MGMT70043 Software Testing

Lesson 4: Test Design Techniques

MGMT70043 LESSON 4

Agenda

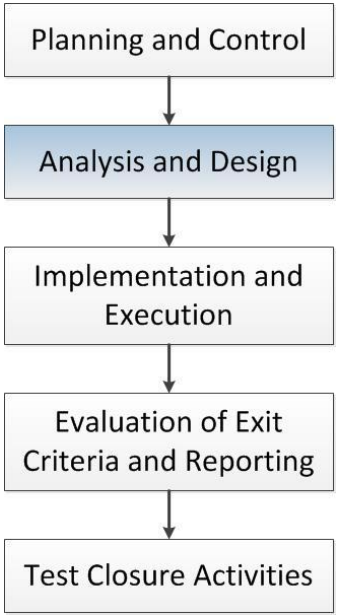
1. Introduction and Overview
    - Quiz 3 based on Lesson 3 (and on textbook chapter 3)
    - Lesson 4: Test Design Techniques

2. Lesson 4: Testing Design Techniques
    - Test development process
    - Test coverage
    - Types of test design techniques
        - Specification-based (black-box) techniques
        - Structural (white-box) techniques
        - Experience-based techniques

MGMT70043 LESSON 4

Test design process

- The test case development process takes place during the Analysis and Design phase of the Fundamental Test Process (FTP).
- It's during the Analysis and Design phase that you review documentation, identify items to be tested, and design the tests, among other things.



MGMT70043 LESSON 4

Test coverage

- **Test coverage** provides a quantitative assessment of the extent and quality of testing.
- Test coverage can be calculated for both black-box and white-box techniques.

- The basic formula is: `(# Coverage Items Tested  /# of Coverage Items) * 100`

Test coverage is important because:

- It provides a way to **measure the quality** of testing (50% of the code vs. 33% of the code).
- It **facilitates estimation**. For example, if it has taken two weeks to test 25% of the use cases, then it will likely take six more weeks to test the remaining 75% of the use cases.
- It can be used as a **test completion criterion** in the test plan. For example, the test plan might say, "We can stop testing once we have tested 100% of the high-priority features and 80% of the medium-priority features."

MGMT70043 LESSON 4

Test design techniques

There are three main categories of test design techniques:

- **Black-box techniques**, also known as **specification-based techniques**. Black-box techniques are based on a review of the test basis documentation, including functional and non-functional requirements.
- **White-box techniques**, also known as **structural techniques**. White-box techniques are based on knowledge of "how" the solution is implemented, including menu structures, interactions between components, and the source code itself. White-box techniques tend to be used at lower levels of testing (unit and component testing).
- **Experience-based techniques**, also known as **ad hoc testing**. Experience-based techniques are based on the tester's own experience with similar systems and/or with testing in general. They tend to be used if there is a time crunch or to supplement the other techniques.

MGMT70043 LESSON 4

Black-box techniques

- A principle of software testing is that exhaustive testing is impossible.
- An efficient test design tests the maximum number of test conditions with as few test cases as possible.
- There are several **black-box techniques** that can be used to increase efficiency while ensuring coverage.
  1. Equivalence partitioning
  2. Boundary value analysis
  3. Decision table testing
  4. State transition testing
  5. Use case testing

MGMT70043 LESSON 4

Equivalence partitioning

- **Equivalence partitioning** is a black-box test design technique
- It is based on the assumption that all the possible input values can be grouped into subsets, and that one input value from each subset can be used to represent the entire subset.
- Each subset is known as an **equivalence partition** or an equivalence class.

Example: let's say the requirements document states the following:

– 1.2 The system shall charge ATM users who have a Basic Plan $1.00 per transaction for the first 10 transactions and $3.00 for each subsequent transaction.

MGMT70043 LESSON 4

Equivalence partitioning

- In order to test the previous example, we could theoretically write dozens of test cases.
- But if we use the equivalence partitioning technique, we see that there are only two **equivalence partitions for valid input values**: 0-10 and >10.
- So we can test the entire requirement with only two input values, say 5 and 15.
- In other words, if we have two test cases (input=5 and input=15), then we would have 100% coverage of this particular requirement.

- We can also define equivalence partitions for **output values**. In this case, there are two equivalence partitions for valid output values: $1.00 and $3.00.
- Similarly, we can define equivalence partitions for **invalid values**. For example, all output values greater than $3.00 would comprise an equivalence partition containing invalid output values.

MGMT70043 LESSON 4

Equivalence partitioning

How to write a test case based on an equivalence partitioning analysis

1. Define the equivalence partitions. Be sure to include both valid and invalid partitions.
2. Write one test case per partition, using a representative value from each partition as the input value

MGMT70043 LESSON 4

In-class exercise

A system is designed to accept values of examination marks as follows:
Fail 0– 39 inclusive
Pass 40– 59 inclusive
Merit 60– 79 inclusive
Distinction 80– 100 inclusive

Which of the following sets of values are all in different equivalence partitions?
(a) 25, 40, 60, 75
(b) 0, 45, 79, 87
(c) 35, 40, 59, 69
(d) 25, 39, 60, 81

MGMT70043 LESSON 4

Boundary value analysis

- **Boundary value analysis** is another black-box test design technique
- It is based on the principle that defects cluster at boundaries.
- Boundary value analysis is often used in conjunction with equivalence partitioning because partitions have boundaries.
- There is a good chance that if the programmer has make a mistake, it is at the boundaries of the partitions.
- We need to identify both valid and invalid boundary values. There are two ways to do this:
  1. The most common way (**two-value approach**) is to identify a valid boundary value, as well as an invalid value, immediately outside the boundary. This is done at each end of the partition.
  2. The **three-value approach** further identifies a valid value inside the boundary. This is also done at each end of the partition.

MGMT70043 LESSON 4

Boundary value analysis

Example: “1.1.3 The fifth time that the user enters an invalid PIN, the system shall keep the card and suspend the account”

In order to define both valid and invalid boundary values, we would do the following:

- Using the more common **two-value approach,** the first partition would have lower boundary values of 0 and 1, and upper boundary values of 4 and 5. At each end of the partition, we have identified a valid value that marks the boundary the partition and an invalid boundary value outside the partition.
- Using the **three-value approach,** the first partition would have lower boundary values of 0, 1 and 2, and upper boundary values of 3, 4 and 5. At each end of the partition, we have identified a valid value that marks the boundary the partition, a valid boundary value just inside the partition, and an invalid boundary value outside the partition.

MGMT70043 LESSON 4

Boundary value analysis

- Sometimes a partition will seem to have no upper boundary (price > $10) but due to technical constraints an upper boundary exists (what if the price = $1,000,000?)

- Boundary values are specified at the level of precision used in the partition
  - For example, let's say we're testing a digital thermometer that is supposed to turn orange if the user has a temperature of 37.8 C.
  - The "orange" partition has a valid boundary value of 37.8, which has a precision of one decimal place.
  - The invalid value immediately outside the partition would be 37.7.
  - The valid value immediately inside the partition would be 37.9.

MGMT70043 LESSON 4

Boundary value analysis

How to write a test case based on a boundary value analysis

1. Identify the boundary values for each partition
2. Write one test case per boundary value, using the boundary value as the input value

MGMT70043 LESSON 4

Group exercise

- A bakery must charge HST if it sells a customer fewer than 12 cupcakes.
- If the bakery sells a customer a dozen or more cupcakes, the transaction is exempt from HST.

1. What would be the boundary values using the two-value approach? The three-value approach?
2. Write a test case using one of the boundary values as the input

MGMT70043 LESSON 4

Decision table testing

- **Decision tables**, also known as cause effect tables, are useful for testing complex combinations of business rules.
- Conditions (inputs) are listed in the top rows; outcomes (outputs) are listed in the bottom rows.
- Each column represents a business rule or scenario. For testing purposes, each column in a decision table becomes a test case.
- To make decision tables more manageable, you can use **limited entry decision tables** showing unique outcomes (outputs) rather than all possible permutations of inputs.

	Business rule 1	Business rule 2	Business rule 3
Condition 1	T	F	T
Condition 2	T	T	T
Condition 3	T	–	F
Action 1	Y	N	Y
Action 2	N	Y	Y

MGMT70043 LESSON 4

Decision table testing

Example 4: Decision table testing

For example, let's say the requirements document states the following:

1.2 The system shall charge ATM users who have a Basic Plan $1.00 per transaction for the first 10 transactions and $3.00 for each subsequent transaction.

1.3 The system shall charge ATM users who have an Enhanced Plan $0.00 per transaction for the first 20 transactions and $2.00 for each subsequent transaction.

- The table on the following slide shows a decision table that includes all possible permutations of input conditions.
- If we wanted to create a limited entry decision table, we could eliminate column 3, which has the same outcome as column 2. We could also eliminate column 5, which has the same outcome as column 4.

MGMT70043 LESSON 4

Decision table testing

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6
CONDITIONS						
Basic Plan	Y	Y	Y	N	N	N
Enhanced Plan	N	N	N	Y	Y	Y
0-10 transactions	Y	N	N	Y	N	N
11-19 transactions	N	Y	N	N	Y	N
20+ transactions	N	N	Y	N	N	Y
ACTIONS						
$0.00 per transaction	N	N	N	Y	Y	N
$1.00 per transaction	Y	N	N	N	N	N
$2.00 per transaction	N	N	N	N	N	Y
$3.00 per transaction	N	Y	Y	N	N	N

MGMT70043 LESSON 4

Decision table testing

How to write a test case based on a decision table analysis

1. Create a decision table. Be sure to include both conditions (user input) and actions (system output).
2. Write one test case per column in the decision table, using the conditions as the input and the actions as the output.

MGMT70043 LESSON 4

Group exercise

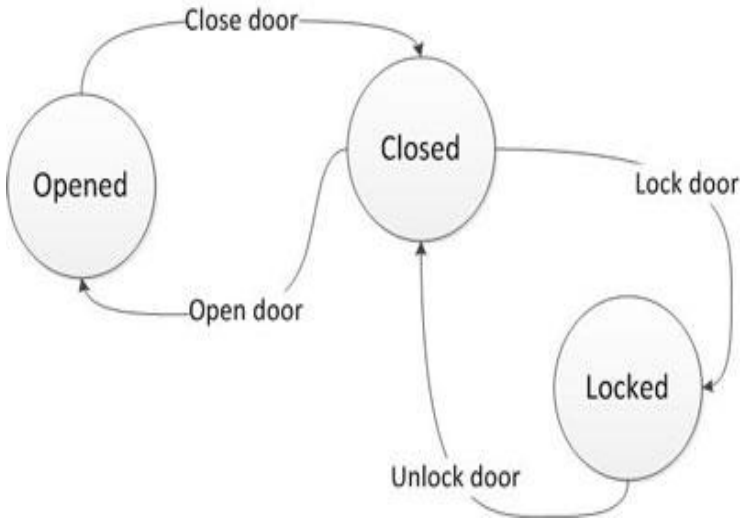
1. Create a decision table for the following:
   - You are building a mortgage calculator.
   - If the user enters the term, the system must calculate the repayment amount
   - If the user enters the repayment amount, the system must calculate the term
   - If the user enters both or neither, the system displays an error.

2. Write a test case based on one column in the decision table

MGMT70043 LESSON 4

State transition testing

- **State transition testing** is useful for testing systems where changes in state are important.
- A **state** is defined as a mode or condition of being. For example: "pending", "approved" and "rejected".
- “State" can refer to the state of the system or to the state of an object within the system.
- A state transition diagram has two main symbols:
  - A circle (or a rectangle with rounded edges), representing a state.
  - An arrow, which represents a transition. The text on the arrow is the action or event that caused the transition.
- The diagram shows which transitions are valid and which are invalid (“open” to “locked” is invalid).



MGMT70043 LESSON 4

State transition testing

- A transition from one state to the next state is a single-state transition.
- Two sequential pairs of transitions are called a two-state transition.
- Transitions in state can also be captured in **state transition tables**, also known as state tables.
- State tables show starting states, inputs (actions & events), ending states and outputs.
- Each combination of starting state and input can be used to generate a test case.

MGMT70043 LESSON 4

State transition testing

For example, let's say the requirements document states the following:

1.1.2 The first four times that the user enters a invalid PIN, the system shall display an error message and prompt the user to try again

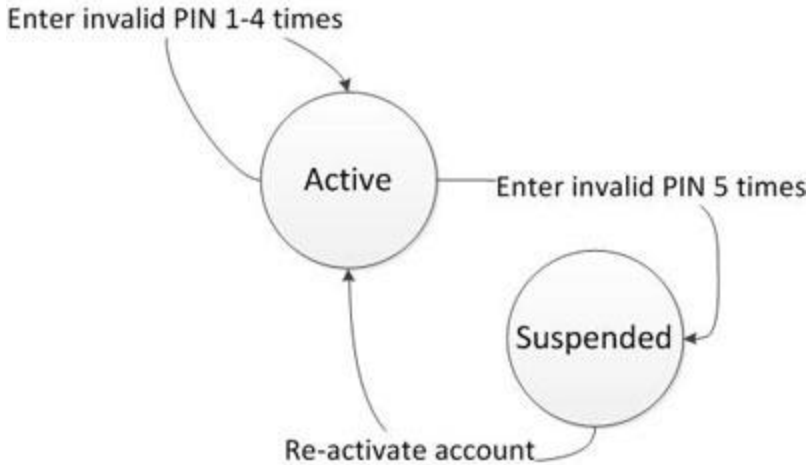
1.1.3 Once the user enters an invalid PIN five times, the system shall keep the card and suspend the account

1.1.4 Suspended ATM users can re-activate their accounts by speaking with a customer service rep and correctly answering security questions

MGMT70043 LESSON 4

State transition testing

- The state transition diagram might look like the figure on the right.
- The diagrams shows two transitions from one state to another (from Active to Suspended and from Suspended back to Active), as well as one action that does not result in a change in state (Entering an invalid PIN 1-4 times).



MGMT70043 LESSON 4

State transition testing

- The state table might look like this
- If the current state is "Active" and the action is "Enter invalid PIN 1-4 times", then the resulting state is "Active" and the output is the "Try again" error message.
- If the current state is "Active" and the action is "Enter invalid PIN 5 times", then the resulting state is "Suspended" and the output is the "Account suspended" error message.

	Test Case 1	Test Case 2	Test Case 3
Starting State	Active	Active	Suspended
Input	Enter invalid PIN 1-4 times	Enter invalid PIN 5 times	Re-activate account
Finish State	Active	Suspended	Active
Expected Output	"Try Again" error message	"Account Suspended" error message	Null

MGMT70043 LESSON 4

State transition analysis

How to write a test case based on state transition analysis

1. Identify all the possible state changes. This may be done via a state transition diagram or a state table.
2. Write one test case per state change. Be sure to include the starting state in the Pre-Conditions and the ending state in the Post-Conditions.

MGMT70043 LESSON 4

Group activity

- When Fred wakes up in the morning, he places his cell phone in vibrate mode to avoid waking his family.
- When he gets to work, he leaves his cell phone in vibrate mode so that it doesn't disturb his colleagues.
- When he gets home from work, he turns the ringer on.
- Before going to bed at night, he places his cell phone on mute.

1. Create a state transition diagram and state table showing each of Fred's actions and the effect of each action on the state of Fred's phone.
2. Write a test case for one of the state changes identified in your analysis. Be sure to include the starting state in the Pre-Conditions and the ending state in the Post-Conditions.

MGMT70043 LESSON 4

Use case testing

- **Use case testing** is a black-box technique based on use cases.
- Use cases describe interactions between a user and a system that produce something of value to the user.
- In addition to system testing, use case testing is useful for testing business processes and for user acceptance testing (UAT), since use cases represent actual likely use of a system.
- System use cases, which describe interactions between different components of the system, can also be used for integration testing.
- We will look at use cases and use case testing in greater detail in the next lesson.

MGMT70043 LESSON 4

Structural (white-box) technique

- **White-box techniques** (also called **structural techniques**) are based on an analysis of the internal structure of the component or system.
- Based on an identified structure of the software or system, for example:
  - **Component level:** the structure is that of the code itself, i.e. statements, decisions or branches.
  - **Integration level:** the structure may be a call tree (a diagram in which modules call other modules).
  - **System level**: the structure may be a menu structure, business process or web page structure.

MGMT70043 LESSON 4

Statement testing

- The purpose of s**tatement testing** is to ensure that all the statements have been exercised.
- Can be used to measure the percentage of statements exercised by a set of test cases.
- The formula for **statement coverage**, also known as **line coverage**, is:

```
(Number of statements exercised/Total number of statements) * 100
```

- For example, if the pseudo code below is tested with the input value X=5, then every line of the code below would be exercised, and the testing would provide 100% statement coverage.

```
READ X
IF X = 5
PRINT "Your account has been suspended"
ENDIF
```

MGMT70043 LESSON 4

Decision testing

- The purpose of **decision testing** is to ensure that all decisions have been exercised.
- For example, if a program contains an IF statement, then this is a decision with two possible outcomes: true and false.
- The program would need to be tested with input values that result in each possible outcome (i.e. true and false).
- Decision testing can be used to measure the percentage of decisions that have been tested by a given set of test cases.

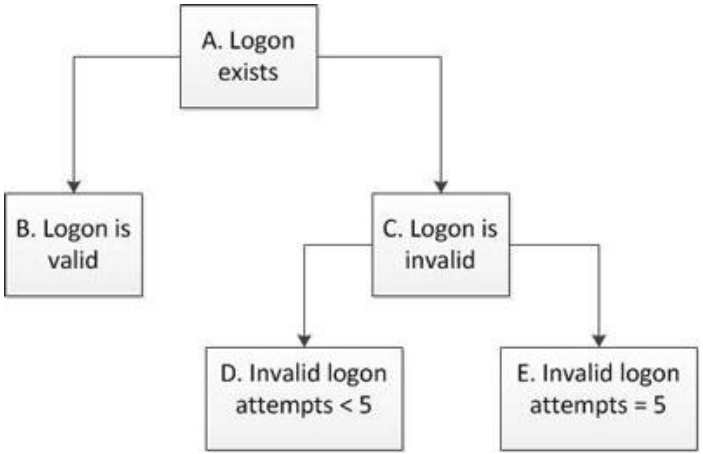
- The formula for **decision coverage**, also known as **branch coverage**, is:

```
(# of decision outcomes exercised/Total # of decision outcomes) * 100
```

MGMT70043 LESSON 4

Decision testing

- In the example on the right, you would need three test cases in order to get 100% decision coverage: one for the A-B path, one for the A-C-D path, and one for the A-C-E path.
- In the earlier pseudo code example, if the code is tested with the input value X=5, then this would make the decision "If X = 5" true, but it would not test the scenario where "If X = 5" is false.
- Since X=5 tests only 1 of 2 possible decision outcomes, we would have 50% decision coverage.
- In order to achieve 100% coverage, we would also need to use an input value that makes the outcome false (for example, X=5 and X=3).



MGMT70043 LESSON 4

Decision testing

How to write a test case based on decision testing analysis

1. Identify all the possible decision paths in the design. The decision paths may be identified via a design flowchart or via the code.
2. Write one test case per decision path.

MGMT70043 LESSON 4

In-class exercise

Given the following fragment of code, how many tests are required for 100% statement coverage and 100% decision coverage?

```
1 Begin
2 Read Time
3 If Time < 12 Then
4 Print(Time, "am")
5 Endif
6 If Time > 12 Then
7 Print(Time -12, "pm")
8 Endif
9 If Time = 12 Then
10 Print (Time, "noon")
11 Endif
12 End
```

MGMT70043 LESSON 4

Experience-based techniques

- **Experience-based testing** is used when there is no formal documentation or if there is not enough time to derive test cases from specifications.
- Even if specifications do exist, it's sometimes a good idea to supplement specification-based testing with experience-based techniques.
  1. **Error guessing**, also known as **ad hoc testing**, is based on the tester's own knowledge and experience.
     - Error guessing works backwards: first testers come up with a list of possible defects and then test the system to see if those defects exist.
     - Error guessing can also start with a list of **known defects and failures**.
  2. **Exploratory testing** is a structured way of testing when specifications are missing and/or there is a time crunch.
     - It is also based on the experience of the testers, but its objectives are defined in a testing charter and each phase is time-boxed.

MGMT70043 LESSON 4

Choosing test techniques

The test technique used for a given project will depend on various factors, including:

- the development methodology
- the documentation available
- the type of system
- regulatory requirements

MGMT70043 LESSON 4

In-class exercise

What test design technique would you use in each of the following situations?

1. You are helping end users plan test cases for UAT and you would like the test cases to capture how the system is actually used in real life
2. The requirements document contains complex combinations of business rules
3. According to the requirements document, changes of state are important to the system
4. Your goal is to break the system by focusing on areas where programmers typically make errors
5. You are working on a safety critical system and the test methodology is subject to regulatory requirements
6. No requirements documents exist and there is no time to create them.

References

- Hambling, B. (2010). Chapter 4: Test Design Techniques. In B. Hambling (Ed.), *Software Testing: An ISTQB-ISEB Foundation Guide* (pp. 74-128). Swindon: British Informatics Society Limited.
- Sehlhorst, S. (2006, January 13). Black Box vs White Box Testing. *Tyner Blain Blog*. Retrieved from tynerblain.com.
- Test Design Techniques. ISTQB Foundation Level Syllabi. Retrieved from astqb.org.