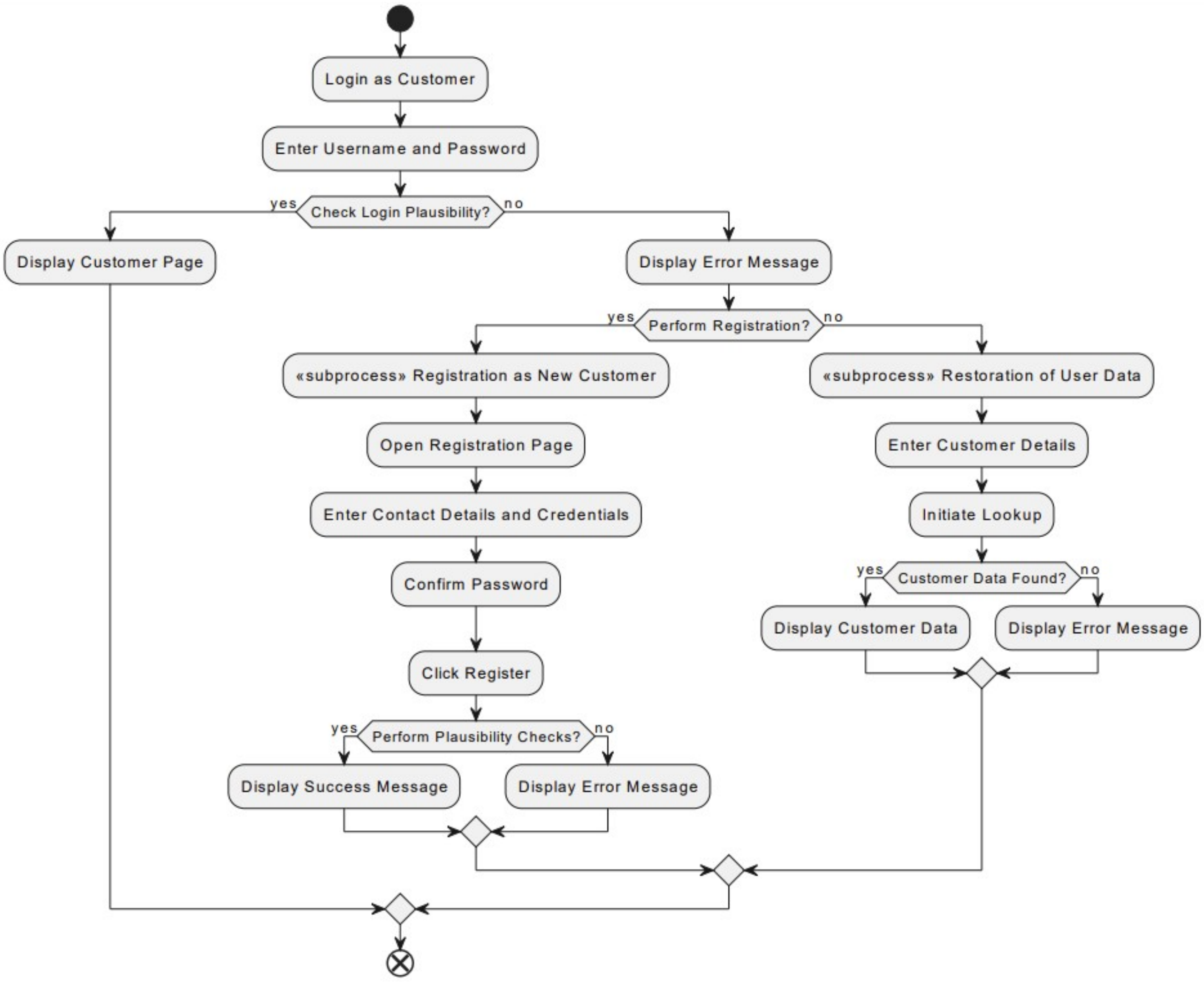
Sub-Activity 1: Registration as New Customer

1. **Start Node**
2. **Activity Node:** "Open Registration Page"
3. **Activity Node:** "Enter Contact Details and Credentials"
4. **Activity Node:** "Confirm Password"
5. **Activity Node:** "Click Register"
6. **Decision Node:** "Perform Plausibility Checks"
    - **Condition 1:** "Successful Plausibility Check"
        - **Activity Node:** "Display Success Message"
    - **Condition 2:** "Failed Plausibility Check"
        - **Activity Node:** "Display Error Message"
7. **End Node**

Sub-Activity 2: Restoration of User Data (Customer Lookup)

1. **Start Node**
2. **Activity Node:** "Enter Customer Details"
3. **Activity Node:** "Initiate Lookup"
4. **Decision Node:** "Customer Data Found?"
    - **Condition 1:** "Customer Data Found"
        - **Activity Node:** "Display Customer Data"
    - **Condition 2:** "Customer Data Not Found"
        - **Activity Node:** "Display Error Message"
5. **End Node**

Diagram:



sdfgsadg

Test Models - Exercise

Description:
▪ Given are the following use cases for the Parabank demo:
a) Registration as a new customer
b) Login as customer
▪ In addition, an initial test model consisting of structure and activity diagrams is provided.

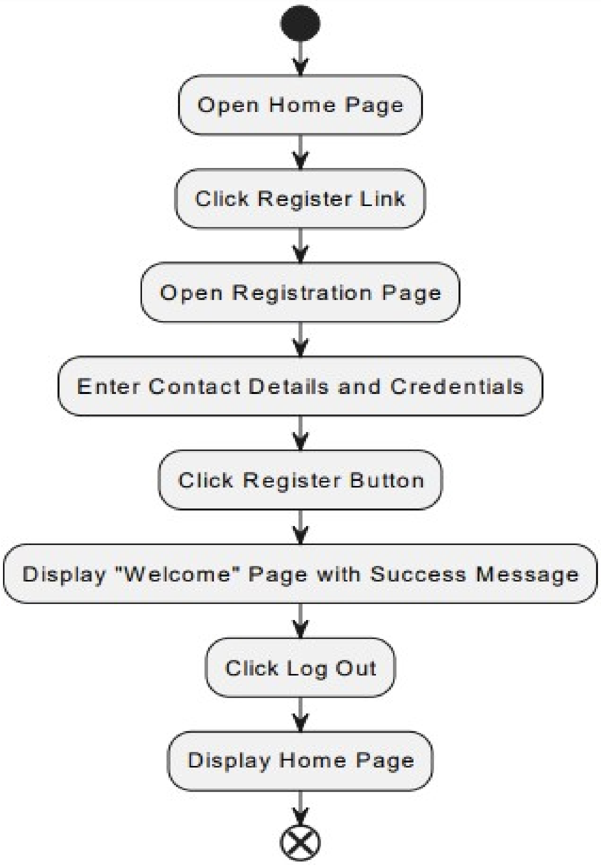
Task:
▪ Based on the given use cases, the test model is to be completed with concrete actions.
▪ For use case (a), no error scenarios are to be considered. I.e. only the successful registration is to be modelled.
▪ In use case (b), in addition to a successful login, a login attempt with an empty user name field and an empty password field should also be specified.
▪ Note: Both use cases start on the home page of the Parabank demo.

Use Case (a): Registration as a New Customer

This use case involves successful registration only, with no error scenarios.

Steps:

1. Open the registration page via hyperlink.
2. Enter contact details, desired username, and password.
3. Click the "Register" button.
4. Display the customer start page with the title "Welcome" and the message "Your account was created successfully. You are now logged in."
5. The customer logs out by clicking on the hyperlink "Log Out."

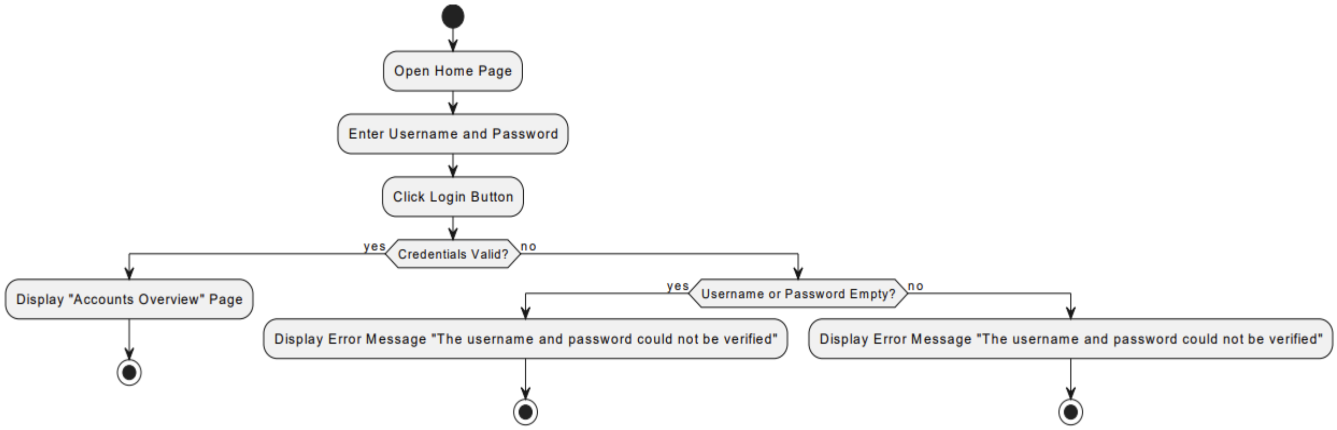


Use Case (b): Login as a Customer

This use case includes successful login, as well as error scenarios for an unknown user and missing password.

Steps:

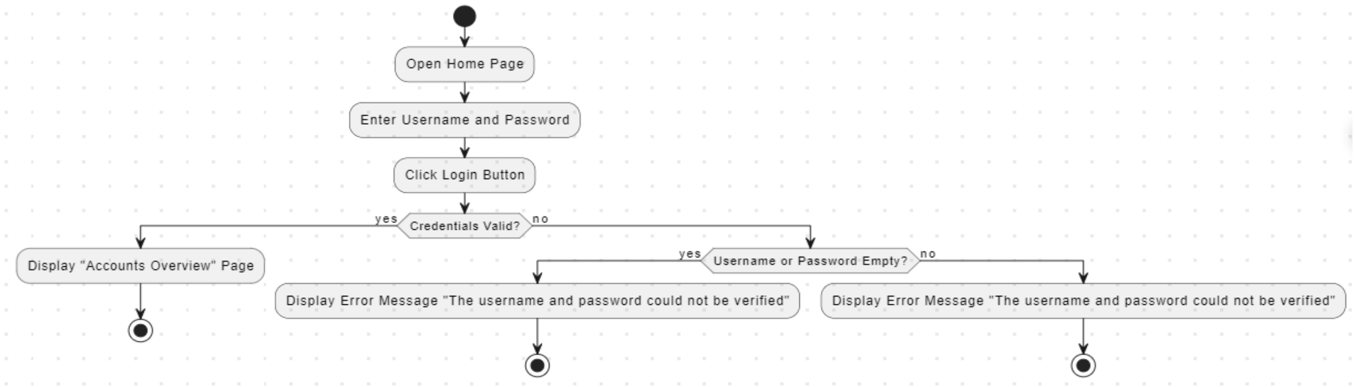
1. The user enters his username and password on the start page and clicks the login button.
2. If login is successful, the customer is shown the customer page with the title "Accounts Overview."
3. If the user is unknown, display the error message "The username and password could not be verified" on the start page.
4. If no password is entered, display the same error message on the start page.

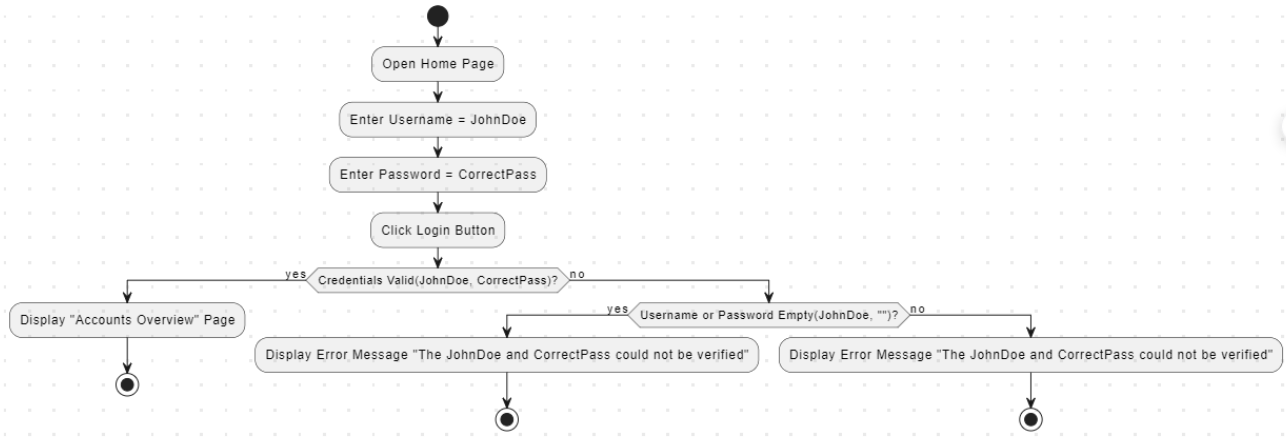

Open Home Page
Enter Username and Password
Click Login Button
yes
Credentials Valid?
no
Display "Accounts Overview" Page
yes
Username or Password Empty?
no
Display Error Message "The username and password could not be verified"
Display Error Message "The username and password could not be verified"

<u>Working With Test Data</u>

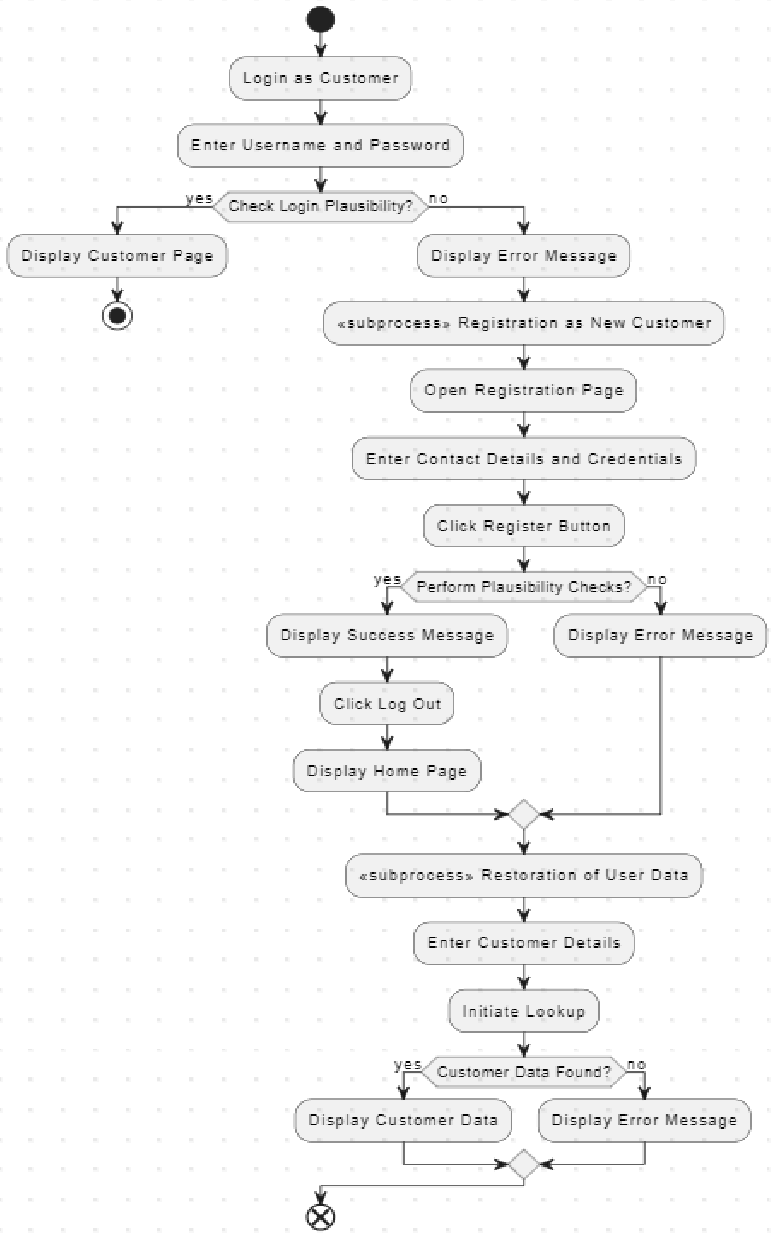
# Working with Test Data

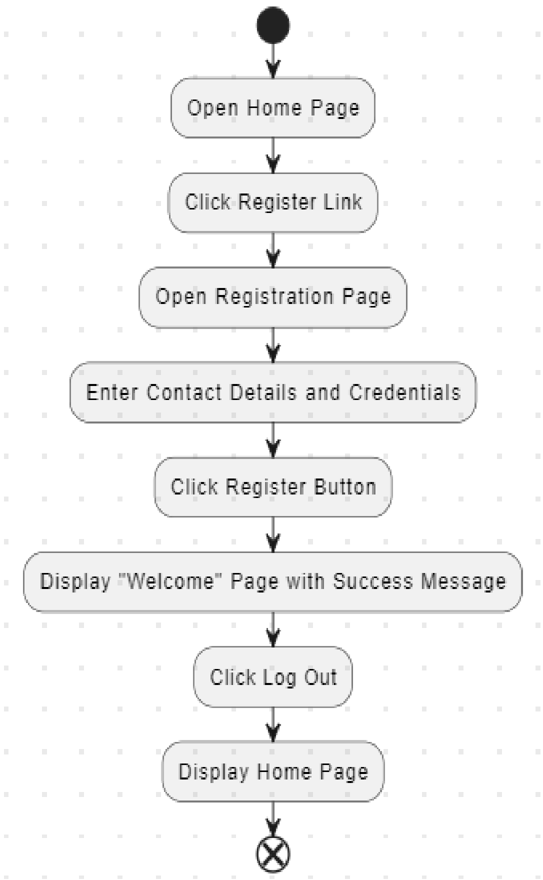
- **Atomic Values & Value Lists**
  - **Inline Specification:**
    - Values are directly included in action properties.
    - No reuse possibility.
  - **Specification as Object:**
    - Values are referenced by action properties.
    - Reusable across different actions.
- **Variables & Action References**
  - **Variables:**
    - Declared with visibility scopes: Local (diagram) or Global (model-wide).
    - Used to assign and access values.
  - **Action References:**
    - Similar to using variables but without declaration.
- **Exercises:**
  - **Exercise A:**
    - **Use Case:** "Login as customer"
    - **Task:** Complete existing actions and decision nodes, specify test data using value lists and variables.


Open Home Page
Enter Username and Password
Click Login Button
yes
Credentials Valid?
no
Display "Accounts Overview" Page
yes
Username or Password Empty?
no
Display Error Message "The username and password could not be verified"
Display Error Message "The username and password could not be verified"


Open Home Page
Enter Username = JohnDoe
Enter Password = CorrectPass
Click Login Button
yes
Credentials Valid(JohnDoe, CorrectPass)?
no
Display "Accounts Overview" Page
yes
Username or Password Empty(JohnDoe, "")?
no
Display Error Message "The JohnDoe and CorrectPass could not be verified"
Display Error Message "The JohnDoe and CorrectPass could not be verified"

- **Exercise B:**
  - **Use Case:** "New customer registration"
  - **Task:** Provide actions with "block" or "skip" preconditions to check error messages. Ensure each error message is checked by one test case.



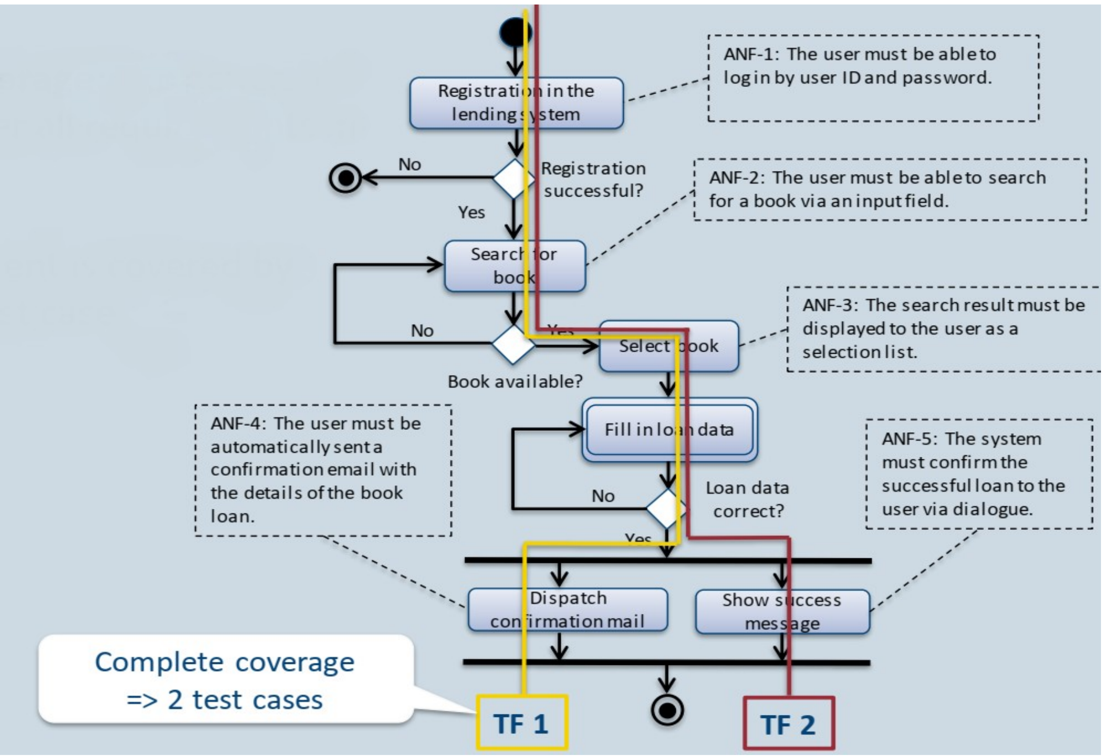


- **External Data Sources**
    - **Form Value List from External Spreadsheet File:**
        - Data read from Excel sheets matching column names with input widget names.
    - **Form Value from Spreadsheet Row:**
        - Data read line by line from Excel sheets, free assignment of columns to fields.
- **Preconditions**
    - **General:**
        - Boolean conditions checked before actions.
        - Two types: "skip" and "block".
    - **"Block" Preconditions:**
        - If not fulfilled, the entire control flow branch is ignored in test case generation.
    - **"Skip" Preconditions:**
        - If not fulfilled, the specific action is skipped.
- **Data Tables**
    - Used for modeling database-like behavior and UI Table Widget contents.
    - Defined in structure diagrams and can be initialized via Excel files.
    - User actions on UI table widgets can derive test steps.

# Test Selection Criteria & Generation

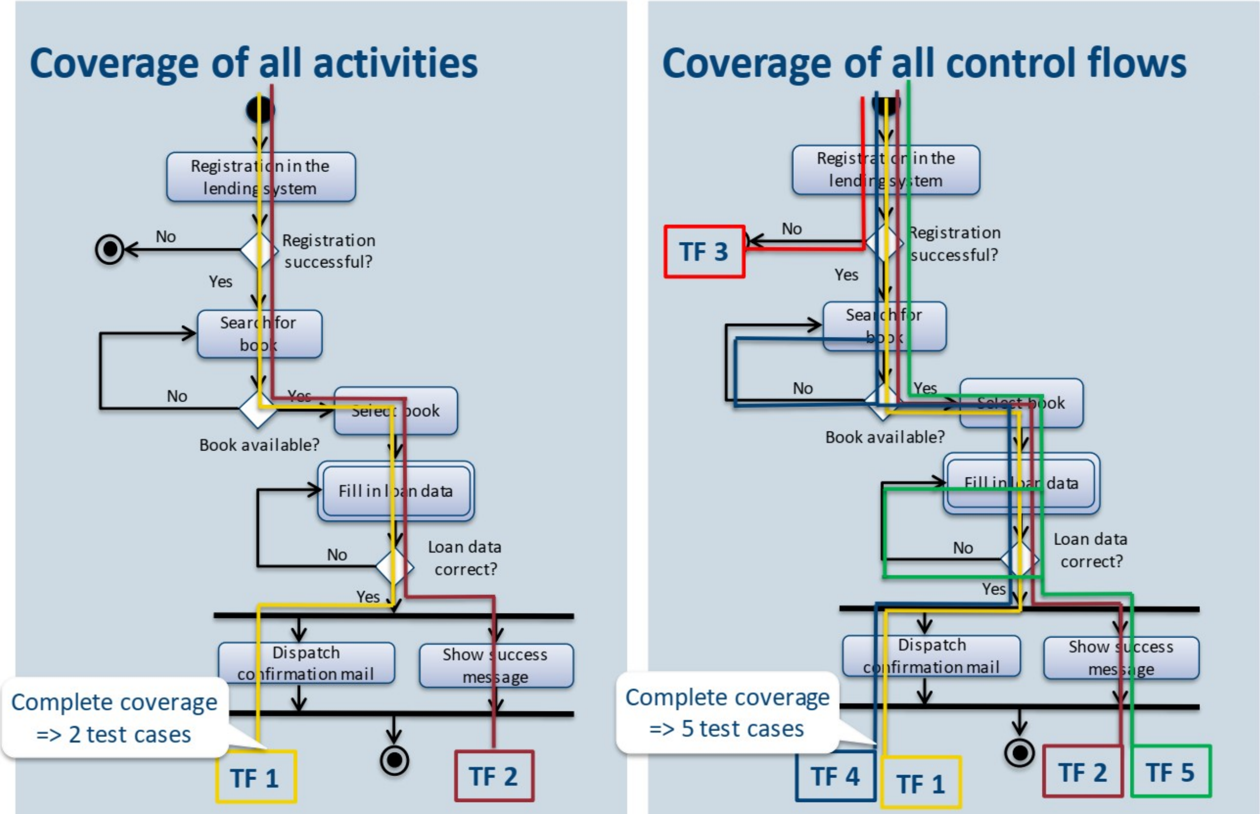
I. Coverage-based Criteria

1. **Requirements-based**
    - **Prerequisite:** Model elements are linked to requirements.
    - **Complete Coverage:** The generated test cases cover all requirements in the model.
    - **Objective:** Each requirement must be covered by at least one test case.


Registration in the lending system
ANF-1: The user must be able to log in by user ID and password.
No
Registration successful?
Yes
ANF-2: The user must be able to search for a book via an input field.
Search for book
No
Yes
Select book
Book available?
ANF-3: The search result must be displayed to the user as a selection list.
ANF-4: The user must be automatically sent a confirmation email with the details of the book loan.
Fill in loan data
No
Loan data correct?
Yes
ANF-5: The system must confirm the successful loan to the user via dialogue.
Dispatch confirmation mail
Show success message
Complete coverage => 2 test cases
TF 1
TF 2

2. **Model Elements-based**

  - **Coverage Basis:** Based on the internal structure of an MBT (Model-Based Testing) model.
  - **Possible Variants:**
    - **State Diagrams:** States, state transitions, and decision nodes.
    - **Activity Diagrams:** Activities, control flow paths, and decision nodes.
    - **Textual Models:** Instructions and conditions.
  - **Goal:** Attempt to cover 100% of the model elements, though a lower degree of coverage can also be sufficient.



Coverage of all activities

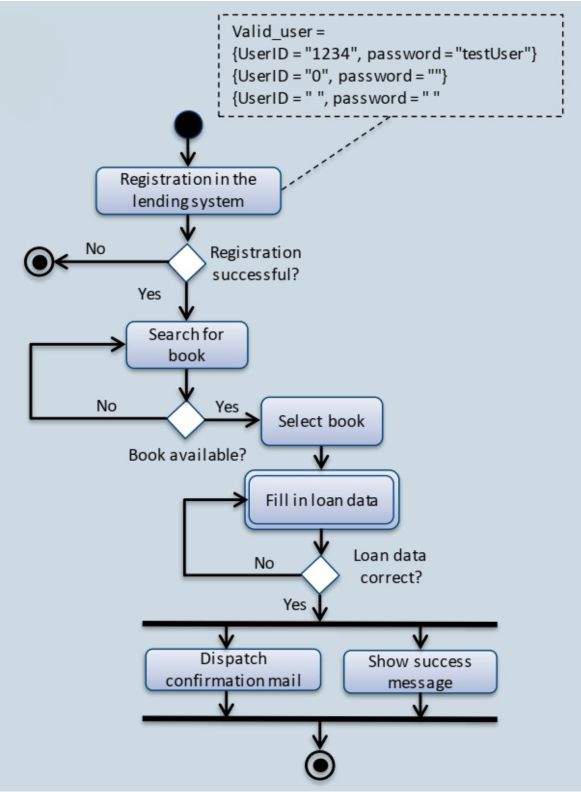
Complete coverage
=> 2 test cases

Coverage of all control flows

Complete coverage
=> 5 test cases

3. **Data-based**

  - **Techniques Required:** Application of test design techniques such as:
    - **Equivalence Partitioning:** Dividing input data into equivalent partitions where test cases are derived.
    - **Boundary Value Analysis:** Testing the boundary values of input data.
  - **Focus:** Inputs and expected results.
  - **MBT Tool Support:** Limited; inputs/outputs should already be considered during modeling.

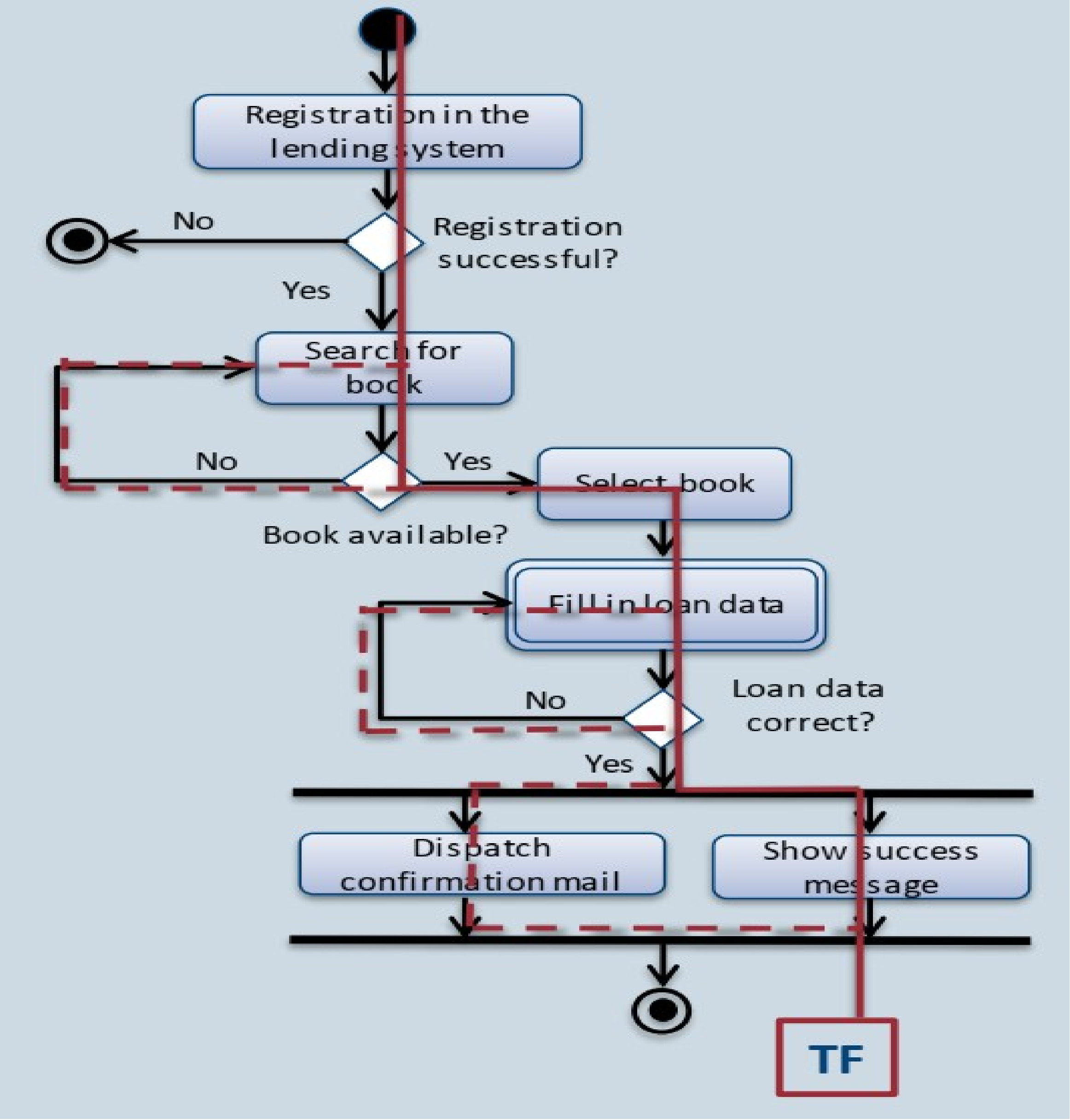


Valid_user =
{UserID = "1234", password = "testUser"}
{UserID = "0", password = ""}
{UserID = " ", password = " "

II. Other Criteria

1. **Random-based (Random)**

- **Path Determination:** Path through the MBT model is chosen comparable to a walkthrough.
- **Focus:** Test coverage is not the main focus.
- **Control Flow Selection:** Done randomly.
- **Special Case:** Probability-based test case selection, often used for usage models.



2. **Scenario/Pattern-based**

- **Scenario-based Selection:**
  - **Path Specification:** Explicit path through an MBT model specified.
  - **Specification Basis:** Through explicitly defined use case or user story.
- **Pattern-based Selection:**
  - **Template Provision:** The MBT tool is provided with a template (sample).

        - **Scenario Definition:** A pattern corresponds to an only partially defined scenario.
        - **Test Case Match:** Each generated test case must match the pattern.
        - **Flexibility:** Patterns are less restrictive and more flexible than scenarios.
3. **Project-specific**
    - **Selection Basis:** Based on project-specific information.
    - **Information Added to MBT Model:**
        - **Risks**
        - **Priorities**
        - **Required Test Environments**
    - **Selection Process:** Typically done manually according to the defined criteria of the project.

→ **When to Use Different Test Selection Criteria**

Coverage-based Criteria

1. **Requirements-based Testing**
    - **When to Use:** To ensure all documented requirements are met.
    - **Example:** Compliance with regulatory or contractual obligations.
2. **Model Elements-based Testing**
    - **When to Use:** To test the internal logic and structure of the system.
    - **Example:** Systems with complex internal states and workflows.
3. **Data-based Testing**
    - **When to Use:** To validate all relevant data partitions and boundary conditions.
    - **Example:** Applications that handle a wide range of input data.

Other Criteria

1. **Random-based Testing**
    - **When to Use:** For exploratory or ad-hoc testing to find unexpected issues.
    - **Example:** Early development stages.
2. **Scenario/Pattern-based Testing**
    - **When to Use:** To validate specific user workflows or scenarios.
    - **Example:** User acceptance testing.
3. **Project-specific Testing**
    - **When to Use:** Tailored testing based on project-specific risks and priorities.
    - **Example:** Projects with unique testing requirements.

Practical Application for Parabank Demo

1. **Requirements-based Testing:** Ensure all registration and login requirements are met.
2. **Model Elements-based Testing:** Cover all states and transitions in registration and login processes.

3. **Data-based Testing:** Validate input combinations and boundary values for registration and login fields.
4. **Random-based Testing:** Explore the system to find unexpected issues.
5. **Scenario/Pattern-based Testing:** Validate specific user scenarios, like a new user registering and logging in.
6. **Project-specific Testing:** Focus on critical areas, such as security in the registration process.

# Test Selection Criteria of the Creator

1. Test Design Configurations

- **Purpose:** Define specific test objectives to be covered.
- **Flexibility:** Multiple configurations can be defined per Creator project.
- **Display:** Test targets are shown in the "Test Targets" view.
- **Grouping:** Test objectives are organized by coverage type.

2. Conformiq Options

- **Purpose:** Provide global settings for a Creator project to control test case generation.
- **Influence:** Test case generation can be directed based on desired test coverage.

Summary

- **Test Design Configurations:** Specify and manage test objectives within a project.
- **Conformiq Options:** Adjust global settings to influence and control test case generation for optimal coverage.

Tutorial:

Test Models - Exercise

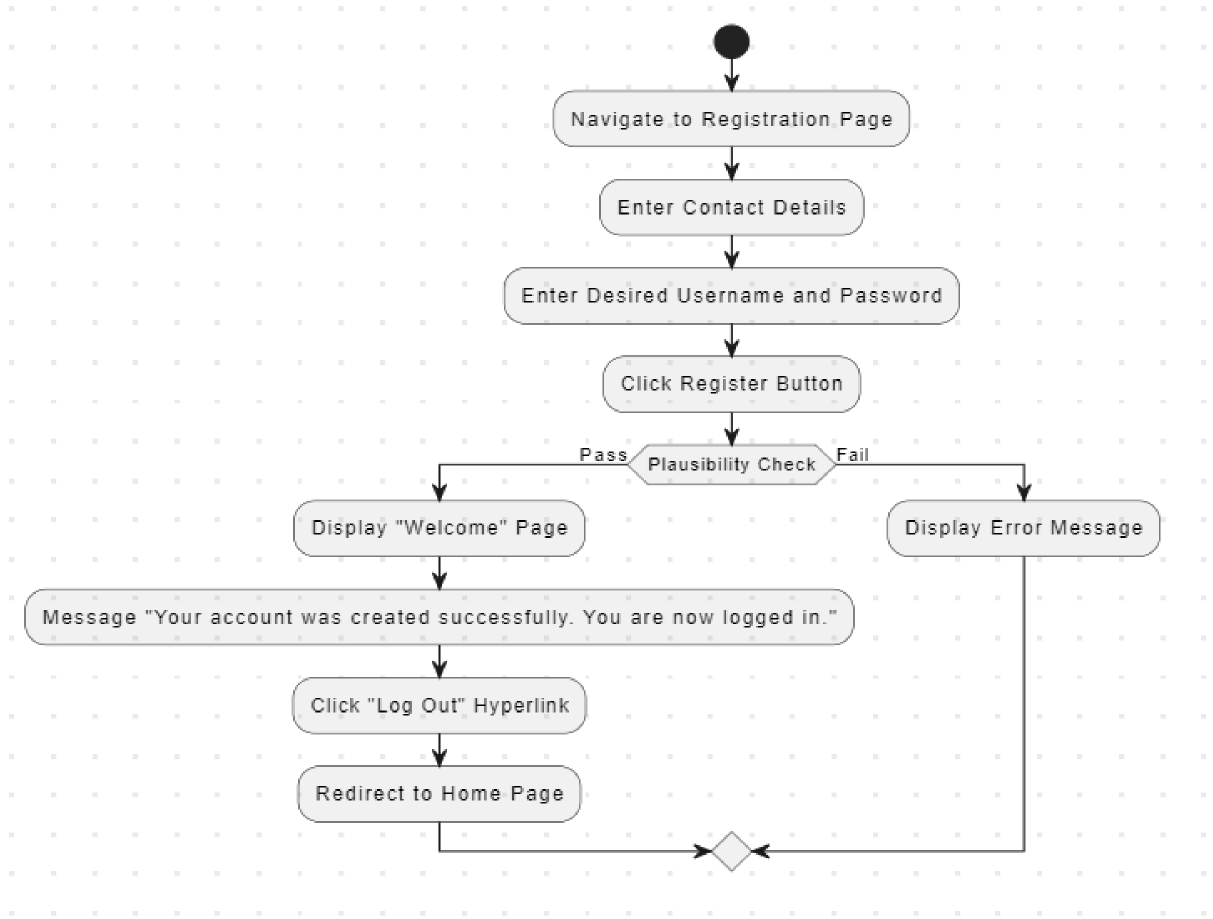
Description:
▪ Given are the following use cases for the Parabank demo: a) Registration as a new customer b) Login as customer
▪ In addition, an initial test model consisting of structure and activity diagrams is provided.

Task:
▪ Based on the given use cases, the test model is to be completed with concrete actions.
▪ For use case (a), no error scenarios are to be considered. I.e. only the successful registration is to be modelled.
▪ In use case (b), in addition to a successful login, a login attempt with an empty user name field and an empty password field should also be specified.
▪ Note: Both use cases start on the home page of the Parabank demo.

Use Case a: Registration as a new customer.

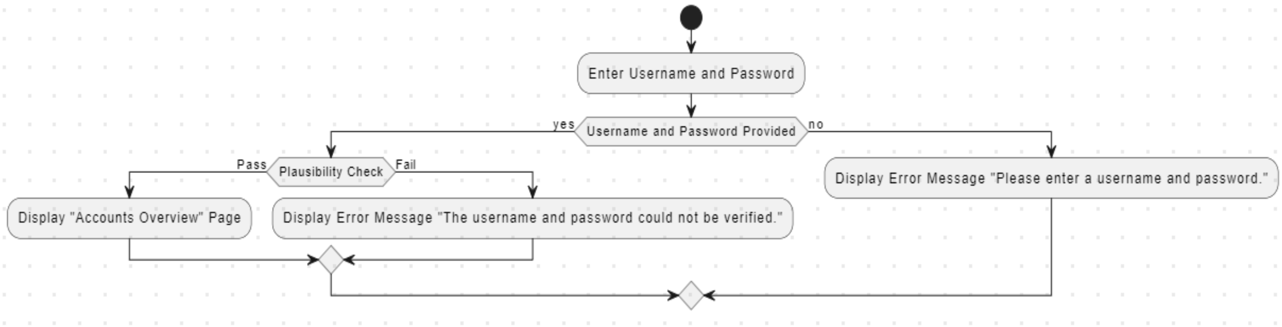
Steps to be taken:
1. Calling up the registration page (via hyperlink); The registration page with the title "Signing up is easy!" is displayed.
2. Enter the contact details, the desired username and password.
3. Complete the registration by clicking the "Register" button.

4. The customer is shown the customer start page with the title "Welcome" and the message "Your account was created successfully. You are now logged in.
5. The customer logs out by clicking on the hyperlink "Log Out".


Navigate to Registration Page
Enter Contact Details
Enter Desired Username and Password
Click Register Button
Pass
Plausibility Check
Fail
Display "Welcome" Page
Display Error Message
Message "Your account was created successfully. You are now logged in."
Click "Log Out" Hyperlink
Redirect to Home Page

Use Case b: Login as customer

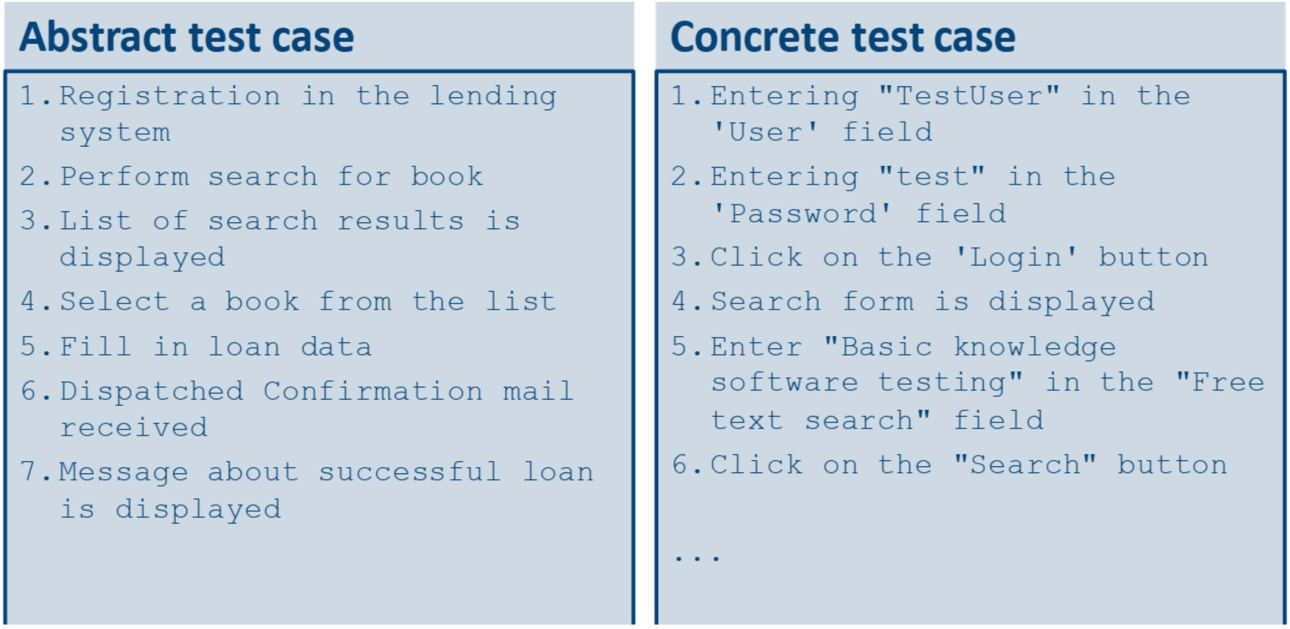
Steps to be taken:
1. The user enters his username and password on the start page and completes the entry by clicking on the login button.
. If the login is successful, the customer will be shown his individual customer page with the title "Accounts Overview" (Note: Only the title should be checked).
3. If the user is unknown, the error message "The username and password could not be verified." is displayed on the start page.
4. If no username or password is entered at login, or if both fields are left blank, the user will see the error message "Please enter a username and password." on the home page.


Enter Username and Password
yes
Username and Password Provided
no
Pass
Plausibility Check
Fail
Display Error Message "Please enter a username and password."
Display "Accounts Overview" Page
Display Error Message "The username and password could not be verified."

Software Quality Management

# Generation of Executable Test Scripts

- **Basic Knowledge**: Understanding the differences between abstract and concrete test cases.
  - **Abstract Test Cases**: Do not include specific values and are not executable. Used for reviews.
    - Example: "Registration in the lending system."
  - **Concrete Test Cases**: Include specific values and detailed steps, suitable for manual or automated execution.
    - Example: "Entering 'TestUser' in the 'User' field."



Abstract test case

```
1.Registration in the lending
  system
2.Perform search for book
3.List of search results is
  displayed
4.Select a book from the list
5.Fill in loan data
6.Dispatched Confirmation mail
  received
7.Message about successful loan
  is displayed
```

Concrete test case

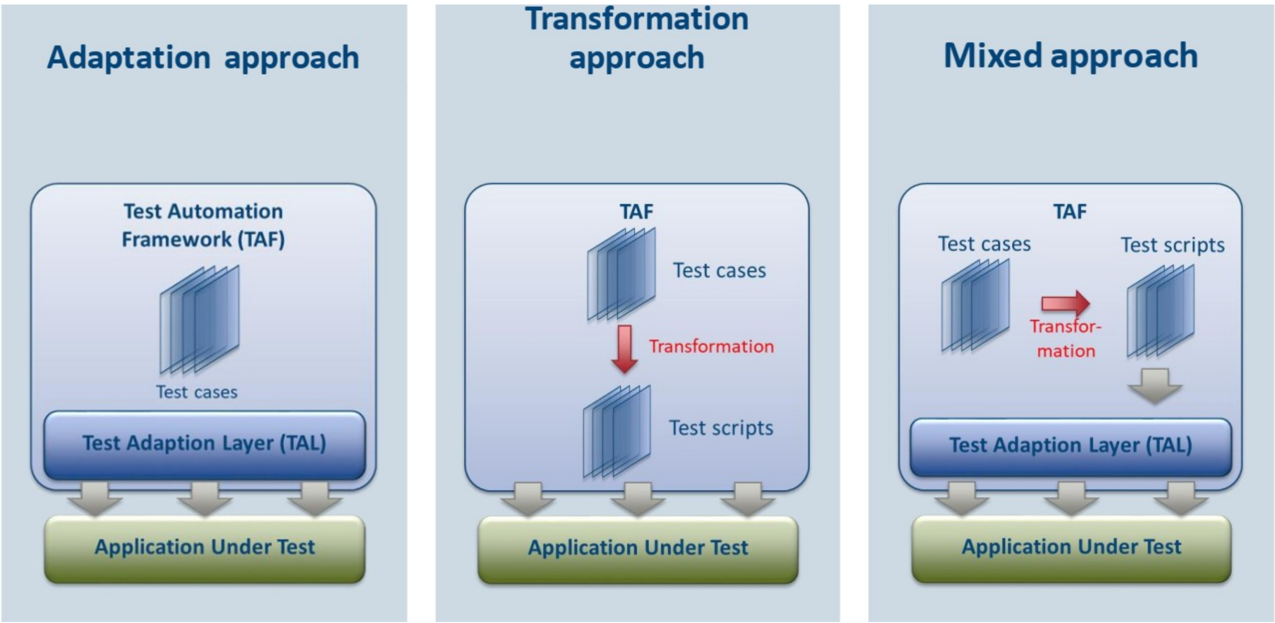
```
1.Entering "TestUser" in the
  'User' field
2.Entering "test" in the
  'Password' field
3.Click on the 'Login' button
4.Search form is displayed
5.Enter "Basic knowledge
  software testing" in the "Free
  text search" field
6.Click on the "Search" button

...
```

→ Types of Test Case Generation

- **Manual Execution**: Test cases must be understandable and can be read by text editor or exported to a management tool. It is manually executed by testers.
- **Automated Execution**: Test cases must be executable by a test automation tool. Offline and online execution possible.

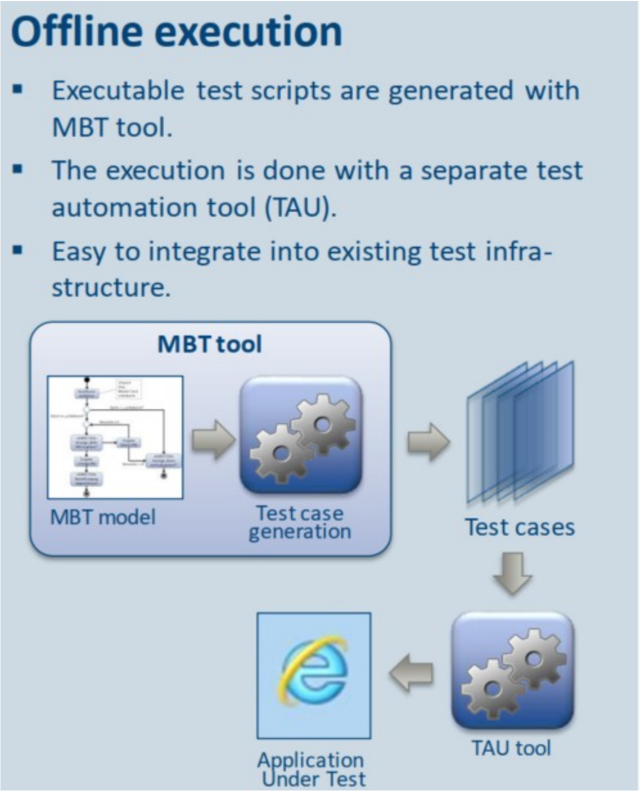
→ Model-Based Testing (MBT) and Automatic Test Execution

- **Test Automation Framework (TAF)**: Consists of function libraries, test data, generated test scripts, test execution tools, and the Test Adaptation Layer (TAL).
- **Variants of TAF**:
  - **Adaptation Approach**: Uses keywords to translate test steps into concrete interactions.
  - **Transformation Approach**: Directly generates executable test scripts.
  - **Mixed Approach**: Combines adaptation and transformation.



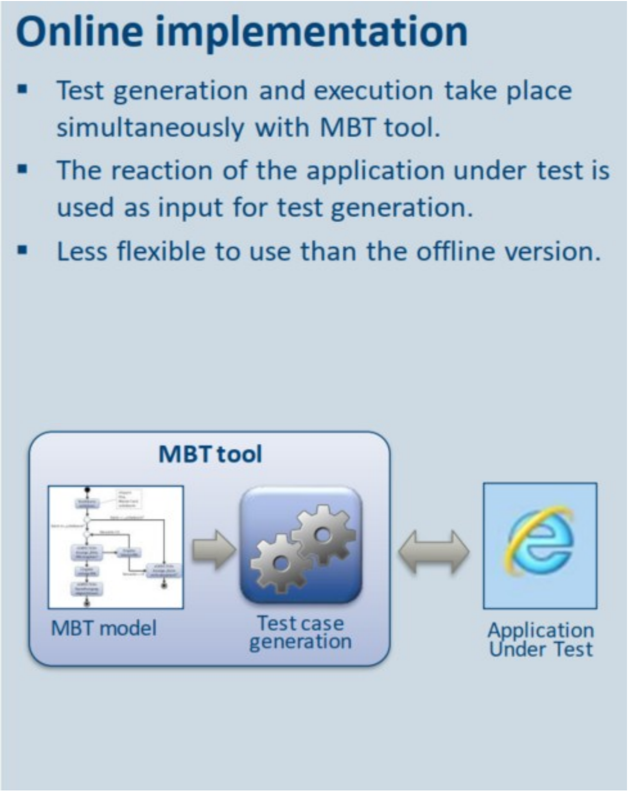
# Offline/Online Test Execution

- **Offline Execution**: Test scripts are generated separately from their execution.
- **Online Execution**: Test generation and execution occur simultaneously, using the application's reactions as input.



Offline execution

- Executable test scripts are generated with MBT tool.
- The execution is done with a separate test automation tool (TAU).
- Easy to integrate into existing test infra-structure.



Online implementation

- Test generation and execution take place simultaneously with MBT tool.
- The reaction of the application under test is used as input for test generation.
- Less flexible to use than the offline version.

# How TAF Supports MBT

1. **Automating Test Case Generation**:
    - MBT tools like Conformiq Creator can generate test scripts directly from the model, reducing manual scripting effort.
2. **Ensuring Consistency and Reusability**:
    - Function libraries and reusable components ensure that the same logic is applied consistently across different test cases.
    - Reduces duplication and maintenance effort.
3. **Managing Test Data**:
    - Centralized management of test data makes it easier to update and maintain.
    - Ensures that tests are run with consistent and accurate data.
4. **Executing Tests Efficiently**:
    - Integration with test execution tools allows for efficient and automated execution of large volumes of test scripts.
    - Supports parallel execution and continuous integration setups.
5. **Handling Changes and Maintenance**:
    - Changes in the MBT model automatically reflect in the generated test scripts, ensuring that the tests are always up-to-date with the latest requirements and functionality.

Example Workflow

1. **Model Creation**:
    - Create an MBT model using tools like Conformiq Creator, defining the test scenarios and conditions.
2. **Test Script Generation**:
    - Use TAF to generate concrete test scripts from the MBT model.
    - The Test Adaptation Layer translates abstract test steps into executable code.
3. **Data Preparation**:
    - Prepare the necessary test data in the test data repository.
4. **Test Execution**:
    - Execute the generated test scripts using the integrated test execution tools.
    - Monitor the execution and collect results.
5. **Review and Maintenance**:
    - Review the test results and update the MBT model or test scripts as needed based on the findings.

# Creating Test Scripts with Conformiq Creator

- **Standard Scripter**: Generates test descriptions and executable test scripts in various formats, such as native script code, Excel sheets, or XML.
- **Restrictions**: Standard scripters cannot generate code for specific UI actions or custom actions; these need manual completion.

# Impact of Changes on MBT

- **Categories of Changes**:
  - **Functional Further Development**: Changes in business processes, requirements, or workflows.
  - **Test Objectives/Conditions**: Changes in the test objectives or conditions.
  - **Technical Further Development**: Changes in the user interface, API, or service interface.

7. Typical Errors in Model Design

- **Wrong Degree of Abstraction**: Using a single activity diagram for multiple aspects leads to complexity.
  - **Correct Approach**: Use hierarchical activity diagrams.
- **Single Test Model Usage**: Avoid using one test model for all aspects to prevent test case explosion.
  - **Correct**: Do not include all aspects to be tested in a single (large) test model!
- **Lack of Traceability**:
  - Problem:
    - **Fragmented Test Models**: Created by different team members.
    - **Complexity**: Hard to maintain an overview of covered requirements/user stories.
    - **Lack of Traceability**: Difficult to ensure all requirements are adequately tested.

Right Approach:

- **Integration with Test Management Tools**:
  - **Action**: Use MBT tools that integrate with test management tools (e.g., JIRA).
  - **Benefit**: Synchronizes requirements between MBT models and test management tools.
- **Linking Requirements in Test Models**:
  - **Action**: Link actions in the test model directly to corresponding requirements.
  - **Benefit**: Ensures clear traceability from model actions to requirements.

- **Test Case Explosion**: Use strategies to prevent generating redundant test cases.

Possible Causes:

1. All aspects to be tested are specified in a single test model.
2. Unnecessarily large amount of test data.
3. Modelling errors.

Consequences:

- High CPU load and memory requirements.
- Creation of redundant test cases.

# False expectations and assumptions

Expectation

• The use of MBT replaces classical test design techniques.

• The MBT tool does all the work.

• The use of MBT enables full automation.

Reality

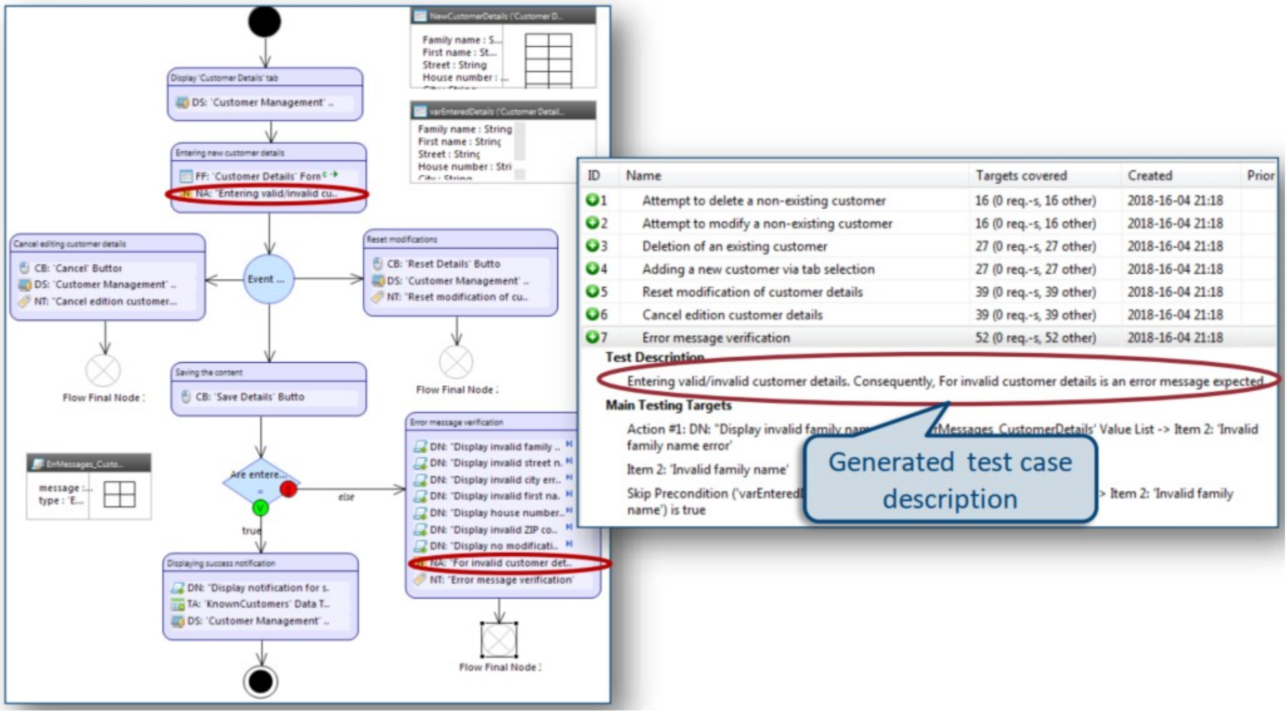
• Test design techniques are needed to specify test data in the test model.

• An MBT tool is just a (useful) tool!

• A mixture of manual and automatic test execution is still recommended.

# Modeling Guidelines

- **Best Practices**:
    - Follow predefined workflow structures.
    - Organize activity diagrams hierarchically.
    - Use speaking names for activity nodes, e.g., "Entering Customer Details".
    - Link verification actions to requirements.
    - Use narrative actions to generate test case descriptions.


Display 'Customer Details' tab
DS: 'Customer Management' ..
NewCustomerDetails ('Customer D...
Family name : S...
First name : St...
Street : String
House number : ...
varEnteredDetails ('Customer Detail...
Family name : String
First name : String
Street : String
House number : Stri
Entering new customer details
FF: 'Customer Details' Form
NA: 'Entering valid/invalid cu..
Cancel editing customer details
CB: 'Cancel' Button
DS: 'Customer Management' ..
NT: 'Cancel edition customer...
Event ...
Reset modifications
CB: 'Reset Details' Butto
DS: 'Customer Management' ..
NT: 'Reset modification of cu..
Flow Final Node :
Saving the content
CB: 'Save Details' Butto
Flow Final Node :
ErrMessages_Custo...
message :...
type : 'E...
Are entere...
else
true
Error message verification
DN: 'Display invalid family ..
DN: 'Display invalid street n.
DN: 'Display invalid city err..
DN: 'Display invalid first na.
DN: 'Display house number..
DN: 'Display invalid ZIP co..
DN: 'Display no modificati..
NA: 'For invalid customer det..
NT: 'Error message verification'
Displaying success notification
DN: 'Display notification for s.
TA: 'KnownCustomers' Data T..
DS: 'Customer Management' ..
Flow Final Node :
Generated test case description

ID	Name	Targets covered	Created	Prior
1	Attempt to delete a non-existing customer	16 (0 req.-s, 16 other)	2018-16-04 21:18	
2	Attempt to modify a non-existing customer	16 (0 req.-s, 16 other)	2018-16-04 21:18	
3	Deletion of an existing customer	27 (0 req.-s, 27 other)	2018-16-04 21:18	
4	Adding a new customer via tab selection	27 (0 req.-s, 27 other)	2018-16-04 21:18	
5	Reset modification of customer details	39 (0 req.-s, 39 other)	2018-16-04 21:18	
6	Cancel edition customer details	39 (0 req.-s, 39 other)	2018-16-04 21:18	
7	Error message verification	52 (0 req.-s, 52 other)	2018-16-04 21:18	

Test Description

Entering valid/invalid customer details. Consequently, For invalid customer details is an error message expected

Main Testing Targets

Action #1: DN: "Display invalid family na ... rMessages_CustomerDetails' Value List -> Item 2: 'Invalid family name error'

Item 2: 'Invalid family name'

Skip Precondition ('varEntered ... > Item 2: 'Invalid family name') is true

# Software Quality Management

# **Definition:** The quality of a system is the degree to which the system satisfies the stated and implied needs of its various stakeholders, and thus provides value.

# Software Quality Management

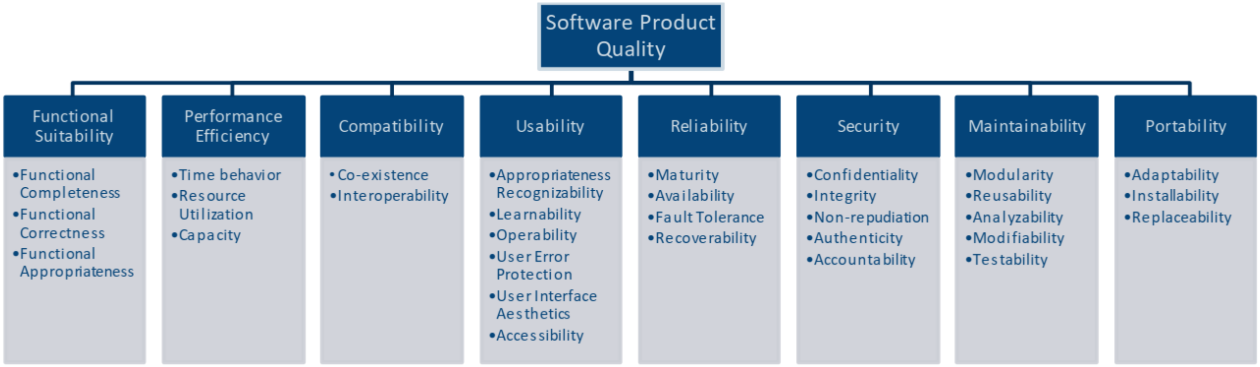
Software Quality Management (SQM) is a comprehensive approach to ensure that the software products meet the specified quality requirements and standards. It involves a combination of activities and processes aimed at managing and improving the quality of software throughout its development life-cycle.

- **Quality Planning:**
  - **Purpose:** Define and plan quality requirements and compliance methods.
  - **Focus:** Prevention of defects.
  - **Nature:** Proactive and strategic.
- **Quality Assurance (QA):**
  - **Purpose:** Ensure processes are adequate to meet quality requirements.
  - **Focus:** Process-oriented; prevention of defects.
  - **Nature:** Proactive and continuous improvement.
- **Quality Control (QC):**
  - **Purpose:** Identify and correct defects in the product.
  - **Focus:** Product-oriented; detection of defects.
  - **Nature:** Reactive and focused on testing.

Example:

- **Quality Planning:** Defining coding standards and performance benchmarks.
- **Quality Assurance:** Implementing peer code reviews and automated unit testing processes.
- **Quality Control:** Conducting functional testing to identify bugs and defects in the software product.

# The ISO/IEC 25010 standard provides a comprehensive model for evaluating the quality of software products. Here’s a summary of its key characteristics:



1.**Functional Suitability:** Measures how well the software meets the specified needs under defined conditions.

2.**Performance Efficiency**: Assesses the performance in relation to the resources used under stated conditions.

3.**Compatibility**: Evaluates the ability of the software to exchange information and function with other systems or components in the same environment.

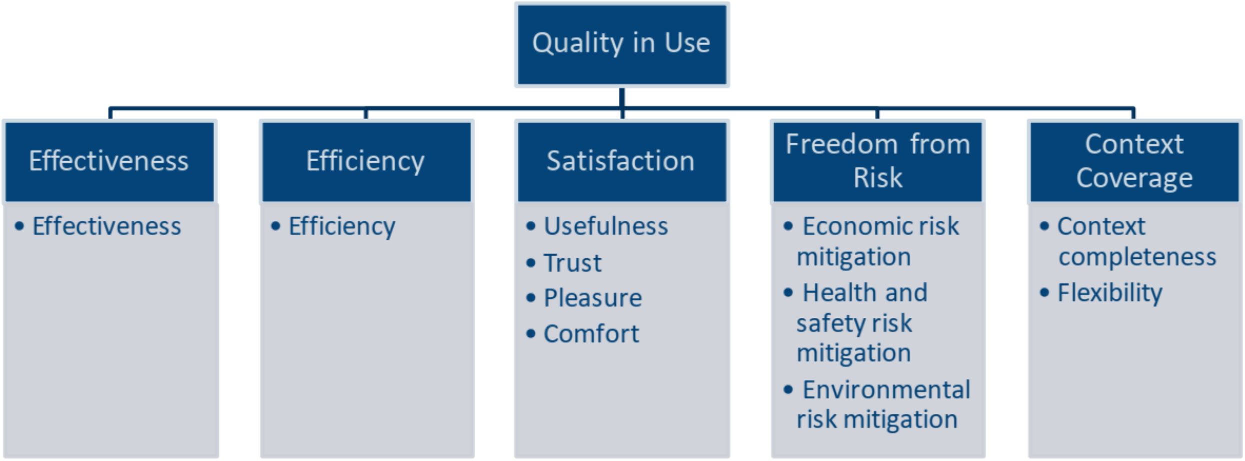
4.**Usability**: Determines how effectively, efficiently, and satisfactorily the software can be used by specified users to achieve specific goals.

5.**Reliability**: Measures the software's ability to perform its required functions under specified conditions for a defined period.

6.**Security**: Assesses how well the software protects information and data, ensuring appropriate access levels.

7.**Maintainability**: Evaluates the ease with which the software can be modified to improve, correct, or adapt to changes.

8.**Portability**: Measures the ease with which the software can be transferred from one environment to another.



# ISO/IEC/IEEE 12207 Software Life Cycle Processes:

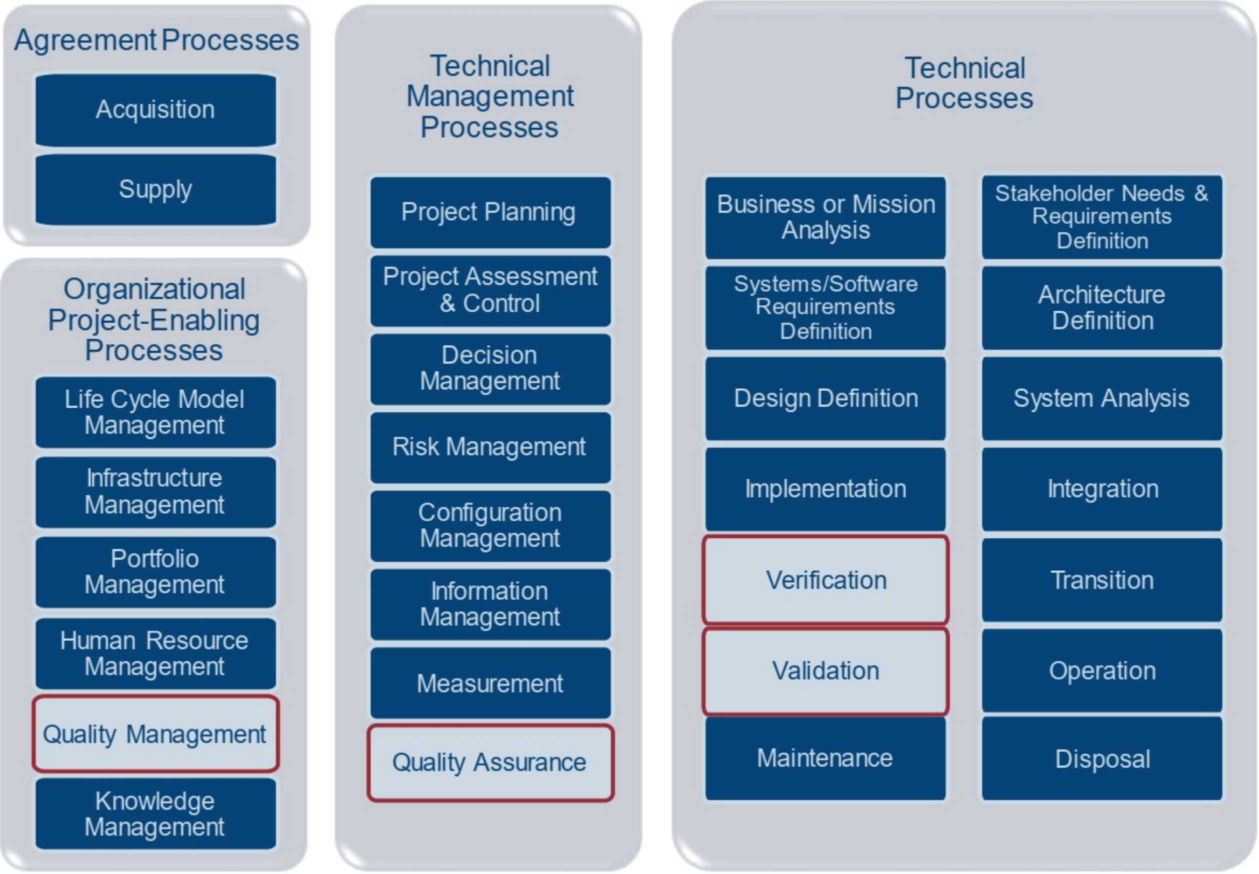
Overview:

- **Purpose:** Defines a comprehensive framework for software life cycle processes using well-defined terminology.
- **Scope:** Covers all key processes and their relationships at a high level of abstraction.
- **Flexibility:** Does not prescribe a specific software lifecycle model or development methodology.

Process Groups:

1. **Agreement Processes:**
    - **Focus:** Establishing agreements between organizations or between a customer and supplier.
    - **Key Activities:** Contract management, supplier agreement management.

2. **Organizational Project-Enabling Processes:**
    - **Focus:** Providing support and infrastructure to projects within the organization.
    - **Key Activities:** Infrastructure management, quality management, knowledge management.
3. **Technical Management Processes:**
    - **Focus:** Managing and controlling the technical aspects of projects.
    - **Key Activities:** Project planning, project assessment, decision management, risk management.
4. **Technical Processes:**
    - **Focus:** The actual development and maintenance of software products.
    - **Key Activities:** Requirements definition, design, implementation, testing, maintenance.



Structure:

- **Processes:** Each process is a collection of activities.
- **Activities:** Each activity comprises a set of tasks.

# Deming Cycle, also known as the PDCA (Plan-Do-Check-Act) Cycle, is a continuous loop used for process improvement and quality management. Here’s a breakdown of each stage:

1.**Plan:**

•Analysis of the current state: Understand the existing process or situation.

•Identification of improvement potential: Identify areas where improvements can be made.

•Development of new concepts: Create new strategies or solutions to address the identified improvements.
2.**Do**:

•Implementation: Execute the plan and implement the new concepts or changes. This stage is about putting the plan into action.
3.**Check**:

•Review and analysis of results: Evaluate the outcomes of the implementation.

•If goals were not met, return to “Plan”: If the desired results are not achieved, go back to the planning stage to refine and improve the plan.
4.**Act**:

•Roll out the new concept/process: If the implementation is successful, standardize the new process and fully integrate it into regular operations.

# ISO/IEC 15504 (SPICE)

SPICE:

- **Full Form:** Software Process Improvement and Capability Determination.
- **Purpose:** Evaluates and improves software processes within an organization.

Key Components:

1. **Process Improvement:** Enhances processes for better performance and quality.
2. **Capability Determination:** Assesses process capability to meet goals and requirements.

Process Assessment:

- Evaluates maturity and capability of software processes.
- Focuses on process definition, management, and optimization.

Improvement Framework:

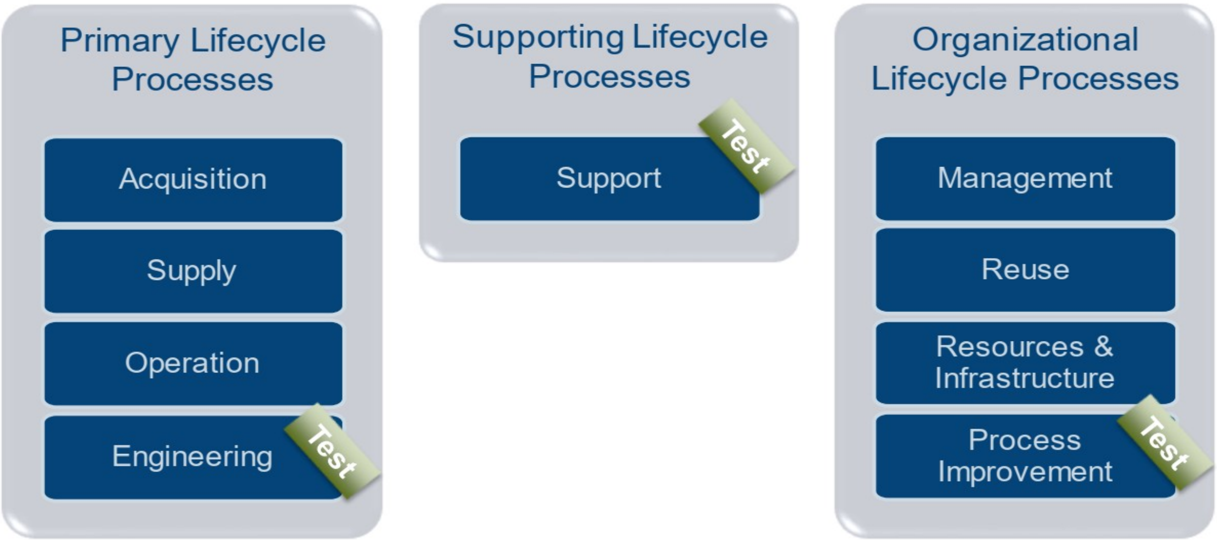
- Provides guidelines for continuous process improvement.
- Enhances software development practices.

Capability Levels:

- **Basic:** Unpredictable and reactive processes.
- **Optimized:** Continuously improved processes based on feedback.

Benefits:

1. **Enhanced Quality:** Produces higher quality software products.
2. **Increased Efficiency:** Streamlined processes reduce development time.
3. **Better Risk Management:** Early identification and mitigation of risks.



Primary Lifecycle Processes

- Acquisition
- Supply
- Operation
- Engineering — Test

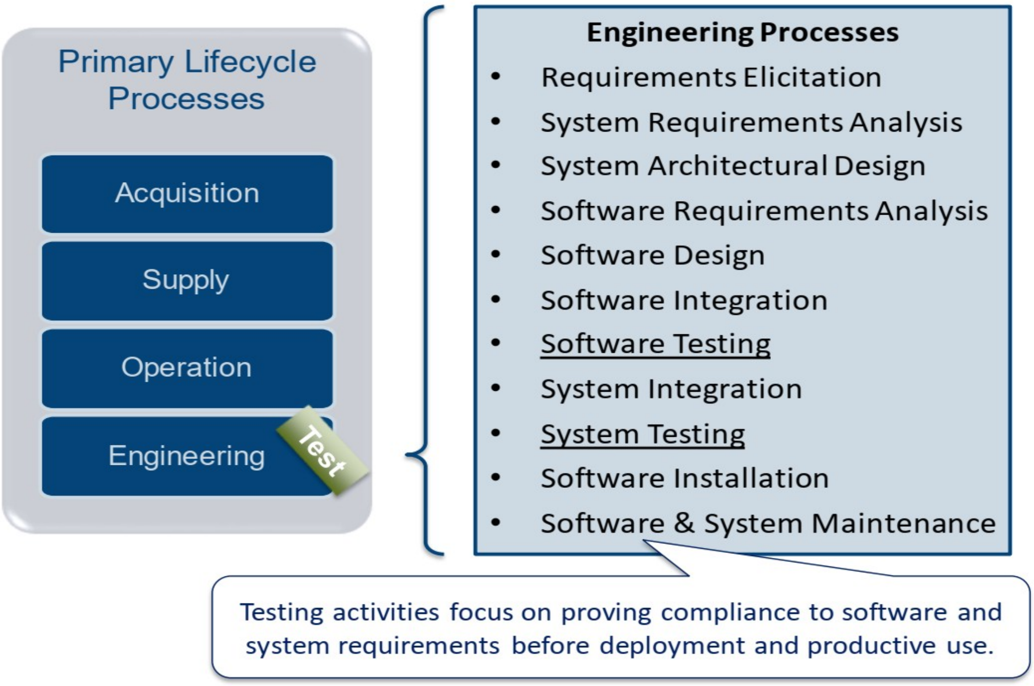
Supporting Lifecycle Processes

- Support — Test

Organizational Lifecycle Processes

- Management
- Reuse
- Resources & Infrastructure
- Process Improvement — Test

Process Categories and Groups



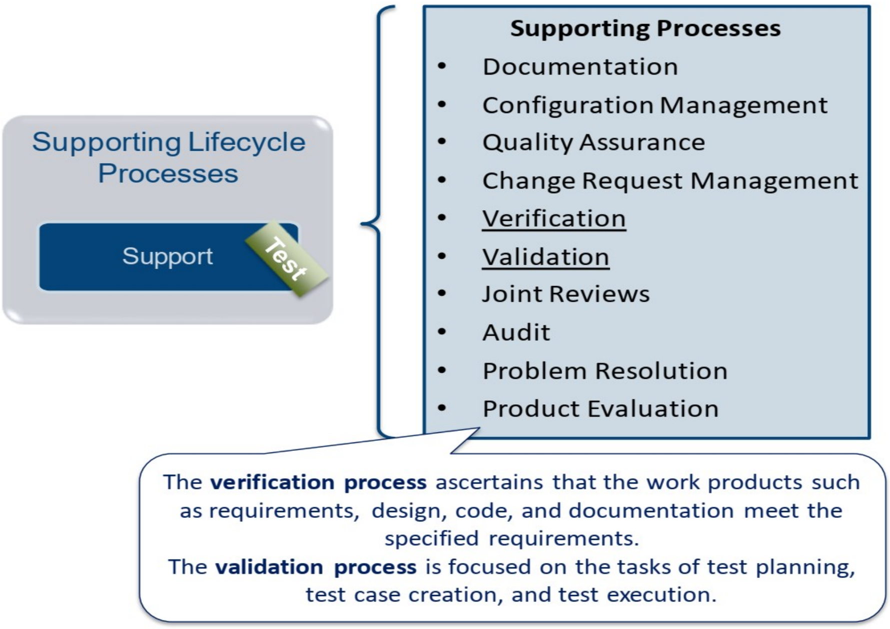
Primary Lifecycle Processes

- Acquisition
- Supply
- Operation
- Engineering — Test

Engineering Processes

- Requirements Elicitation
- System Requirements Analysis
- System Architectural Design
- Software Requirements Analysis
- Software Design
- Software Integration
- Software Testing
- System Integration
- System Testing
- Software Installation
- Software & System Maintenance

Testing activities focus on proving compliance to software and system requirements before deployment and productive use.



Supporting Lifecycle Processes

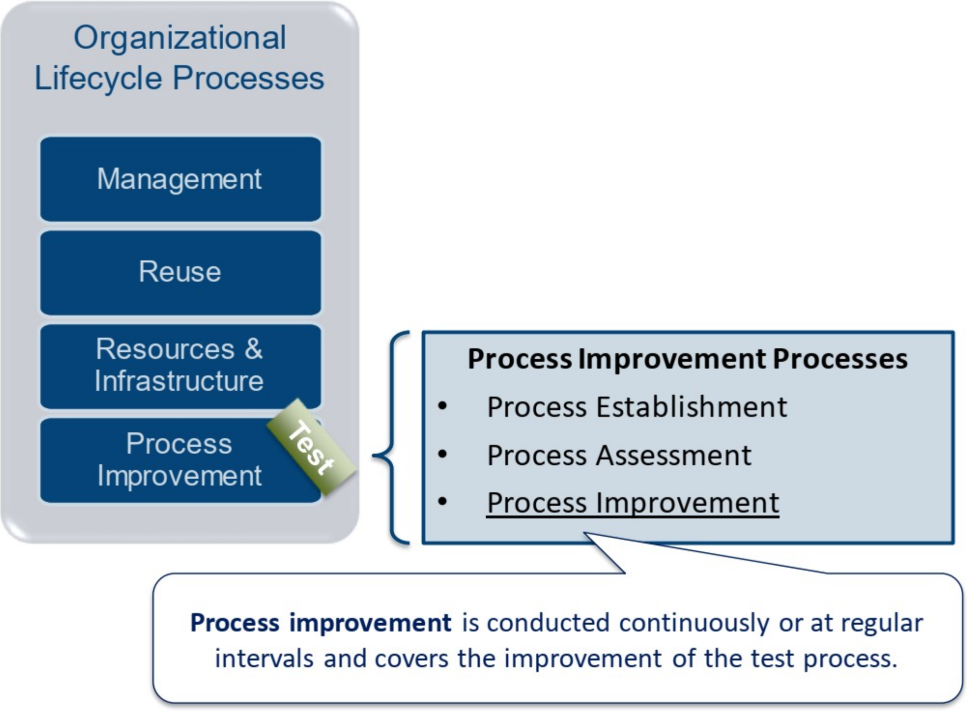
- Support — Test

Supporting Processes

- Documentation
- Configuration Management
- Quality Assurance
- Change Request Management
- Verification
- Validation
- Joint Reviews
- Audit
- Problem Resolution
- Product Evaluation

The **verification process** ascertains that the work products such as requirements, design, code, and documentation meet the specified requirements.

The **validation process** is focused on the tasks of test planning, test case creation, and test execution.

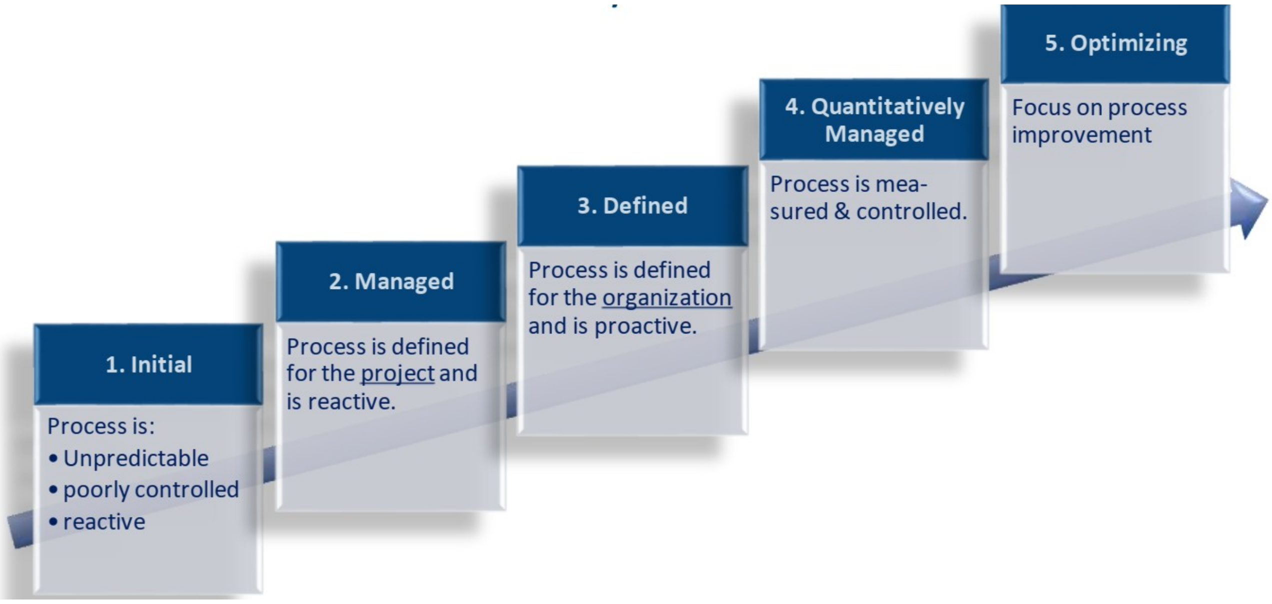

Organizational Lifecycle Processes
Management
Reuse
Resources & Infrastructure
Process Improvement
Test
Process Improvement Processes
• Process Establishment
• Process Assessment
• Process Improvement
Process improvement is conducted continuously or at regular intervals and covers the improvement of the test process.

# **The Capability Maturity Model Integration (CMMI)** is a process improvement framework designed to help organizations improve their processes and ultimately achieve better results.

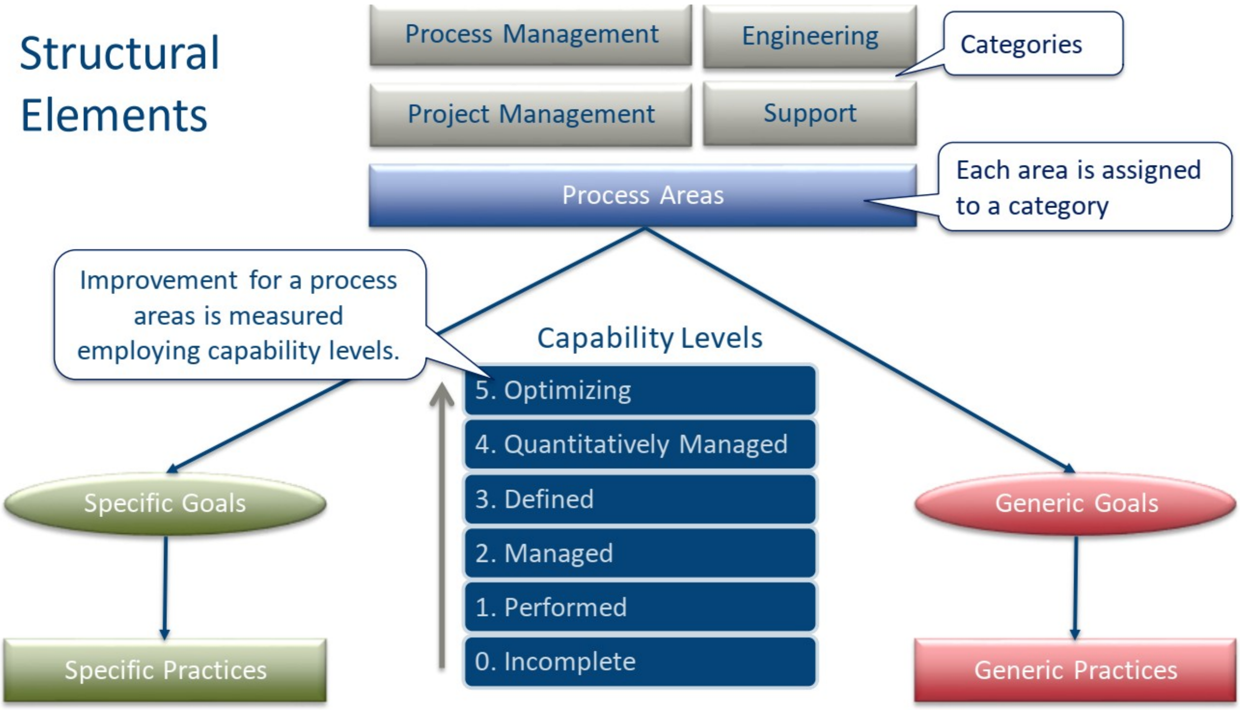
It's a successor to the Capability Maturity Model (CMM), developed at Carnegie Mellon University in 1987. Both models provide guidelines for choosing appropriate improvement strategies.

Key Features:

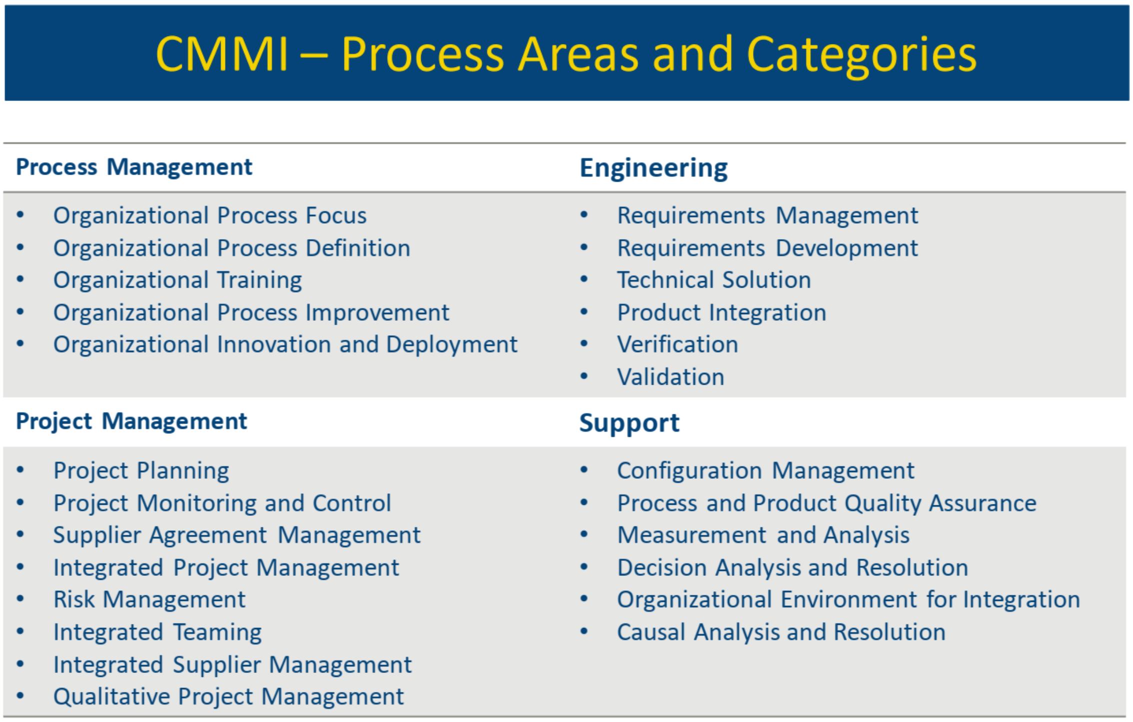
- **Process Improvement Framework:** CMMI provides a structured approach for organizations to assess their current processes, identify areas for improvement, and implement changes to achieve higher levels of maturity.
- **Two Representations:**
  - **Staged Representation:** This representation defines five levels of process maturity, ranging from "Initial" to "Optimizing." It's backward compatible with the original CMM.


1. Initial
Process is:
• Unpredictable
• poorly controlled
• reactive
2. Managed
Process is defined for the project and is reactive.
3. Defined
Process is defined for the organization and is proactive.
4. Quantitatively Managed
Process is mea-sured & controlled.
5. Optimizing
Focus on process improvement

- **Continuous Representation:** This representation allows organizations to focus on specific areas of interest, such as requirements management, software development, or project management.



- Application Areas (Constellations): CMMI is tailored to different application areas, including:
  - CMMI-DEV: Product and Service Development
  - CMMI-SVC: Service Establishment and Management
  - CMMI-ACQ: Product and Service Acquisition



CMMI – Process Areas and Categories

Process Management	Engineering
• Organizational Process Focus	• Requirements Management
• Organizational Process Definition	• Requirements Development
• Organizational Training	• Technical Solution
• Organizational Process Improvement	• Product Integration
• Organizational Innovation and Deployment	• Verification
	• Validation
Project Management	**Support**
• Project Planning	• Configuration Management
• Project Monitoring and Control	• Process and Product Quality Assurance
• Supplier Agreement Management	• Measurement and Analysis
• Integrated Project Management	• Decision Analysis and Resolution
• Risk Management	• Organizational Environment for Integration
• Integrated Teaming	• Causal Analysis and Resolution
• Integrated Supplier Management	
• Qualitative Project Management	