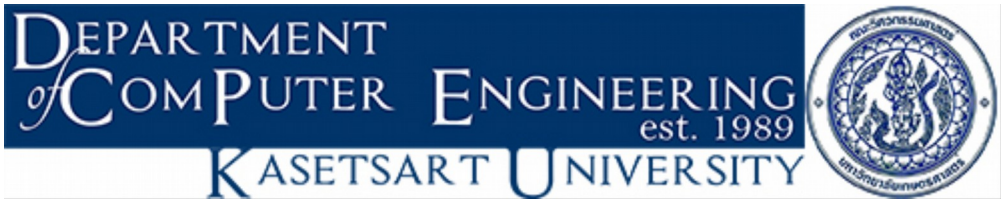


Software Testing

Lesson 5 – Dynamic Testing I

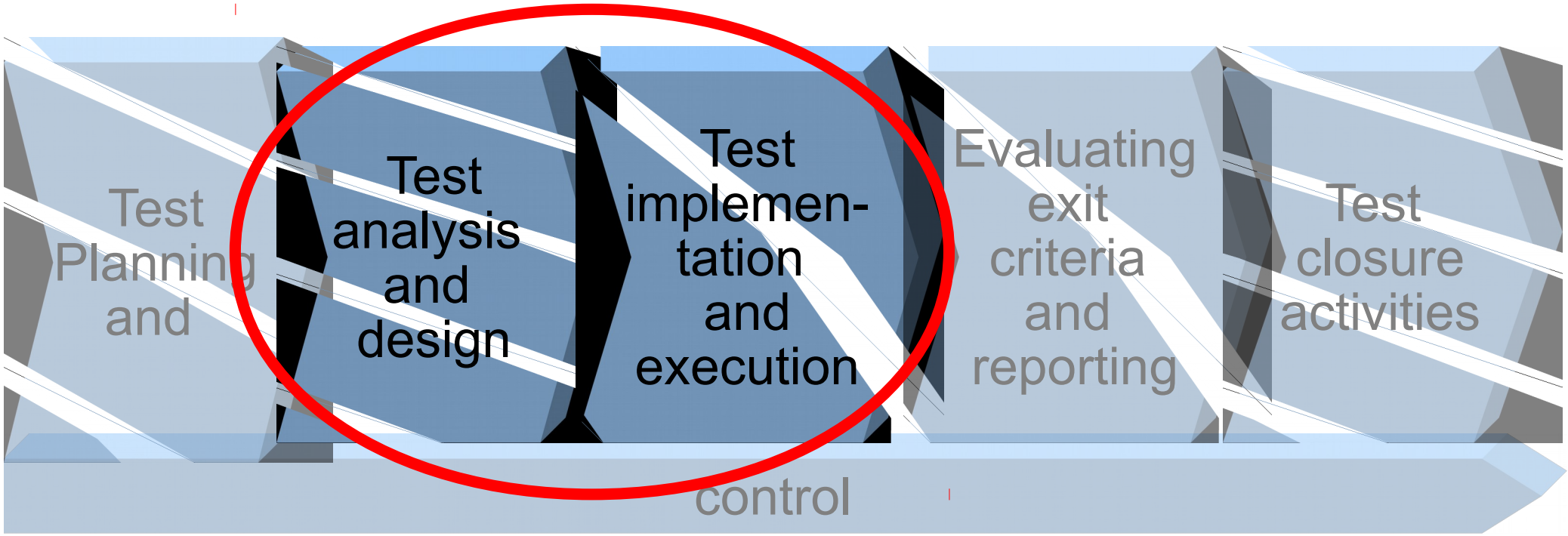
Uwe Gühl

Winter 2015 / 2016

Contents

- Test Design Techniques – Dynamic Testing I
  - Test Development Process
  - Categories of Test Design Techniques
  - 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

Test Development Process



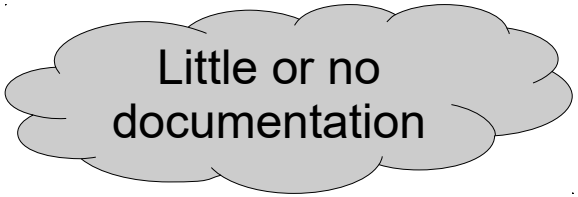
Test Development Process is the core of the Fundamental Test Process

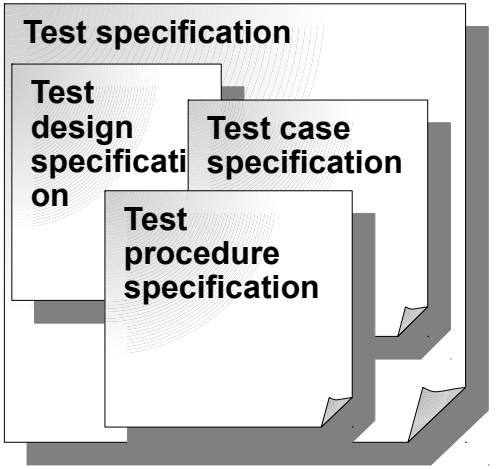
Test Development Process

Test Development Process covers

- Test analysis
- Test design
- Test implementation
- Test execution

Test Development Process





Informal ... to very formal

… depending on the context of the testing

- maturity of testing and development processes
- time constraints
- safety or regulatory requirements
- people involved

Test Development Process
Test analysis

- Analysis of test basis documentation
  - ⇨ What to test?
  - ⇨ What are the test conditions*?
- Requested: Bidirectional traceability between

Specifications and requirements ⇔ Test conditions

  - for impact analysis when requirements change
  - to determine requirements coverage for a set of tests

* Test condition = An item or event of a component or system that could be verified by one or more test cases, e. g. a function, transaction, feature, quality attribute, or structural element [ISTQB-GWP15].

Test Development Process
Test design

Creation / Specification of test cases and test data

- **Test Case:** A test case consists of a set of
  - input values
  - execution preconditions
  - expected results
  - execution postconditions

  to cover a certain test objective(s) or test condition(s).

Expected results should

- include outputs, changes to data and states, any other consequences of the test
- ideally be defined before tests get executed

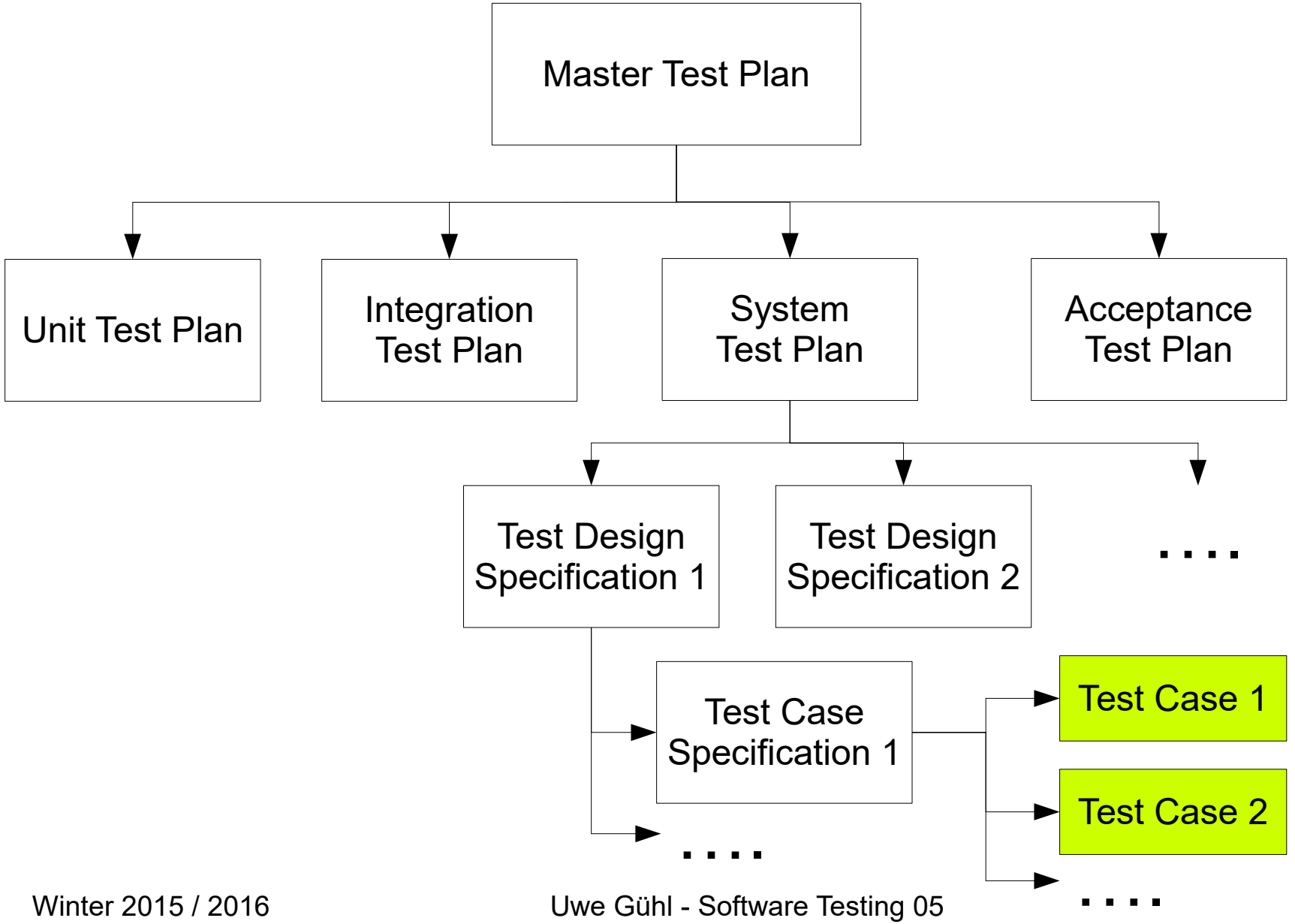
If expected results are **not defined,** then a plausible, but erroneous, result may be interpreted as the correct one

Test Development Process
Test design

Out of ‘Standard for Software Test Documentation’
[IEEE STD 829-1998]

- Test design specification [ISTQB-GWP15]
  “Document that specifies the test conditions (coverage items) for a test item, the detailed test approach and the associated high level test cases.”
  A test plan could content several test design specifications.
- Test case specification [ISTQB-GWP15]
  “Document that specifies a set of test cases (objective, inputs, test actions, expected results and execution pre conditions) for a test item.“
  A test design specification could content several test case specifications.

Test Development Process
Test design



Test Development Process
Test implementation

- During test implementation test cases are
  - developed
  - implemented
  - prioritized
  - organized in the test procedure specification
- Test procedure specification [ISTQB-GWP15]
  "Document that specifies a sequence of actions for the execution of a test (test script or manual test scripts)."

Test Development Process
Test execution

- Test execution schedule
  - contents and defines the execution order of
    - test procedures
      … specifies the sequence of actions for a test execution
    - automated test procedures (automated test scripts)
      … if a test automation tool is used, contents sequence of actions
  - takes into account factors like
    - regression tests
    - prioritization
    - technical dependencies
    - logical dependencies

Test Design Techniques

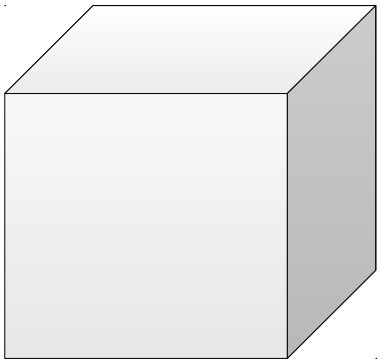
- Purpose of a test design technique is to identify
  - test conditions
  - test cases
  - test data
- Combination of test design techniques to improve testing
  - Black-box and white-box testing may also be combined with experience-based techniques to effectively use the experience of stakeholder

Test Design Techniques

- Overview:
  - White-box
  - Black-box
  - Grey-box
  - Specification-based
  - Structure-based
  - Experience-based
- Combination of test design techniques to do best testing, e.g. Black-box and white-box testing with experience-based techniques to effectively use the experience of stakeholder

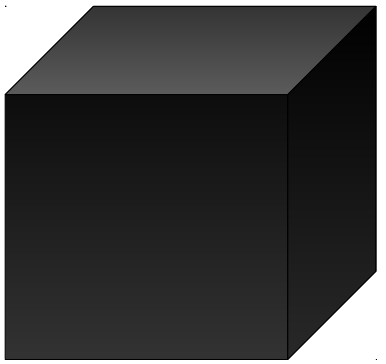
Test Design Techniques

- White-box test design techniques
  Synonyms: Structural or structure-based techniques, glass box, open box
  - based on an analysis of the structure of the component or system
  - **uses** any information regarding the internal structure of the component or system to be tested



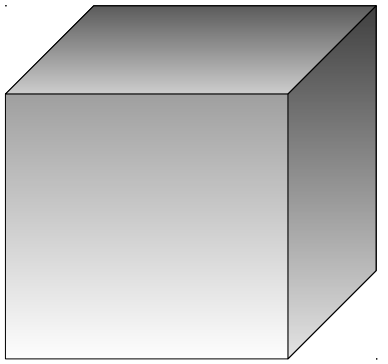
Test Design Techniques

- Black-box test design techniques
  Synonym: Specification-based techniques
  - based on an analysis of the test basis documentation
  - include both functional and non-functional testing
  - **does not use** any information regarding the internal structure of the component or system to be tested



Test Design Techniques

- Gray-box test design techniques [Wik16a]
  - based partly on internals of a software, involves knowledge of internal data structures and algorithms
  - execute defined tests at the user, or black-box level
  - **uses some** information about the inside, to better test from the outside
  - is important with web applications



Test Design Techniques

Specification-based test design techniques

- Models, either formal or informal, are used for
  - the specification of the problem to be solved
  - the software
  - the software components
- Test cases can be derived systematically from these models

Test Design Techniques

Structure-based test design techniques

- Information about how the software is constructed is used to derive the test cases (e.g., code and detailed design information)
- The extent of coverage of the software can be measured for existing test cases, and further test cases can be derived systematically to increase coverage

Test Design Techniques

Experience-based test design techniques

- based on the knowledge and experience about
  - the software
  - its usage
  - its environment
  - likely defects and their distribution

White-box Techniques

- White-box testing is based on an identified structure of the software or the system:
  - **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 could be based on

- data flow
- required functionality

White-box Techniques Structural Coverages

Structural Coverage Metrics discussed:

- Statement testing
- Decision testing*

More Structural Coverage Metrics (see e.g. [ISTQB-CTALSTTA12] for details) are

- Condition testing
- Multiple condition testing
- Condition determination testing
- Linear Code Sequence and Jump (LCSAJ) or loop testing
- Application Programming Interface (API) testing

* Similarities to branch testing

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

- Statement coverage is done in component testing

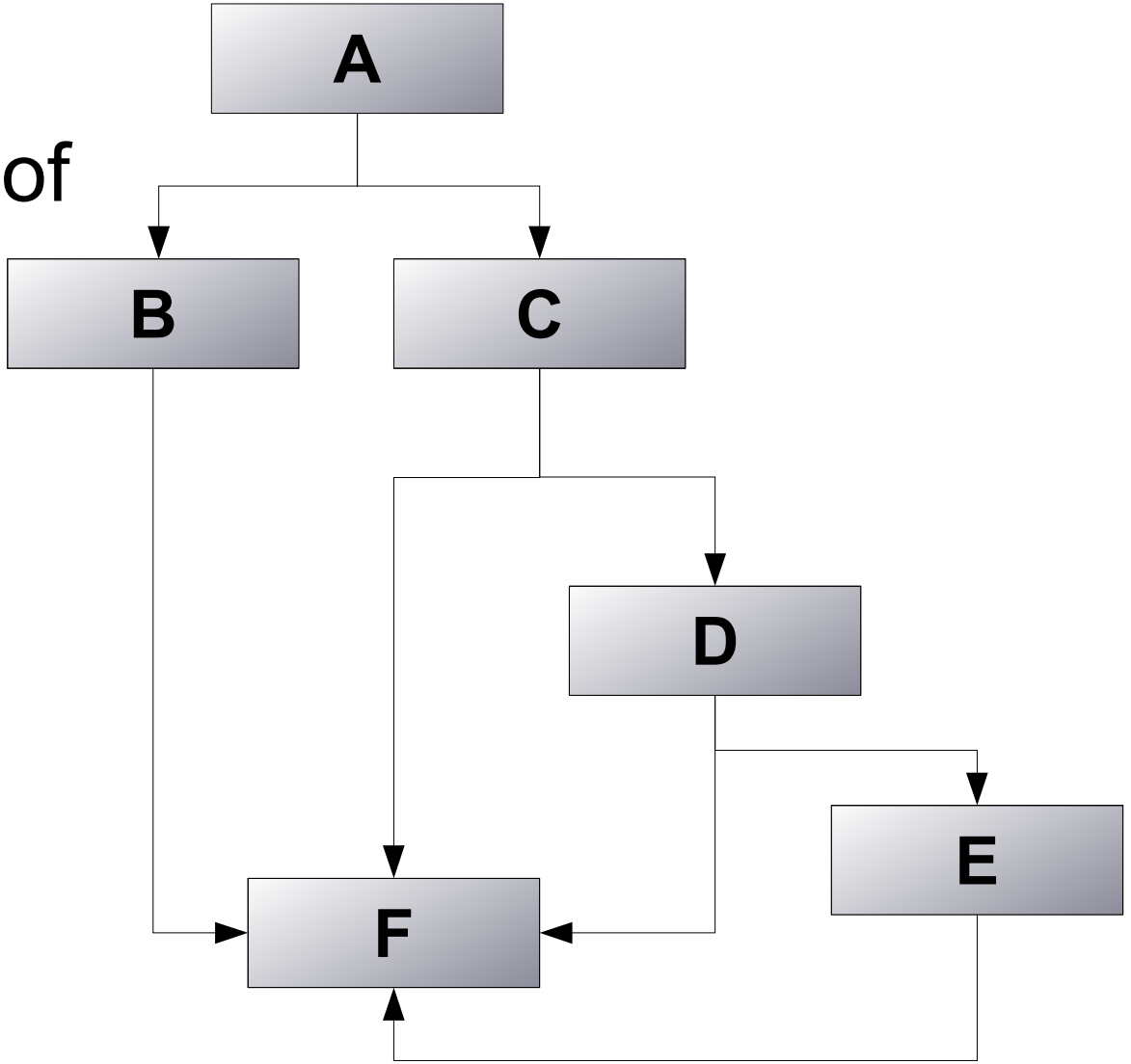
White-box Techniques Statement Testing and Coverage

Example 1

- Required number of test cases for 100 % statement coverage:

  2

  - A, B, F
  - A, C, D, E, F



White-box Techniques Statement Testing and Coverage

Example 2

- Required number of test cases for 100 % statement coverage:

  1

  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 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

Excerpt:

- Branch coverage is determined by
  $$\frac{testedBranches}{allBranches}$$
  - testedBranches = number of all branch outcomes covered by (designed or executed) test cases
  - AllBranches = number of all possible branch outcomes in the code under test
- Branch testing is a form of control flow testing as it follows a specific flow of control through all branches

White-box Techniques
Decision Testing and Coverage

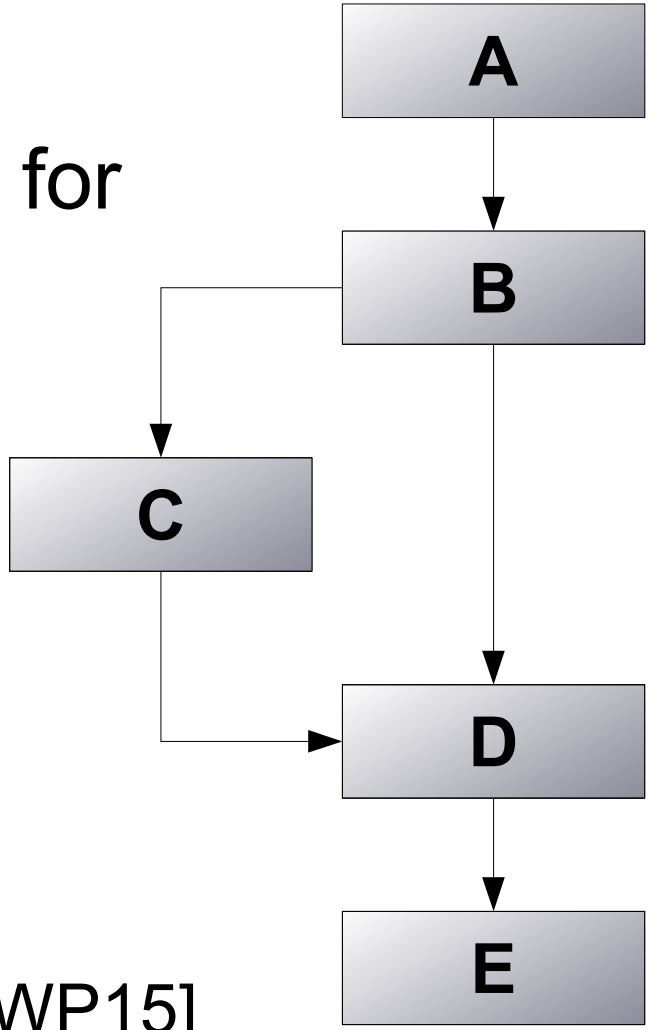
Example 3

- Required number of test cases for 100 % decision coverage:

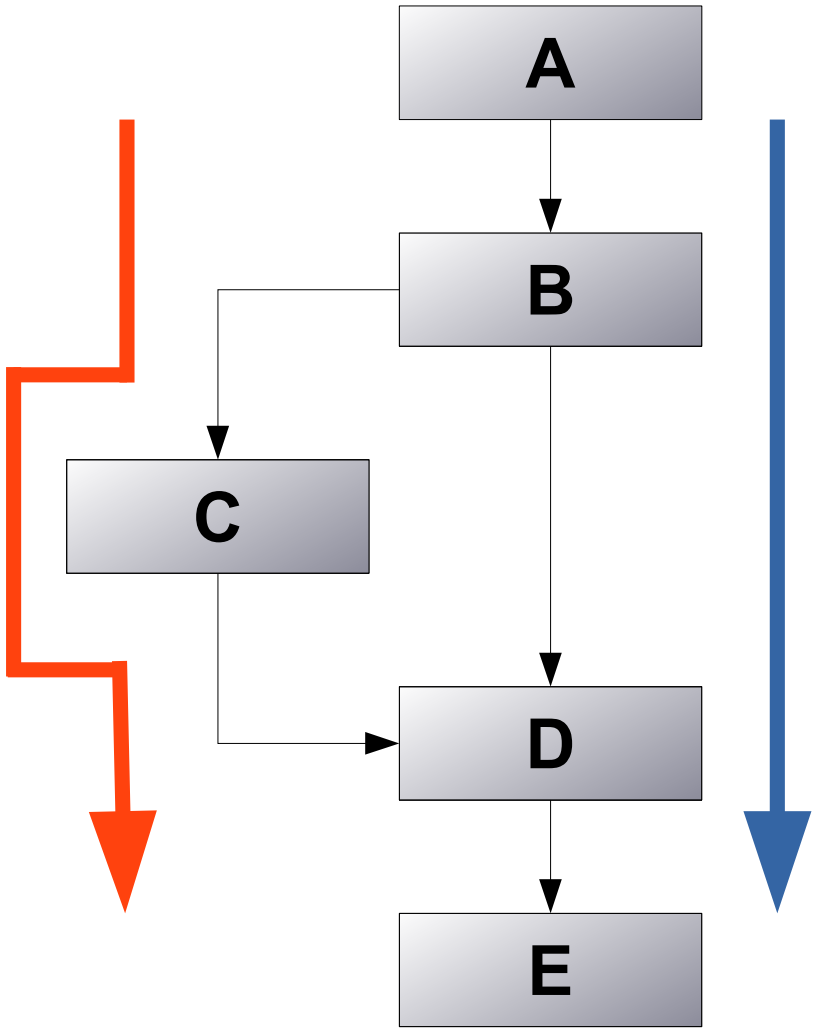
  2

  - A, B, D, E
  - A, C, D, E, F

- Hint:
  100 % decision coverage implies both
  100 % branch coverage and
  100 % statement coverage [ISTQB-GWP15]



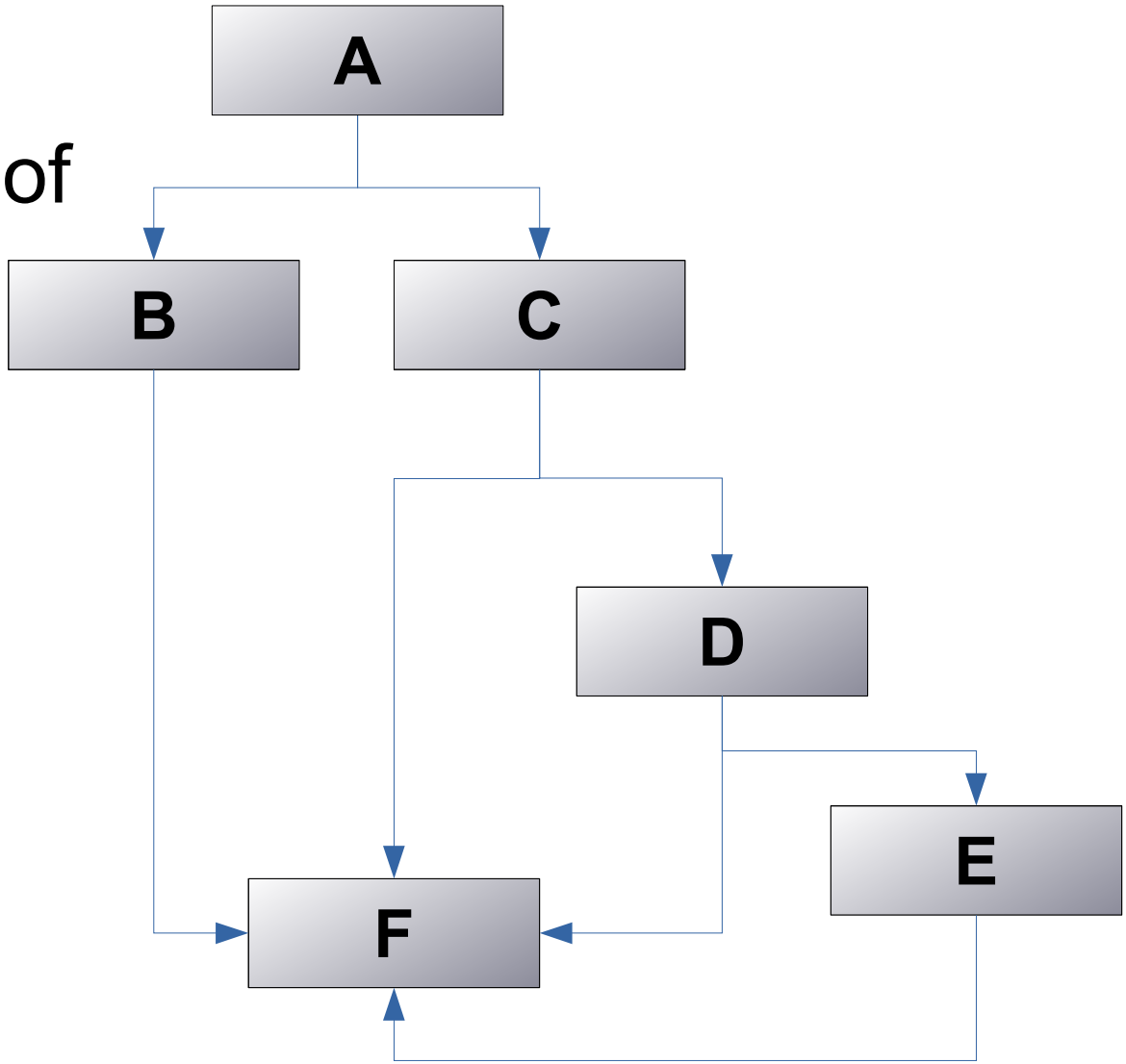
White-box Techniques
Decision Testing and Coverage



Example 3

- Decision Coverage
  - 1 Test case A, B, D, E
    50 % Decision test coverage
  - 1 Test case A, B, C, D, E
    50 % Decision test coverage
- Branch Coverage
  - 1 Test case A, B, D, E
    60 % Branch test coverage
  - 1 Test case A, B, C, D, E
    80 % Branch test coverage

White-box Techniques
Decision Testing and Coverage



Example 4

- Required number of test cases for 100 % decision coverage:

  4

  - A, B, F
  - A, C, F
  - A, C, D, F
  - A, C, D, E, F

White-box Techniques
Decision Testing and Coverage

Example 5

- Required number of test cases for 100 % decision coverage:

  2

  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 coverage
  also
  50% Statement coverage?
  ==> No!

- TC1: x=3
  50 % Decision coverage
  75 % Statement coverage
- TC2: x=2
  50 % Decision coverage
  50 % Statement coverage

Code example

```
int foo(int x) {
   int a = 0;
   if (x>2) {
      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. 44 Mye12]
- 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
  - Multiple condition coverage
- The concept of coverage can also be applied at other test levels, e.g., at the integration level
  - Percentage of modules exercised → Module coverage
  - Percentage of components exercised → Component coverage
  - Percentage of classes exercised → 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)

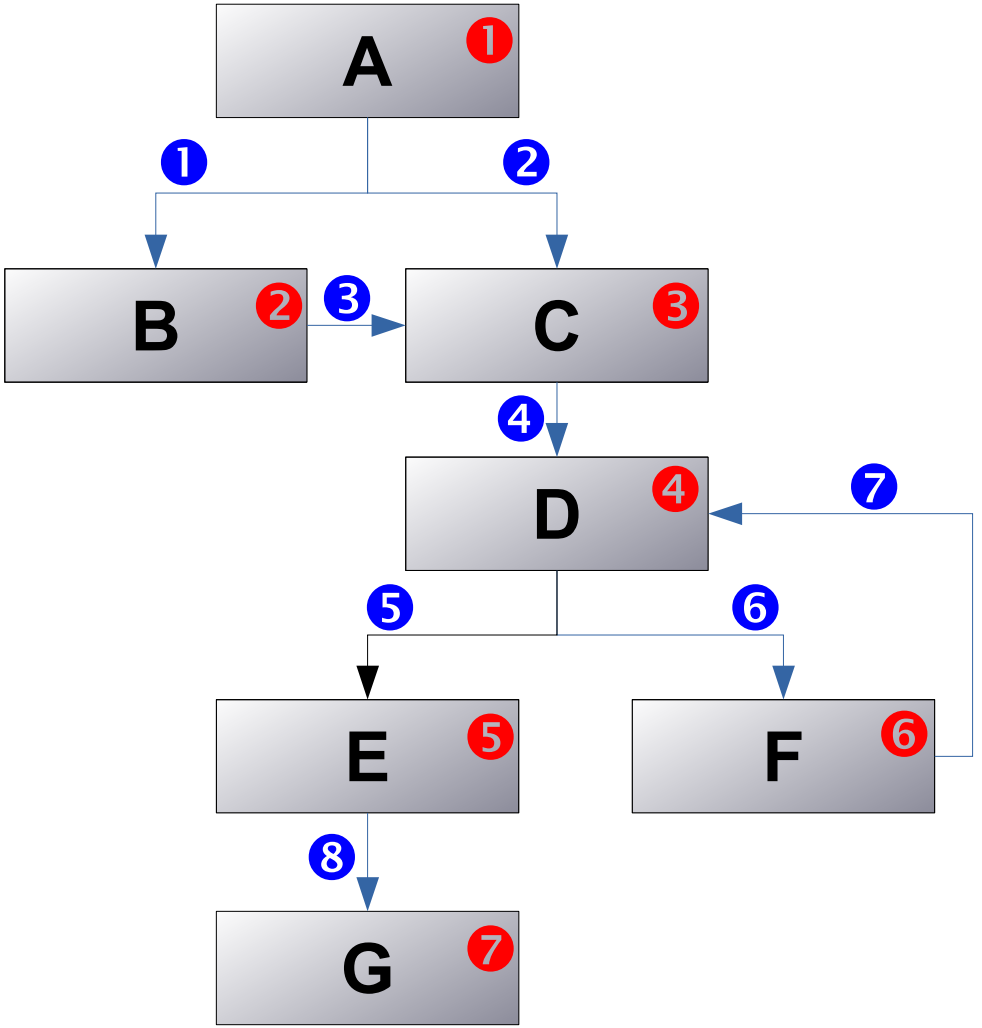
White-box Techniques
Cyclomatic Complexity

Example:

M = L – N + 2P

= 8 – 7 + 2

= 3



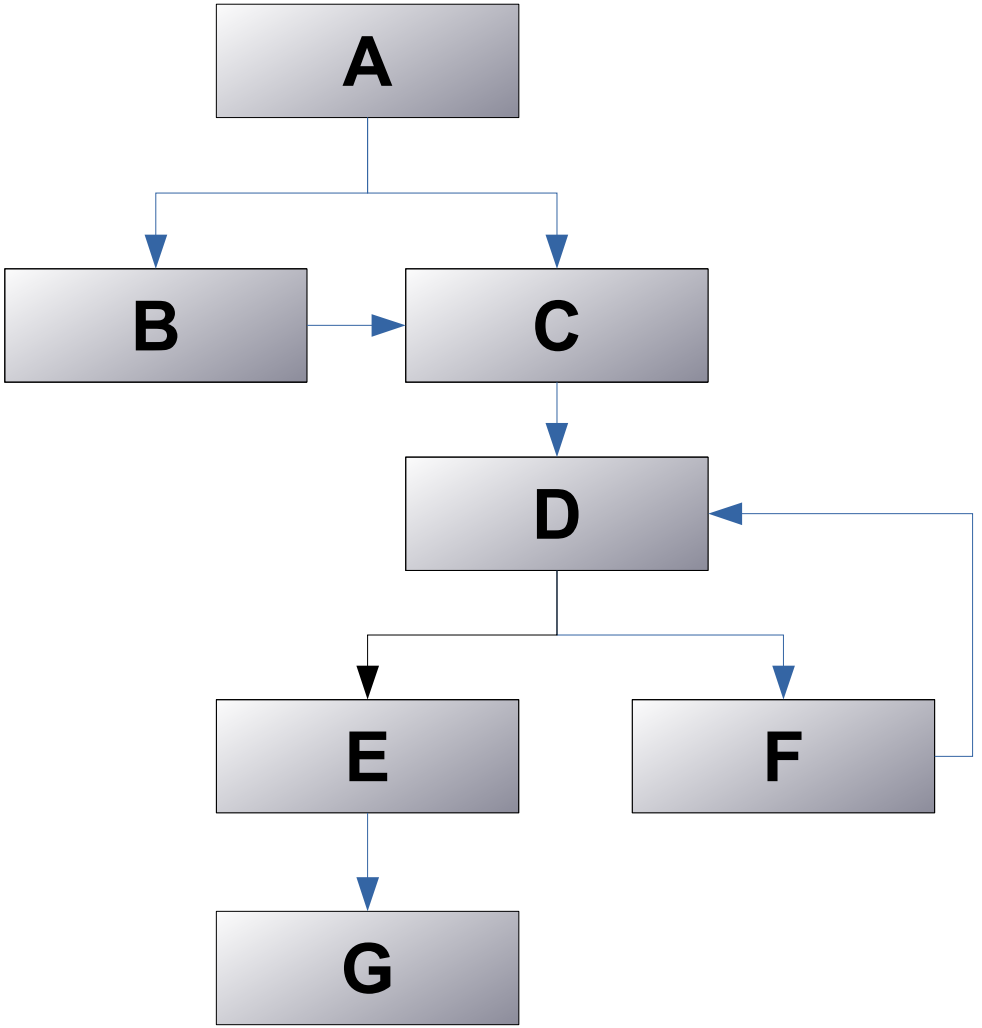
White-box Techniques
Cyclomatic Complexity

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

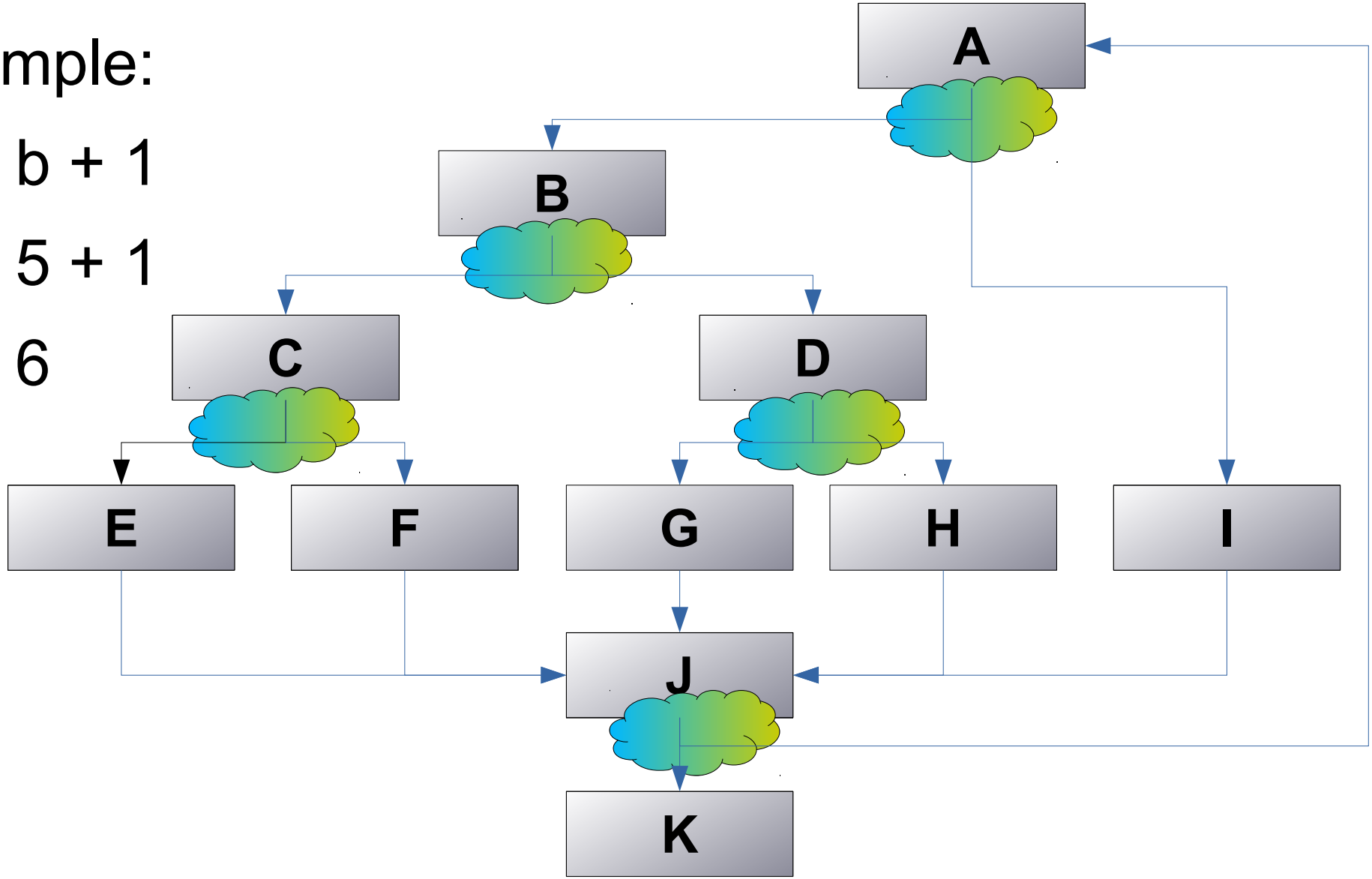
White-box Techniques
Cyclomatic Complexity

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.

Sources (1/2)

- [ISTQB-CTFLS11] International Software Testing Qualifications Board: Certified Tester Foundation Level Syllabus, Released Version 2011, http://www.istqb.org/downloads/syllabi/foundation-level-syllabus.html
- [ISTQB-GWP15] Glossary Working Party of International Software Testing Qualifications Board: Standard glossary of terms used in Software Testing, Version 3.01, 2015, http://www.istqb.org/downloads/glossary.html
- [ISTQB-CTALSTTA12] International Software Testing Qualifications Board: Certified Tester Advanced Level Syllabus, Technical Test Analyst, Version 2012, http://www.istqb.org/downloads/syllabi/advanced-level-syllabus.html
- [IEEE Sttd 829-1998] IEEE Std 8229™ (1998) IEEE Standaard for Softwware Test Documentation ,
- [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/

Sources (2/2)

- [McC76] T. McCabe, A complexity measure, in: IEEE Transactions on Software Engineering, Vol. 2, pp. 308-320, 1976.
- [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
- [Mye12] Glenford J. Myers: The Art of Software Testing, Third Edition, John Wiley & Sons, Inc., 2012
- [Sha08] Rich Sharpe: McCabe Cyclomatic Complexity: the proof in the pudding, 2008, http://www.qualitydev.net/?p=27
- [Wal79] Walsh, T., “A Software Reliability Study Using a Complexity Measure,” AFIPS Conference Proceedings, AFIPS Press, 1979.
- [Wik16] Wikipedia, Gray box testing, 2016, https://en.wikipedia.org/wiki/Gray_box_testing