Test Design Techniques

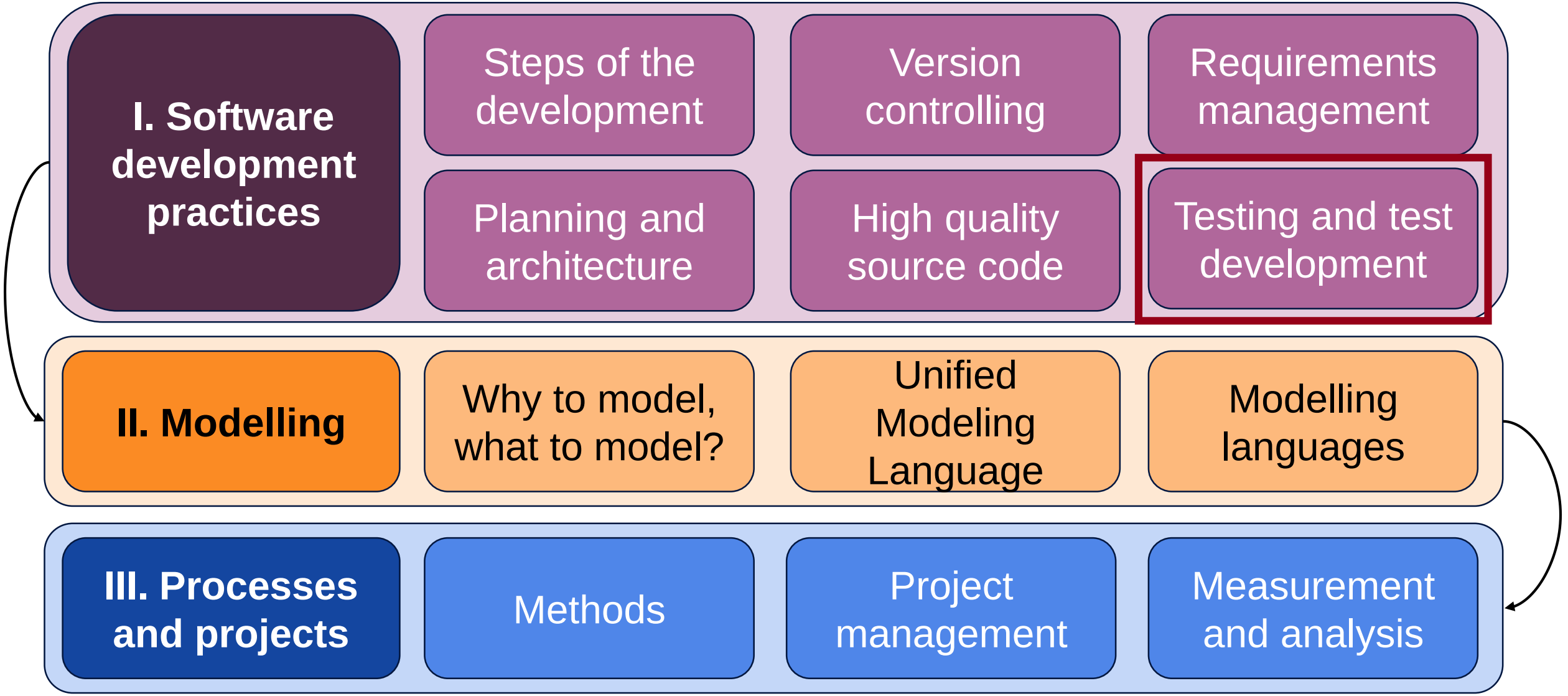
HUSZERL Gábor
huszerl@mit.bme.hu

MŰEGYETEM 1782
BME VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR
Méréstechnika és Információs Rendszerek Tanszék
ftsrg Critical Systems Research Group

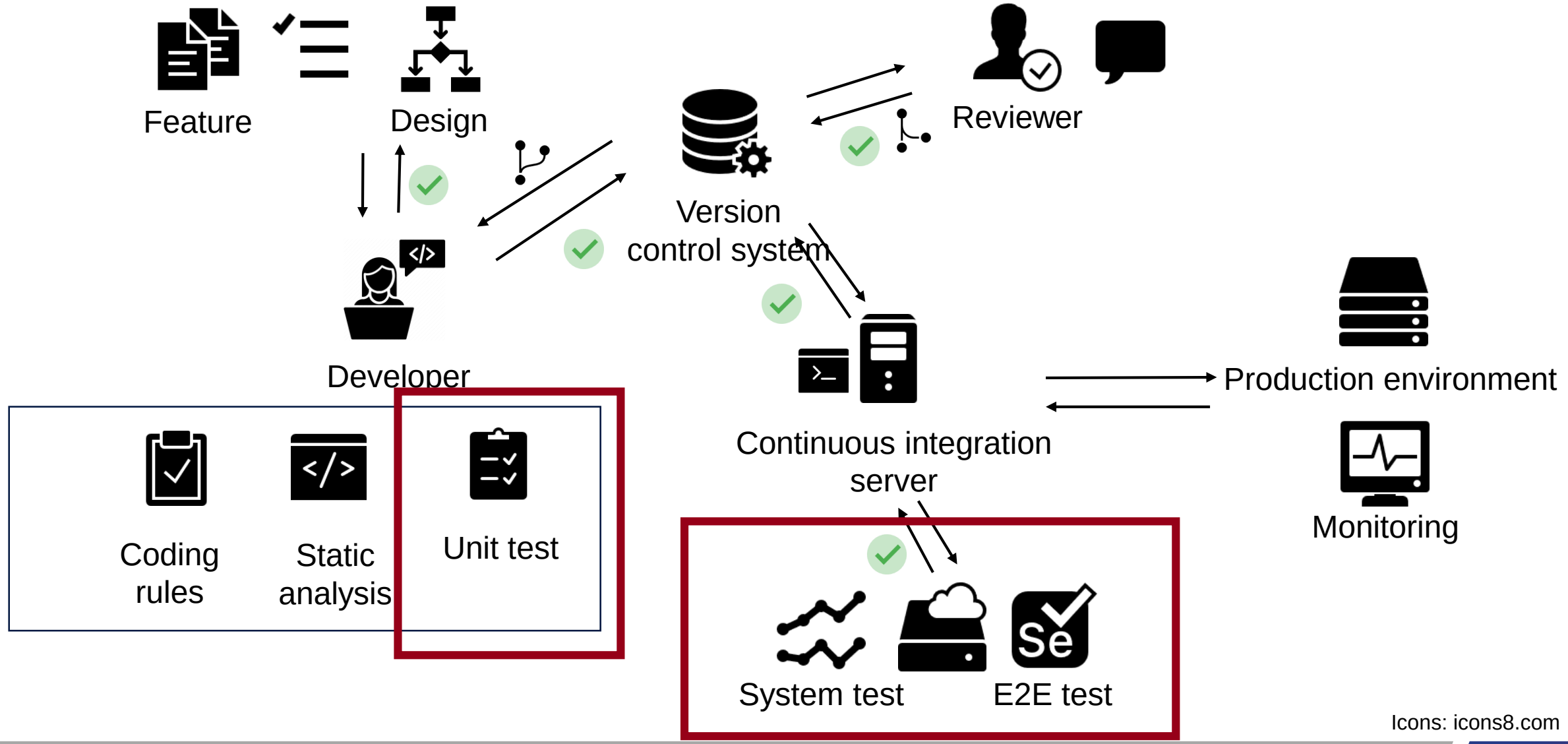
Learning Outcomes

- At the end of the lecture the students are expected to be able to
- (K2) summarize the challenges of test design,
- (K3) apply basic specification-based test design techniques,
- (K3) apply basic structure-based test design techniques.

Further Topics of the Subject



Typical Development Workflow

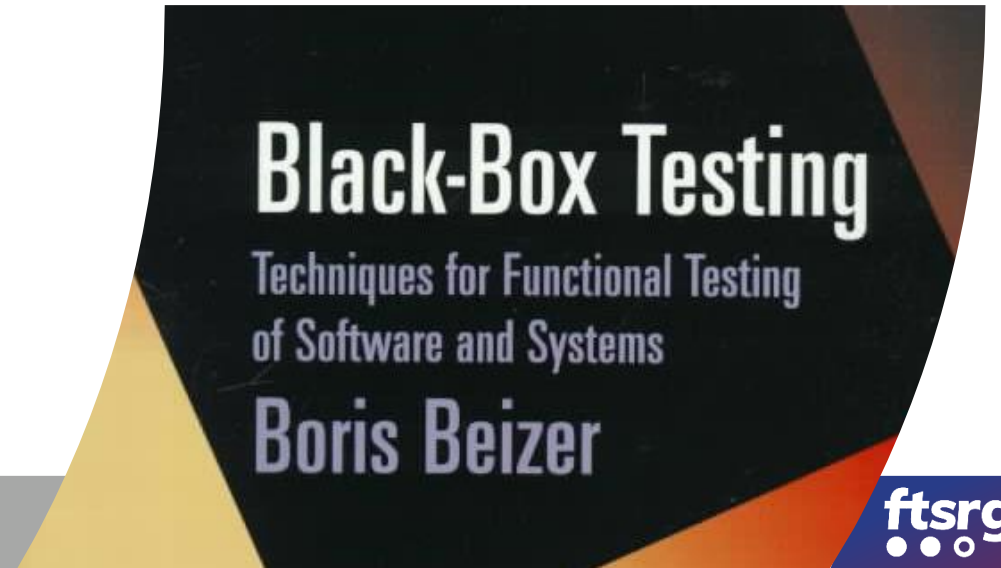


Icons: icons8.com

Basic Design Principles

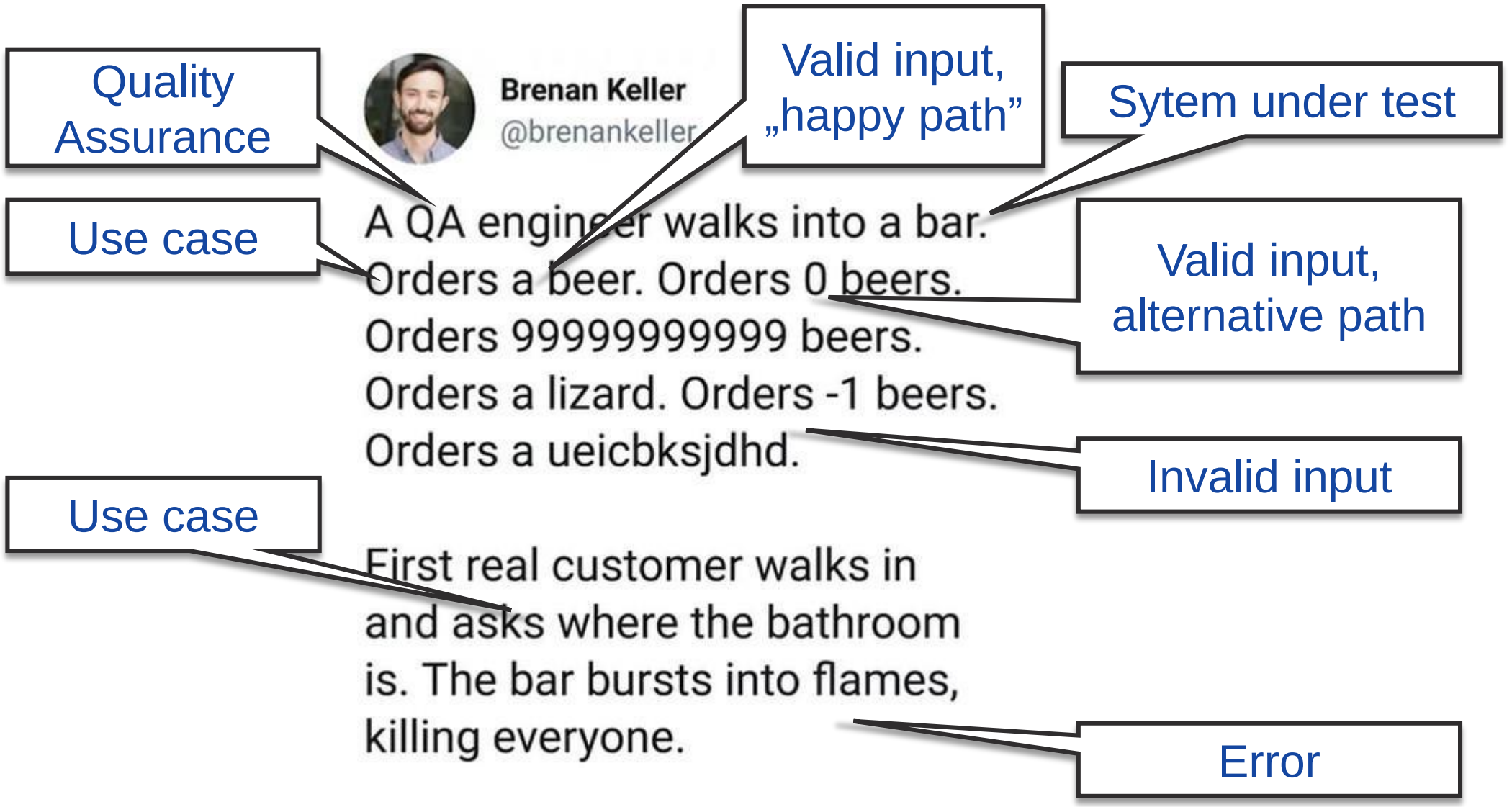
Why is Test Design Important?

„More than the act of testing, the act of designing tests is one of the best bug preventers known.”

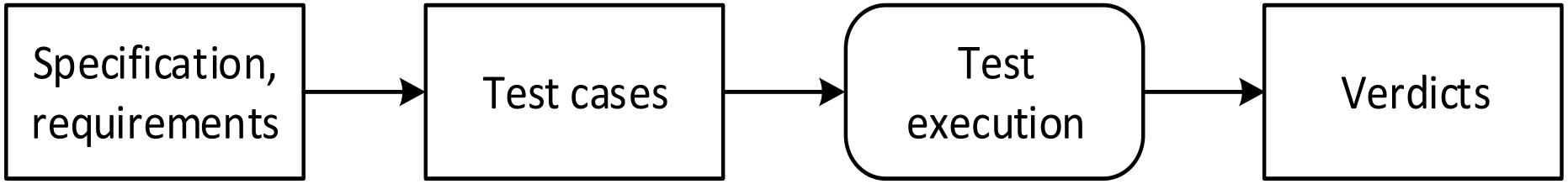


- Is the system well designed?
  Can it be tested?
- “Wearing the glasses of the tester”
  A completely different mind-set.

The Different Mind-Set



Basic Concepts of Testing (Reminder)



- SUT: system under test
- Test case
  - a set of inputs, implementation conditions and expected results developed for a specific purpose
- Test suite
- Test oracle
  - A principle or method that helps the tester decide whether the test was successful
- Verdict: pass / fail / error / inconclusive…

Tasks to Solve (Reminder)

Test selection	Oracle problem	Exit criteria
• What input values and data to use?	• Where to find a reliable oracle?	• When to stop testing?

Test Design Techniques

Goal: Select test cases based on test objectives

Specification-based

- SUT: black box
- Only specification known
- Testing specified functionality

Structure-based

- SUT: white box
- Inner structure known
- Testing based on internal structure and behaviour

Coverage Metrics

- What % of testable elements have been tested

- Testable element
  - Specification-based: requirement, functionality, …
  - Structure-based: statement, decision, …

- Coverage criterion: X% for Y coverage metric

- This is not fault coverage!

How to Use Coverage Metrics?

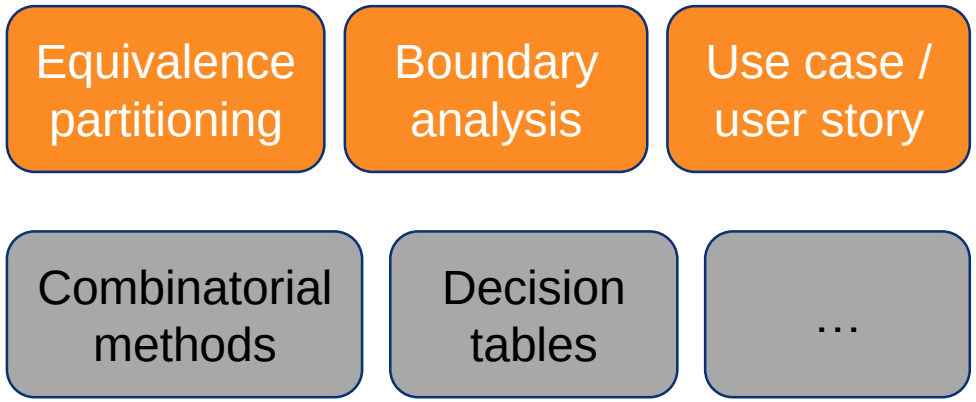
Evaluation (measure)

- Evaluate quality of existing tests
- Find missing tests

Selection (goal)

- Design tests along the metrics
- Satisfy criteria by reaching given %

Specification-Based Testing



Test Design Techniques

Goal: Select test cases based on test objectives

Specification-based

- SUT: black box
- Only specification known
- Testing specified functionality

Structure-based

- SUT: white box
- Inner structure known
- Testing based on internal structure and behaviour

Specification-Based Techniques

Equivalence partitioning

Boundary analysis

Use case / user story

Combinatorial methods

Decision tables

...

Equivalence Class Partitioning

- Input and output equivalence classes:
  - Data that are expected to cover the same faults (cover the same part of the program)
  - Goal: Each equivalence class is represented by one test input (selected data)

- Highly context-dependent
  - Needs to know the domain and the SUT!
  - Depends on the skills and experience of the tester

Selecting Equivalence Classes

- Selection uses heuristics
  - Initially: valid and invalid partitions
  - Next: refine partitions
- Typical heuristics
  - Interval (e.g. 1-1000)
    - <min, min-max, >max
  - Set (e.g. RED, GREEN, BLUE)
    - valid elements, invalid elements
  - Specific format (e.g. first character is @)
    - condition true, condition false
  - Custom (e.g. the month February)

Deriving Test Cases from Equivalence Classes

- Combining equivalence classes of several inputs

- For valid (normal) equivalence classes:
  - test data should cover as much equivalence classes as possible

- For invalid equivalence classes:
  - first covering each invalid equivalence class separately
    - to avoid their effects cancelling out each other
  - then combining them systematically

EXERCISE: Equivalence Class Partitioning

- **Requirement:** The loan application shall be denied if the requested amount is larger than 1M Ft and the customer is a student, unless the amount is less than 3M Ft and the customer has repaid a previous loan (of any kind).

- Input parameters? Equivalence classes?
- Any questions regarding the requirement?

Specification-Based Techniques

Equivalence partitioning

Boundary analysis

Use case / user story

Combinatorial methods

Decision tables

...

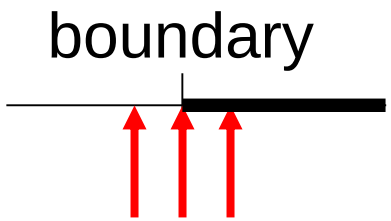
Boundary analysis

- Examining the boundaries of data partitions
  - Focusing on the boundaries of equivalence classes
  - Both input and output partitions

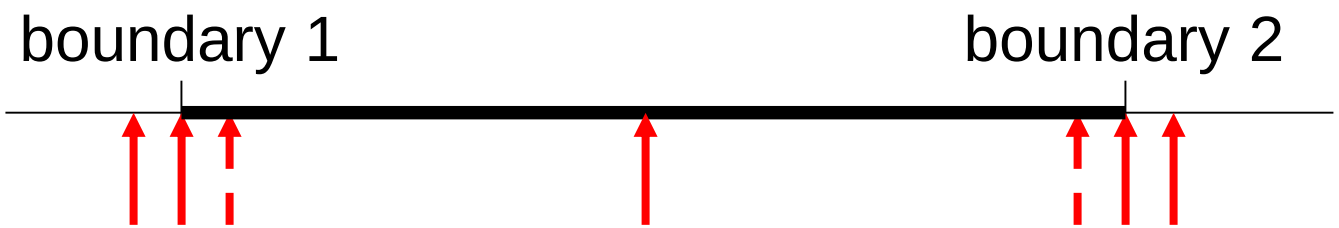
- Typical faults to be detected:
  - Faulty relational operators,
  - Conditions in cycles,
  - Size of data structures,
  - …

Typical Test Data for Boundaries

- One boundary requires 3 tests:



- An interval requires 5-7 tests:



EXERCISE: Boundary Analysis

Requirement: The customer has to order at least 3 books to get a discount price. A customer ordering 10 books is given additional discount.

- What values to use for testing?
- Any other questions regarding the requirement?

Specification-Based Techniques

Equivalence partitioning

Boundary analysis

Use case / user story

Combinatorial methods

Decision tables

...

Deriving Test Cases from Use Cases

- Typical test cases:
  - 1 test for main path („happy path”, „mainstream”)
    - Oracle: checking post-conditions
  - Separate tests for each alternate path
  - Tests for violating pre-conditions

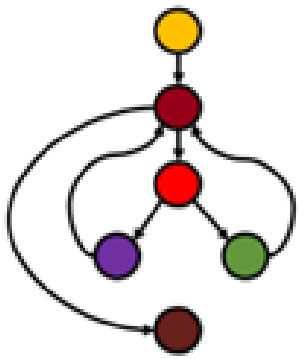
- Mainly higher testing levels (system test, acceptance test, …)

Structure-Based Testing

Forráskód:

```
int a = read();
while(a < 16) {
  if(a < 10) {
      a += 2;
  } else {
      a++;
  }
}
a = a * 2;
```

Control-flow graph (CFG):



Test Design Techniques

Goal: Select test cases based on test objectives

Specification-based

- SUT: black box
- Only specification known
- Testing specified functionality

Structure-based

- SUT: white box
- Inner structure known
- Testing based on internal structure and behaviour

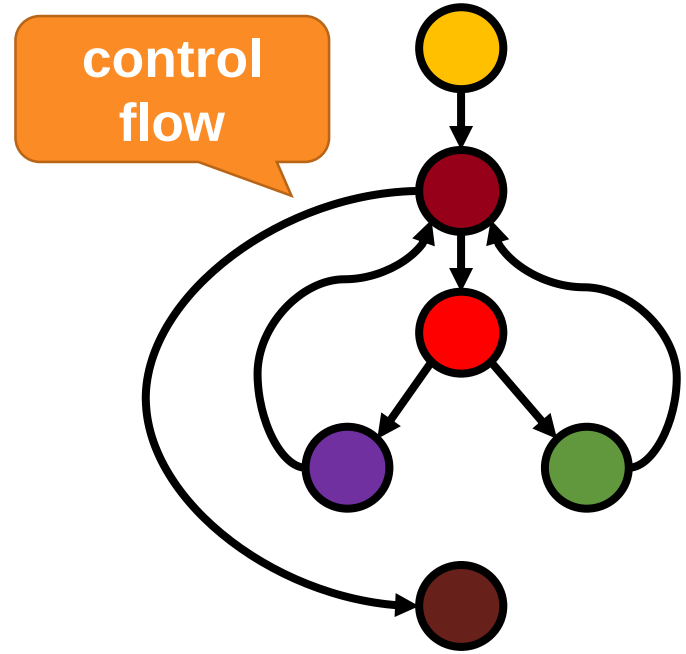
What Is “Internal Structure”?

- In case of code: structure of the code (CFG)

Source code:

```
int a = read();
while(a < 16) {
  if(a < 10) {
      a += 2;
  } else {
      a++;
  }
}
a = a * 2;
```

Control-flow graph (CFG):



Note: We will not go in details for constructing CFG

Basic Concepts

```
int t = 1;
Speed s = SLOW;

if (! started){
   start();
}

if (t > 10 && s == FAST){
   brake();
} else {
   accelerate();
}
```

Statement

Block

Condition

Decision

Decision branch

Basic Concepts

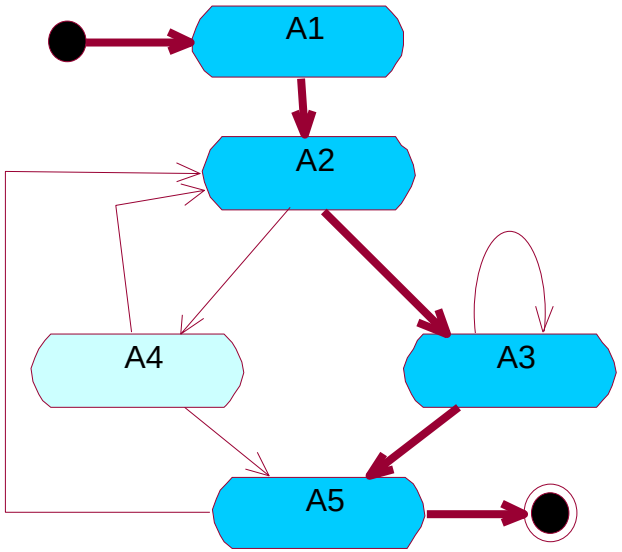
- Statement
- Block
  - A sequence of one or more consecutive executable statements containing no branches or function calls
- Condition
  - Logical expression without logical operators (Boolean operators like *and*, *or*, …)
- Decision
  - A logical expression consisting of one or more conditions combined by logical operators
- Decision branch
  - One possible outcome of a decision
- Path
  - A sequence of events, e.g., executable statements, of a component typically from an entry point to an exit point.

1. Statement Coverage

$$\frac{\text{Number of statements executed during testing}}{\text{Number of all statements}}$$

Do NOT use “line coverage”

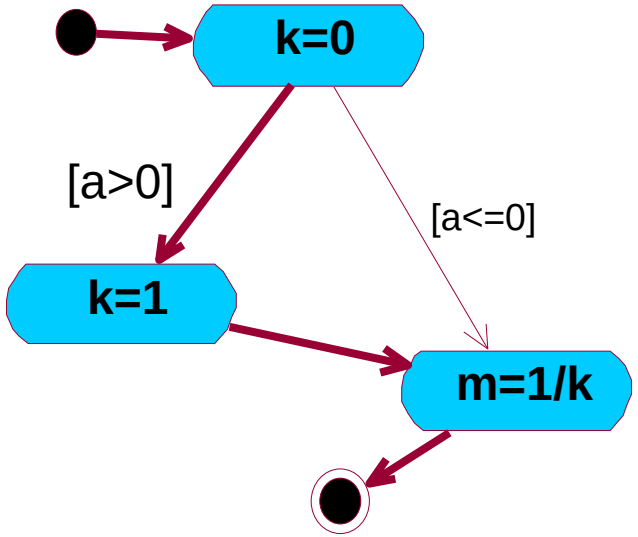
- Whole program in one single line?
- Empty lines?
- Comments?
- Multi-line strings?
- Conditional operator in C?



Statement coverage: 4/5 = 80%

Assessing Statement Coverage

Each statement is executed at least once



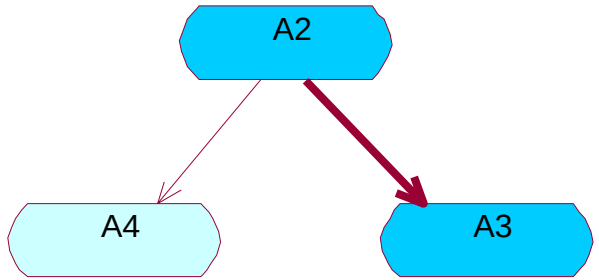
Statement coverage: 100%

BUT: [a<=0] branch is left out

Does not guarantee coverage of empty branches

2. Decision Coverage

$$\frac{\text{Number of decision outcomes taken during testing}}{\text{Number of all possible outcomes}}$$



Decision coverage: 50%

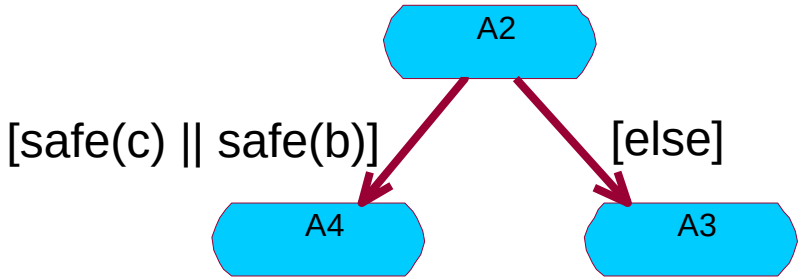
How many outcomes can a decision have?

Considers decision in isolation, no combinations!

Assessing Decision Coverage

Each statement is executed at least once

All decision outcomes are covered (also the empty ones)



#	safe(c)	safe(b)	safe(c) || safe(b)
1	T	F	T
2	F	F	F

100% decision coverage

BUT: *safe(b)* = *true* case is left out!

Does not take all combinations of conditions into account!

Additional Coverage Criteria (see MSc)

- Condition Coverage
- Condition/Decision Coverage (C/DC)
- Modified Condition/Decision Coverage (MC/DC)
- Multiple Condition Coverage (MCC)
- Loop Coverage
- …
- All-Defs Coverage
- All-Uses Coverage
- …

EXERCISE: Structure-Based Testing

```
1 int pow(int n, int k) {
2   if (n < 0 || k < 0) {
3     return -1;
4   }
5   int p = 1;
6   for (int i = 0; i < k; i++) {
7     p *= n;
8   }
9   return p;
  }
```

Construct the CFG of the code
Design test cases for:

- 100% statement coverage
- 100% decision coverage

Calculating Coverage in Practice

- Every tool uses different definitions
- Implementation
  - Instrument source/binary code
  - Adding instructions to count coverage

```
if (a > 10){
  CoveredBranch(1, true);
  b = 3;
} else {
  CoveredBranch(1, false);
  b = 5;
}
send(b);
```

See also: Is bytecode instrumentation as good as source code instrumentation, 2013.

Using Test Coverage Criteria

Can be used for

- Find untested parts of the program
- Measure “completeness” of test suite
- Can be basis for exit criteria

Cannot be used for

- Finding/testing missing or not implemented requirements
- Only indirectly connected to code quality

Using Test Coverage Criteria

- Experience from Microsoft
  - „Test suite with high code coverage and high assertion density is a good indicator for code quality.”
  - „Code coverage alone is generally not enough to ensure a good quality of unit tests and should be used with care.”
  - „The lack of code coverage to the contrary clearly indicates a risk, as many behaviors are untested.”

    (Source: „Parameterized Unit Testing with Microsoft Pex”)
- Related case studies
  - „*Coverage Is Not Strongly Correlated with Test Suite Effectiveness*”, 2014. DOI: 10.1145/2568225.2568271
  - „*The Risks of Coverage-Directed Test Case Generation*”, 2015. DOI: 10.1109/TSE.2015.2421011

Summary

Summary on Test Design Techniques

- Specification and structure based techniques
  - Many orthogonal techniques
  - Every techniques need practice!

- Combination of techniques is useful
  - Example (How We Test Software at Microsoft):

    specification based: 83% code coverage

    + exploratory: 86% code coverage

    + structural: 91% code coverage

Summary

Test Design Techniques

Goal: Select test cases based on test objectives

Specification-based

- SUT: black box
- Only specification known
- Testing specified functionality

Structure-based

- SUT: white box
- Inner structure known
- Testing based on internal structure and behaviour

Specification-Based Techniques

- Equivalence partitioning
- **Boundary analysis**
- Use case / user story
- Combinatorial methods
- Decision tables
- …

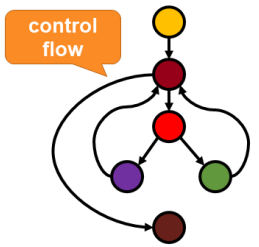
What Is “Internal Structure”?

- In case of code: structure of the code (CFG)

Source code:

```
int a = read();
while(a < 16) {
  if(a < 10) {
     a += 2;
  } else {
     a++;
  }
}
a = a * 2;
```

Control-flow graph (CFG):



Note: We will not go in details for constructing CFG