ISTQB – Foundation Level

CHAPTER 4: TEST DESIGN TECHNIQUES

Prepared by: Vu Nguyen
April 2010
Updated by: Minh Ha
Dec 2012

AGENDA

- 4.1 Test development process (K2)
- 4.2 Categories of test design techniques (K2)
- 4.3 Specification-based or black-box techniques (K3)
- 4.4 Structure-based or white-box techniques (K3)
- 4.5 Experience-based techniques (K2)
- 4.6 Choosing test techniques (K2)

4.1 Test development process

- Dynamic testing:
  - Execution of the test object on a computer
  - The test object (program) must be executable
- Test bed:
  - Also called as Test environment
  - Platform or environment where testing is being done
  - Testers must often setup test bed
- Test design technique:
  - Systematic approach at determination of the test cases
  - To verify as many requirements as possible with as little expense as possible

4.1 Test development process

- Conditions, preconditions, and goals:
  - At the beginning, test basis is analyzed to determine what must be tested -> test condition
  - Determine the test objectives for demonstrating that requirements are met
  - Failure risk should especially be taken into account
- Traceability:
  - Exists between specifications and test cases
  - Allows impact analysis of the effects of other specifications on the test process
  - Helps to identify where many of test cases overlap
  - Identify test cases affected by changes in specifications

4.1 Test development process

- Test case specification:
  - Consists of a set of input values, preconditions, expected results and post-conditions
  - Developed to cover certain test conditions
  - Determined by test design technique
  - Preconditions, expected results and expected postconditions are important in determining if there is a failure
- Expected result/behavior:
  - Part of a test case and include outputs, changes to data and states, and any other consequences of the test.
  - Should ideally be defined prior to test execution

4.1 Test development process

- Test scenario (test sequence, test procedure spec):
  - Test cases should be grouped in such a way that a whole sequence of test cases is executed
  - Test script can be required
- Black box and white box techniques:
  - Several different approaches are available for testing the test object
  - Can be categorized into two groups: black box and white box testing
  - Are test case design techniques
  - Support the identification of the respective test cases

4.1 Test development process

- Black box and white box techniques:
  - Black box:
    - Test object is seen as a black box
    - Test cases are designed based on specification or requirements of test object
    - Behavior of test object is watched from the outside
    - Operating sequence of the test object can only be influenced by choosing appropriate input test data or by setting appropriate preconditions
    - Used for higher levels of testing
    - Any test design before the code is written (test-first programming, test-driven development) is black box driven

4.1 Test development process

- Black box and white box techniques:
  - White box:
    - Sometimes called “glass box testing” or “open box testing” or structural testing
    - The source code is known and used for test design
    - While executing test cases, internal processing of the test object and output is analyzed
    - Test cases are designed to cover the program structure of the test object
    - Can be applied at the lower levels of the testing, i.e., component and integration test

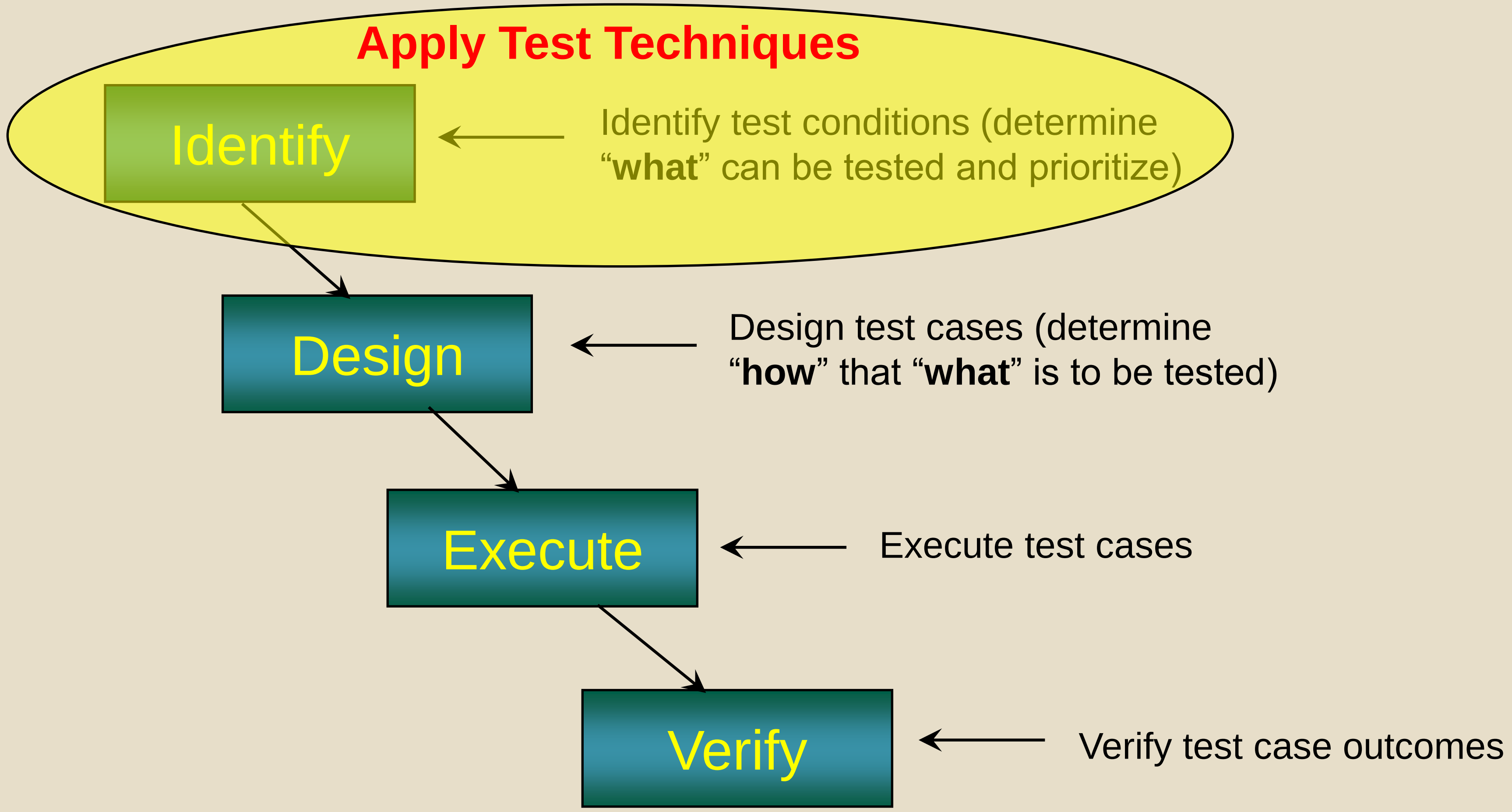
4.2 Categories of test design techniques

- Objectives
  - LO-4.2.1 Recall reasons that both specification-based (black-box) and structure-based (white box) approaches to test case design are useful, and list the common techniques for each. (K1)
  - LO-4.2.2 Explain the characteristics and differences between specification-based testing, structure-based testing and experience-based testing. (K2)

Introduction of Test Technique

- A test technique is a formalized approach to choose test conditions that give a high probability of finding defects.
- A testing technique helps us select a good set of tests from the total number of all possible tests for a given system.

Life Cycle of Test Cases



Identify test conditions (determine "**what**" can be tested and prioritize)

Design test cases (determine "**how**" that "**what**" is to be tested)

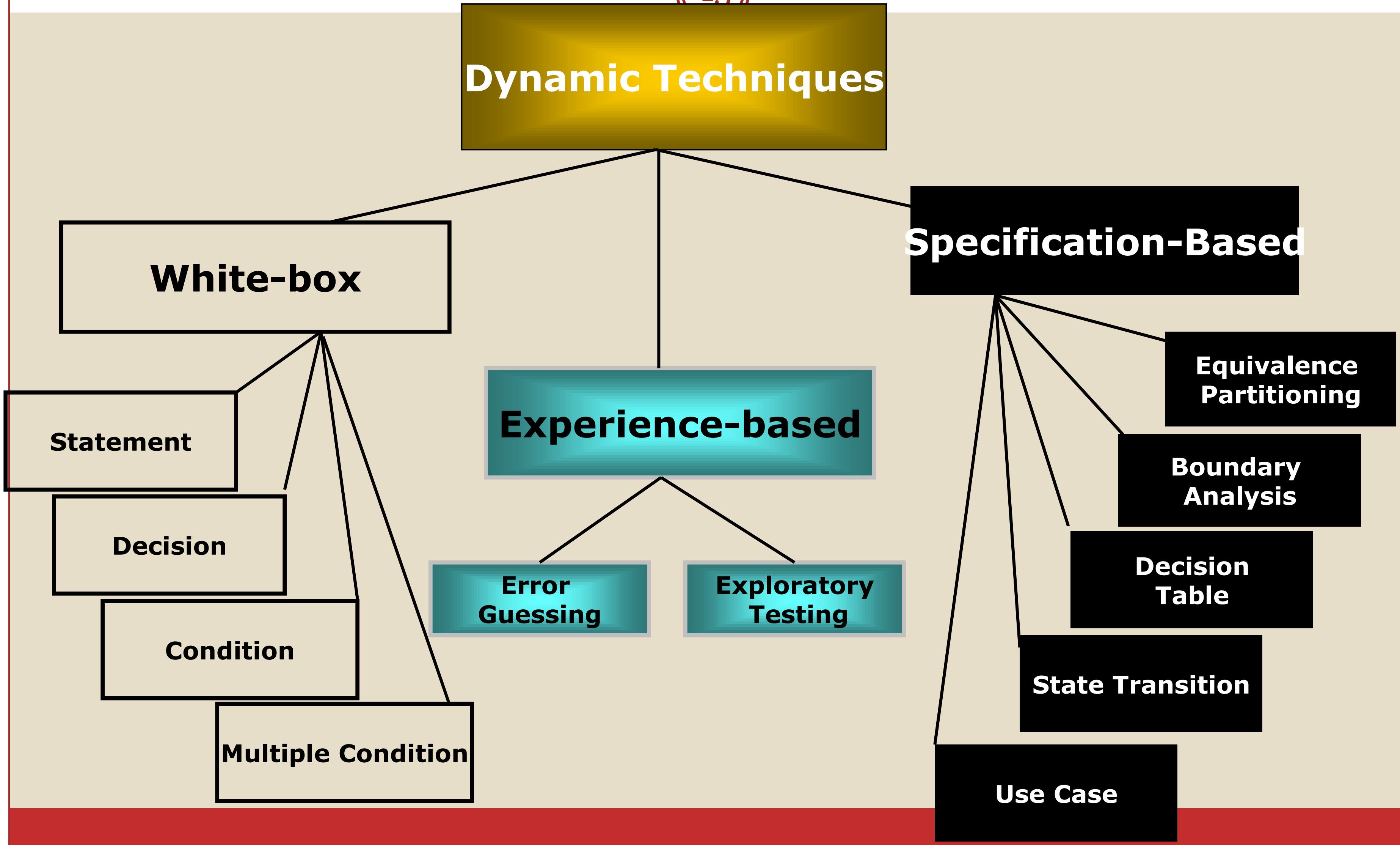
Execute test cases

Verify test case outcomes

Categories of test design techniques

- Categories of testing techniques
  - Static techniques → Chapter 3
  - Dynamic
    - Structure-based
    - Specification-based
    - Experience-based

Dynamic Testing



4.3 Specification-based techniques

- Objectives
  - LO-4.3.1 Write test cases from given software models using the following test design techniques: (K3)
    - equivalence partitioning;
    - boundary value analysis;
    - decision table testing;
    - state transition testing.
  - LO-4.3.2 Understand the main purpose of each of the four techniques, what level and type of testing could use the technique, and how coverage may be measured. (K2)
  - LO-4.3.3 Understand the concept of use case testing and its benefits. (K2)

Specification-based or black-box techniques

- Alias: Specification-based, Behavioral technique
- It is a procedure to select test cases based on software specification without reference to its internal structure.
- Black-box test techniques are appropriate at all levels of testing where a specification exists.

Specification-based or black-box techniques

- Black-box Techniques
  - Equivalence Partitioning
  - Boundary Value Analysis
  - Decision Table
  - State Transition

Equivalence Class Partitioning

- Input domains are divided into equivalence classes:
  - Equivalence class is a group of data values where tester assumes that test object processes them in the same way
  - The test of one representative of the equivalence class is seen as sufficient
  - Besides equivalence classes for correct input, those for incorrect input values must be tested as well
- Equivalence classes with incorrect values:
  - Besides correct input values, incorrect values must be tested
  - Equivalence classes for incorrect input values must be derived
  - Test cases with representatives of these classes must be executed

Equivalence Class Partitioning

- Example:

A function to calculate Christmas bonus of employees depending on the affiliation to the company. Here is description of the requirements:

"Employees receive a Christmas bonus equal to 50% of their monthly income if they have been working for the company for more than three years, employees who have been employed for more than five years receive a 75% bonus, and those with more than eight years of employment are awarded a 100% bonus."

Equivalence Class Partitioning

- Four different equivalence classes with correct input values can be derived by considering the length of employment

Table 5-1. Correct equivalence classes and representatives

Parameter	**Equivalence classes**	**Representative values**
Bonus calculation program, duration of employment in years	vEC1: 0 <= x <= 3 vEC2: 3 < x <= 5 vEC3: 5 < x <= 8 vEC4: x > 8	2 4 7 12

Equivalence Class Partitioning

- There are the following two equivalence classes with invalid values

Table 5-2. Invalid equivalence classes and representatives

Parameter	**Equivalence classes**	**Representative values**
Bonus calculation program, duration of employment in years	iEC1: x < 0 ("Negative" – thus incorrect – staff membership in a company)	-3
	iEC2: x > 70 (Unrealistically long and incorrect staff membership in a company[39])	80

Equivalence Class Partitioning

- The same principle of dividing into equivalence classes can also be used for:
  - Output data
  - Behavior
- Equivalence classes of input values, which are not basic data types:
  - Besides basic data types, data structures and sets of objects can occur
  - Must be decided in each case with which representative values to execute the test case

Equivalence Class Partitioning

- Example: A traveler can be a child, a teenager, an adult, a student, a person on welfare, or a retired person.
- Test object is to calculate the fare, and depends on the type of person:
  - 6 different test cases for the traveler must be provided
  - Fare is calculated differently for each traveler
  - Details must be looked up in the requirements
- Test object to handle seat reservation:
  - Sufficient to choose only one representative, e.g., an adult, for the traveler

Equivalence Class Partitioning

- Combination of the representatives:
  - Usually, the test object has more than one input parameter
  - A decision must be made about which of the available values should be combined together to form test cases
  - The number of "valid" test cases is the product of the number of valid equivalence classes per parameter
  - Need to restrict the number of test cases:
    - Combine test cases and sort by frequency of occurrence (typical usage profile)
    - Prioritize test cases in this order
    - Only the "relevant" test cases are tested
    - Ensure that every representative of an equivalence class appears in at least one test case

Equivalence Class Partitioning

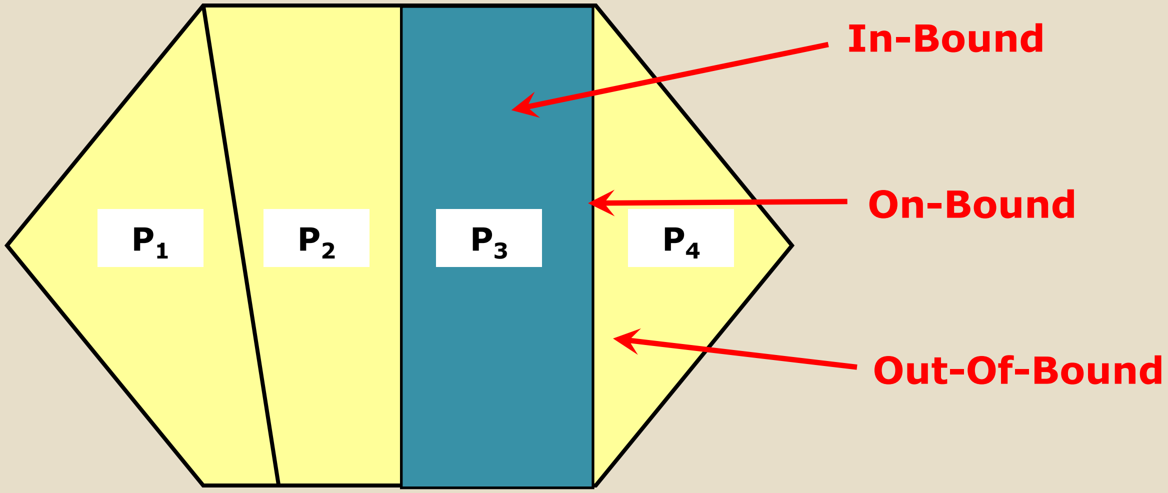
- Test invalid values separately:
  - An incorrect value should only be combined with “correct” ones
  - This is usually independent of values of other parameters
  - If a test case combines more than one incorrect value, defect masking may result

Boundary Value Analysis

- Boundary Value Analysis is a test technique in which test cases are designed based on boundary values which are the maximum and minimum values of a partition
- The tests can be designed to cover both valid and invalid boundary values:
    - Valid partition -> Valid boundary values
    - Invalid partition -> Invalid boundary values
- Test cases are designed to cover each boundary value
- Boundaries not existing for sets

Boundary Value Analysis

- Boundaries define 3 sets of data:
  - Good (In-Bound)
  - Bad (Out-Of-Bound)
  - On-the-border (On-Bound)
- It is easy to apply and its defect finding capability is high.



Boundary Value Analysis

Example:

A printer has an input option of the number of copies to be made, from 1 to 99.

	Invalid	Valid	Invalid
Partition	<1 (page)	1 – 99 (page)	> 99 (page)
Boundary Value	0	1 99	100

->*We will have 5 boundary values:* 0, 1, 50, 99, 100.

EP & BVA Relations

- BVA is considered as an extension of EP. Boundary values are identified based on defined partitions.
- It's recommended that you test the partitions separately from boundaries.
- Depend on your test objectives, you can decide to exercise which partitions and boundaries and set their priority if need:
  - **The most thorough approach:** valid partition -> invalid partition -> valid boundaries -> invalid boundaries.
  - **Typical transactions with a minimum number of tests:** valid partition only.
  - **Find as many as defects as possible:** both valid and invalid boundaries.
  - **Ensure that the system will handle bad inputs correctly:** invalid partitions and boundaries.

Decision Table

- Decision Table is a test technique in which test cases are designed to execute the combinations of inputs and/or causes shown in a decision table.
- Decision Table:
  - Capture requirements that contain logical conditions.
  - Provide a systematic way of stating complex business rules.
  - Each column of the table corresponds to a business rule that define a unique combination of conditions.
- Test cases are designed to have at least one test per rule in the table.

Decision Table

- Each combination of conditions in decision table is referred to as a rule and have a corresponding outcome.

Conditions	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**
Condition #1	**T**	**T**	**F**	**F**
Condition #2	**T**	**F**	**T**	**F**
Actions/Outcomes				
Outcome #1	**Y**		**Y**	

- The strength of this technique is discover omissions and ambiguities in specification.

Decision Table

- Example:
- If you are a new customer opening a credit card account, you will get a 15% discount on all your purchases today.
- If you are an existing customer and you hold a loyalty card, you get a 10% discount.
- If you have a coupon, you can get 20% off today (but it can’t be used with the ‘new customer’ discount).
- Discount amounts are added, if applicable.

Decision Table

-> We will have 6 test cases to cover all possible rules in the decision table.

Conditions	R1	R2	R3	R4	R5	R6	R7	R8
New customer (15%)	T	T	T	T	F	F	F	F
Loyalty card (10%)	T	T	F	F	T	T	F	F
Coupon (20%)	T	F	T	F	T	F	T	F
Actions								
Discount (%)	🟨	🟨	20	15	30	10	20	0
	N/A	N/A						

State Transition Testing

- A system may exhibit a different response depending on current conditions or previous history
- State transition testing is much used within the embedded software industry and technical automation in general. However, the technique is also suitable for modeling a business object having specific states or testing screen-dialogue flows
- Example: software installer

State Transition Testing

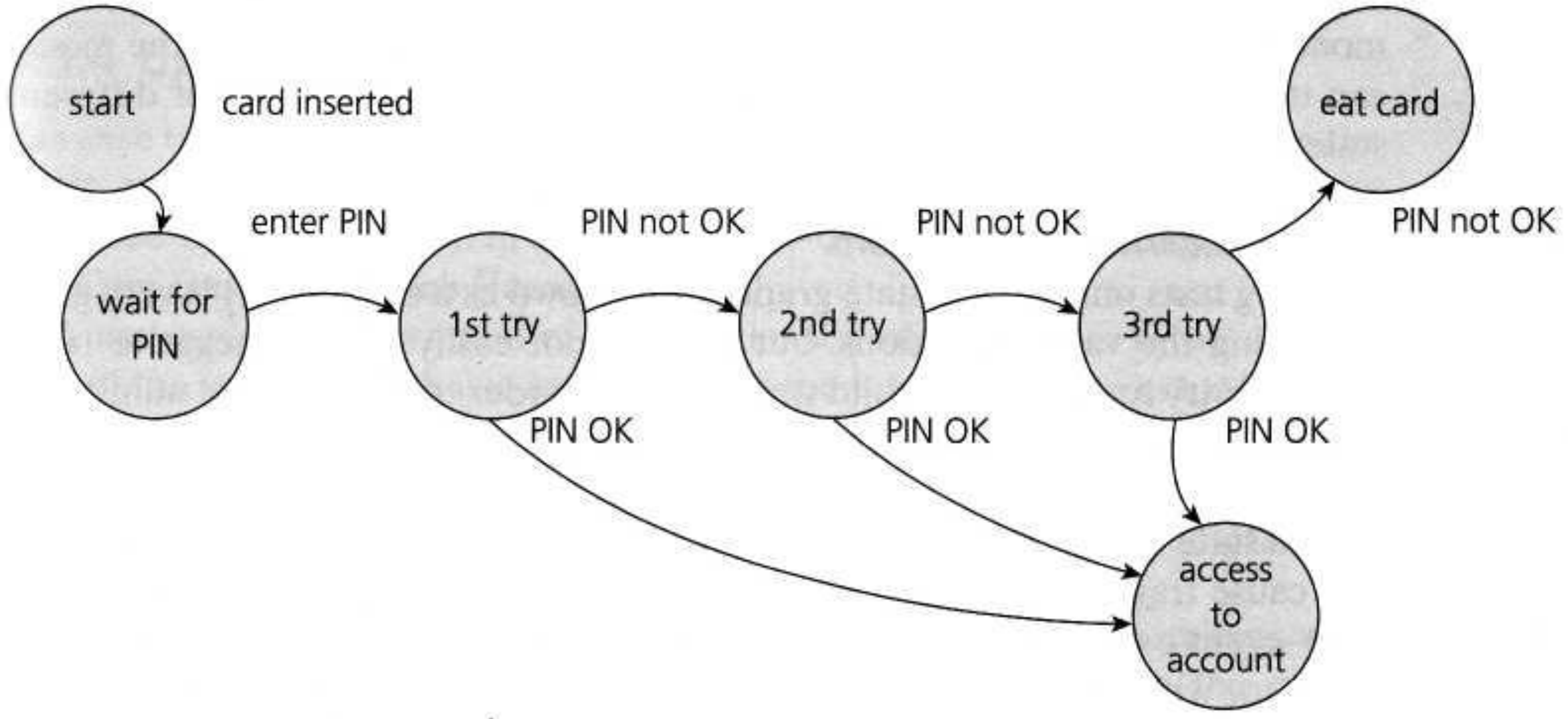


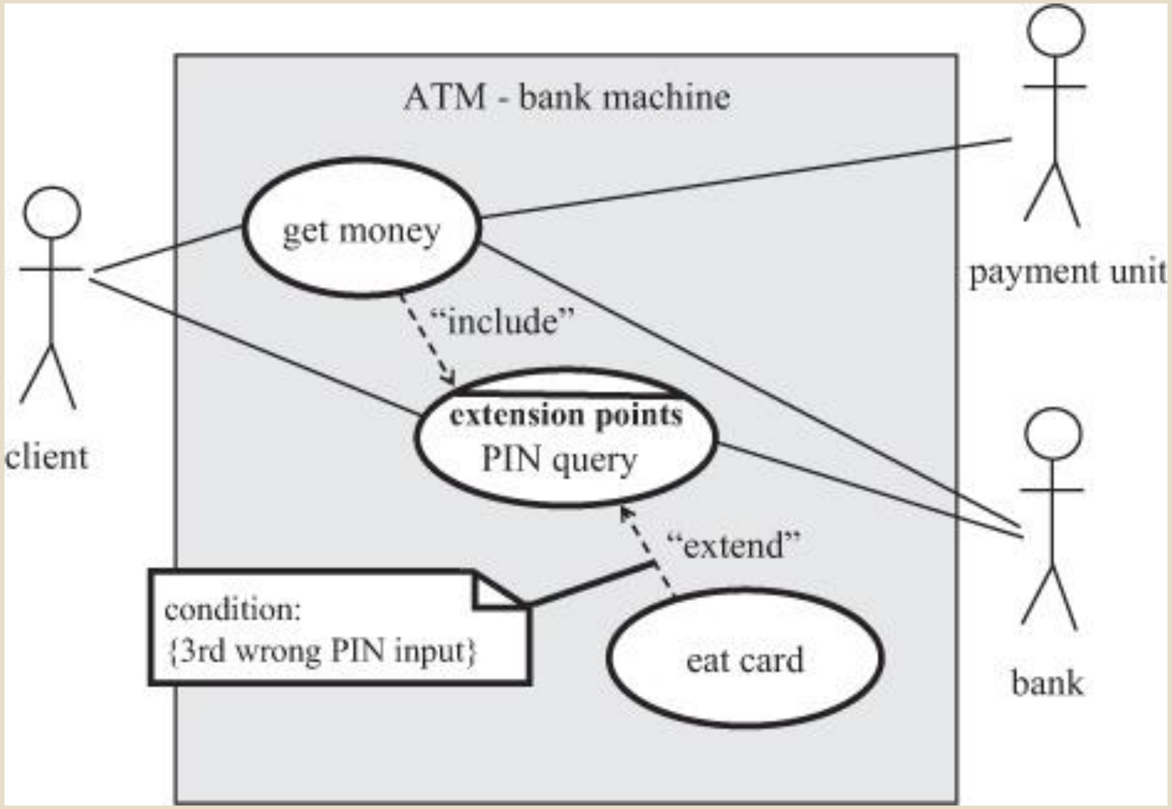
FIGURE 4.2 State diagram for PIN entry

Use Case Testing

- Use case testing is a technique that helps us identify test cases that exercise the whole system on a transaction by transaction basis from start to finish.
- UML is often used
- Use case diagrams serve the purpose of defining requirements on a relatively abstract level and describing typical user system interactions
- Use cases describe the process flows through a system based on its most likely use.

Use Case Testing

- Individual use cases in this example are "Get money", "PIN query", and "Eat card."



4.4 Structure-based or white-box techniques (K3)

- Objectives:
  - LO-4.4.1 Describe the concept and importance of code coverage. (K2)
  - LO-4.4.2 Explain the concepts of statement and decision coverage, and understand that these concepts can also be used at other test levels than component testing (e.g. on business procedures at system level). (K2)
  - LO-4.4.3 Write test cases from given control flows using the following test design techniques:
    - statement testing;
    - decision testing. (K3)
  - LO-4.4.4 Assess statement and decision coverage for completeness. (K3)

Statement Coverage

- Statement Coverage is the assessment of the percentage of executable statements that have been exercised by a test suite.

- Statement testing derives test cases to execute specific statements, normally to increase statement coverage.

Statement Coverage

$$\text{Statement Coverage} = \frac{\text{Number of statements exercised}}{\text{Total number of statements}} \times 100\%$$

- A statement may be on single line or spread over several lines.
- One line may contain more than one statement, just one statement or only part of a statement.

Statement Coverage

Example:

Let's look at a code sample below:

```
1. READ A
2. READ B
3. C = A + 2*B
4. IF C > 50 THEN
5.         PRINT "Large C"
6. END
```

We'll have 3 tests:

- Test 1: A=2, B=3
- Test 2: A=0, B=25
- Test 3: A=47, B=1

Statement Coverage

- Statement Coverage:
  - Test 1: A=2, B=3
  - Test 2: A=0, B=25
  - Test 3: A=47, B=1

⇒ 85% (5/6 statements)

- Increase coverage to 100% ?

⇒ Test 4: A=20, B=25

Pseudo-code:

```
1. READ A
2. READ B
3. C = A + 2*B
4. IF C > 50 THEN
5.     PRINT "Large C"
6. END
```

Decision Coverage

- Decision/Branch Coverage is the assessment of the percentage of decision outcomes (e.g. the True and False options of an IF statement) that have been exercised by a test suite.

- Decision testing derives test cases to execute specific decision outcomes, normally to increase decision coverage.

Decision Coverage

$$\text{Decision Coverage} = \frac{\textbf{Number of decision outcomes exercised}}{\textbf{Total number of decision outcomes}} \times 100\%$$

- A decision is a statement (If, Do-While, Repeat-Until, Case) where there are 2 or more possible exits/outcomes from the statement.
- Decision coverage is stronger than statement coverage: 100% decision coverage guarantees 100% statement coverage, but not vice versa.

Decision Coverage

Example:

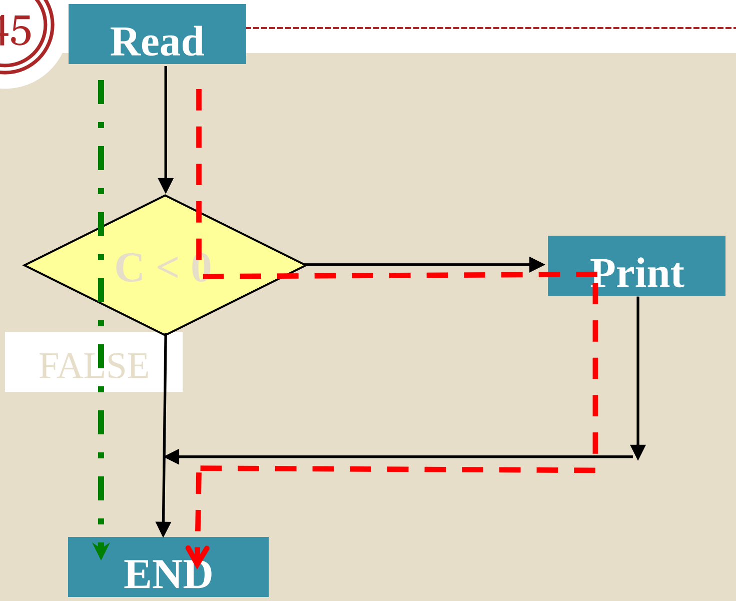
Let’s look at a code sample below:

```
1. READ A
2. READ B
3. C = A - 2*B
4. IF C < 0 THEN
5.     PRINT “C negative”
6. END
```

We’ll have 1 test:

A=20, B=15

Decision Coverage



- Decision Coverage:
  - Test 1: A=20, B=15
  - ⇒ 50% (1/2 decisions)

- Increase coverage to 100% ?
  - ⇒ Test 2: A=10, B=2

Pseudo-code:

```
READ A
READ B
C = A - 2*B
IF C < 0 THEN
    PRINT "C negative"
END
```

Other White-box Techniques

- Some other white-box techniques:
  - Branch coverage = Decision coverage
  - Condition coverage: Atomic condition coverage
  - Multiple condition coverage:
    - Combinations of atomic conditions
    - Includes statement and branch coverage

4.5 Experience-based techniques (K2)

- Objectives:
  - LO-4.5.1 Recall reasons for writing test cases based on intuition, experience and knowledge about common defects. (K1)
  - LO-4.5.2 Compare experience-based techniques with specification-based testing techniques.(K2)

4.5 Experience-based techniques (K2)

- Experienced-based testing is where tests are derived from the tester's skill and intuition and their experience with similar applications and technologies.

- Experience-based techniques are used to complement white-box and black-box techniques and are also used when there is no specification is inadequate or out of date.

Error Guessing

- Error guessing is a test technique where the experience of the tester is used to:
  - Anticipate what defects might be present in the system under test as a result of errors made
  - Design tests specifically to expose these defects

- The defect and failure lists can be built based on:
  - Tester's experience
  - Available defects and failure data
  - Common knowledge about why software fails

Error Guessing

- Some samples of error situations:
  - Initialization of data
  - Wrong kind of data
  - Handling of real data
  - Error management
  - Restart/Recovery
  - Proper handling of concurrent procedure

Exploratory Testing

- Exploratory testing is a test technique where the tester:
  - Actively controls the design of the tests as those tests are performed
  - Uses information gained while testing to design new and better tests.
- This is an approach that is most useful where:
  - There are few or inadequate specifications and severe time pressure
  - Or in order to complement other, more formal testing.

4.6 Choosing test techniques (K2)

- Objectives:
  - LO-4.6.1 List the factors that influence the selection of the appropriate test design technique for a particular kind of problem, such as the type of system, risk, customer requirements, models for use case modeling, requirements models or tester knowledge. (K2)

4.6 Choosing test techniques (K2)

- The choice of which test techniques to use depends on a number of factors:
  - The Type of System
  - Regulatory Standards
  - Customer or Contractual Requirements
  - Level of risk, Type of Risk
  - Test Objective
  - Documentation Available
  - Knowledge of Testers
  - Time and Budget
  - Development Life Cycle...

Summary

- Each test technique has its own benefit:
  - Black-box techniques can find parts of the specification that are missing from the code.
  - White-box techniques can find things in the codes that aren't supposed to be there.
  - Experience-based techniques can find things missing from the code and the specification.

- Using a variety of techniques will help ensure that a variety of defects are found.

References

- Rex Black, Foundations of Software Testing
- ISTQB Foundation Syllabus.pdf
- Slides Software Testing Techniques from TTC

Q & A

Glossary

- **Test basis:** All documents from which the requirements of a component or system can be inferred. The documentation on which the test cases are based. If a document can be amended only by way of formal amendment procedure, then the test basis is called a frozen test basis.
- **Test case:** A set of input values, execution preconditions, expected results and execution postconditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement.
- **Test case specification:** A document specifying a set of test cases (objective, inputs, test actions, expected results, and execution preconditions) for a test item.

Glossary

- **Test design specification:** A document specifying the test conditions (coverage items) for a test item, the detailed test approach and identifying the associated high level test cases.
- **Test design technique:** Procedure used to derive and/or select test cases.
- **Test execution:** The process of running a test on the component or system under test,
- producing actual result(s).
- **Test implementation:** The process of developing and prioritizing test procedures, creating test data and, optionally, preparing test harnesses and writing automated test scripts
- **Test procedure specification:** A document specifying a sequence of actions for the execution of a test. Also known as test script or manual test script.

Glossary

- **Test specification:** A document that consists of a test design specification, test case specification and/or test procedure specification.
- **Specification:** A document that specifies, ideally in a complete, precise and verifiable manner, the requirements, design, behavior, or other characteristics of a component or system, and, often, the procedures for determining whether these provisions have been satisfied. [After IEEE 610]
- **Specification-based testing:** See black box testing.
- **Specification-based technique**: See black box test design technique.
- **Specification-based test design technique:** See black box test design technique.

Glossary

- **Black-box technique:** See black box test design technique.
- **Black-box testing:** Testing, either functional or non-functional, without reference to the internal structure of the component or system.
- **Black-box test design technique:** Procedure to derive and/or select test cases based on an analysis of the specification, either functional or non-functional, of a component or system without reference to its internal structure.

Glossary

- **Structure-based testing:** See white-box testing.
- **Structure-based technique:** See white box test design technique.
- **White-box test design technique:** Procedure to derive and/or select test cases based on an analysis of the internal structure of a component or system.
- **White-box testing:** Testing based on an analysis of the internal structure of the component or system.