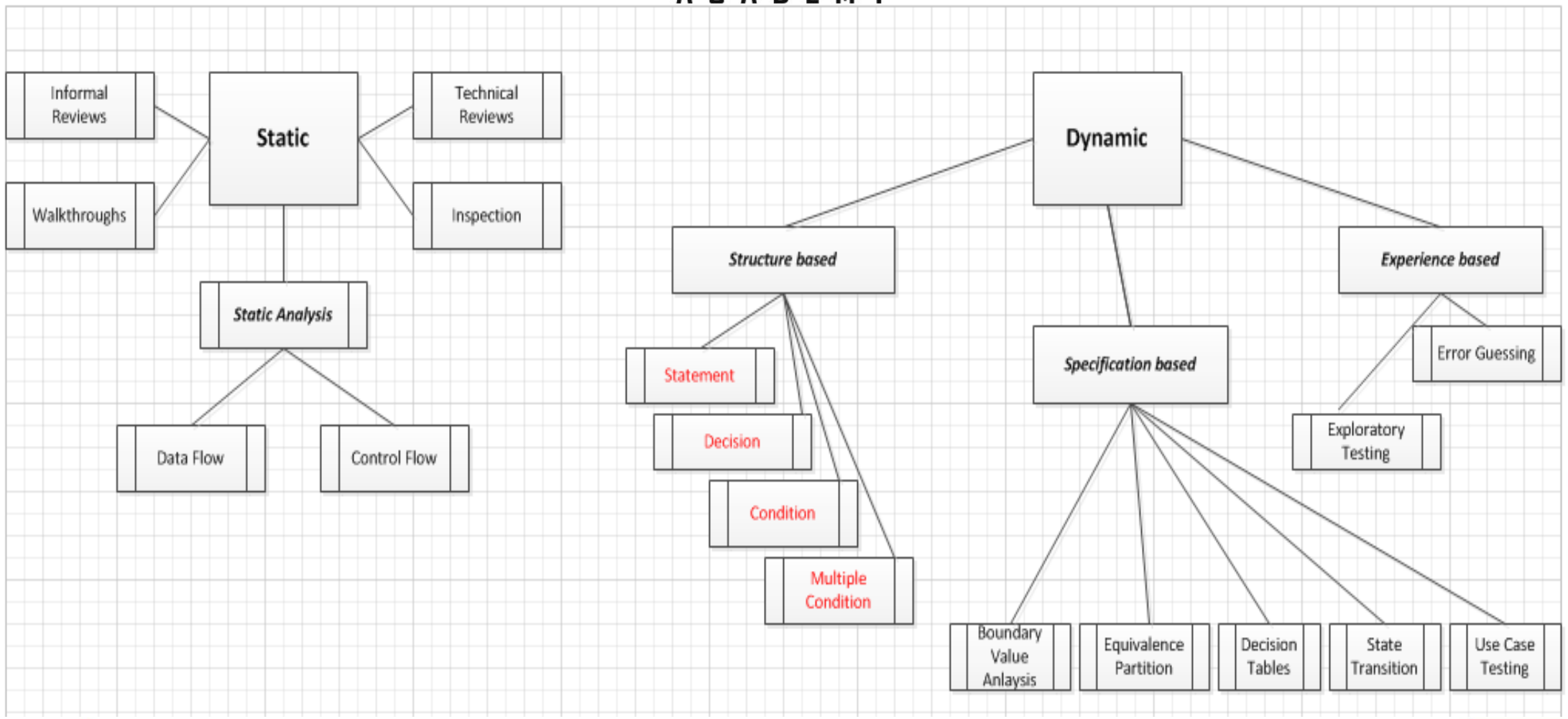
Software Testing

Testing Techniques

Agenda

- Testing techniques classification
- Experience-based techniques
- Specification-based techniques (black box)
- Structure-based techniques (white box)
- Static analyses techniques


Informal Reviews
Walkthroughs
Static
Technical Reviews
Inspection
Static Analysis
Data Flow
Control Flow
Dynamic
Structure based
Statement
Decision
Condition
Multiple Condition
Specification based
Boundary Value Anlaysis
Equivalence Partition
Decision Tables
State Transition
Use Case Testing
Experience based
Error Guessing
Exploratory Testing

Experience-based testing techniques

- Experience-based testing techniques rely on tester’s experience rather than per-defined test scope and formal test documentation
- Tester verifies and validates the software product quality using his/her past experience of testing the similar type of product in the respective domain
- Experience-based testing techniques are often useful when:

  - combined with other formal techniques

  - there is none or insufficient product documentation

  - there is limited knowledge on the software product

  - there is very limited amount of time left for testing

Experience-based testing techniques

- **Exploratory testing** – both testing technique and progressive learning approach; during an exploratory session, tester constantly analyses how software behaves and chooses what to test next based on current findings
- **Error guessing** - tester relies on past experience to identify the vulnerable areas of the software product that are likely to be more error-prone; similar to risk-based approach, where high, medium and low risk areas are identified, and tests are designed to target high-risk ones
- **Checklist-based testing** - experienced tester prepares a checklist, which serves as a manual to direct the testing process; the checklist is often the only tool to measure test coverage
- **Attack testing / Fault injection** – introduces faults in to the software, to help test the decisions / find failures; fault can be injected at compile time (change the source code) or run-time (example: memory corruption, registry corruption, etc.)

Specification-based testing techniques

- Specification-based testing techniques are also known as ‘black box’ techniques. They view the software as a black-box, or sets of inputs / outputs. Tests are designed based on the specification (requirements) and do consider the internal structure of the system or component, i.e. how input is processed “inside the box”.
- Commonly used specification-based testing techniques:
  - Equivalence partitioning
  - Boundary value analyses
  - Pairwise testing
  - Decision tables
  - State transition testing

Equivalence partitioning

Purchase amount	Discount (%)
>=999	5
>=1999	10
>=3999	15
>=5999	25
>=7999	35
>=9999	50

- Separate data set in partitions based on given rules
- Under certain conditions, the software must apply the same logic to each element of the equivalent class.
- Different sets are used in different parts of software
- Partitions can be valid and invalid

Invalid Partition	Valid					
	Partition 1	Partition 2	Partition 3	Partition 4	Partition 5	Partition 6
0%	5%	10%	15%	25%	35%	50%

Boundary value analysis

- Test edges of equivalent classes, since software tends to break on boundaries of equivalent classes; effective when combined with Equivalence partitioning technique

  Example:
  Requirement: *Input field for person's age accepts values between 18-65 as valid to proceed with account creation*
  Test inputs: *17 (invalid), 18 (valid), 19 (valid); 64 (valid), 65 (valid), 66 (invalid)*

- For each equivalent class, we may have only lower, only upper, or both lower & upper boundaries

0%	5%	10%	15%	25%	35%	50%
(Invalid)	(Valid)	(Valid)	(Valid)	(Valid)	(Valid)	(Valid)
0 - 998	999 - 1998	1999 - 3998	3999 - 5998	5999 - 7998	7999 - 9998	9999 –>

Pairwise / All-Pairs Testing

- A black-box test design technique in which test cases are designed to execute all possible discrete combinations of each pair of input parameters.
- Imagine a simple piece of software to test, with 10 drop-down fields, each having 10 possible values. In order to test them all, you need to check 10^10 = **10000000000** combinations.
- Exhaustive testing is impossible, and pairwise testing technique can help us in this case.

Pairwise / All-Pairs Testing

Example: Game launcher application with following options:

- Game [drop-down]: Deadly Race; Bubble Gum; MegaCity Tycoon
- Version [drop-down]: 1.0; 2.0; 3.0; 4.0
- OS [drop-down]: Android; iOS; Windows OS
- Device [drop-down]: Smartphone; Tablet, PC

If we try to test each possible combination, we need to do

3 x 4 x 3 x 3 = **108 tests**

Now let’s try the same using pairwise testing technique.

Version	Game	OS	Device
1.0	Deadly Race	Android	Phone
1.0	Bubble Gum	iOS	Tablet
1.0	Mega City Tycoon	Windows	PC
2.0	Deadly Race	iOS	Phone
2.0	Bubble Gum	Android	PC
2.0	Mega City Tycoon	Windows	Tablet
3.0	Deadly Race	Windows	Phone
3.0	Bubble Gum	Android	Tablet
3.0	Mega City Tycoon	iOS	PC
4.0	Deadly Race	iOS	Tablet
4.0	Bubble Gum	Windows	Phone
4.0	Mega City Tycoon	Android	PC

Version	Game	OS	Device
1.0	Deadly Race	Android	Phone
1.0	Bubble Gum	iOS	Tablet
1.0	Mega City Tycoon	Windows	PC
2.0	Deadly Race	iOS	Phone
2.0	Bubble Gum	Android	PC
2.0	Mega City Tycoon	Windows	Tablet
*2.0	*Deadly Race*	*Android	*PC*
3.0	Deadly Race	Windows	Phone
3.0	Bubble Gum	Android	Tablet
3.0	Mega City Tycoon	iOS	PC
*3.0	*Mega City Tycoon*	*Android	*Phone*
4.0	Deadly Race	iOS	Tablet
4.0	Bubble Gum	Windows	Phone
4.0	Mega City Tycoon	Android	PC

Decision tables

- Tabular representation of business logic and rules – shows combination of inputs and outputs
- Decision tables consist of Conditions, Actions/Rules and Outcomes
- Very useful for designing tests for software with complex logic, i.e. when there are multiple combinations of conditions and rules

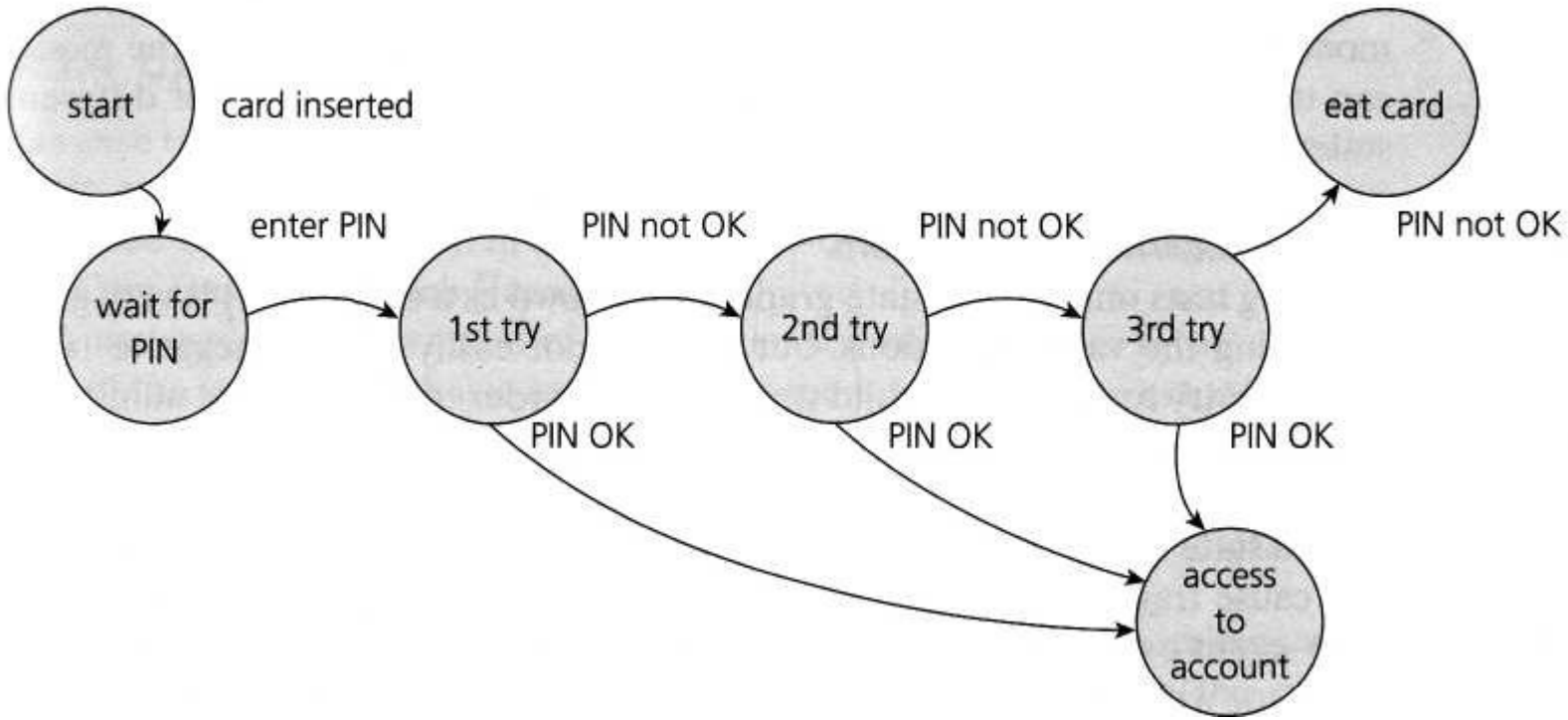
Decision Table Example	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8	Rule 9	Rule 10	Rule 11	Rule 12
Conditions												
Passed credit check	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N
Existing policy	Y	Y	Y	N	N	N	Y	Y	Y	N	N	N
Total years as homeowner	>5	1-5	<1	>5	1-5	<1	>5	1-5	<1	>5	1-5	<1
Outcomes												
OC001 Policy A	X	X	-	X	-	-	-	-	-	-	-	-
OC002 Policy B	X	X	X	X	X	-	-	-	-	-	-	-
OC003 Policy C	X	-	X	-	X	x	-	-	-	-	-	-
OC004 Decline	-	-	-	-	-	-	X	X	X	X	X	X

Decision table example

State transition

- A system may exhibit a different response depending on current conditions or previous history (its state)
- Using state transition diagrams or tables, testers can more easily identify system states and transition between them

- State transition technique focuses on identifying and testing:
  - Different states of the tested object
  - Moving from one state to another
  - Events that cause state transition
  - Aftermath of state transition



State transition diagram example

Structure-based techniques

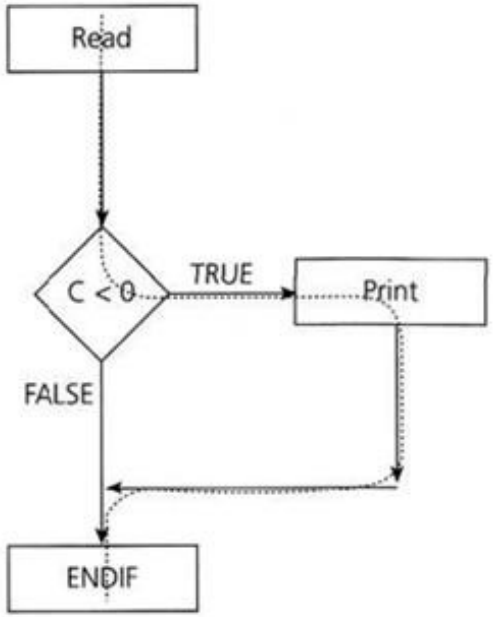
- Structure-based techniques are also known as ‘white-box’ techniques
- Tests are designed based on the internal structure of the software / code
- White box techniques:
  - Statement coverage
  - Decision/Branch coverage
  - Path/Condition coverage

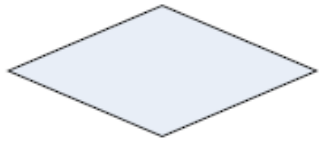
Testing for code coverage

- Structured-based techniques aim at measuring different types of code coverage
- Test for **Statement coverage** - exercise every statement (output)
- Test for **Decision/Branch coverage** - execute every decision outcome (branch), i.e. both TRUE and FALSE outcomes should be covered
- Test for **Condition coverage** - cover all possible conditions
- Test for **Path coverage** – covers all possible path options

- Code coverage is measured by software tools
- Often used in designing unit tests

```
Read C
    IF C < 0
        Print C
    END IF
```



	This is the point of start/end of the flowchart. This block contains text with the name of the Web page, or "Start"/"End," or whatever text is needed.
	This block denotes some process – e.g., "Fill up and submit Page 1."
	This is decision point – i.e., depending on certain conditions, the process can take different passes from this point on. The text of the condition – e.g., "Data valid?" should always be specified in this block.

Flowchart example

Static techniques

- Static techniques analyze the code without it being executed
- They can be applied much earlier in the development life-cycle, as the code itself is being written
- Different types of static techniques:

  - **Code reviews** (informal, walkthrough, technical review, inspection) – done by people
- - **Static analyses** (LCSAJ - Linear Control Sequence and Jump, Data flow analyses, etc.) - done by software tools

SOFT ACADEMY