INTRODUCTION TO SOFTWARE ENGINEERING

09. TESTING

Bui Thi Mai Anh
anhbtm@soict.hust.edu.vn

Content

1. **Testing overview**
2. Blackbox Testing
3. Whitebox Testing

1.1. Testing

- "[T]he means by which the presence, quality, or genuineness of anything is determined; a means of trial." – dictionary.com
- A *software test* executes a program to determine whether a property of the program holds or doesn't hold
- A test ***passes*** [***fails***] if the property **holds** [**doesn't hold**] on that run

Software Quality Assurance (QA)

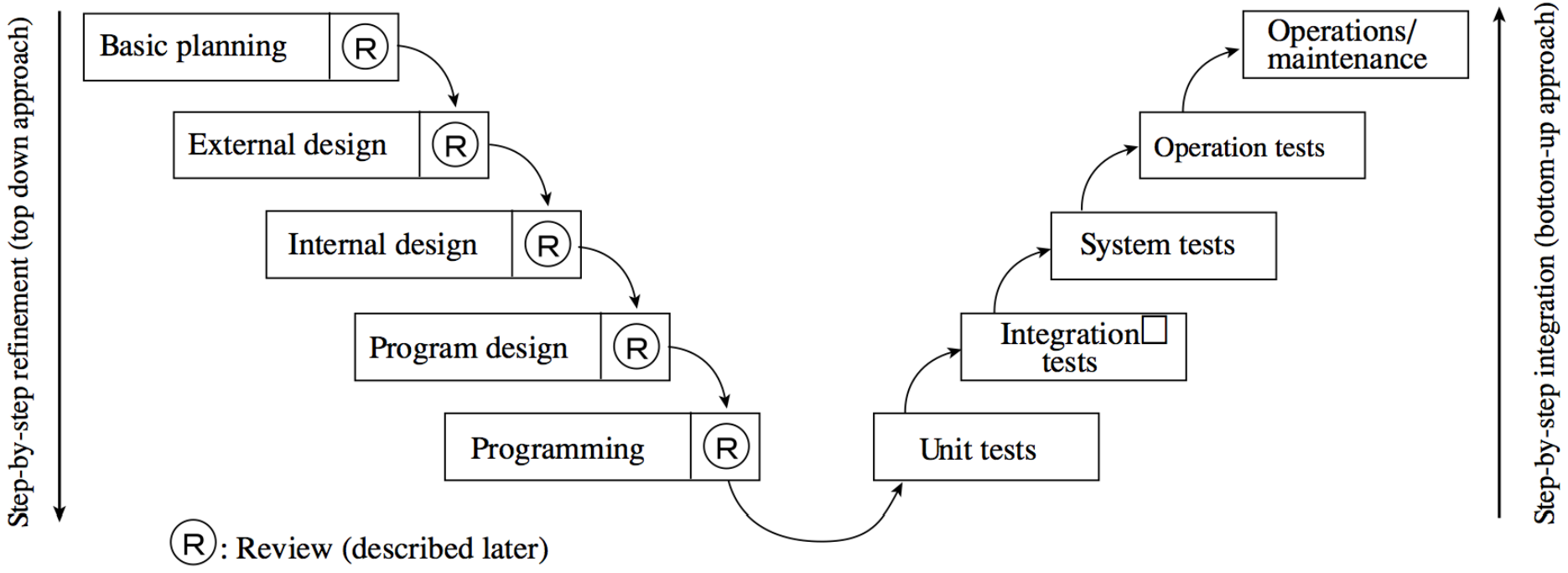
Testing plus other activities including

- Static analysis (assessing code without executing it)
- Proofs of correctness (theorems about program properties)
- Code reviews (people reviewing others' code)
- Software process (placing structure on the development lifecycle)
- …and many more ways to find problems and to increase confidence

No single activity or approach can guarantee software quality

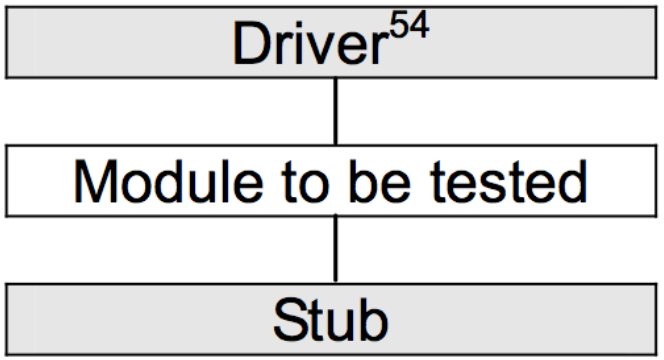
V Model – Different test level

- Unit test: ONE module at a time
- Integration test: The linking modules
- System test: The whole (entire) system



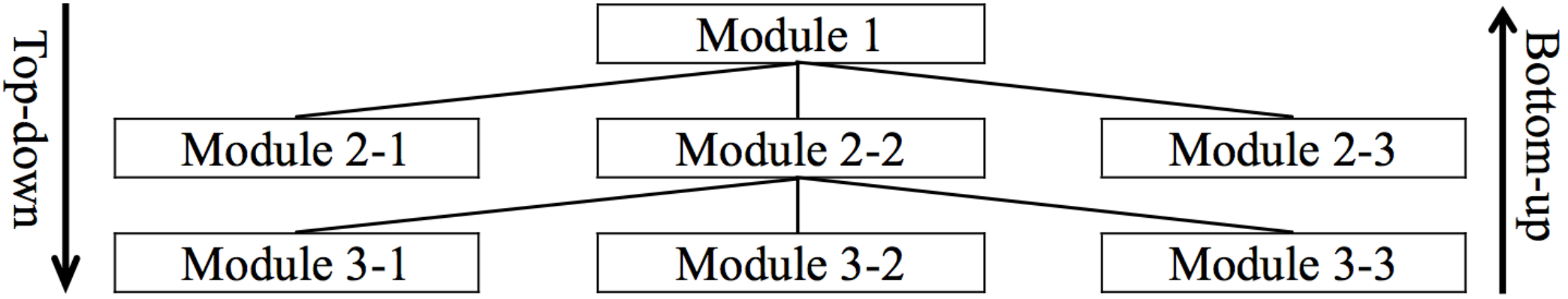
Test levels

- Unit Testing: Does each unit (class, method, etc.) do what it supposed to do?
  - Smallest programming units
  - Approaches: Black box and white box testing
  - Techniques, Tools
- Integration Testing: do you get the expected results when the parts are put together?
  - Approaches: Bottom-up, top-down testing
- System Testing: does it work within the overall system?
  - Approaches: Black box testing
- Acceptance Testing: does it match to user needs?



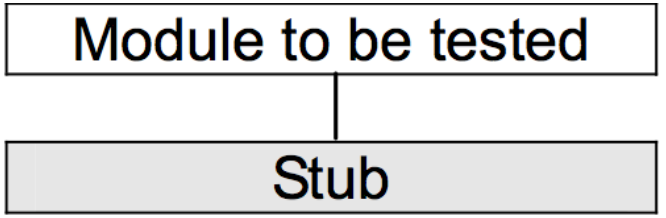
1.2. Integration test

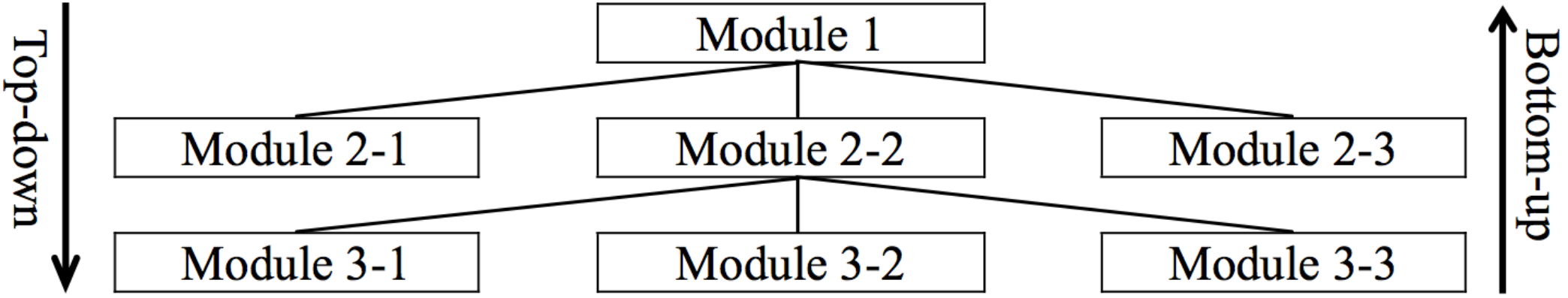
- Examine the interface between modules as well as the input and output
- Stub/Driver:
  - A program that simulates functions of a lower-level/upper-level module



1.2. Integration Test
Top-down approach

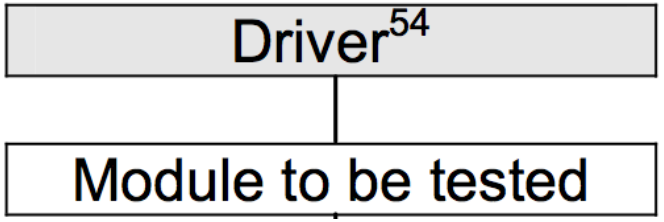
- Defects based on misunderstanding of specification can be detected early
- Effective in newly developed systems
- Need test stubs (can be simply returning a value)

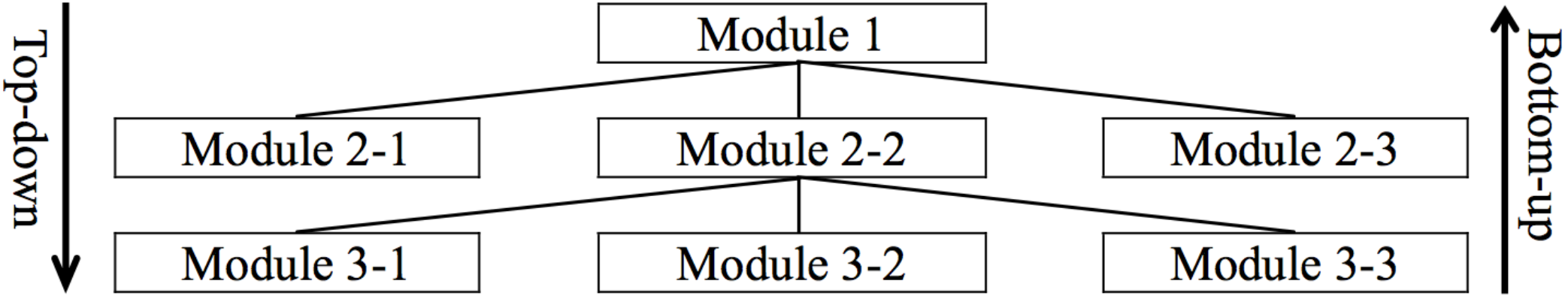




1.2. Integration Test
Bottom-up approach

- Lower modules are independent => test independently and on a parallel
- Effective in developing systems by modifying existing systems
- Need test drivers (more complex with controlling)





Other integration test techniques

- Big-bang test
  - Wherein all the modules that have completed the unit tests are linked all at once and tested
  - Reducing the number of testing procedures in small-scale program; but not easy to locate errors
- Sandwich test
  - Where lower-level modules are tested bottom-up and higher-level modules are tested top-down

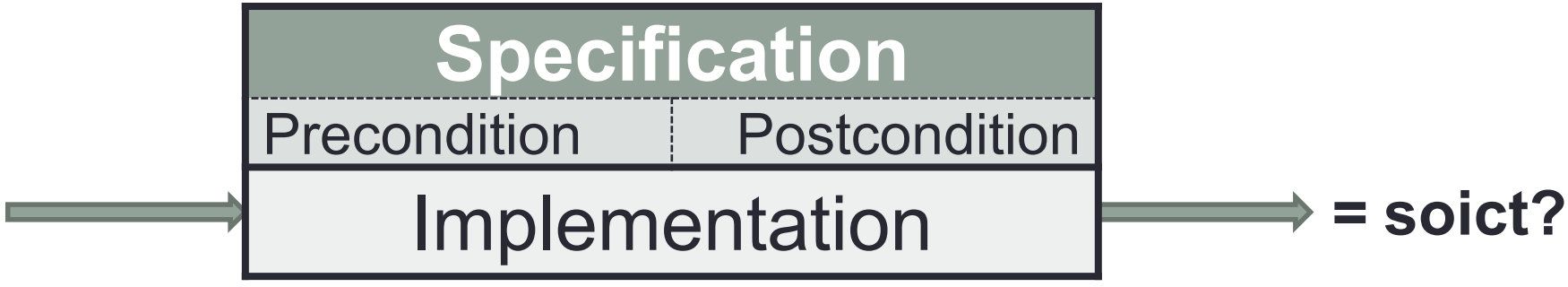
1.3. Regression test

"When you fix one bug, you introduce several new bugs"

- Re-testing an application after its code has been modified to verify that it still functions correctly
  - Re-running existing test cases
  - Checking that code changes did not break any previously working functions (side-effect)
- Run as often as possible
- With an automated regression testing tool

1.4. Black box and White box testing

A. Choose input data (“test inputs”)
B. Define the expected outcome (“soict”)
C. Run the unit (“SUT” or “software under test”) on the input and record the results
D. Examine results against the expected outcome (“soict”)



Black box
Must choose inputs *without knowledge* of the implementation

White box
Can choose inputs *with knowledge* of the implementation

It's not black-and-white, but…

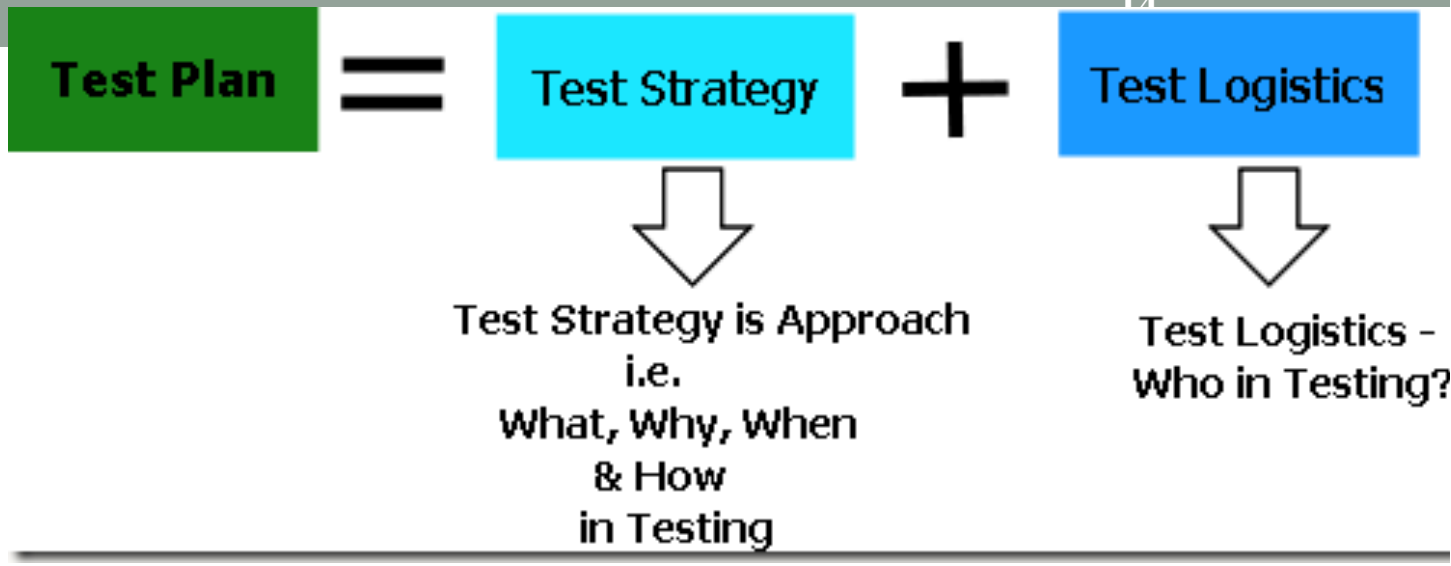
Black box
Must choose inputs *without knowledge* of the implementation

- Has to focus on the behavior of the SUT
- Needs an "soict"
  - Or at least an **expectation** of whether or not an exception is thrown

White box
Can choose inputs *with knowledge* of the implementation

- Common use: *coverage*
- Basic idea: if your test suite never causes a statement to be executed, then that statement might be buggy

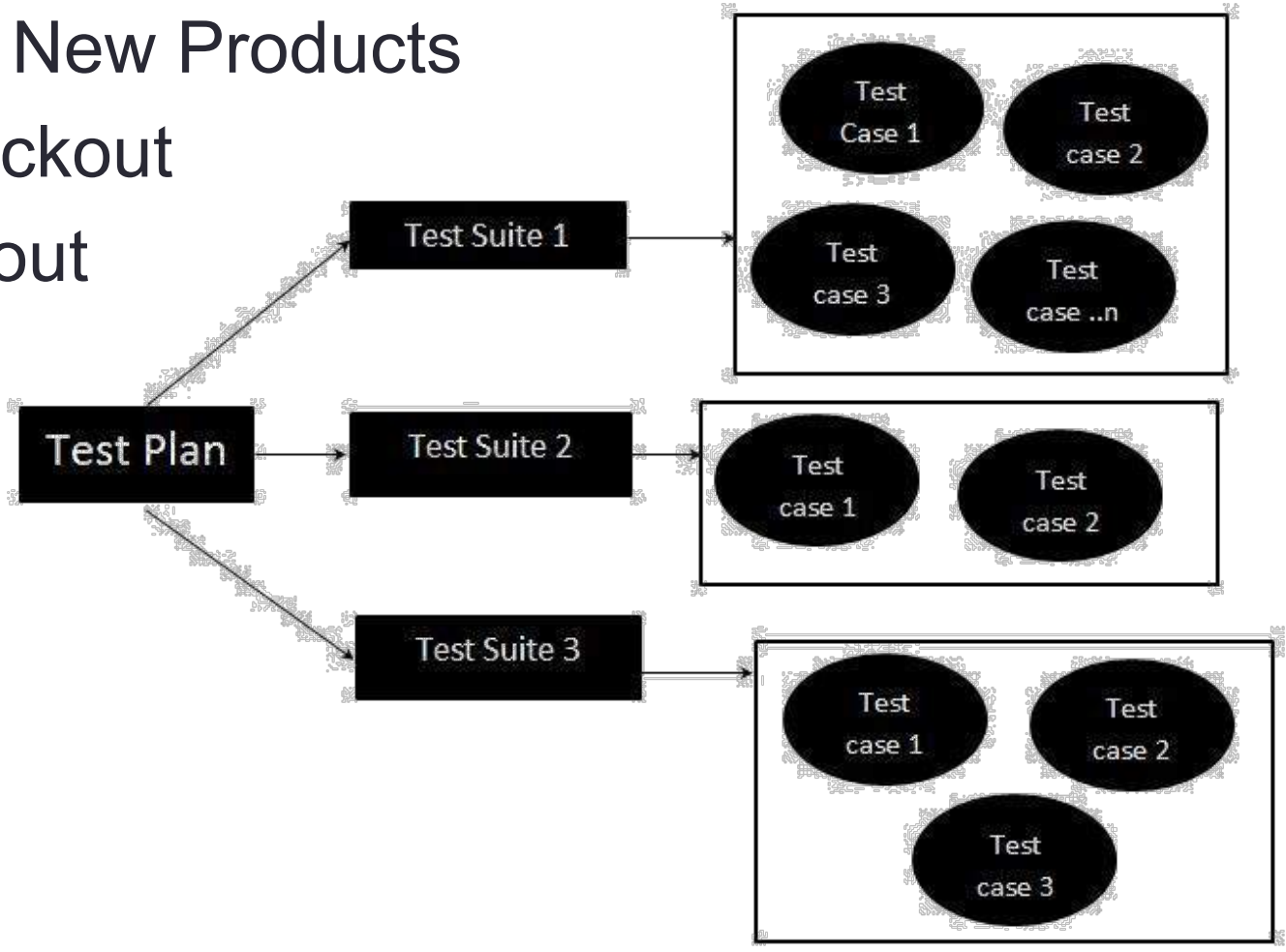
Terms



- Test case
  - a set of conditions/variables to determine whether a system under test satisfies requirements or works correctly
- Test suite
  - a collection of test cases related to the same test work
- Test plan
  - a document which describes testing approach and methodologies being used for testing the project, risks, scope of testing, specific tools

Test suite

- Example of test suite
  - Test case 1: Login
  - Test case 2: Add New Products
  - Test case 3: Checkout
  - Test case 4: Logout



Unit & System Testing techniques

- For test case design
- (2.2) Test Techniques for Black Box Test
  - Equivalence Partitioning Analysis
  - Boundary-value Analysis
  - Decision Table
  - Use Case-based Test
- (2.3) Test Techniques for White Box Test
  - Control Flow Test with C0, C1 coverage
  - Sequence chart coverage test

Content

1. Testing overview
2. **Blackbox Testing**
3. Whitebox Testing

2.1. Equivalence Partitioning

- Create the encompassing test cases by analyzing the input data space and dividing into equivalence classes
  - Input condition space is partitioned into equivalence classes
  - Every input taken from a equivalence class produces the same result

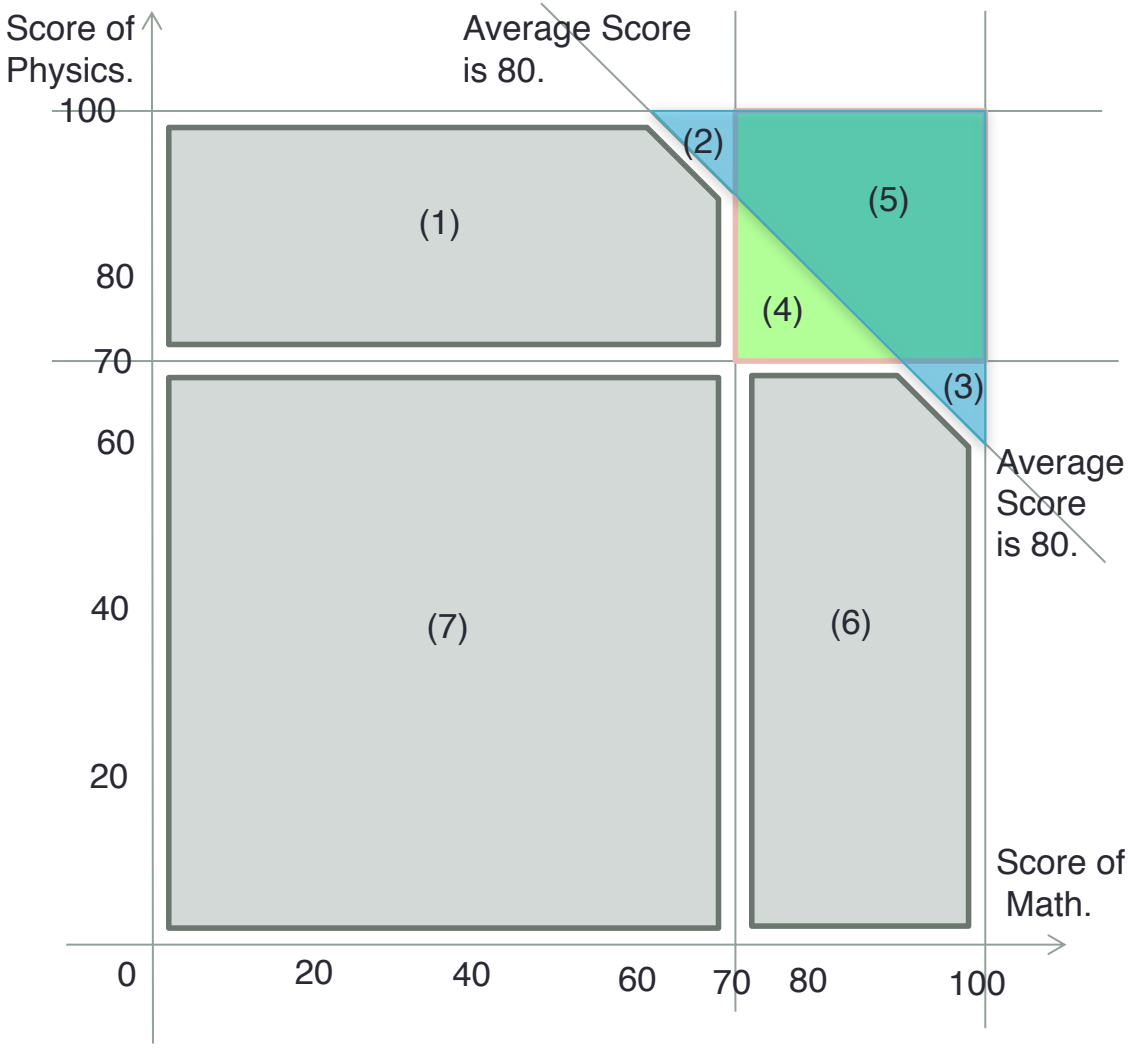
Example: Examination Judgment Program

- Program Title: “Examination Judgment Program”
- Subject: Two subjects as Mathematics, and Physics Judgment
- Specification:
  - Passed if
    - scores of both mathematics and physics are greater than or equal to 70 out of 100

      or,
    - average of mathematics and physics is greater than or equal to 80 out of 100
  - Failed => Otherwise

Equivalence Partitioning of Input space and test cases

- 7 equivalence classes => at least 7 test cases/data



Score	Math.	Physics	Result
(1)	55	85	Failed
(2)	67	97	Passed
(3)	96	68	Passed
(4)	77	80	Passed
(5)	85	92	Passed
(6)	79	58	Failed
(7)	52	58	Failed

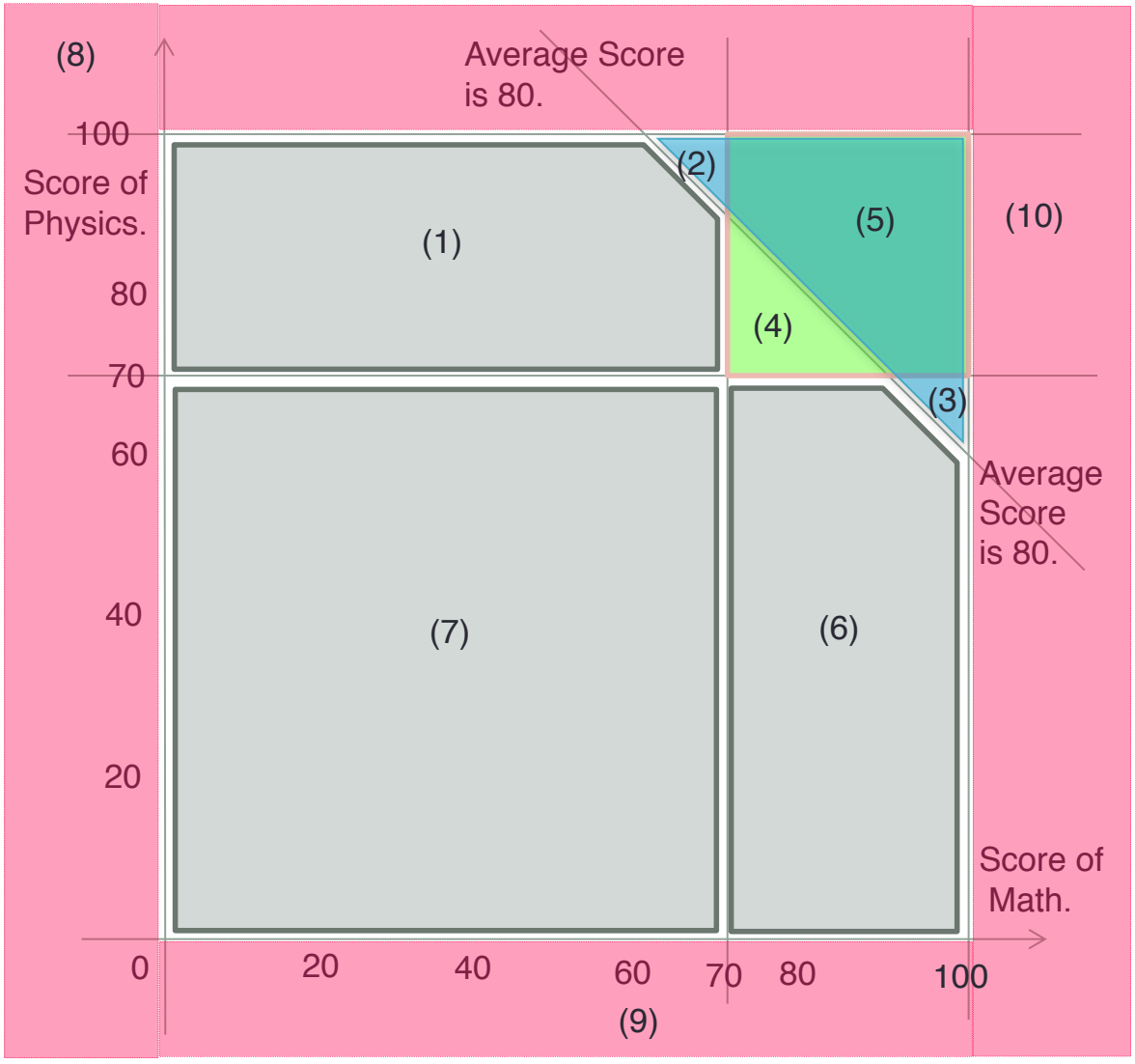
Equivalence Partitioning

Discussions and additional analysis

- Are we successful?
  - No we don't! Why?
  - →One thing is missing!
- The scope of input space analyzed is not enough!
- We must add "Invalid value" as the test data.
  - For example, some patterns of "Invalid value".
    - (8) Math = -15, Physics = 120 Both score are invalid.
    - (9) Math = 68, Physics = -66 Physics score is invalid.
    - (10) Math = 118, Physics = 85 Math score is invalid.

More equivalent classes

- Additional 3 test cases/data



Some invalid data are added.

Score	Math.	Physics	Result
(1)	55	85	Failed
(2)	67	97	Passed
(3)	96	68	Passed
(4)	77	80	Passed
(5)	85	92	Passed
(6)	79	58	Failed
(7)	52	58	Failed
(8)	-15	120	Invalid
(9)	68	-66	Invalid
(10)	118	85	Invalid

Analysis and discussions

- We tried to create encompassing test cases based on external specification.
  - Successful? “Yes” !

- Next question. The test cases/data are fully effective?
  - We have to focus on the place in which many defects are there, don’t we?
  - Where is the place ?

→ “Boundary-value analysis”

2.2. Blackbox Testing Techniques
2.2.2. Boundary-value analysis

- Extract test data to be expected by analyzing boundary input values => Effective test data
  - Boundary values can detect many defects effectively

→E.g. mathematics/physics score is 69 and 70

- The programmer has described the following wrong code:

```
if (mathscore > 70) {
        ....................
}
```

- Instead of the following correct code;

```
if (mathscore >= 70) {
        ....................
}
```

Example: Boundary-value analysis

- Boundary values of the mathematics score of small case study:

- What about the boundary value analysis for the average of mathematics and physics?

2.2. Blackbox Testing Techniques

2.2.3. Decision Table

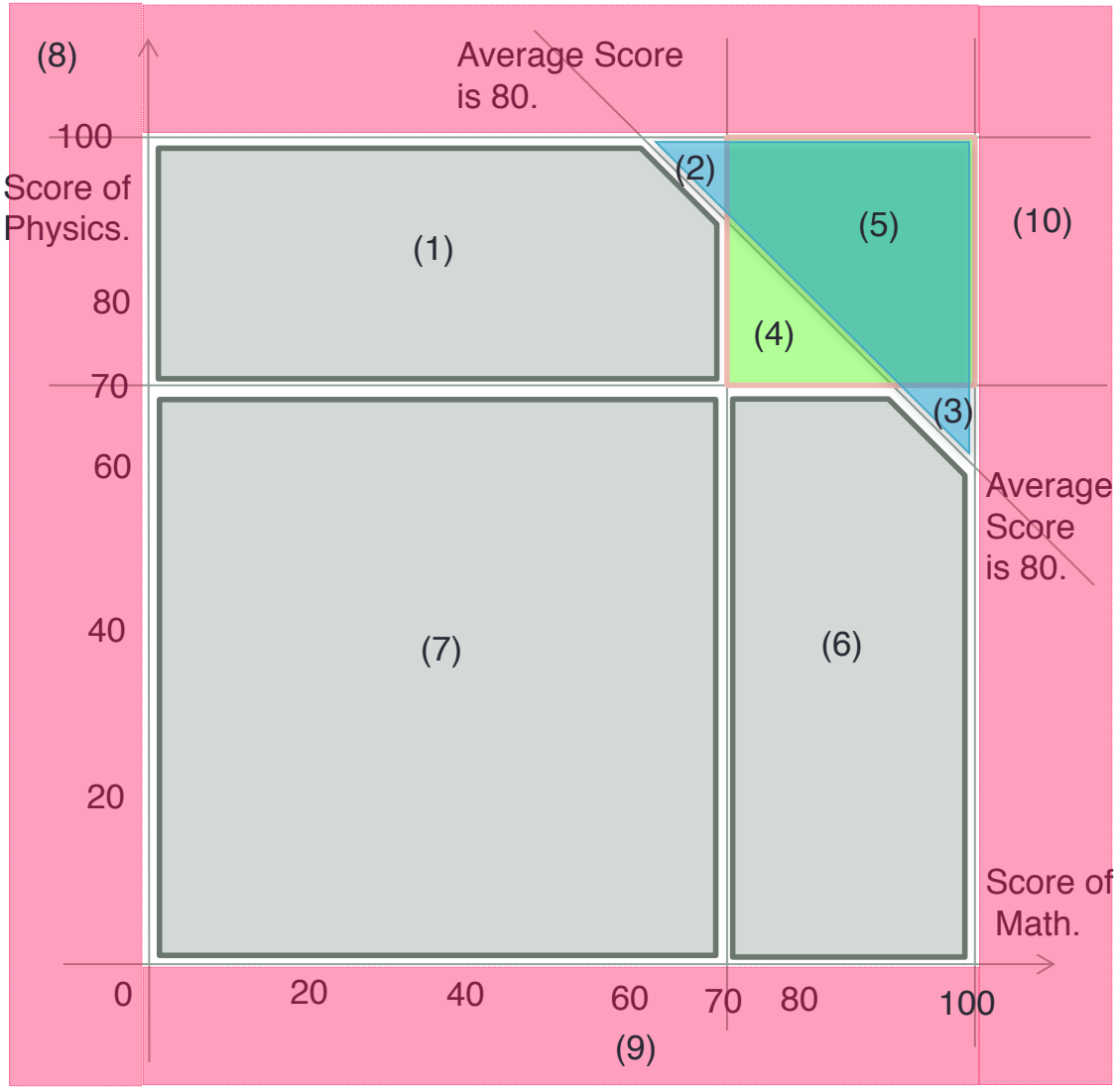
- Relations between the conditions for and the contents of the processing are expressed in the form of a table
- A decision table is a tabular form tool used when complex conditions are combined

Example: Decision table

- The conditions for creating reports from employee files

Under age 30	Y	Y	N	N
Male	Y	N	Y	N
Married	N	Y	Y	N
Output Report 1	-	X	-	-
Output Report 2	-	-	-	X
Output Report 3	X	-	-	-
Output Report 4	-	-	X	-

Decision Table for "Examination Judgement"???



Test Data from Equivalence Analysis

Score	Math.	Physics	Result
(1)	55	85	Failed
(2)	67	97	Passed
(3)	96	68	Passed
(4)	77	80	Passed
(5)	85	92	Passed
(6)	79	58	Failed
(7)	52	58	Failed
(8)	15	120	Invalid
(9)	68	-66	Invalid
(10)	118	85	Invalid

Decision Table for “Examination Judgement”

Condition1: Mathematics score=>70

Condition2: Physics score=>70

Condition3: Average of Mathematics, and Physics =>80

	TC5	TC4	TC3	TC6	TC2	TC1	TCNG	TC7
Condition1	True	True	True	True	False	False	False	False
Condition2	True	True	False	False	True	True	False	False
Condition3	True	False	True	False	True	False	True(none)	False
“Passed”	Yes	Yes	Yes	---	Yes	---	N/A	--
“Failed”	---	---	---	Yes	---	Yes	N/A	Yes

Decision Table for “Examination Judgement”

- Invalid input data (integer)
  - Condition1: Mathematics score = valid that means “0=< the score =< 100”
  - Condition2: Physics score = valid that means “0=< the score =< 100”

	TCI1	TCI2	TCI3	TCI4
Condition1	Valid	Invalid	Valid	Invalid
Condition2	Valid	Valid	Invalid	Invalid
“Normal results”	Yes	---	---	---
“Error message math”	---	Yes	---	Yes
“Error message phys”	---	---	Yes	Yes

If both of mathematics score and physics score are invalid, two messages are expected to be output. Is it correct specifications? Please confirm it?

2.2. Blackbox Testing Techniques
2.2.4. Testing for Use case

- ???
- E.g. Decision table for Login
  - Conditions
    - ???
  - Results
    - ???
- E.g. Boundary Value Analysis
  - ?

Test cases for “Log in”

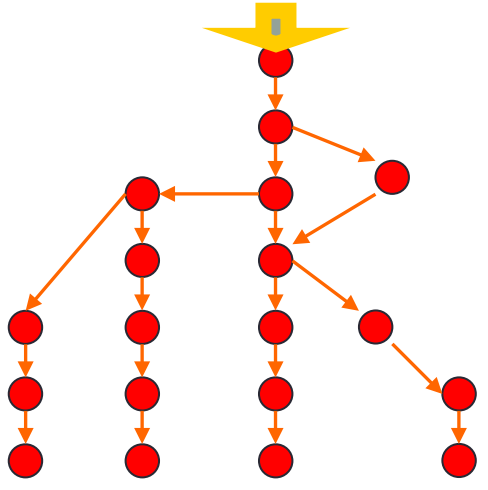
- “Thành công”
  - Mã PIN đúng
- “Thất bại”
  - Mã PIN sai và số lần sai < 3
- “Khoá tài khoản”
  - Mã PIN sai và số lần sai = 3

Mã PIN đúng	**Y**	**Y**	**N**	**N**
Số lần sai < 3	**Y**	**N**	**Y**	**N**
“Thành công”	x	N/A	-	-
“Thất bại”	-	N/A	x	-
“Khoá tài khoản”	-	N/A	-	x

- Phân tích vùng biên? Số lần sai = 2, 4 (?)

Creating test cases from use cases

- Identify all of the scenarios for the given use case
- Alternative scenarios should be drawn in a graph fo each action
- Create scenarios for
  - a basic flow,
  - one scenario covering each alternative flow,
  - and some reasonable combinations of alternative flows
- Create infinite loops



Content

1. Testing overview
2. Blackbox Testing
3. **Whitebox Testing**

3. Whitebox Testing

- Test cases should cover all processing structure in code

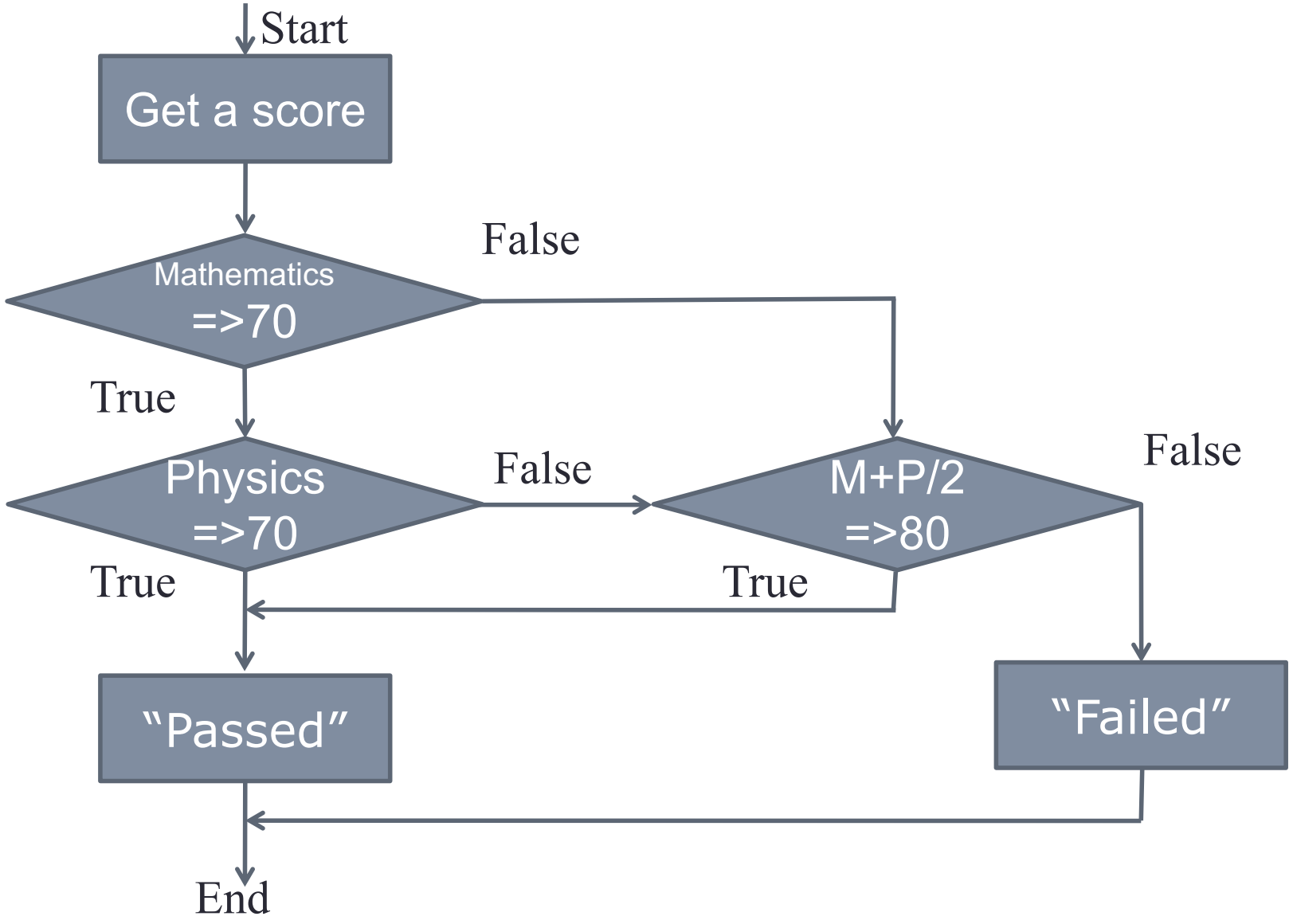
=> Typical test coverage

  - C0 measure: Executed statements #/all statements #
    - C0 measure at 100% means “all statements are executed”
  - C1 measure: Branches passed #/all blanches #
    - C1 measure at 100% means “all branches are executed”

=> Prevent statements/blanches from being left as non-tested parts

=> Cannot detect functions which aren't implemented

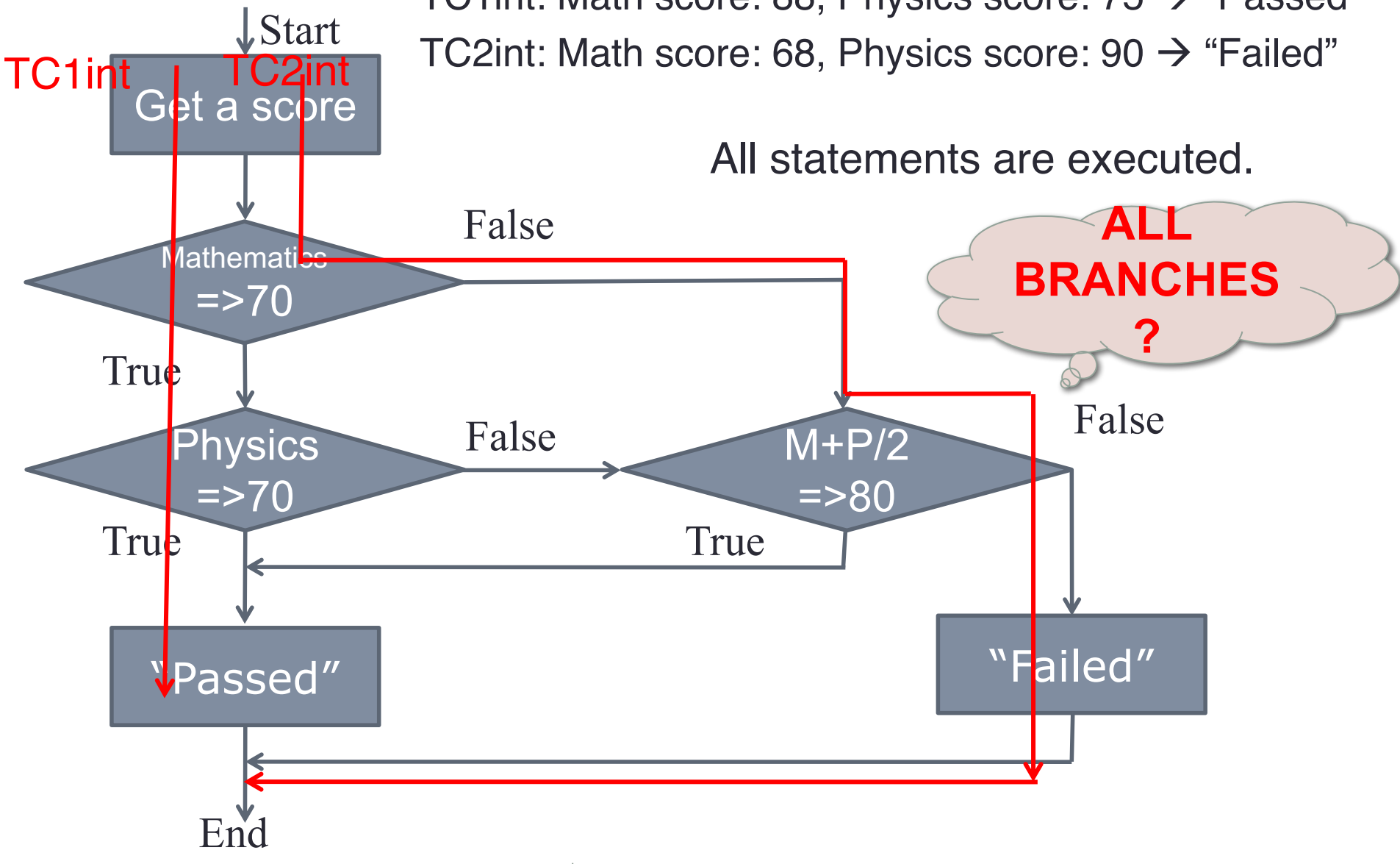
E.g. Control flow test of “Examination Judgment Program”



E.g. Control flow test of “Examination Judgment Program” – 100% C0 coverage

TC1int: Math score: 88, Physics score: 75 → “Passed”

TC2int: Math score: 68, Physics score: 90 → “Failed”

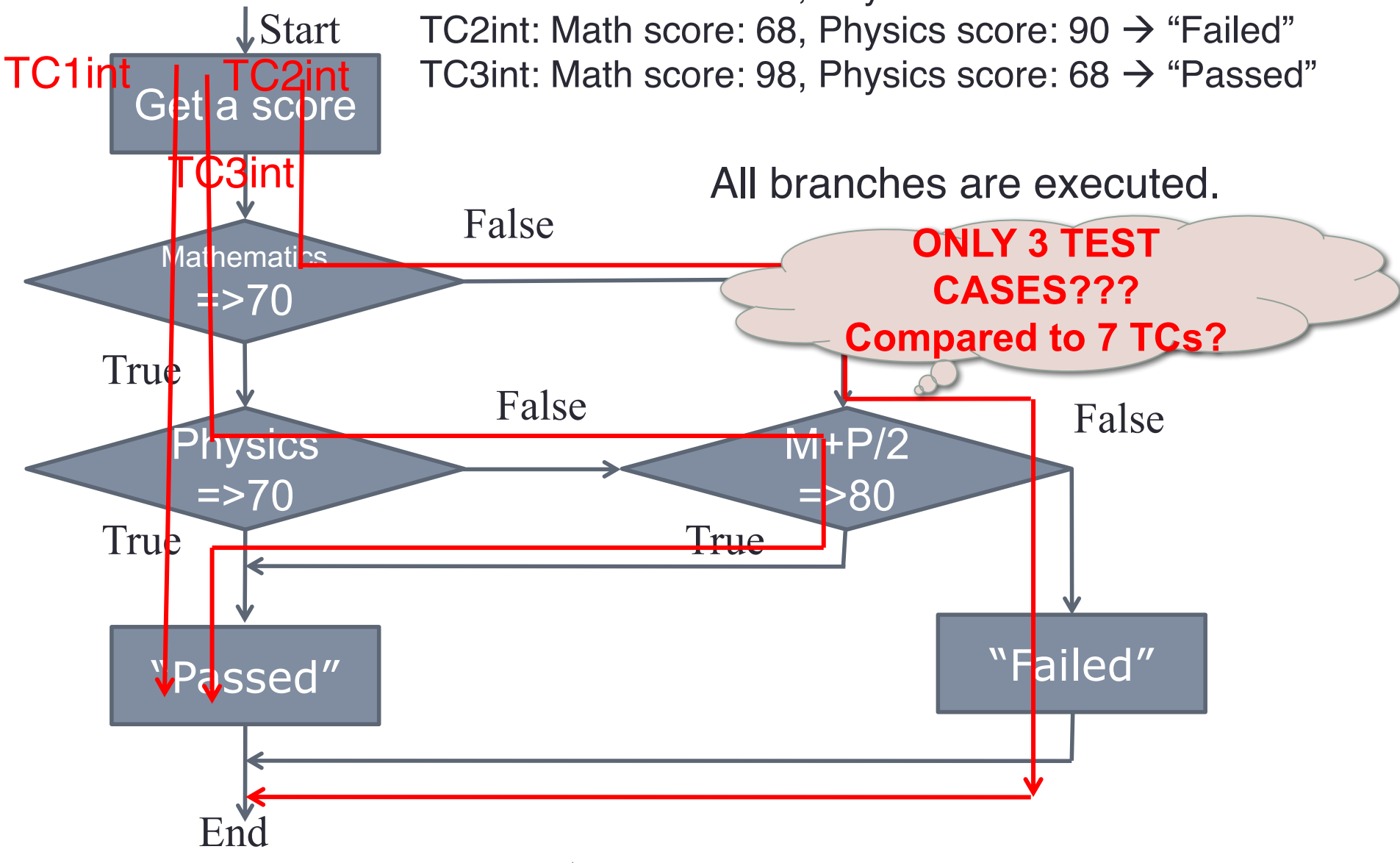


All statements are executed.

ALL BRANCHES ?

E.g. Control flow test of “Examination Judgment Program” – 100% C1 coverage

TC1int: Math score: 88, Physics score: 75 → “Passed”
TC2int: Math score: 68, Physics score: 90 → “Failed”
TC3int: Math score: 98, Physics score: 68 → “Passed”



All branches are executed.

ONLY 3 TEST CASES???
Compared to 7 TCs?

Decision Table for “Examination Judgement”

Condition1: Mathematics score=>70
Condition2: Physics score=>70
Condition3: Average of Mathematics, and Physics =>80

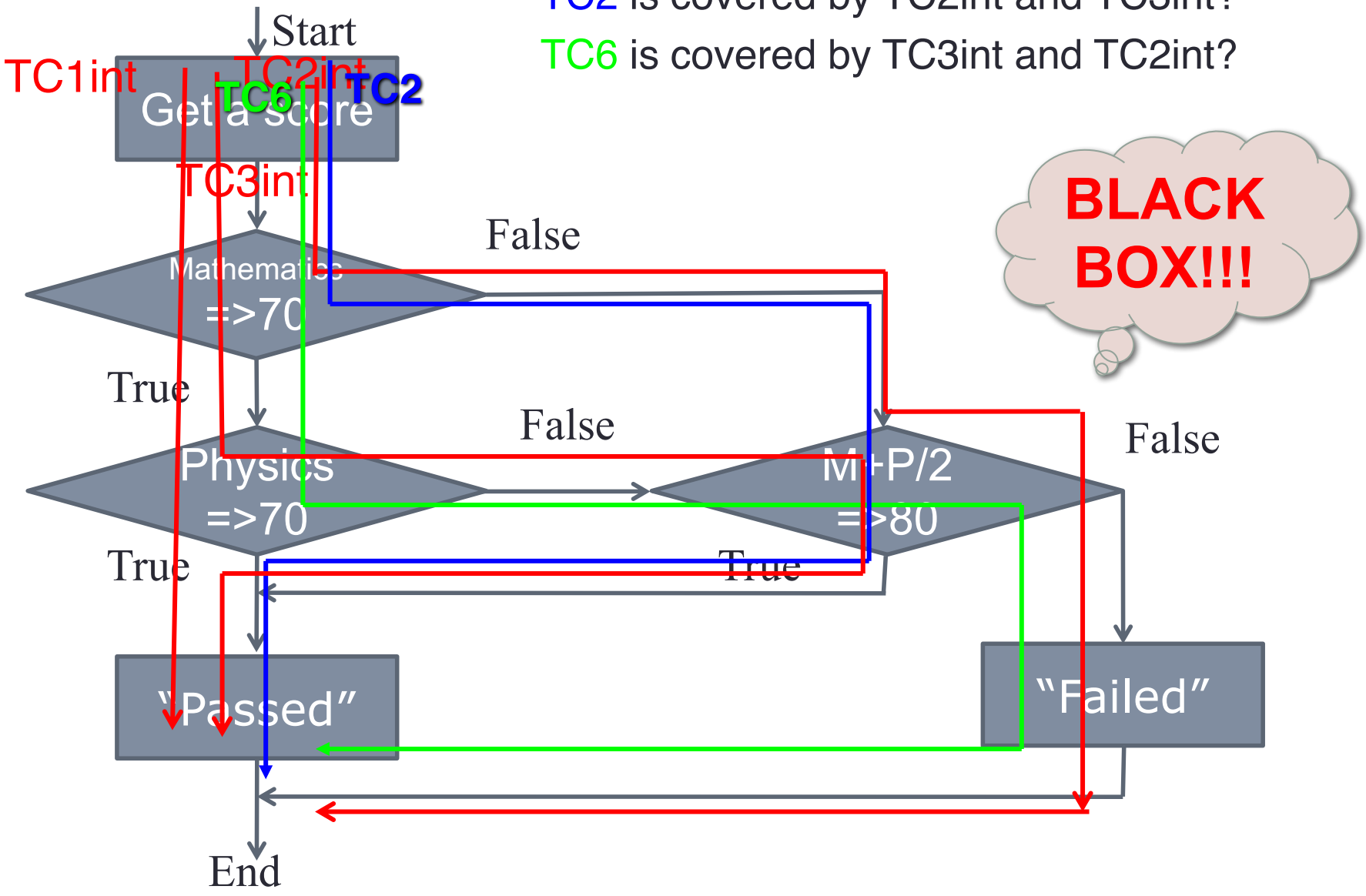
	TC5	TC4	TC3	TC6	TC2	TC1	TCNG	TC7
Condition1	True	True	True	True	False	False	False	False
Condition2	True	True	False	False	True	True	False	False
Condition3	True	False	True	False	True	False	True(none)	False
“Passed”	Yes	Yes	Yes	---	Yes	---	N/A	--
“Failed”	---	---	---	Yes	---	Yes	N/A	Yes

- One TCxint can cover plural TCs, based on the correct control flow structure
  - TC1int covers TC5 and TC4
  - TC2int covers TC1 and TC7
  - TC3int covers to TC3.
- TC2 and TC6 are left in no execution.

E.g. Control flow test of “Examination Judgment Program” – 100% C1 coverage

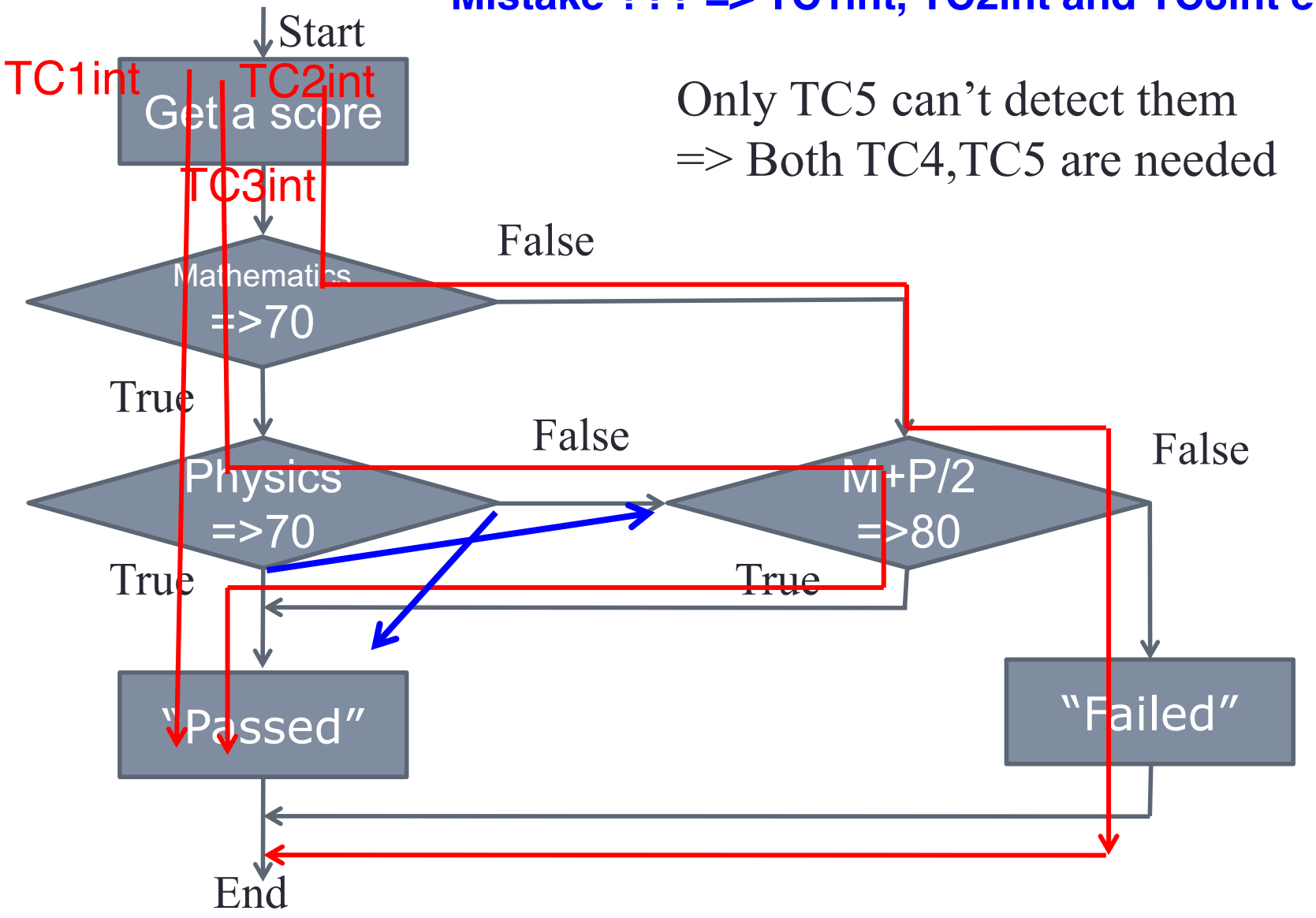
TC2 is covered by TC2int and TC3int?

TC6 is covered by TC3int and TC2int?



E.g. Control flow test of “Examination Judgment Program” – 100% C1 coverage

Mistake ??? => TC1int, TC2int and TC3int enough?



Only TC5 can’t detect them
=> Both TC4,TC5 are needed

Data/message path test for integrated test

- Execute white box test using sequence chart for integration test.

⇒Execute every message path/flow

⇒100% message path/flow coverage

- Can apply to other data/message path/flow charts or diagrams

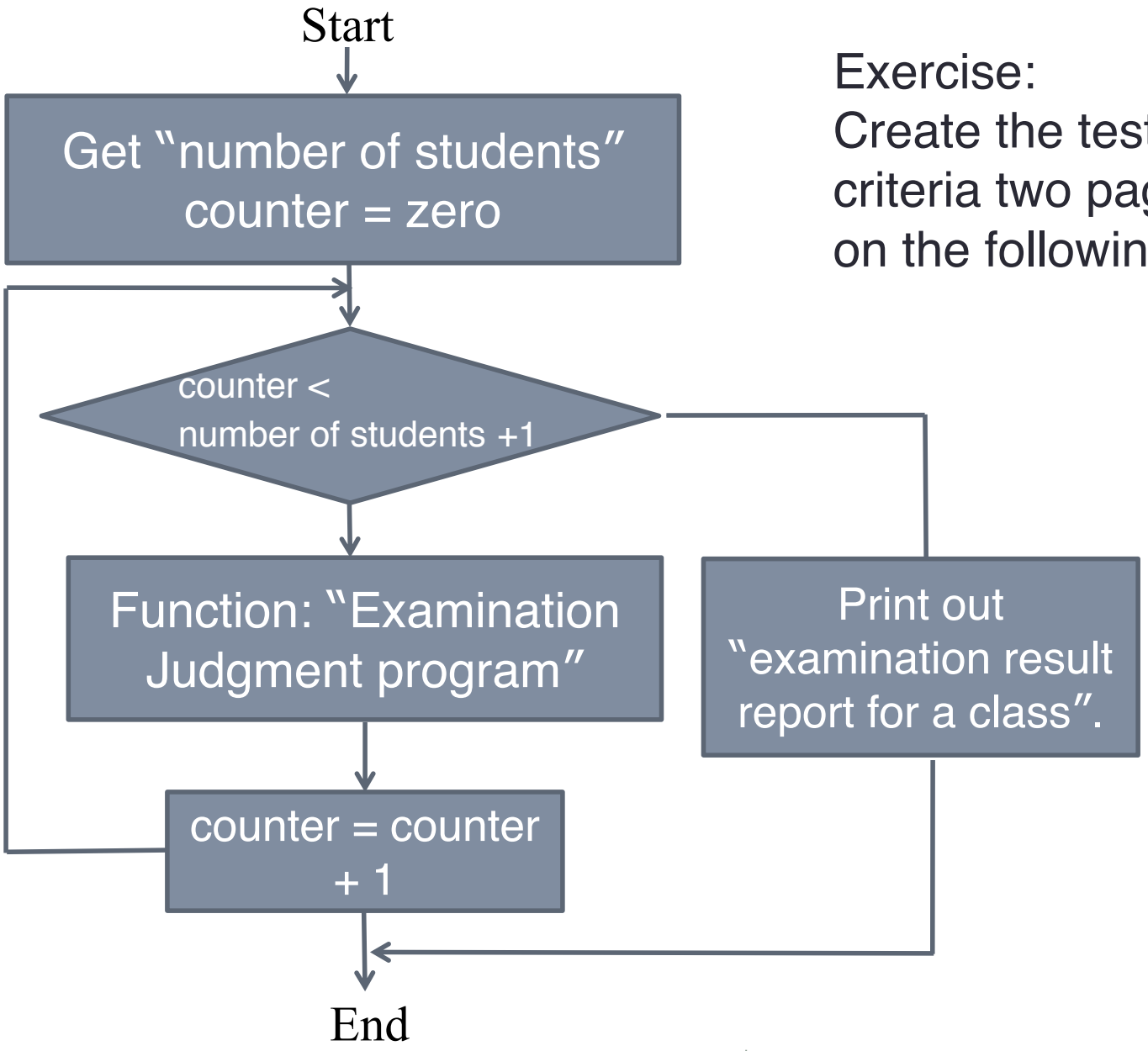
How to test a loop structure program

- For the control flow testing in the software including a loop, the following criteria are usually adopted instead of C0/C1 coverage measures.
  - Skip the loop.
  - Only one pass through the loop.
  - Typical times m passes through the loop
  - n, n-1, n+1 passes through the loop
    - n is maximum number, m is typical number (m<n)
- Example: 6 cases based on boundary-value analysis:

Examples for “Examination Judgment Program”

- Input two subjects scores, Mathematics and Physics, for each member of one class.
  - The input form is “tabular form”.
  - Class members can be allowed only 0 (zero) through 50.
- Output/Print out the “Examination result report for a class”.
  - The output form is also “tabular form” that has the columns such as student name, scores (Math., Physics), passed or failed.

Examples for “Examination Judgment Program”



Exercise:
Create the test cases using the criteria two pages before based on the following assumptions.

1. “Examination Judgment program” are already tested.
2. Input data of this module are already checked, and valid.

Examples for “Examination Judgment Program”

Loop test cases of the module are; n = 50.

“number of students” = 0,

“number of students” = 1,

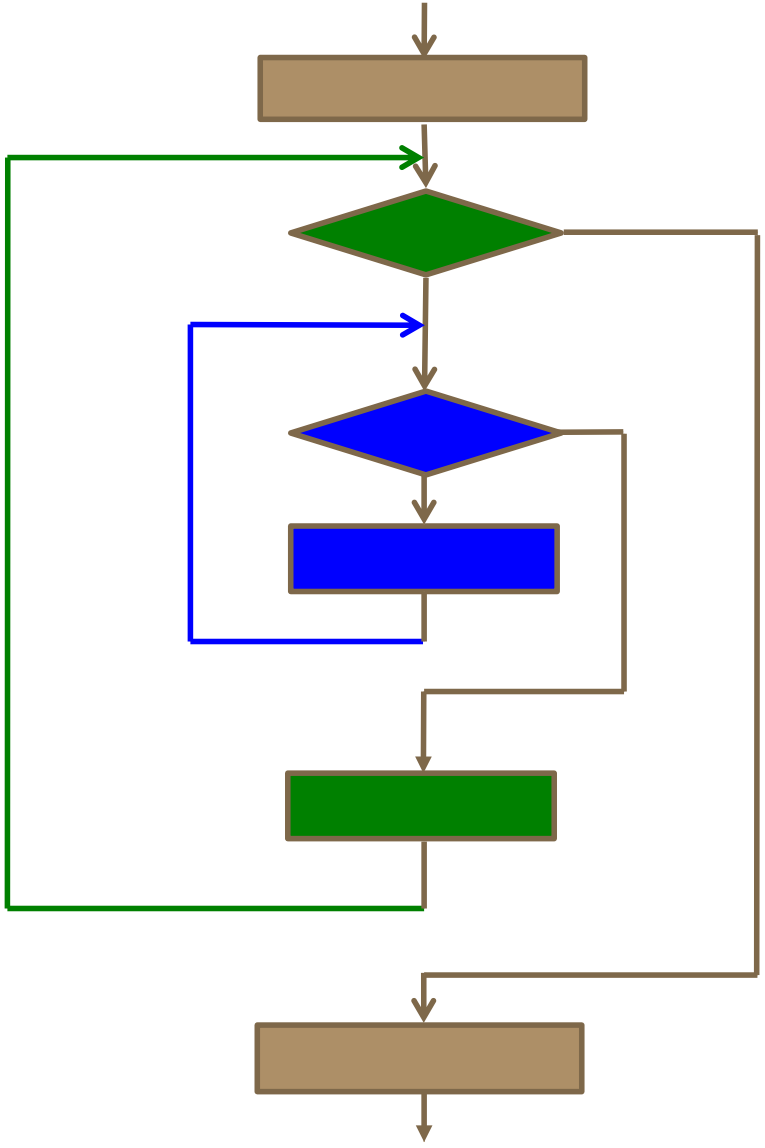
“number of students” = 20,

“number of students” = 49,

“number of students” = 50,

“number of students” = 51 → Invalid.

How to test for nested loops structure



At first, first loop's control number is determined at typical number, and second loop is tested as a simple loop.

Next, second loop's control number is determined at typical number, and first loop is tested as a simple loop.

Combination of Black/White Box test

- Advantage of Black box
  - Encompassing test based on external specification
  - Very powerful and fundamental to develop high-quality software
- Advantage of White box
  - If any paths/flows don’t appear in the written specifications, the paths/flows might be missed in the encompassing tests => White box test
    - for data of more than two years before => alternative paths
    - “0 =< score =< 100” => code: “if 0 =< score ” and “if score =< 100”

How to carry out efficient and sufficient test

- First, carry out tests based on the external specifications
  - If all test cases are successful

  => All external specifications are correctly implemented
- Second, carry out tests based on the internal specifications
  - Add test cases to execute the remaining paths/flow, within external specifications
  - If all test cases are successful with coverage = 100%

  => All functions specified in the external specification are successfully implemented without any redundant codes