IT Quality and Software Test

Lesson 6
Test Design Techniques
Dynamic Testing II
V1.1

Uwe Gühl

Winter 2011/ 2012

Contents

- Test Design Techniques – Dynamic Testing II
  - White-box Techniques
    (or Structure-based Techniques)
    - Structural Coverages
    - Statement Testing and Coverage
    - Decision Testing and Coverage
    - Statement Coverage / Decision Coverage
    - Other Structure-based Techniques
    - Structural Coverages – Challenges and Hints
    - Cyclomatic Complexity
  - Experience-based Techniques
  - Choosing Test Techniques

White-box Techniques

- White-box testing is based on an identified structure of the software or the system, as seen in the following examples:
  - **Component level:** The structure of a software component, as for example
    - statements,
    - branches,
    - decisions,
    - distinct paths.
  - **Integration level:** The structure may be a call tree (a diagram in which modules call other modules).
  - **System level:** The structure may be a
    - menu structure,
    - web page structure,
    - business process.

White-box Techniques
Structural Coverages

Structural Coverage

- based on control flow analysis,
- gives no advice concerning test case creation,
- good starting point for thorough testing.

Other criteria for designing tests should be included in an effective testing strategy, based on

- data flow, and
- required functionality

White-box Techniques
Structural Coverages

Structural Coverage Metrics cover

- Statement testing
- Decision testing

- Branch testing – Many sources mention that Decision testing is same like Branch testing, but ISTQB syllabus says: Note that decision and branch testing are the same at 100% coverage, but can be different at lower coverage levels.

White-box Techniques
Structural Coverages

More Structural Coverage Metrics* are

- Branch testing
- Condition testing
- Multiple condition testing
- Condition determination testing
- Linear Code Sequence and Jump (LCSAJ) or loop testing
- Path testing

* Not discussed – see Syllabus ISTQB Advanced Level

White-box Techniques
Statement Testing and Coverage

- Statement coverage
  - done in component testing
  - assessment of the percentage of executable statements that have been covered by a test case suite.
  - Goals:
    - Execution of all statements of a program at least once
    - Ensuring there is no unreachable code (“dead code”)

White-box Techniques
Statement Testing and Coverage

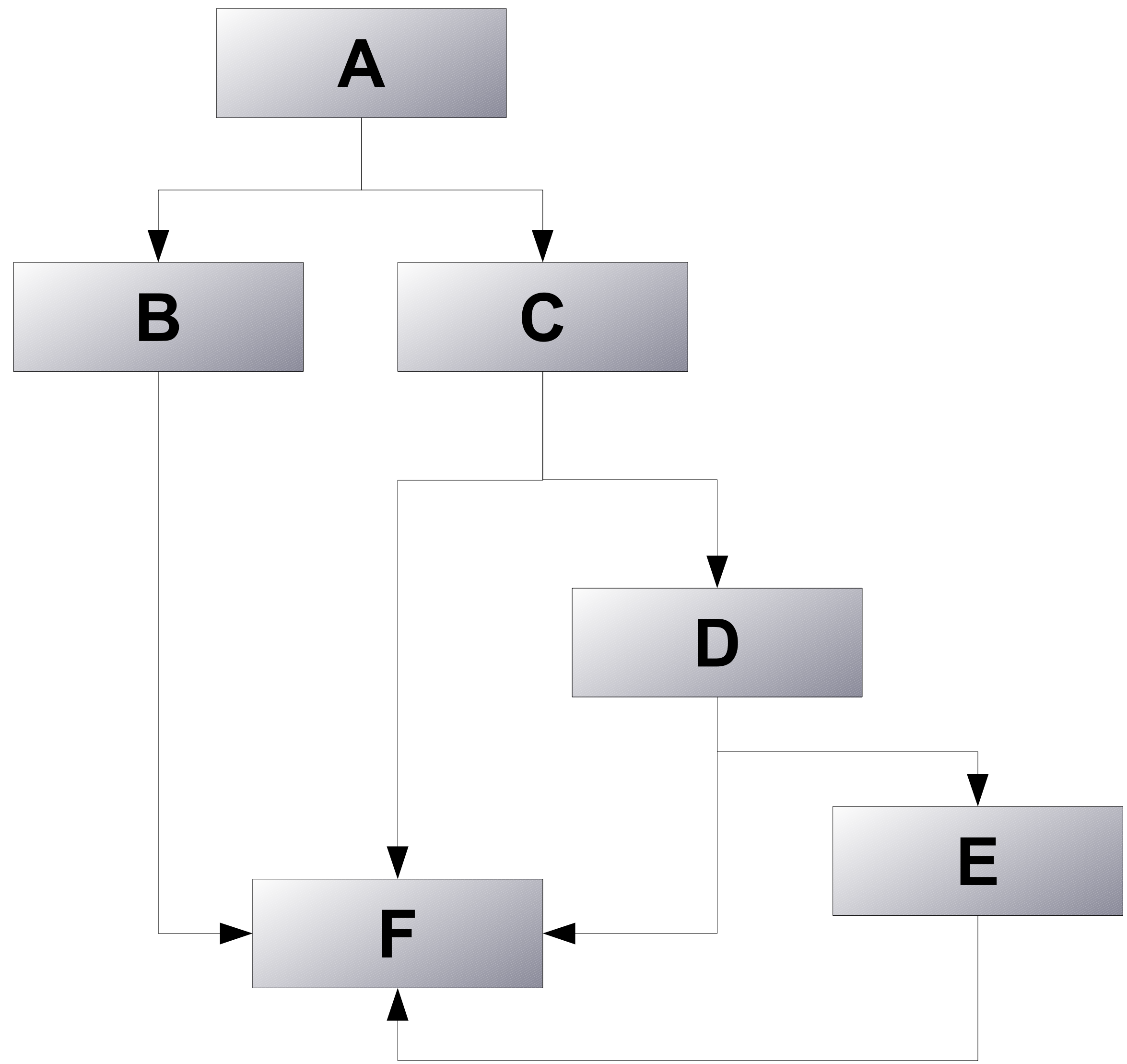
- Statement coverage is determined by

$$\frac{testedStatements}{allStatements}$$

  - testedStatements = number of executable statements covered by (designed or executed) test cases
  - allStatements = number of all executable statements in the code under test.

White-box Techniques Statement Testing and Coverage

- Example 1
  2 Test Cases for 100 % Statement Coverage
  - A, B, F
  - A, C, D, E, F



White-box Techniques
Statement Testing and Coverage

- Example 2
  1 Test Case for
  100 % Statement
  Coverage

  TC1: x = 1, y =2
  Result: z = 3

```
/* z is greater value+1*/
int foo(int x, int y) {
    int z = x;
    if (y > x) {
        z = y;
    }
    z = z +1;
    return z;
}
```

White-box Techniques Decision Testing and Coverage

- Decision coverage, related to branch testing, is the assessment of the percentage of decision outcomes (e.g., the True and False options of an IF statement) that have been exercised by a test case suite.
- The decision testing technique derives test cases to execute specific decision outcomes.
- Branches originate from decision points in the code and show the transfer of control to different locations in the code.

White-box Techniques Decision Testing and Coverage

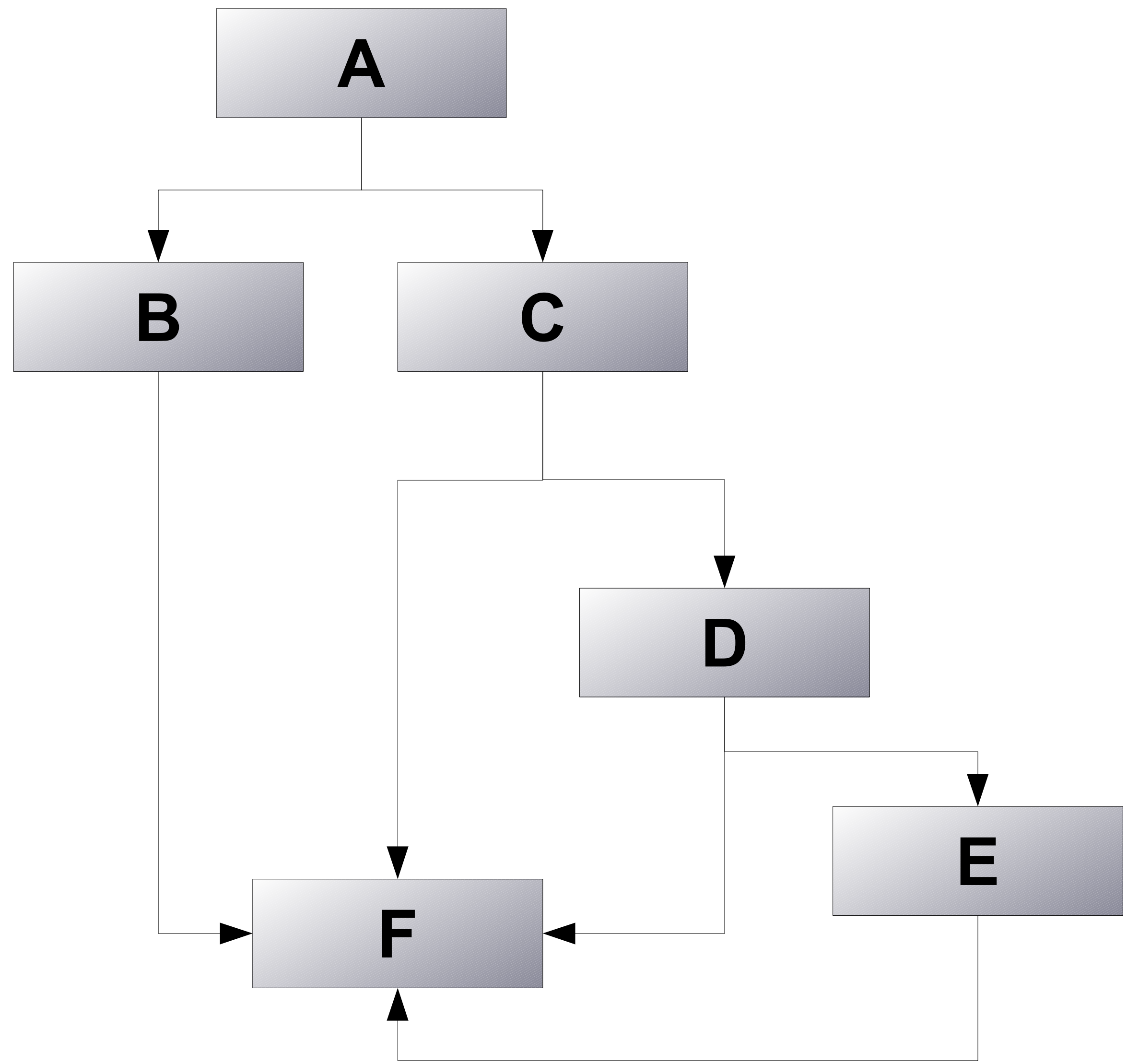
- Decision coverage is determined by

$$\frac{testedDecisions}{allDecisions}$$

  - testedDecisions = number of all decision outcomes covered by (designed or executed) test cases
  - allDecisions = number of all possible decision outcomes in the code under test.
- Decision testing is a form of control flow testing as it follows a specific flow of control through the decision points.

White-box Techniques
Decision Testing and Coverage

- Example 1
  4 Test Cases for 100 % Decision Coverage
  - A, B, F
  - A, C, F
  - A, C, D, F
  - A, C, D, E, F



White-box Techniques
Decision Testing and Coverage

- Example 2
  2 Test Cases for
  100 % Decision
  Coverage

  TC1: x = 1, y =2
  Result: z = 3

  TC2: x = 3, y = 2
  Result: z = 4

```
/* z is greater value+1*/
int foo(int x, int y) {
    int z = x;
    if (y > x) {
        z = y;
    }
    z = z +1;
    return z;
}
```

White-box Techniques

Statement Coverage / Decision Coverage

- Decision coverage is stronger than statement coverage:
  - 100% decision coverage guarantees 100% statement coverage,
  - but not vice versa.

White-box Techniques
Statement Coverage / Decision Coverage

- Means
  50 % Decision /
  Branch coverage
  also
  50% State coverage?

  ==> No!

Code example

```
int foo(int x, int y) {
    int a = 0;
    if (x>0) {
        a = a+1;
        a = a+1;
    } else
        a = a+1;
}
```

[Büc10]

White-box Techniques
Statement Coverage / Decision Coverage

Assessment

- Both statement and decision coverage are weak criteria
- “Statement-coverage criterion is so weak that it is generally considered useless.” [p. 37 Mye04]
- Statement coverage and decision coverage should be considered as a minimal requirement.

White-box Techniques
Other Structure-based Techniques

- There are stronger levels of structural coverage beyond decision coverage, for example,
  - Condition coverage and
  - Multiple condition coverage.
- The concept of coverage can also be applied at other test levels.
- For example, at the integration level the percentage of modules, components or classes that have been exercised by a test case suite could be expressed as module, component or class coverage.

White-box Techniques
Structural Coverages

Challenges [Büc10]

- Different metrics definitions around
- Sometimes you can't achieve 100 % coverage
- Coverage metrics have different names (e.g. Abbreviations have different meanings, like C0 or C1 for statement coverage)
- Not always clear, how coverages were measured (important when using tools)
- Kind of coding influences results of coverage analysis

White-box Techniques
Structural Coverages

Hints [Büc10]

- Clarify, that you talk about the same structural coverage definitions
- Clarify in using coverage measuring tools, how these work
- Don't be relaxed because of 100% code coverage

White-box Techniques
Cyclomatic Complexity

- Complexity
  The degree to which a component or system has a design and / or internal structure that is difficult to understand, maintain and verify.
- The more complex a component or a system is, the higher the probability that
  - test coverage is not complete
  - defects occur
  - maintenance gets more difficult

White-box Techniques Cyclomatic Complexity

- Cyclomatic complexity metric
  - could be used to measure the complexity of a module's decision structure.
  - is the number of linearly independent paths and therefore, the minimum number of paths that should be tested.

White-box Techniques Cyclomatic Complexity

- Cyclomatic complexity [McC76]:
  The number of independent paths through a program. Cyclomatic complexity M is defined as:

  $$M = L - N + 2P$$, where

  - L = number of edges/links in a graph
  - N = number of nodes in a graph
  - P = number of disconnected parts of the graph (e.g. a called graph or subroutine)

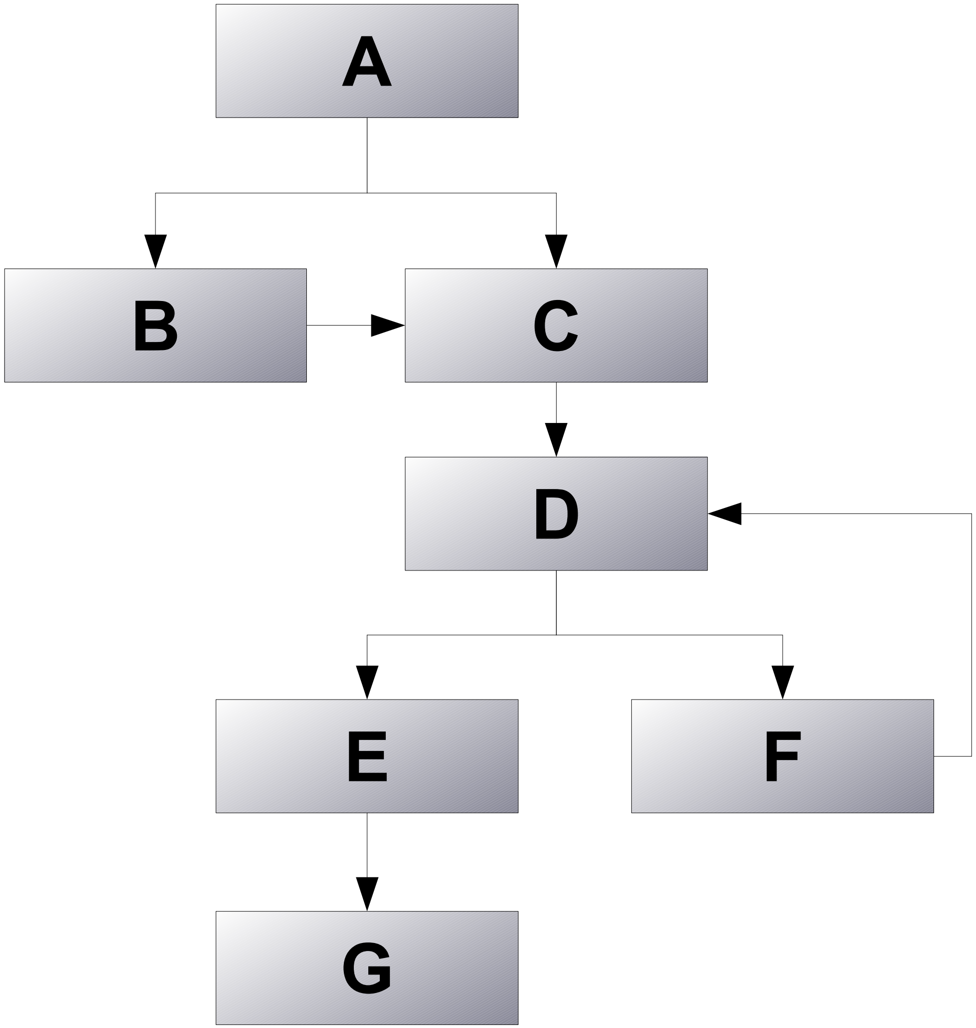
White-box Techniques
Cyclomatic Complexity

Example:

$M = L - N + 2P$

$= 8 - 7 + 2$

$= 3$



White-box Techniques Cyclomatic Complexity

- Cyclomatic complexity [McC76]:
  Alternative calculation, if you have a program with binary conditions only:
  $M = b + 1$, where
  - b = number of binary conditions

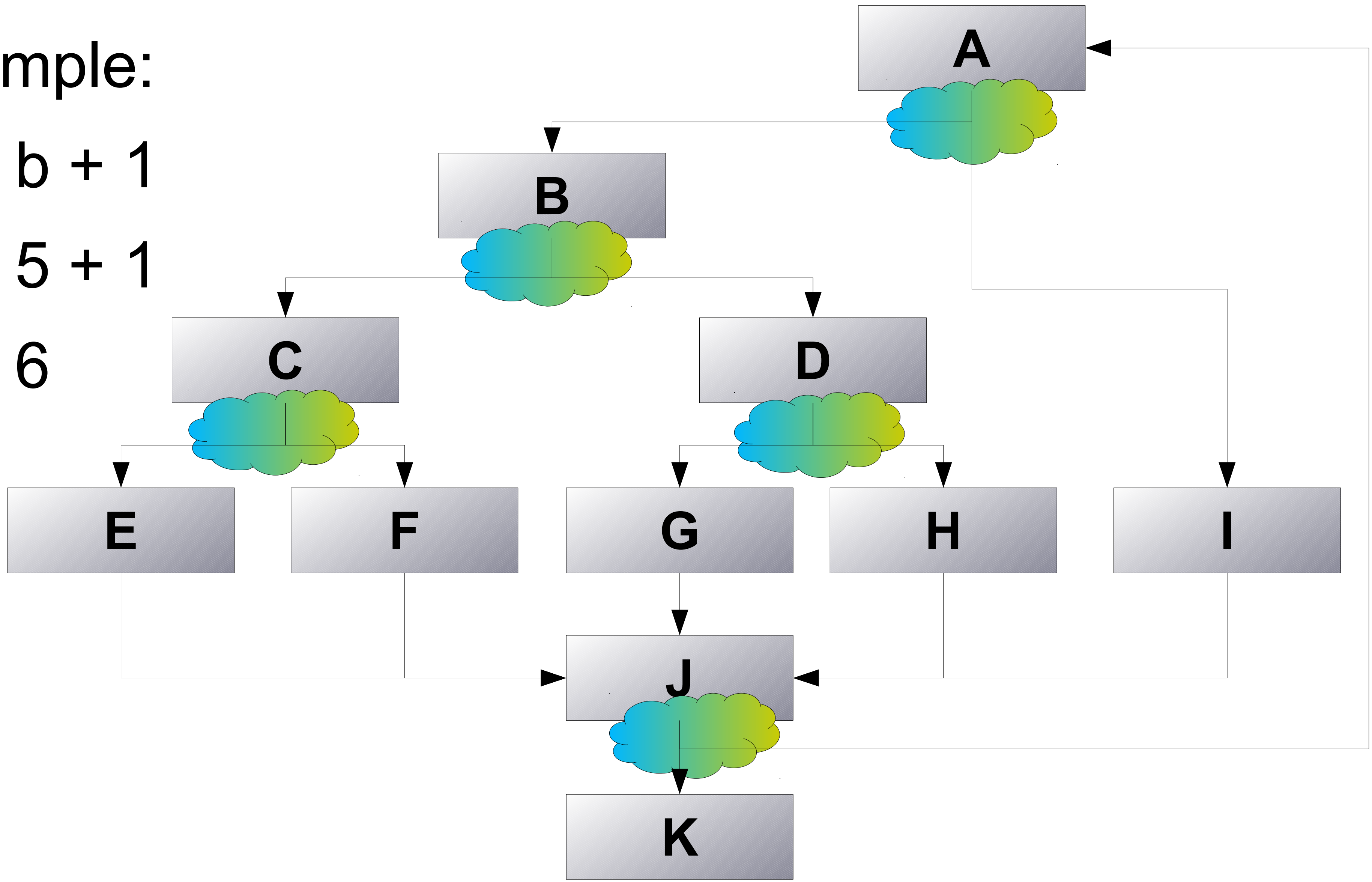
White-box Techniques Cyclomatic Complexity

Example:

M = b + 1

= 5 + 1

= 6



White-box Techniques Cyclomatic Complexity

Cyclomatic Complexity M

- M is the upper bound for the number of test cases for decision coverage.
- M > 10 should be prevented (following McCabe)

White-box Techniques Cyclomatic Complexity

- The higher M, the higher the probability of errors
  - Studies of Sharpe [Sha08] have shown
    - M = 11 had lowest probability of 28% of being fault-prone.
    - M = 38 had a probability of 50% of being fault-prone.
    - M ≥ 74 had 98 % plus probability of being fault-prone.
  - Walsh collected data of 276 modules [McC96, Wal79]:

    ≈ 50 % had M < 10 with 4,6/100 statements error rate

    ≈ 50 % had M ≥ 10 with 5,6/100 statements error rate

White-box Techniques Cyclomatic Complexity

- Weakness
  - Assumption that faults are proportional to decision complexity does not consider processing complexity and database structure.
  - It does not differ between different kinds of decisions, which is counter intuitive
    - An "IF-THEN-ELSE" statement is treated the same as a relatively complicated loop
    - Also CASE statements are treated the same as nested IF statements
  - It's possible that a program gets a high value for M, but is easy understandable (see example next page).

White-box Techniques Cyclomatic Complexity

Example:

```
const String monthsName (const int nummer) {
  switch(nummer)  {
   case 1: return "January";
   case 2: return "February";
   case 3: return "Mars";
   case 4: return "April";
   case 5: return "May";
   case 6: return "June";
   case 7: return "July";
   case 8: return "August";
   case 9: return "September";
   case 10: return "October";
   case 11: return "November";
   case 12: return "December";
  }
  return "unknown month number";
}
```

Program has a high cyclomatic complexity M = 13.

But it is easy to understand.

Experience-based Techniques

- Experience-based testing is where tests are derived from the tester’s skill and intuition and their experience with similar applications and technologies.
- When used to augment systematic techniques, these techniques can be useful in identifying special tests not easily captured by formal techniques, especially when applied after more formal approaches.
- However, this technique may yield widely varying degrees of effectiveness, depending on the testers’ experience.

Experience-based Techniques Error guessing

- Commonly used experience-based technique
- Testers anticipate defects based on experience.
- A structured approach called “fault attack”
  - Enumerate a list of possible defects, based on
    - experience,
    - available defect data,
    - common knowledge about why software fails.
  - Design tests that attack these defects.

Experience-based Techniques
Exploratory testing

Exploratory testing is

- concurrent
  - test design,
  - test execution,
  - test logging and learning,
- based on a test charter containing test objectives,
- carried out within time-boxes.

Experience-based Techniques Exploratory testing

- Approach is useful under following conditions
  - Only few or inadequate specifications available
  - Severe time pressure,
  - In order to augment or complete other, more formal testing.
- It can serve as a check on the test process, to help ensure that the most serious defects are found.

Choosing Test Techniques

- The choice of which test techniques to use depends on a number of factors, including
  - the type of system,
  - regulatory standards,
  - customer or contractual requirements,
  - level of risk,
  - type of risk,
  - test objective,
  - documentation available,

Choosing Test Techniques

- The choice of which test techniques to use depends on a number of factors, including (c'td)
  - knowledge of the testers,
  - time and budget,
  - development life cycle,
  - use case models and
  - previous experience with types of defects found.

Choosing Test Techniques

- Some techniques are more applicable to certain situations and test levels; others are applicable to all test levels.
- When creating test cases, testers generally use a combination of test techniques including
    - process,
    - rule and data-driven techniques

  to ensure adequate coverage of the object under test.

Sources

- International Software Testing Qualifications Board: Certified Tester Foundation Level Syllabus, Released Version 2011, http://istqb.org/display/ISTQB/Foundation+Level+Documents
- International Software Testing Qualifications Board: Certified Tester Advanced Level Syllabus, Released Version 2007, http://istqb.org/display/ISTQB/Advanced+Level+Documents
- [Büc10] Frank Büchner: Irrtümer über Code Coverage, http://www.elektronikpraxis.vogel.de/index.cfm?pid=890&pk=247210&p=1; http://www.elektronikpraxis.vogel.de/themen/embeddedsoftwareengineering/testinstallation/articles/252993/
- [McC76] T. McCabe, A complexity measure, in: IEEE Transactions on Software Engineering, Vol. 2, pp. 308-320, 1976.

Sources

- [McC96] NIST Special Publication 500-235, Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, Computer Systems Laboratory NIST Gaithersburg, MD 20899-0001, September 1996, http://www.mccabe.com/pdf/mccabe-nist235r.pdf
- [Mye04] Glenford J. Myers: The Art of Software Testing, Second Edition, John Wiley & Sons, Inc., 2004
- [Sha08] Rich Sharpe: McCabe Cyclomatic Complexity: the proof in the pudding, 2008, http://www.enerjy.com/blog/?p=198
- [Wal79] Walsh, T., “A Software Reliability Study Using a Complexity Measure,” AFIPS Conference Proceedings, AFIPS Press, 1979.
- [Wik12] Wikipedia.org, Code coverage, 2012, http://en.wikipedia.org/wiki/Code_coverage