Software Quality Assurance and Testing (SQAT)

White Box Testing (Part 1)

dr. Joost Schalken-Pinkster
Windesheim University of Applied Science
The Netherlands

joost@schalken.me

The contents of these course slides is (in great part) based on:
Neil Taylor (2019) *Course presentations for Software Quality Assurance and Testing (SQAT)*, Aberystwyth University.

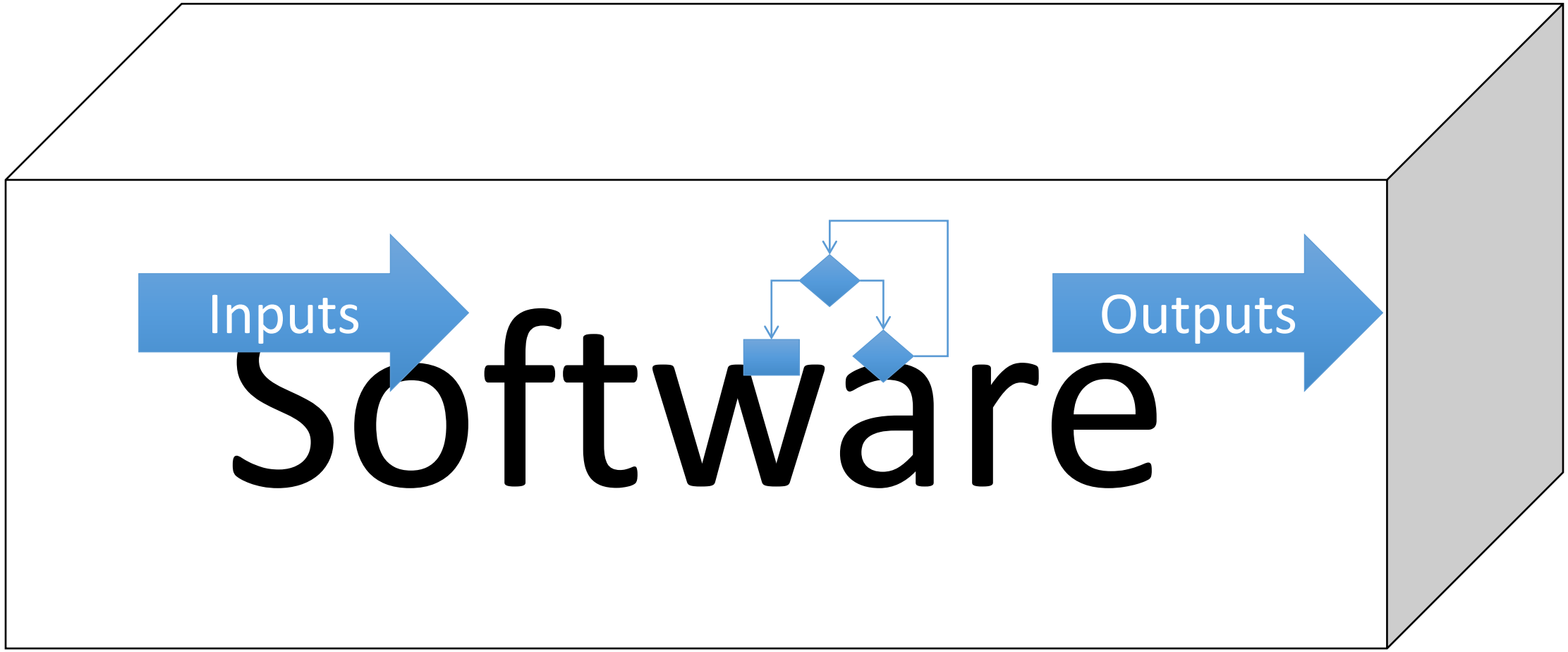
Introduction

- How do we identify test cases? (same question that we had for black-box)
- One general approach: White-box testing – testing based on the logic in a program
- In this part, we will look at code coverage

Learning Objectives

- Describe the purpose of white-box testing
  - What are the advantages?
  - What are the disadvantages?
- Consider techniques to test the structure of the code
  - Statements
  - Decisions
  - Conditions

Software as a white box



Tests & Implementation

- White-box is based on the implementation
  - Tests are written when we have code that we can run
- Black-box testing works well to generate for high-level test cases that check the general logic of the application
  - Reasonable confidence that testing will find problems
- For high confidence, e.g. in safety critical software, need to look at the code and make sure every possibility is covered
- Still worth doing black-box testing then look at EXTRA tests based on white-box approaches

Tests & Implementation

- White-box is based on the in
  - Tests are written when we ha
- Black-box testing works well
  cases that check the general
  - Reasonable confidence that t
- For high confidence, e.g. in safety critical software, need to look at the code and **make sure every possibility is covered**
- Still worth doing black-box testing then look at EXTRA tests based on white-box approaches

What does “every possibility is covered” mean?

Every Possibility?

Does this mean…?

- Every possible path through a program
  - OK for simple programs, but what about tests for programs with loops and complex logic?
- Every statement in a program
- Every decision in a program
- Every condition in a program
- Every combination of decisions/conditions

White box testing types

- Static Analysis
- Dynamic Testing with path coverage of code

Static Analysis

- Discussed during the Software Lifecycle course
- Code Inspection
  - Engineers arrange meetings to review sections of code
  - Ask questions about how the code works with specific (test) cases
- The analysis cannot be automated
  - Difficult to repeat the same analysis each time
- BUT, code inspections are an effective way to detect potential problems

Dynamic Testing: Path Coverage

- Analyse the different ways that the code can be executed
- Write tests so that every path through the program is tested
  - Remember earlier, statements, decisions, conditions
- The tests can be automated and repeated
- Tool support to check coverage
- BUT, we need to update tests when the code changes

Path Coverage

Basic principles of path coverage:

- All independent paths in a module must be traversed at least once
- All conditions (e.g. if statements) are tested for the true and false outcomes
- Review whether the tests cover the internal data structures used
- Test that loops work for their operational range (difficult to try ALL possible values for loops for the same reason that you cannot try ALL values for parameters).

Decision / Branch Coverage (1)


```
if(x > 400) {
   print "a";
} else {
   print "b";
}
```

- Example test cases for coverage of this code:
  (i) x = 401 and (ii) x = 400
- Still worth thinking about boundary conditions
- Need to decide what the correct answer is for each case, not what answer the code will give you

- Ideally, your tools help you with this
- You can use coverage tools to monitor the tests that run and find out which paths in the code are NOT covered
  - Then, you can write the tests to cover those paths
- However, there are still problems – see the following example

Branch Coverage (2)

```
MyClass pointer = null;
if(x > 400) {
    pointer = new MyClass();
}
if(y > 20){
    pointer.myMethod();
}
```

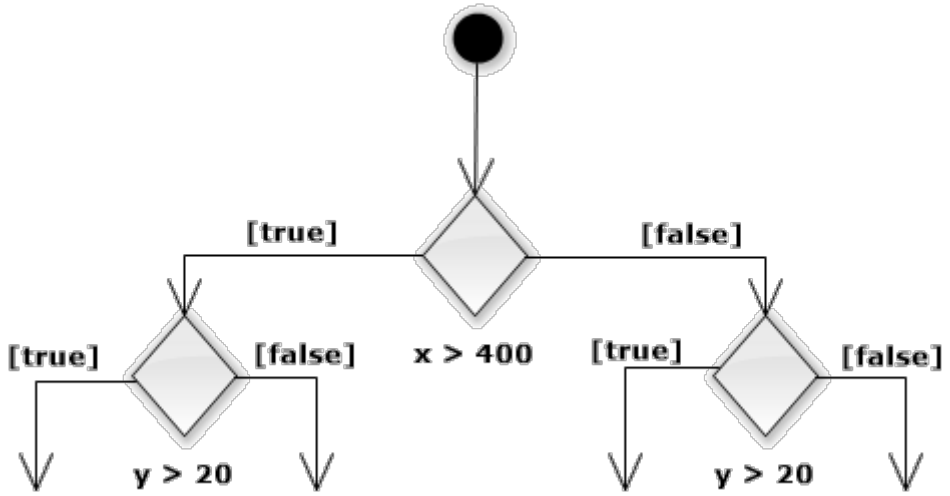
- This works OK if you test decision coverage with two cases (x = 10 and y = 10; x = 401 and y = 21).
  - It fails if you give values (x = 10 and y = 21).
  - So, you need to test all four possibilities.
- This gets harder with more combinations.

How do you work out the combinations?

Draw a decision tree

Consider tests for all possible combinations

What happens when you have a lot of combinations?



Condition	T1	T2	T3	T4
x>400	Y	Y	N	N
y>20	Y	N	Y	N

Paths through Triangle case

Condition	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
C1: ab ≤ ac + bc	F	T	T	T	T	T	T	T	T	T	T
C2: ac ≤ ab + bc		F	T	T	T	T	T	T	T	T	T
C3: cb ≤ ab + ac			F	T	T	T	T	T	T	T	T
C4: ab = ac				T	T	T	T	F	F	F	F
C5: ab = bc				T	T	F	F	T	T	F	F
C6: bc = ac				T	F	T	F	T	F	T	F
A1: Not a triangle	X	X	X								
A2: Scalene											X
A3: Isosceles							X		X	X	
A4: Equilateral				X							
A5: Impossible					X	X		X			

Example based on table 7.5 in Jorgensen book, 4th Edition, page 120

Use the decision table to build up a list of test cases. Note that some of them are not logically possible.

Summary

- What is white-box testing?
- Techniques to help specify test cases based on the code
- Static Analysis
- Dynamic Analysis
  - In this part – looking at decisions in the code

Any Questions?