INF 3121 Software Testing

Test Design Techniques

Chapter 4 – Part 1

1. The test development process
2. Categories of test design techniques
3. Specification-based testing (Black-box)

Test Design Techniques

Recall the difference between static and dynamic test techniques:

- **Static testing** - manual examination and automated analysis of the code or documentation, without executing the software under test.

- **Dynamic testing** - requires execution of the software under test.

The test development process

✓ Differentiate between specifications for: **test design**, **test case** and **test procedure**

✓ LO: Define and compare: test condition, test case and test procedure

✓ LO: Design test cases starting from a set of software requirements

✓ LO: Organize test cases into a well-structured test procedure specification

✓ LO: Evaluate the quality of test cases in terms of **traceability** to the requirements and expected results

Background

Before we start testing, vi need to know

- What are we trying to test?
- What are the inputs?
- What are the results that should be produced by those inputs?
- How do we prepare the tests?
- How do we run the tests?

To answer these questions we will look at

- Test conditions
- Test cases
- Test procedures

Background

The test design process can be done in different ways, from very informal (little or no documentation), to very formal.
The level of formality depends on the context of the testing, including:

- The maturity of testing process
- The maturity of development process
- The organization
- Time constraints
- People involved

Test analysis

The test basis documentation is analyzed in order to determine *what* to test, i.e. to *identify the test conditions*.

Test condition *(Def.)* = an item or event that could be verified by one or more test cases

Test analysis

Examples

- A function
- A transaction
- A quality characteristic
- Other structural element (menus in web pages, i.e.)

Test possibilities

“Throw the net wide”

First; identify as many test conditions as possible

Second; select which one to develop in more detail

We can't test everything (P2). We have to select a subset of all possible tests, but this subset must have a high probability of finding most of the defects in the system.

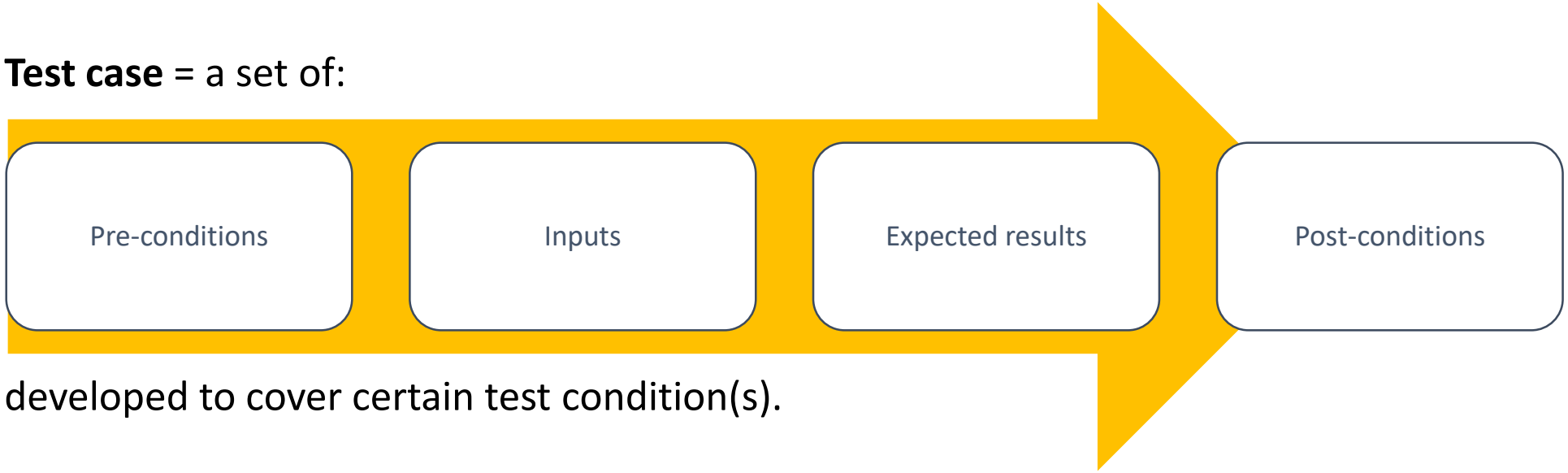
We need a suitable test design technique to guide our selection and to prioritize the test conditions.

Test design

During test design:

are created and specified.

Test case = a set of:

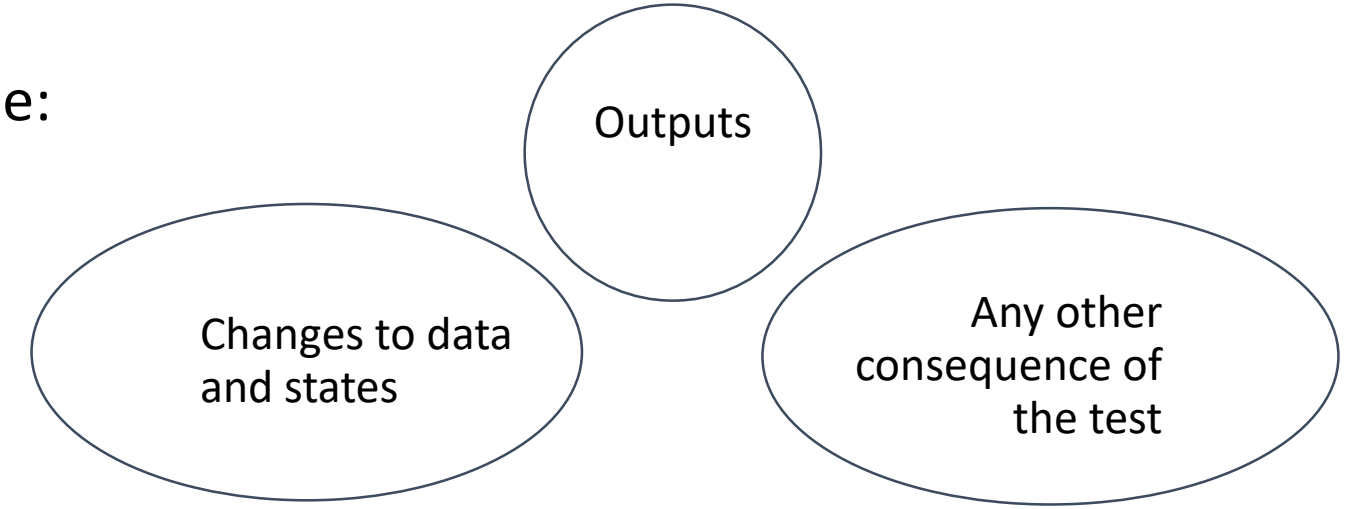


developed to cover certain test condition(s).

Test oracle

In order to know what the system *should* do, we need to have a source of information about the correct behavior of the system – an oracle.

Expected results include:

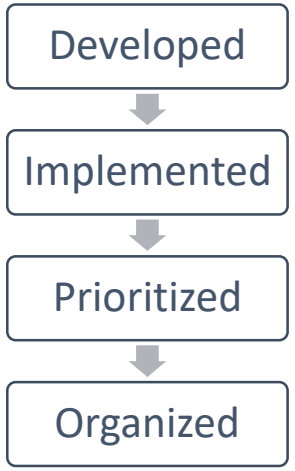


If expected results have not been defined, then a plausible but erroneous result may be interpreted as the correct one.

Expected results should ideally be defined prior to test execution.

Test implementation

During test implementation the test cases are:

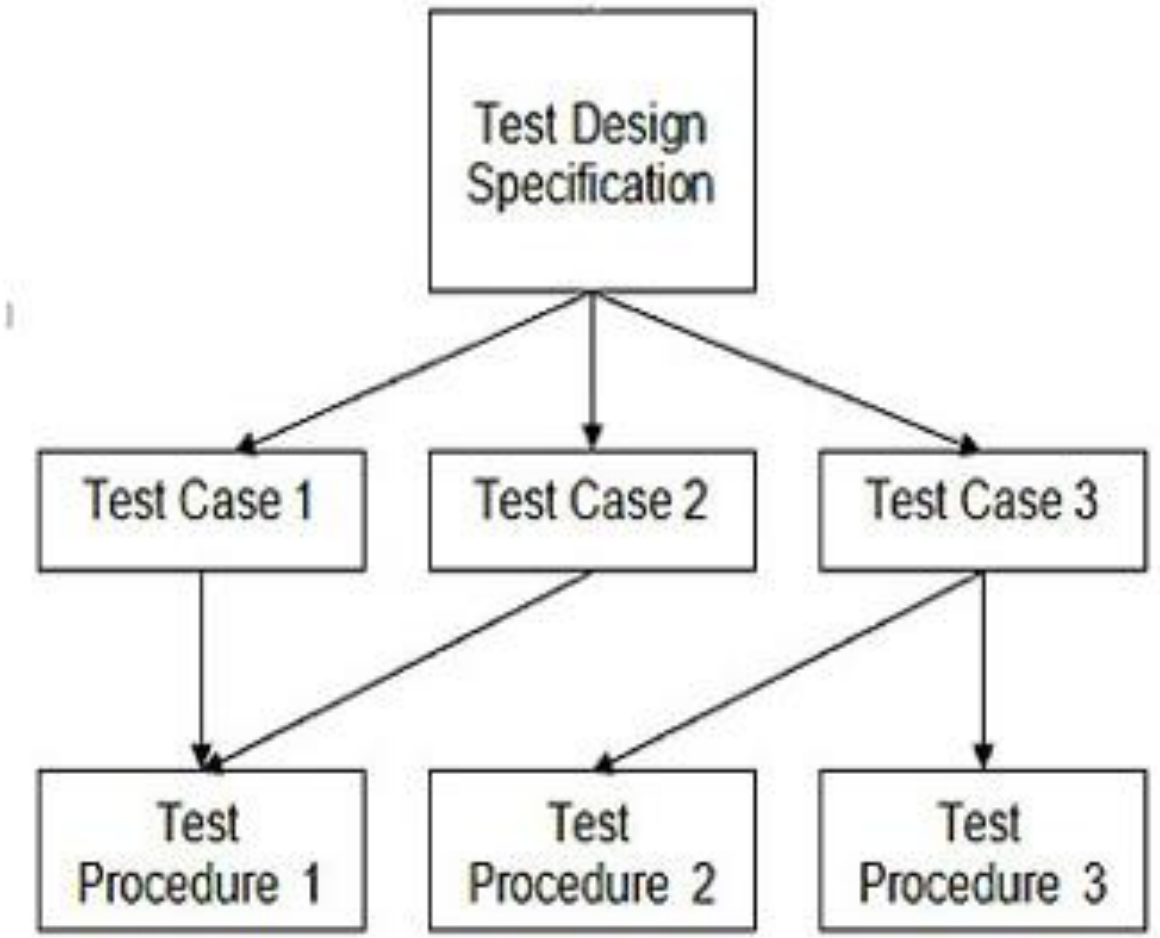


in the test procedure specification.

The test procedure (= **manual test script**)
specifies the sequence of action for the execution of a test.

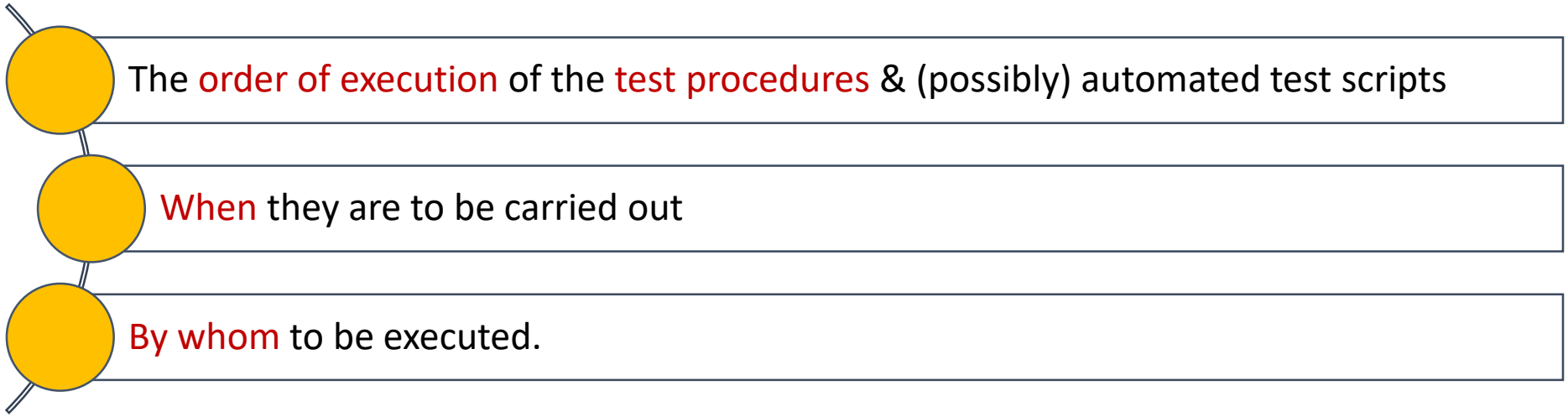
Test script (= **automated test procedure**)
If tests are run using a test execution tool, the sequence of actions is specified in a test script.

Test implementation



Test implementation

The test execution schedule defines:



The test execution schedule will take into account such factors as:

- risks
- regression tests
- prioritization
- technical and logical dependencies

Test implementation

Writing the test procedure is another opportunity to prioritize the tests, to ensure that the best testing is done in the time available.

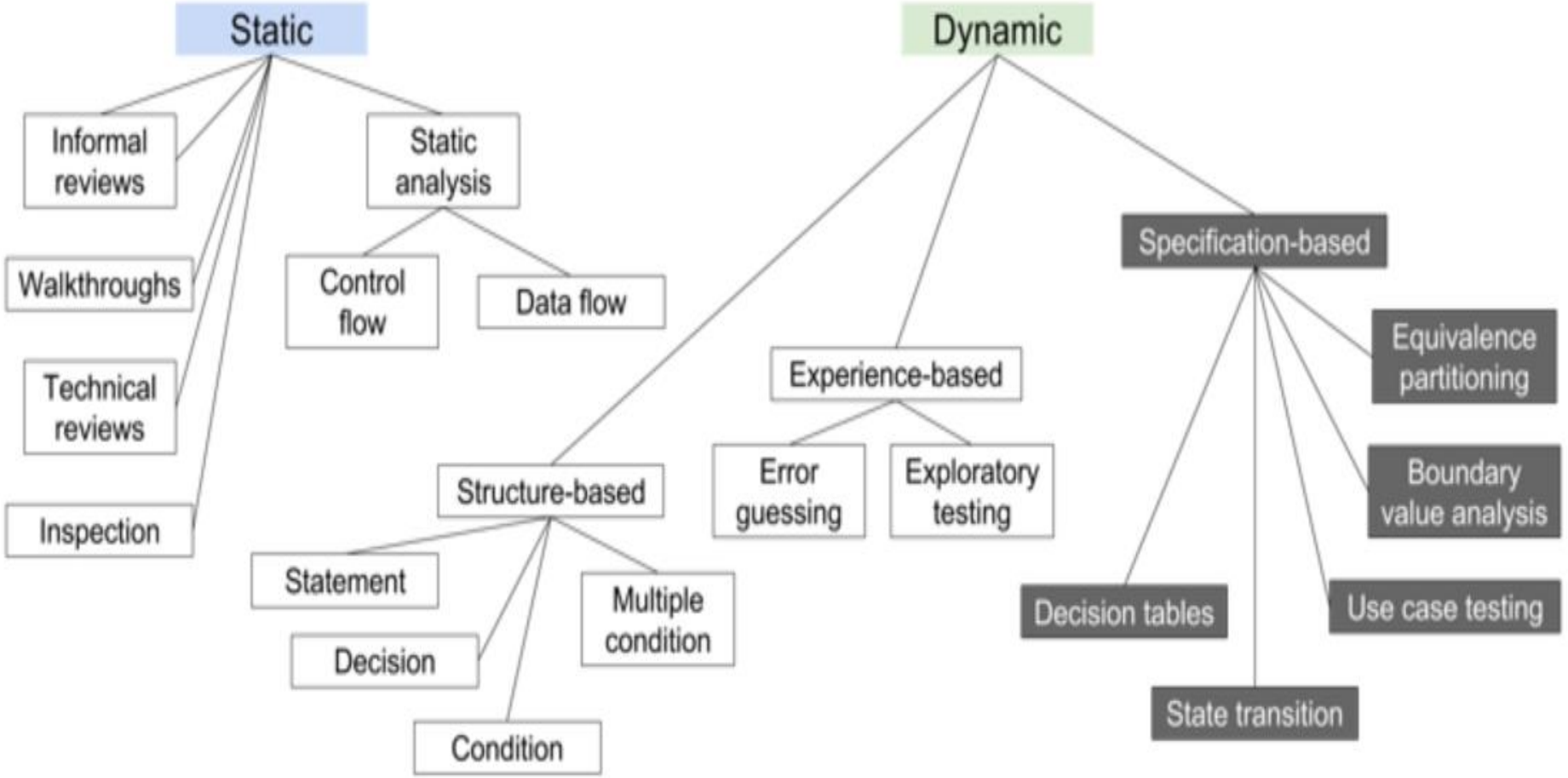
A good **rule of thumb** is 'Find the scary stuff first'. However the definition of what is 'scary' depends on the business, system or project and depends up on the risk of the project.

Categories of test design techniques

✓ LO: Recall the purpose of both black-box testing and white-box testing. Give example of techniques for each type of technique.

✓LO: Explain the characteristics, differences and cases in which to use black-box, white-box and experience based testing techniques

Categories of test design techniques



Common features of black-box techniques

- We use models (formal or informal) to specify the problem to be solved, the software or its components.
- We derive systematically the test cases from these models

Common features of white-box techniques

- The test cases are derived from information about how the software is constructed for example: code and design.
- For the existing test cases, we can measure the test coverage of the software
- Further test cases can be derived systematically to increase the test coverage.

Common features of experience-based techniques

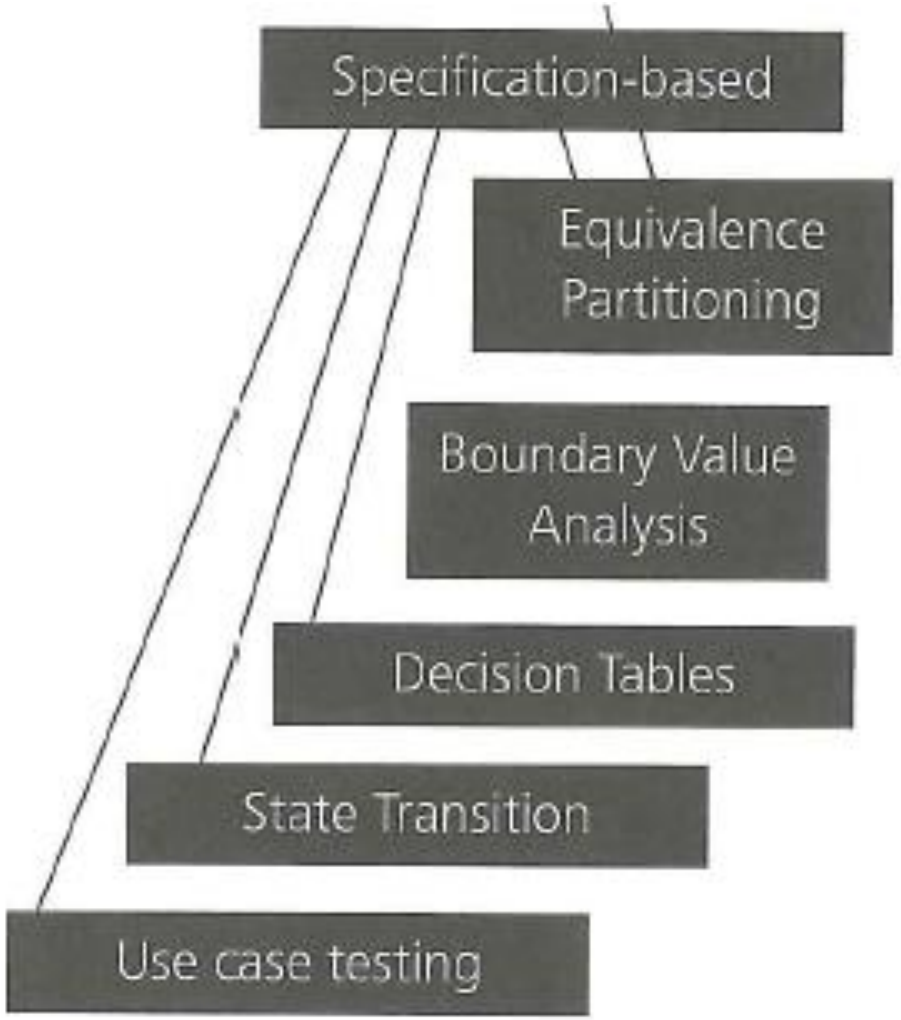
The test cases are derived from the knowledge and experience of people:

- Knowledge of testers, developers, users and other stakeholders about the software, its usage and its environment
- Knowledge about likely defects and their distribution

Specification-based testing
Black-box tecniques

- ✓ LO: Write test cases from given software models using: EP, BVA, DT and ST test design techniques
- ✓ LO: Explain the purpose of each black-box test design technique
- ✓ LO: Explain the concept of use-case testing and its benefits

Black-box techniques



Equivalence partitioning

- The **basic idea** is to divide a set of test conditions into sub-groups or sub-sets (partitions) that can be considered the same.
- It is important that the different partitions do not have common elements.
- We need only to test one condition from each partition, because all the conditions in the same partition will be treated in the same way by the software.

Equivalence partitioning

Typical problem

A **savings account** in a bank earns a different rate of interest depending on the balance in the account.
In order to test the software that calculates the interest due, we can identify the ranges of balance values that earn the different rates of interest.
If a balance in the range $0 up to $100.00 has a 3% interest rate, a balance over $100.00 and up to $1000.00 has a 5% interest rate, and balances of $1000.00 and over have a 7% interest rate, we would initially identify three valid equivalence partitions and one invalid partition as shown below.

Invalid partition	Valid (for 3% interest)		Valid (for 5%)		Valid (for 7%)
–$0.01	$0.00	$100.00	$100.01	$999.99	$1000.00

Equivalence partitioning

We might choose to calculate the interest of one balance in each partition:

-$10.00, $50.00, $260.00, $1348.00

Invalid partition	Valid (for 3% interest)		Valid (for 5%)		Valid (for 7%)
–$0.01	$0.00	$100.00	$100.01	$999.99	$1000.00

Equivalence partitioning

Technique

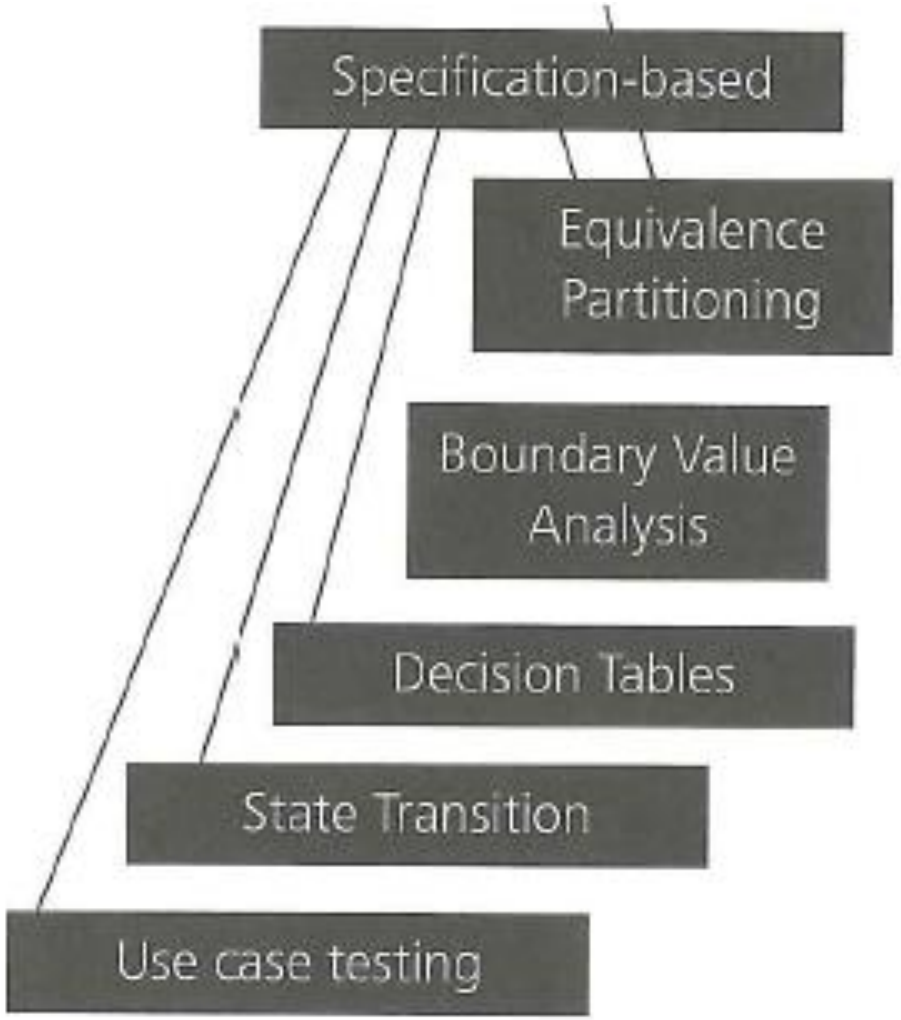
Inputs/outputs/internal values of the software are divided into groups that are expected to exhibit similar behavior.

Equivalence partitions (or classes) can be found for both valid data and invalid data, i.e. values that should be rejected.

Notes

- Tests can be designed to cover more than one partitions.
- Equivalence partitioning is applicable at all levels of testing.
- Equivalence partitioning as a technique can be used to achieve input and output coverage.

Black-box techniques



Boundary value analysis

Behavior at the edge of each equivalence partition is more likely to be incorrect than behavior within the partition.

Boundary are an area where testing is likely to yield defects.

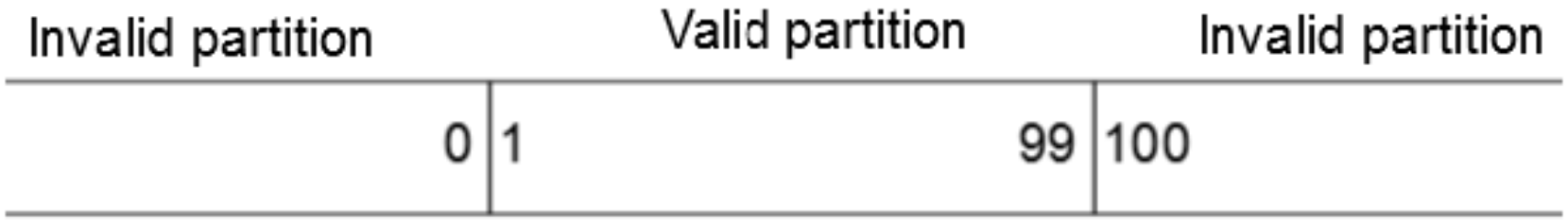
Example

A printer has an input option of number of copies to be made:

Invalid partition	Valid partition		Invalid partition
0	1	99	100

Boundary value analysis

The maximum and minimum values of a partition are its boundary values.



Valid boundary values: 1, 99
Invalid boundary values: 0, 100

When designing test cases, a test for each boundary value is chosen.

Boundary value analysis

Another example with two decimals:

Invalid partition	Valid partition		Invalid partition
- 0.01	0.00	100.00	100.01

Valid boundary values: 0.00, 100.00
Invalid boundary values: - 0.01, 100.01

Boundary value analysis

Boundary analysis

Analysis at the edge of each equivalence partition.

Why? Because there, the results are more likely to be incorrect.

The maximum and minimum values of a partition are its boundary values.

Valid and invalid boundary

- A boundary value for a valid partition is a valid boundary value.
- A boundary value for an invalid partition is an invalid boundary value.
- Tests can be designed to cover both valid and invalid boundary values.

Boundary value analysis

Notes

- Boundary value analysis can be applied at all test levels.
- It is relatively easy to apply and its defect finding capability is high.
- Detailed specifications are helpful.
- This technique is often considered as an extension of equivalence partitioning
- Boundary values are used for test data selection.

Equivalence partitioning and boundary

Why do both equivalence partitioning and boundary value analysis?

Boundary values are usually extreme values

To gain confidence to the system we want to test it under normal circumstances.

1. The test development process

- *1.1 Background*
- *1.2 Test analysis*
- *1.3 Test design*
- *1.4 Test implementation*

2. Categories of test design techniques

3. Specification-based techniques (black-box)

- *3.1 Equivalence partitioning*
- *3.2 Boundary value analysis*
- *3.3 Decision table testing*
- *3.4 State transition testing*
- *3.5 Use case testing*

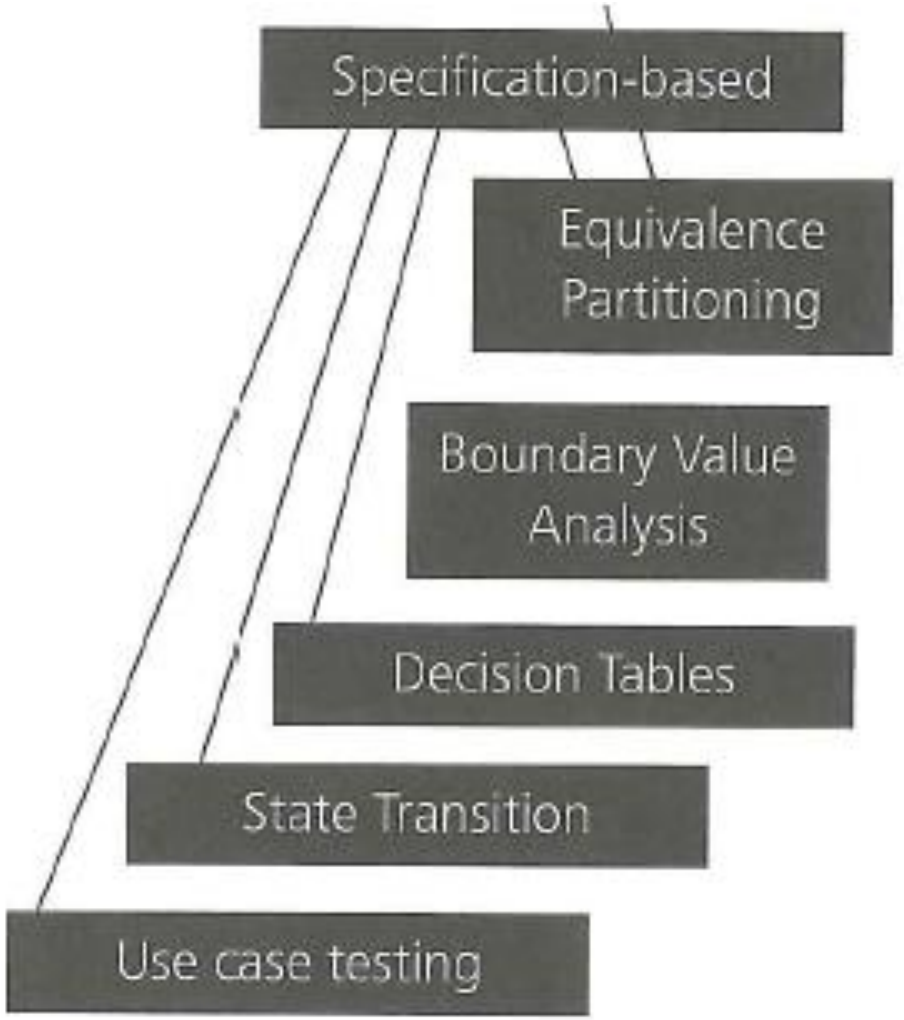
4. Structure-based techniques (white-box)

- *4.1 Statement testing and coverage*
- *4.2 Decision testing and coverage*
- *4.3 Other structure-based techniques*

5. Experience-based techniques

6. Choosing test techniques

Black-box techniques



Decision table testing

Decision tables are a good way

- to capture system requirements that contain *logical conditions*
- to document internal system design
- to record complex business rules that a system is to implement

Decision table testing

- When creating decision tables, the specification is analyzed, and actions of the system are identified.
- The input conditions and actions are most often stated in such a way that they can either be true or false (Boolean).

Decision table testing

The decision table contains the triggering conditions, often combinations of true and false for all input conditions, and the resulting actions for each combination of conditions.

Decision table testing

Example

- Betingelser:
  GSK: Generell studiekompetanse
  R1: Bestått matematikk R1 fra videregående skole
  KP: Konkurransepoeng over årets grense

- Aksjoner:
  SP: Tilbud om studieplass
  V: Settes på venteliste
  FK: Tilbud om forberedende kurs i R1.
  A: Avslag, dvs. avslag på studieplass og forkurs, samt heller ikke på venteliste.

	Regel 1	Regel 2	Regel 3	Regel 4	Regel 5	Regel 6	Regel 7	Regel 8
GSK	true	true	true	true	false	false	false	false
R1	true	true	false	false	true	true	false	false
KP	true	false	true	false	true	false	true	false
Aksjon	**SP**	**V**	**FK**	**A**	**A**	**A**	**A**	**A**

Decision table testing

Svar: Vi ser av tabellen at når søkeren ikke har generell studiekompetanse, så får vedkommende avslag, uavhengig av om vedkommende har R1 eller ikke, eller om vedkommende har konkurransepoeng over eller under årets inntaksgrense. Følgelig kan regel 5, 6, 7 og 8 slås sammen, og vi får følgende tabell:

	Regel 1	Regel 2	Regel 3	Regel 4	Regel 5
GSK	true	true	true	true	false
R1	true	true	false	false	----
KP	true	false	true	false	----
Aksjon	**SP**	**V**	**FK**	**A**	**A**

Vi har nå redusert antall regler fra 8 til uten å miste noen testtilfeller. Regel 5 er uavhengig av verdiene til betingelsene R1 og K.

Decision table testing

Example - Fictional wine monopole store:

	Rule 1	Rule 2	Rule 3	Rule 4
Conditions				
Oslo resident?	False	True	True	True
Over 18 years?	Don't care	False	True	True
Happy hour?	Don't care	Don't care	False	True
Actions				
Can buy wine?	False	False	True	True
Offer 10% discount?	False	False	False	True

Each column of the table corresponds to a business rule that defines a unique combination of conditions and the resulting actions.

Decision table testing

- The coverage standard commonly used with decision table testing is to have at least one test per column, which typically involves covering all combinations of triggering conditions.

- The strength of decision table testing is that it creates combinations of conditions that might not otherwise have been exercised during testing.

- It may be applied to all situations when the action of the software depends on several logical decisions.