Structure Based or White Box Techniques for Test Design

Hans Schaefer
hans.schaefer@ieee.org
http://www.softwaretesting.no/

What means white box-test?

Systematic execution of all parts of the program (its structure).

Component / module: The code
Integration: Call tree, interfaces, global data flow
System: Every function, every menu, every business process, ...

In practice: code coverage

How much do you know about some code that has never been executed during testing?

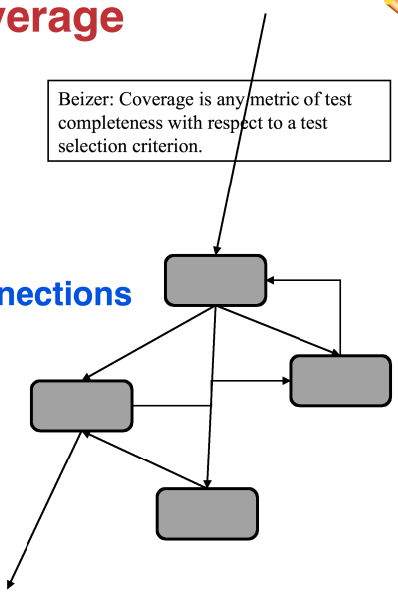
Basic thinking for coverage

Beizer: Coverage is any metric of test completeness with respect to a test selection criterion.

For every type of graphic:

- **(C0) All boxes**
- **(C1) All connections**
- **(Cn) Combinations of connections**

(Control)Flow diagram
Data structure
Data flow
Architecture
Use case
State diagram
Process flow
...



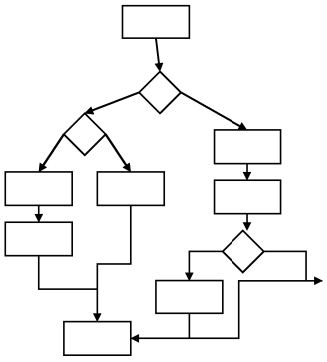
Definition of coverage criteria for code

Theory: The more details one tests, the more faults are found.

Statement Coverage
Branch Coverage (or decision coverage)

Not in exam:
Condition Coverage
Multi-Condition Coverage
.....
Path Coverage

Data (flow) Coverage
Interface Coverage

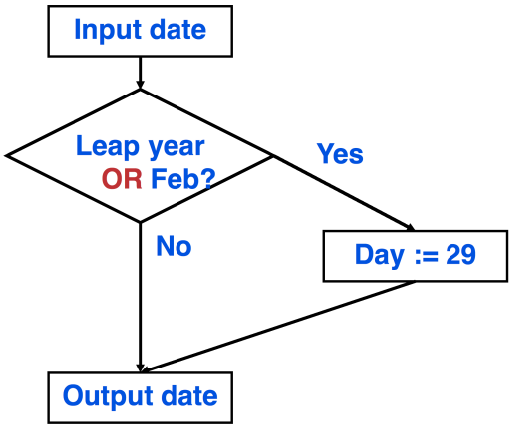


Program example Correction of date for leap year

```
Begin
Read date
If (YY = leapyear AND MM = February)
   then
      DD := 29
End if
Print DDMMYY
End
```

And now we create a fault:
Switch AND and OR...

Flow diagram (program with fault)



Statement Coverage

Percent of statements executed during testing.

Only one test case required for 100%.

Input: 150204
Expected output: 290204
Actual output: 290204
Result: OK

Wrong: OR Instead of AND

```
Begin
Read date
If (YY = leapyear OR MM = February)
        then
                DD := 29
End if
Print DDMMYY
End
```

Executed all statements -> 100% statement coverage
Does NOT find the fault!

Interactive exercise:
Is it possible to make a test case which does not execute all statements?

Branch Coverage / Decision Coverage

Percent of decisions / branches executed during testing.

It requires two test cases to run all 100%.

Input: 150204
Expected output: 290204
Actual output: 290204
Result: OK

Input: 310305
Expected output: 310305
Actual output: 310305
Result: OK

Executed all branches -> 100% branch coverage
Does NOT find the fault!

Wrong: OR Instead of AND

```
Begin
Read date
If (YY = leapyear OR MM = Feb)
        then
        DD := 29
End if
Print DDMMYY
End
```

Decision-/Branch Coverage - Why?

Every IF, WHILE, CASE statement run in every possible direction.

We find that the ELSE-part is missing.

Branch coverage requires
- 2 test cases for IF
- 2 executions for WHILE / FOR / Repeat
  - (reachable with 1 test case if intelligently chosen)
- n+1 test cases for CASE / SWITCH

This is a tougher requirement than statement coverage!

Decision-/branch Coverage - Exercise

1.) Make another test case
- How many of the lines (statements) are executed?
- How many of the branches are executed?

2.) Make a collection of test cases executing all statements and all branches AND the fault is found.

Condition Coverage (not in exam)

Percentage of conditions executed during testing.

Two test cases required for 100% condition coverage.

Input:	150204
Expected output:	290204
Actual output:	290204
Result:	OK

Input:	310305
Expected output:	310305
Actual output:	310305
Result:	OK

Wrong: OR instead of AND

Every single condition is executed both ways -> 100% condition coverage
Fault NOT found!

```
Begin
Read date
If (YY = leapyear OR MM = February)
                then
                                DD := 29
End if
Print DDMMYY
End
```


Condition Coverage - Why? (not in exam)

Every single condition executed in every direction.

We find that the condition itself is wrong. For example if the check is for March instead of February, or unequal instead of equal...

Condition coverage requires
2 test cases for every single condition.

More than statement coverage!
Not necessarily more than branch coverage!

How to find the fault? (not in exam)

Possibility1: Choose different values.

Possibility2: Require a higher test coverage criterion.

Multi-condition coverage (multiple condition coverage): Every combination of conditions is tested.

Requires four test cases in our example.

Path Coverage

Coverage of all combinations of branches in a flow diagram

Practically impossible if loops in the program (may be indefinite number of test cases)!

If no loops -> often a large number!

Which faults are found?

Faults that depend on a combination of where you come from and where you go to.

Try yourself: All possible ways from here to the bus or railway station!

Problem with White Box-Test

Test cases can cover “everything”, but still not find the fault.

Test coverage is no 100% guarantee!

In addition: Not good as test design method, because it is used AFTER the code is written, not before.

What is Good?

We get an idea about how much test is run.

Good as an exit criterion.

Easy to measure with tools.

At least SOME relation to quality.

You see the “holes” in the test.

You find extra complexity in the code.

Side-effect of coverage measurement: Bottlenecks can be found.

Practical Coverage Criteria

Near 100% statement coverage and near 85% branch coverage.

For integration: Coverage of all interfaces (CALL etc). coverage of data flow between the components.

Review everything that is not executed.

What about high risk code? (not for exam)

Higher level coverage criteria

Condition coverage (in addition to statement and branch)

Combination of conditions

LCSAJ - Linear Code Sequence and Jump (software segment of executable code)

Data flow coverage (every variable from places it is set to places it is used)

Not much used!

Coverage Measurement in Practice

Tools available for most languages, often as part of IDE (integrated development environment, compiler).

Three parts:

- Preprocessor: Puts probes (counters) into the code.
- Runtime routine: Counts execution and writes counting into a table or file.
- Postprocessor: Result summary and listing.

Problems

- Code will be slower
- It works in practice until about 50.000 lines.
- Tools require a coding standard.
- Extra work to analyze and document coverage.

Good

- Better control with the test
- Better test
- Finds timing problems, bottlenecks etc.

Exercise

Below is a summary of coverage, as typical from a tool. Which components do you want to have an extra look at and why?

Component no.	# program lines	# branches	covered % of branches
1	200	15	93
2	50	10	100
3	50	5	40
4	120	15	80
5	600	5	80
6	100	48	98
7	100	10	90
8	80	30	30
9	50	5	100
10	50	5	20
11	80	7	0
12	60	10	80
13	400	125	74

Types of defects not found by white box testing

Missing requirements
Data dependent bugs
Wrong order of calling modules
Is the whole structure correct?

Summary

- Instrument the code and measure the test coverage!
- Whatever criterion is better than none.
- Check what has not been executed!
- Find the holes in the test!
- Identify extra (dead) code!

- You must be able to tell more than just ”I have tested it”.

Literature and information

- Linz, Schaefer, Spillner, Software Testing Foundations, 2nd ed. Rocky Nook, 2007.
- Glenford Myers, The Art of Software Testing, 1st ed., Wiley, 1979
- Boris Beizer, Software Testing Techniques, Van Nostrand Reinhold, 1990 and Black box-testing, 1995
- Thomas McCabe: Structured Testing, IEEE Tutorial, IEEE Catalog No. EHO 200-6, 1983
- Hewlett Packard Journal, June 1987, pg. 13 ff
- Paul C. Jorgensen, Software Testing - A Craftman's Approach, 2nd ed., CRC Press 2002
- Keith Miller, A modest proposal for software testing, IEEE Software, 2/2001
- Harry M. Sneed, "Data Coverage Measurement in Program Testing", Proceedings of the Workshop on Software Testing, 15. - 17. July 1986, IEEE Catalog no. 86TH0144-6, pg. 34 ff.
- Joseph R. Horgan, Saul London, Michael R. Lyu, "Achieving Software Quality with Testing Coverage Measures", IEEE Computer, Sept. 1994
- British Standard 7925-2, "Standard for Component Testing"
- Brian Marick: An article on pitfalls with code coverage at the Testing Craft site. http://www.testing.com/writings/coverage.pdf
- A tool supplier's very detailed description of coverage criteria, background, risks and benefits: http://www.bullseye.com/coverage.html