Software Testing

Attribution: some slides from Ian Cottingham, RAIK 383H

Interview with Sean McDonald

Oregon State University

When does it not work?

- Fundamental question:
  - if it does work, how do you know it (always) works?
  - if it doesn't work, why doesn't it work and how do you fix it?

- Testing provides a mechanism for consistently demonstrating that software does what it is supposed to do:
  - the *mechanism* itself can be verified to demonstrate that it "covers" most scenarios.

Why is it important?

- Software faults cost $59.5 billion a year (NIST 2003).
- $22.2b could be saved through application of simple testing by reducing:
  - number of faults,
  - development costs,
  - time to market,
  - maintenance costs.
- Two primary goals:
  - demonstrate that software meets requirements;
  - discover bugs in software.

Oregon State University

Terminology

- Failure:
- Fault:
- Error:
- Bug:
- Debugging:
- Testing:
- Oracle:

Oregon State University

Terminology

- Failure: incorrect program behavior that is externally visible.
- Fault: incorrect portions of code that can lead to a failure.
- Error: a mistake made by a developer that introduces a fault.
- Bug: informal way to describe a failure or fault.
- Debugging: process of finding a fault which causes a failure.
- Testing: the process of finding failures.
- Oracle: a device or procedure for determining correctness of output.

Verification vs. Validation

- Two primary activities:
  - Validation: “are we building the right product?”
  - Verification: “are we building the product right?”

Not a matter of “either or” but of “both and”

Oregon State University

Verification vs. Validation

- Two primary activities:
  - Validation: process of checking whether the specification captures the customers' needs
  - Verification: process of checking that the software meets the specification

Not a matter of "either or" but of "both and"

Oregon State University

Confidence

- Goal of testing: *provide confidence* the system is 'fit for a purpose
  - Subjective: if it compiles it ships??
- Factors that influence confidence:
  - Software purpose - how critical the software is to the users' organization
    - Domain (nuclear vs. candy crush)
    - Number of users
  - User expectations - if expectations are low, testing will reflect that
  - Tested-ness of third party libraries
  - Marketing environment - how quickly a product needs to get to market, faults may become more tolerable.
- Confidence is about understanding the context

Testing

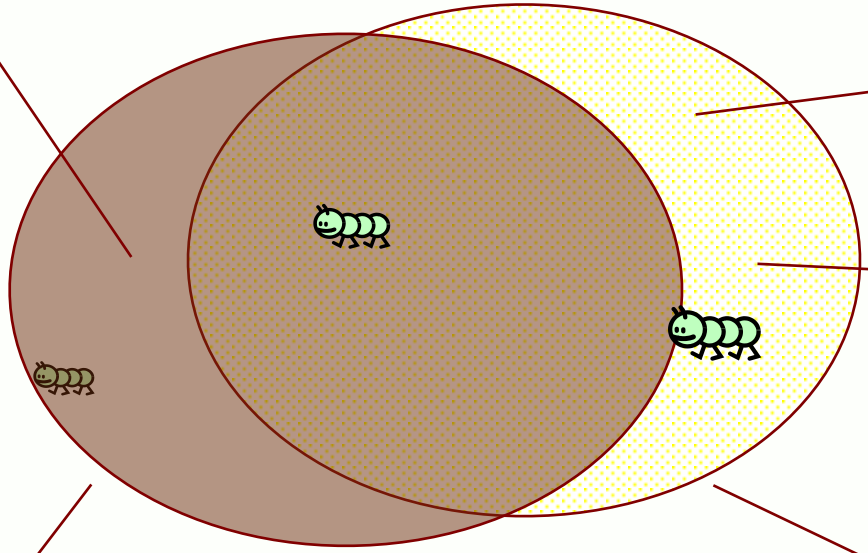
- Expose bugs that occur due to:
  - invalid or faulty inputs,
  - unexpected interactions between systems or components,
  - threading or timing issues,
  - runtime performance
  - …

Three Basic Types of testing

- **Development Testing**
  - testing is done while software is being developed, both dedicated testers and developers are involved in this process.
- Release Testing
  - testing that is done prior to releasing the system or some milestone outside of the development team.
- User Testing
  - testing that is done by potential end-users of the system to determine if the system meets their requirements.

Oregon State University

Black vs. White Box testing



Oregon State University

Black Box testing vs. White Box testing

- Tests are derived from reqts.
- Driven by providing inputs and observing expected outputs,
- Require no knowledge of the internal structure of the code,
- Can be functional or nonfunctional,
- Generally done at a higher level.

- Tests are derived from code,
- Evaluated in terms of code coverage,
- Inputs are selected to test paths of execution,
- Techniques: control flow, data flow, branch, and path testing.
- Generally done at a lower level.

Development Testing: by developers

- White box testing processes identifies and help correct faults early on:
  - expose “little” faults that can be quickly (and inexpensively) corrected before they become “big”.
- Can sometimes involve a dedicated testing team that is responsible for developing and executing test suites.
- Can be carried out in three ways: unit, integration, and system testing

Oregon State University

Unit Testing

- Verification of individual “units” of software
  - supply inputs and assert that expected outputs are generated.
- Unit tests should be created for each class in a system, and test:
  - execution of all methods defined by the class,
  - values on all attributes defined by the class,
  - all states of an instance of the class.

Choosing Inputs

- Testing effectiveness is based on how good your tests are

Inputs

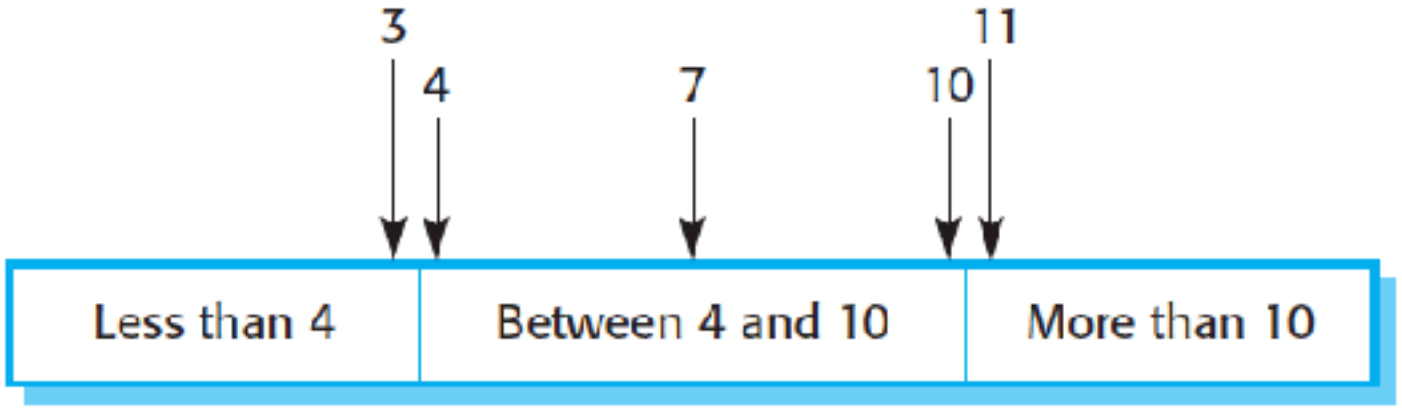
```
if (x<4) then bla
   elseIf x>10 then blabla
else someother_bla
```

Less than 4	Between 4 and 10	More than 10

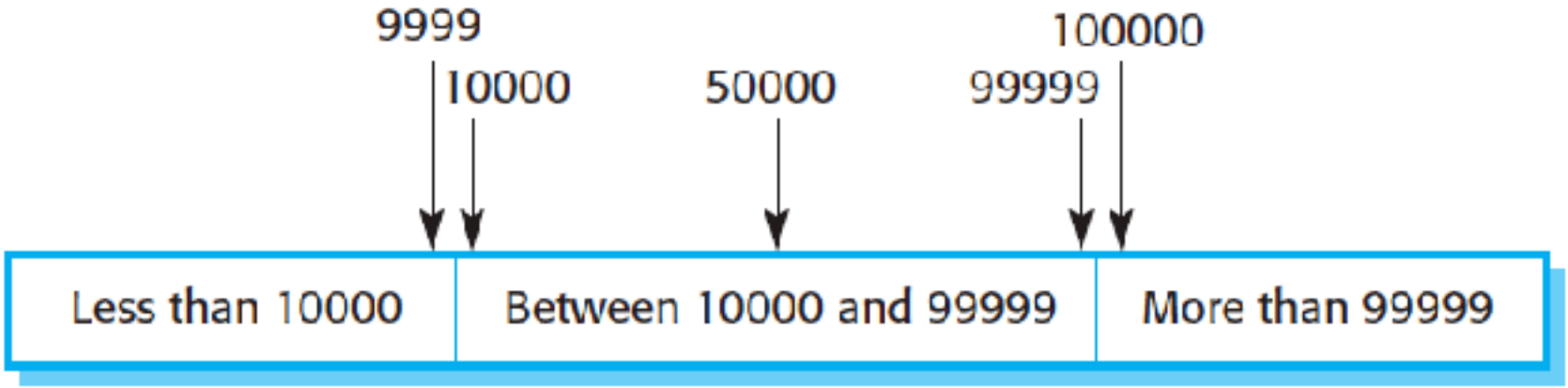
Number of input values

Less than 10000	Between 10000 and 99999	More than 99999

Input values



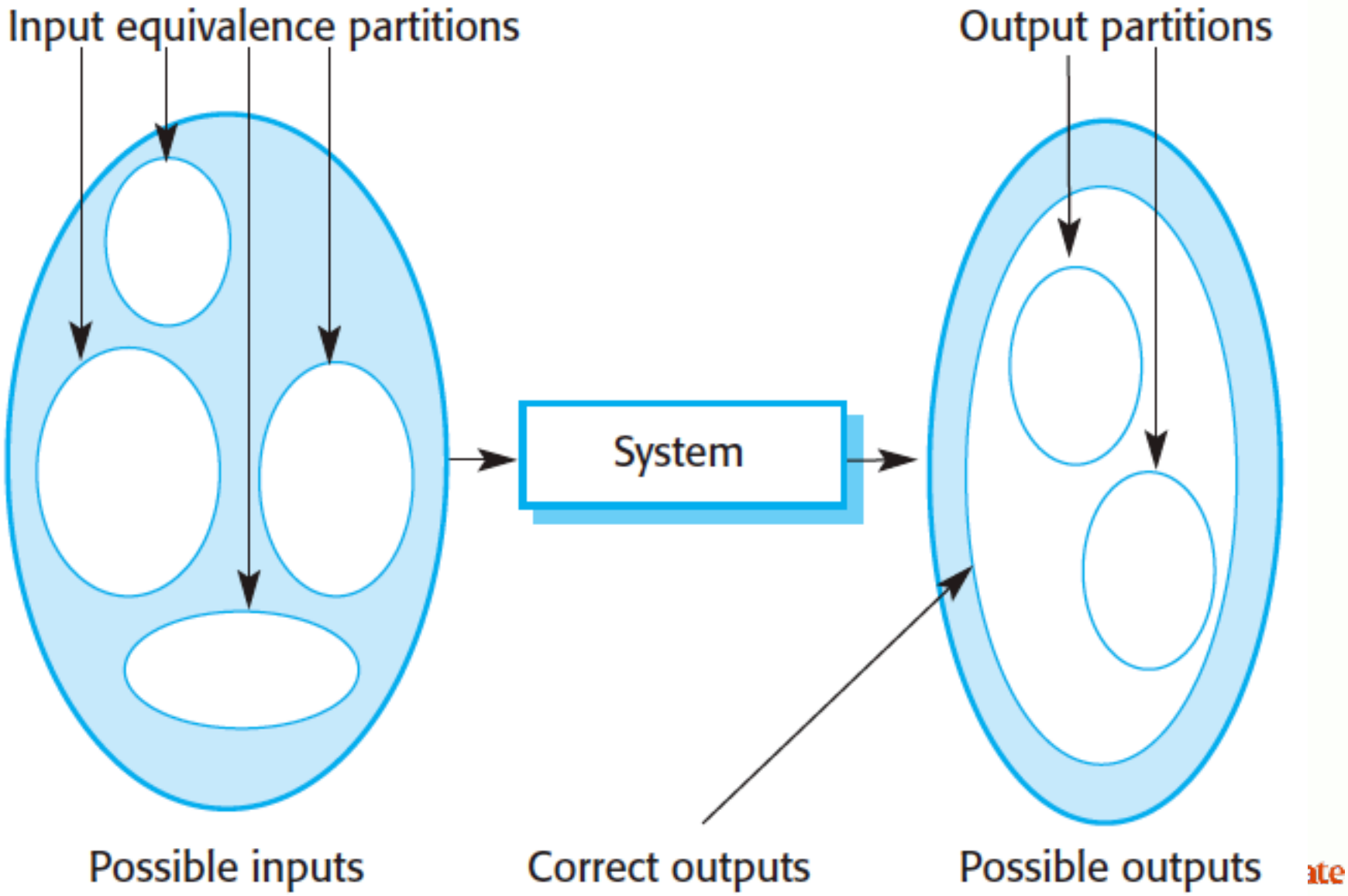
Number of input values



Input values

Choosing Inputs

- Testing effectiveness is based on how good your tests are
- New tests should add value to existing ones
  - Equivalence partitioning: partitioning a set of inputs based on expected outputs and testing across each partition
  - if inputs fall into the same partition (i.e. are equivalent) only one set should be used in a test


Input equivalence partitions
Output partitions
System
Possible inputs
Correct outputs
Possible outputs

Some Adequacy Criteria

- Requirements coverage
- **Statement coverage**
- **Branch coverage**
- Condition coverage
- **Path coverage**
- Boundary value coverage
- Dataflow coverage

Statement Coverage

Test cases that ensure coverage of every executable program statement

```
PROGRAM GCD
begin
1    read(x)
2    read(y)
3    while (x <> y) do
4              if (x > y) then
5                    x = x - y
               else
6                    y = y - x
               endif
      endwhile
7    print x
end
```

Try:

$x = 8, y = 4$

$x = 1, y = 1$

$x = 2, y = 2$

$x = 2, y = 4$

Oregon State University

Statement Coverage – A Weakness

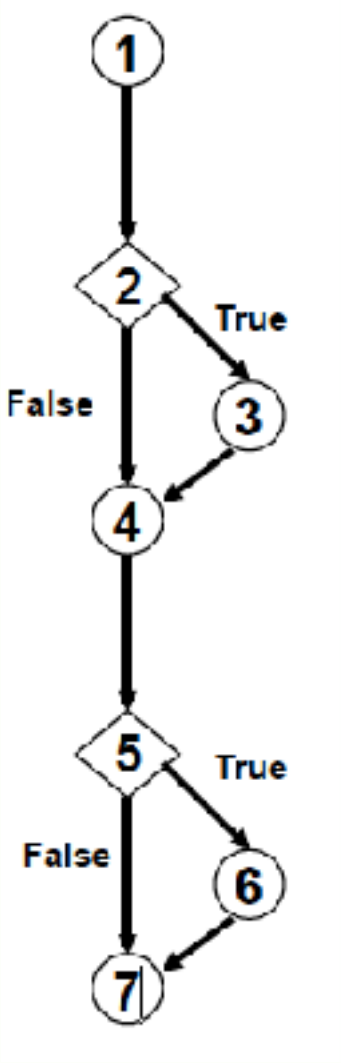
What's the weakness?

```
1    read(x)
2    read(y)
3     if (x > y) then
4           x = x – y
6    print x
end
```

- Do-While not covered
- Else path (if no statement) not checked
- No test logical operators (short circuit)
- Consecutive switch(case) statements not tested

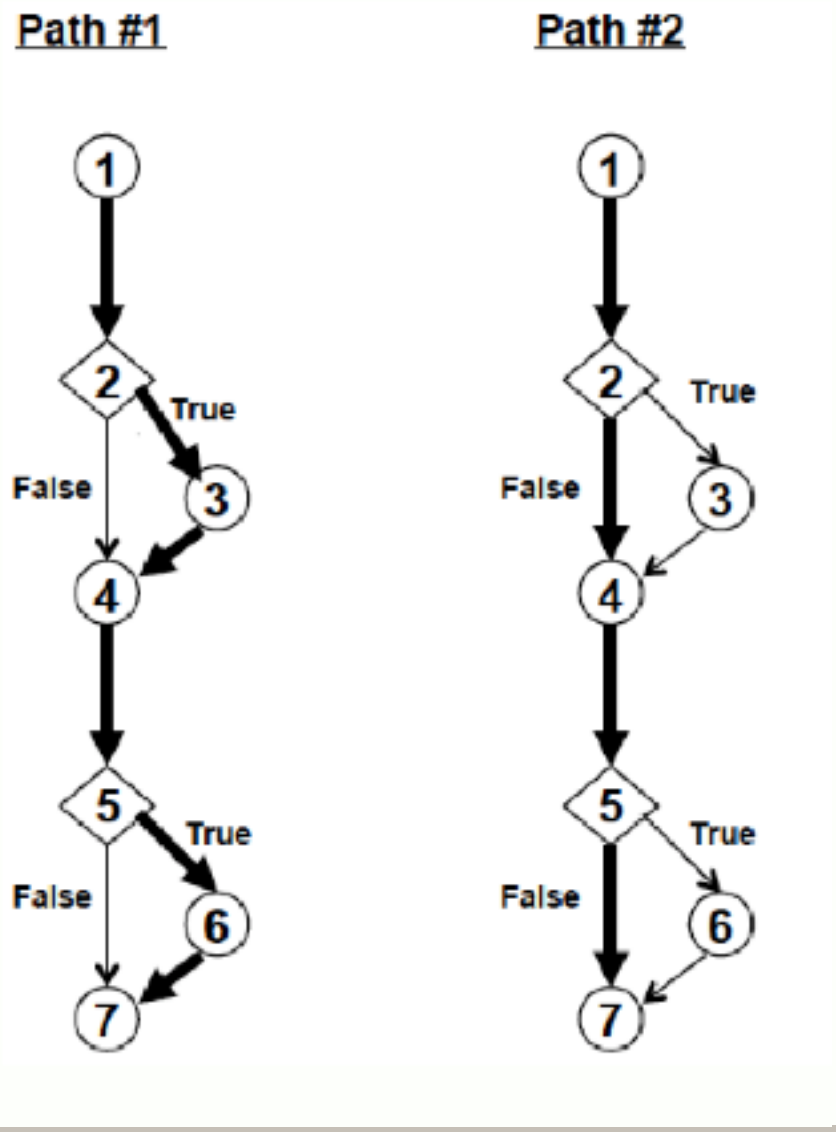
Oregon State University

Program Control-Flow Graph



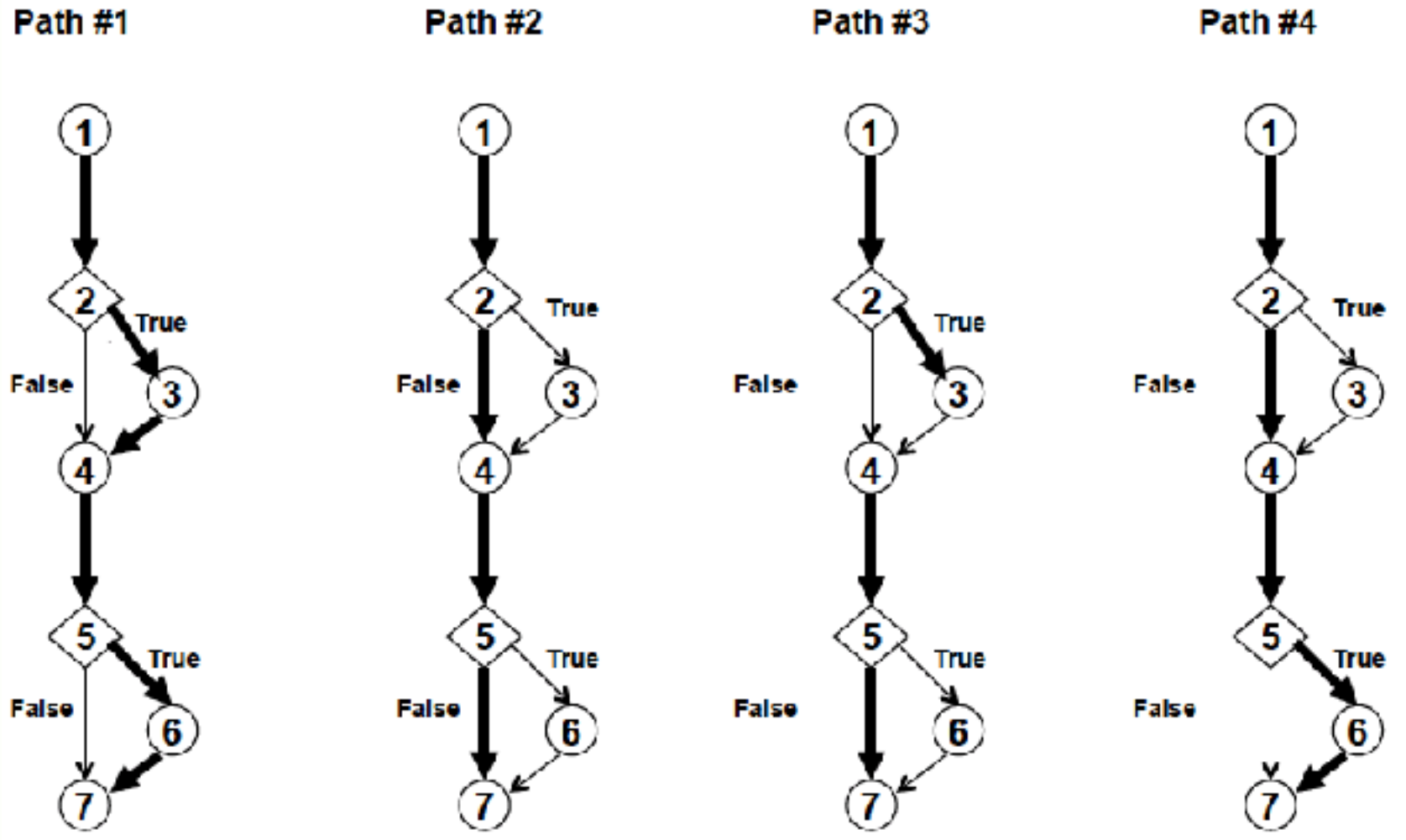
Oregon State University

Branch Coverage



Oregon State University

Path Coverage



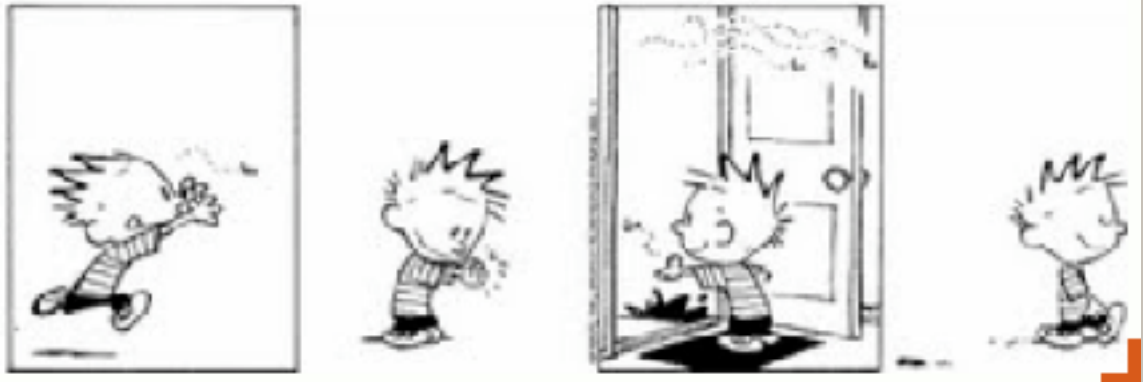
Post Unit testing

- Interface Testing
  - verifies module functionality
- Integration Testing
  - verifies combination of modules
- System Testing
  - verifies the entire application
- Usability Testing
  - validates user satisfaction
- **Regression**
  - verifies that changes didn't break something
- **Acceptance Testing**
  - validates product / requirements
- Installation Testing
  - verifies deployment environment
- Robustness Testing
  - verifies error / anomaly handling
- Performance Testing
  - verifies and validates overall system resource usage within tolerance

Oregon State University

Regression Testing

- Testing the system to ensure that *new changes have not broken any existing functionality.*
- If manual process, regression testing is expensive and time consuming
- A change cannot be accepted until the entire test suite passes.
- Regression testing is one of the most effective way to save time debugging and preventing faults in software.



User Testing: by users

- Testing by potential end users of the system.
- Test cases are not defined, the system is put through "normal" usage:
  - Can ask to focus on specific aspects of the system
- Takes place on more complete builds of the system, proceeds release testing
- Typical builds that would undergo user testing would be alpha or beta releases
- Acceptance testing: final stage with testing by clients

Participation Quiz