Cpt S 422: Software Engineering Principles II

Testing Fundamentals – Part 2

Dr. Venera Arnaoudova

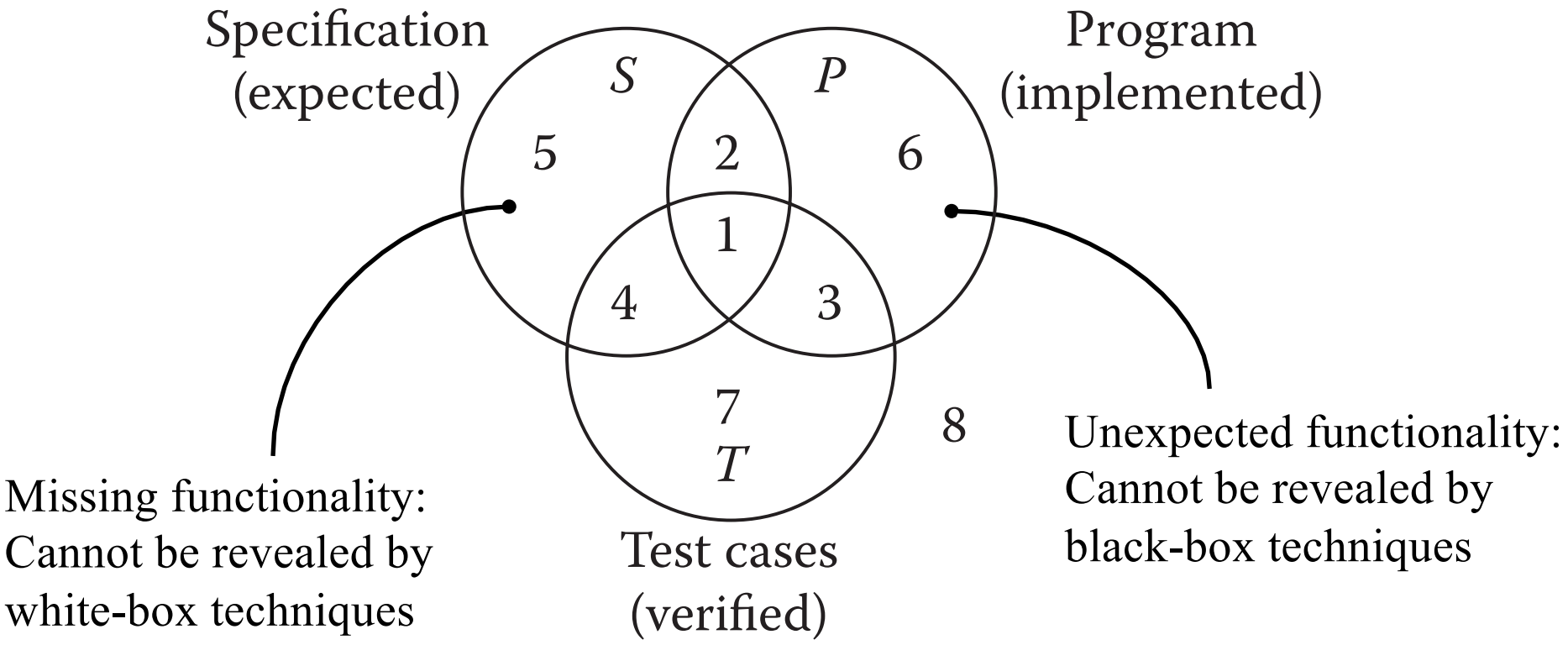
WASHINGTON STATE UNIVERSITY

Exhaustive Testing

- Exhaustive testing, i.e., testing a software system using all the possible inputs, is most of the time impossible.

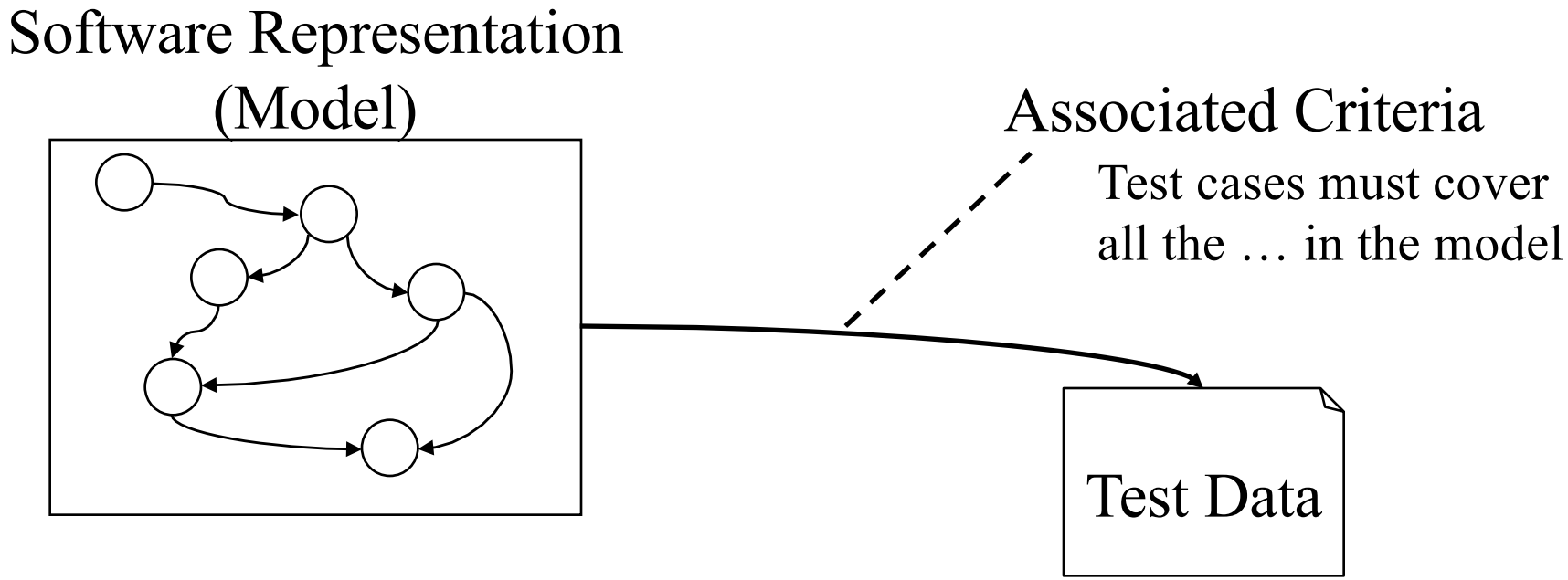
- Examples:
  - A program that computes the factorial function (n!=n.(n-1).(n-2)…1)
    - ✓ Exhaustive testing = running the program with 0, 1, 2, …, 100, … as an input!
  - A compiler (e.g., javac)
    - ✓ Exhaustive testing = running the (Java) compiler with any possible (Java) program (i.e., source code)

What do we want to test?



- the specification ==>Black-Box Testing
- the implementation ==>White-Box Testing

Test Coverage

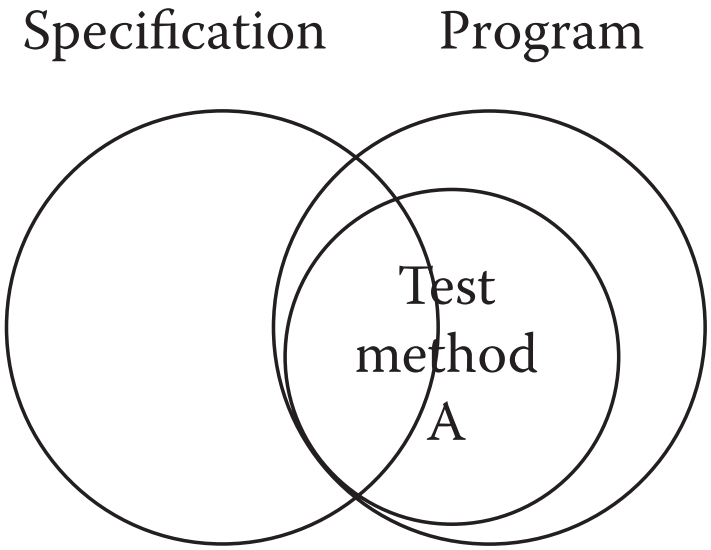


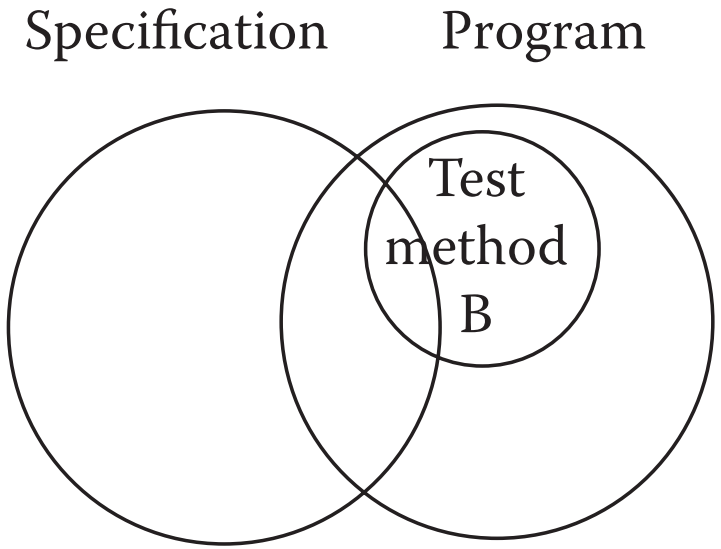
Complete Coverage: White-Box

```
if x > y then
   Max := x;
else
   Max :=x ;   // fault!
end if;
```

- T1: {x=3, y=2; x=2, y=3}?
- T2: {x=3, y=2; x=4, y=3; x=5, y=1}?

Comparing white-box testing techniques



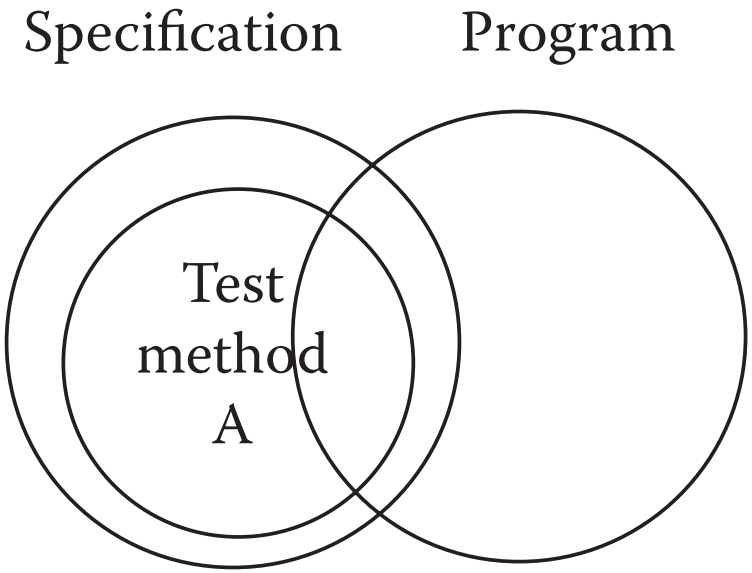


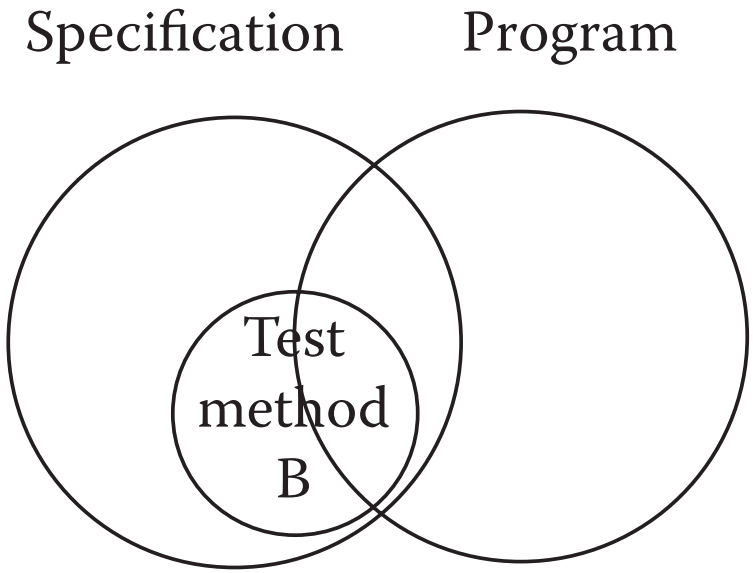
Complete Coverage: Black-Box

- Specification for a function that computes the factorial of a number:
  - If the input value n is < 0, then an appropriate error message must be printed.
  - If 0 <= n < 20, then the exact value of n! must be printed.
  - If 20 <= n < 200, then an approximate value of n! must be printed in floating point format, e.g., using some approximate method of numerical calculus. The admissible error is 0.1% of the exact value.
  - If n>=200, the input can be rejected by printing an appropriate error message.

- Divide the input domain into the following classes and use one data point from each class:
  - {n<0},
  - {0<= n <20},
  - {20 <= n < 200},
  - {n >= 200}.

Comparing black-box testing techniques





Black-box vs White-box Testing

- Black-box

  + Check conformance with specifications

  + It scales up (different techniques at different granularity levels)

  + If the implementation changes the test cases are still valid

  + Test cases and implementation can be done in parallel

  – It depends on the specification notation and degree of detail

  – Do not know how much of the system is being tested

  – What if the software performed some unspecified, undesirable task?

- White-box

  + It allows you to be confident about test coverage

  + It is based on control or data flow coverage

  – It does not scale up (mostly applicable at unit and integration testing levels)

  – Unlike black-box techniques, it cannot reveal missing functionalities (part of the specification that is not implemented)

Static versus dynamic testing

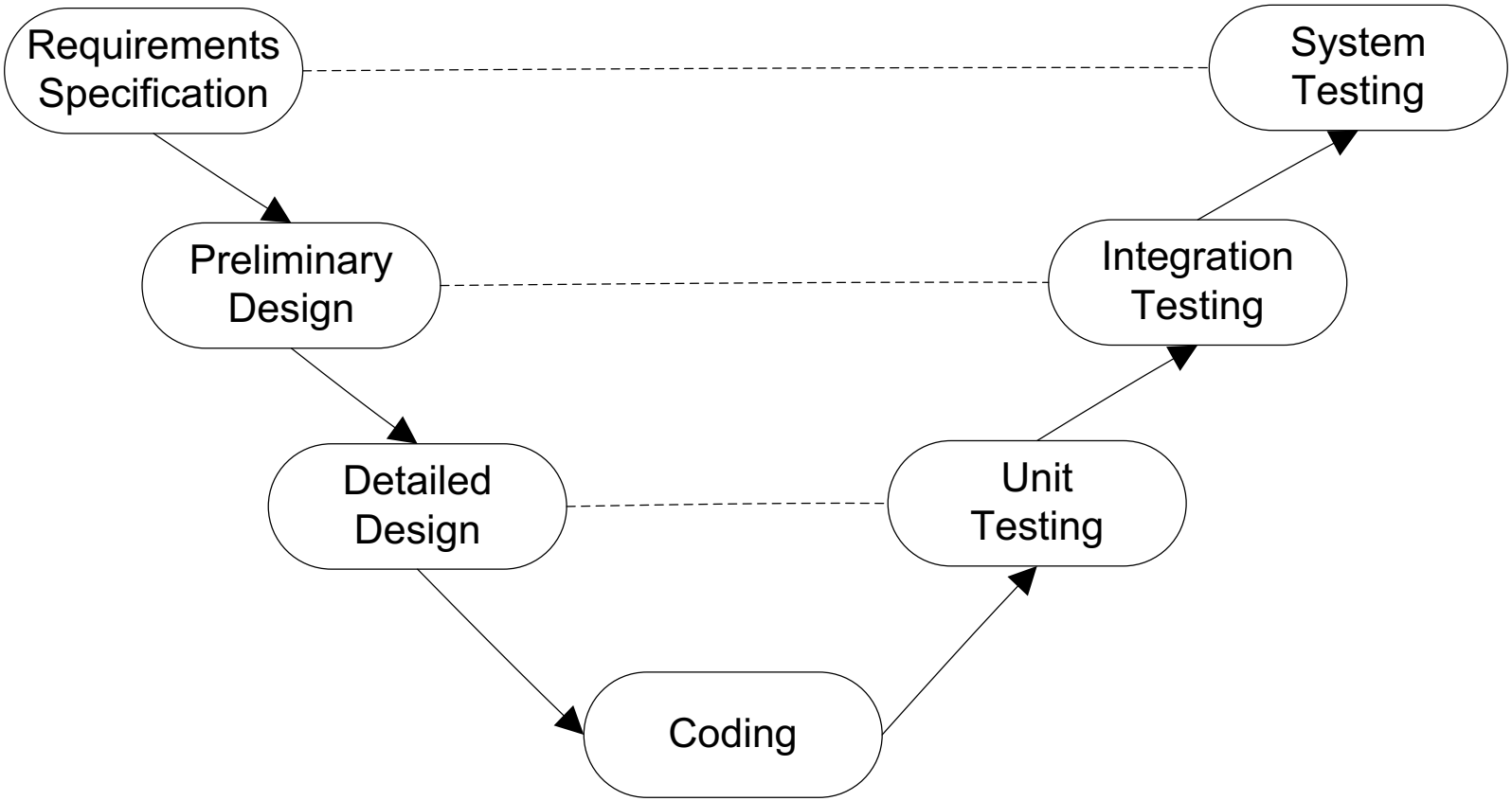
- **Static Testing** : Testing without executing the program
  - This include software inspections and some forms of analyses
  - Very effective at finding certain kinds of problems – especially "potential" faults, that is, problems that could lead to faults when the program is modified
- **Dynamic Testing** : Testing by executing the program with real inputs

Life cycle based testing

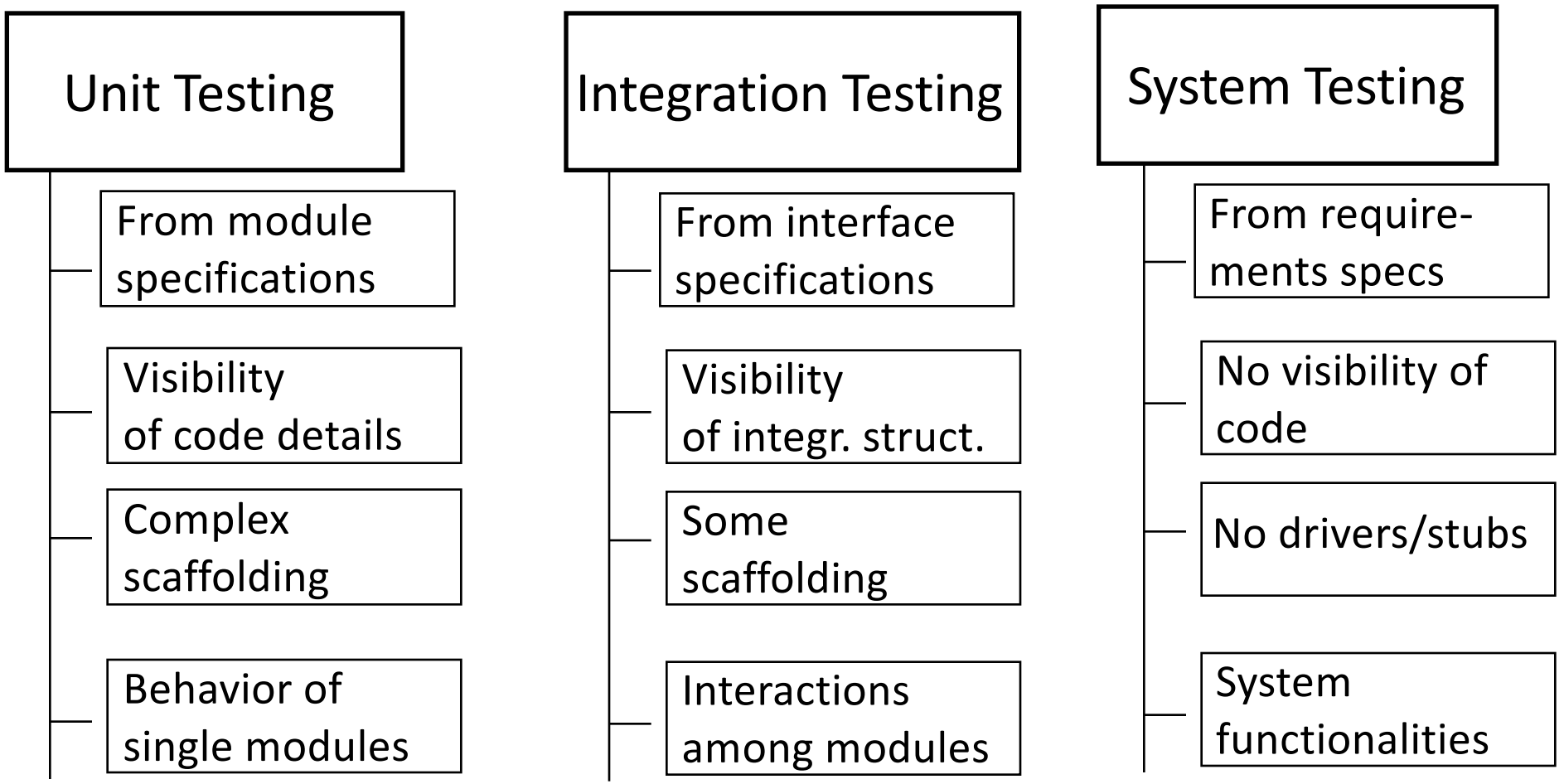
Many Causes of Failures

- The specification may be wrong or have a missing requirement
- The specification may contain a requirement that is impossible to implement given the prescribed software and hardware
- The system design may contain a fault
- The program code may be wrong

Levels of abstraction in testing



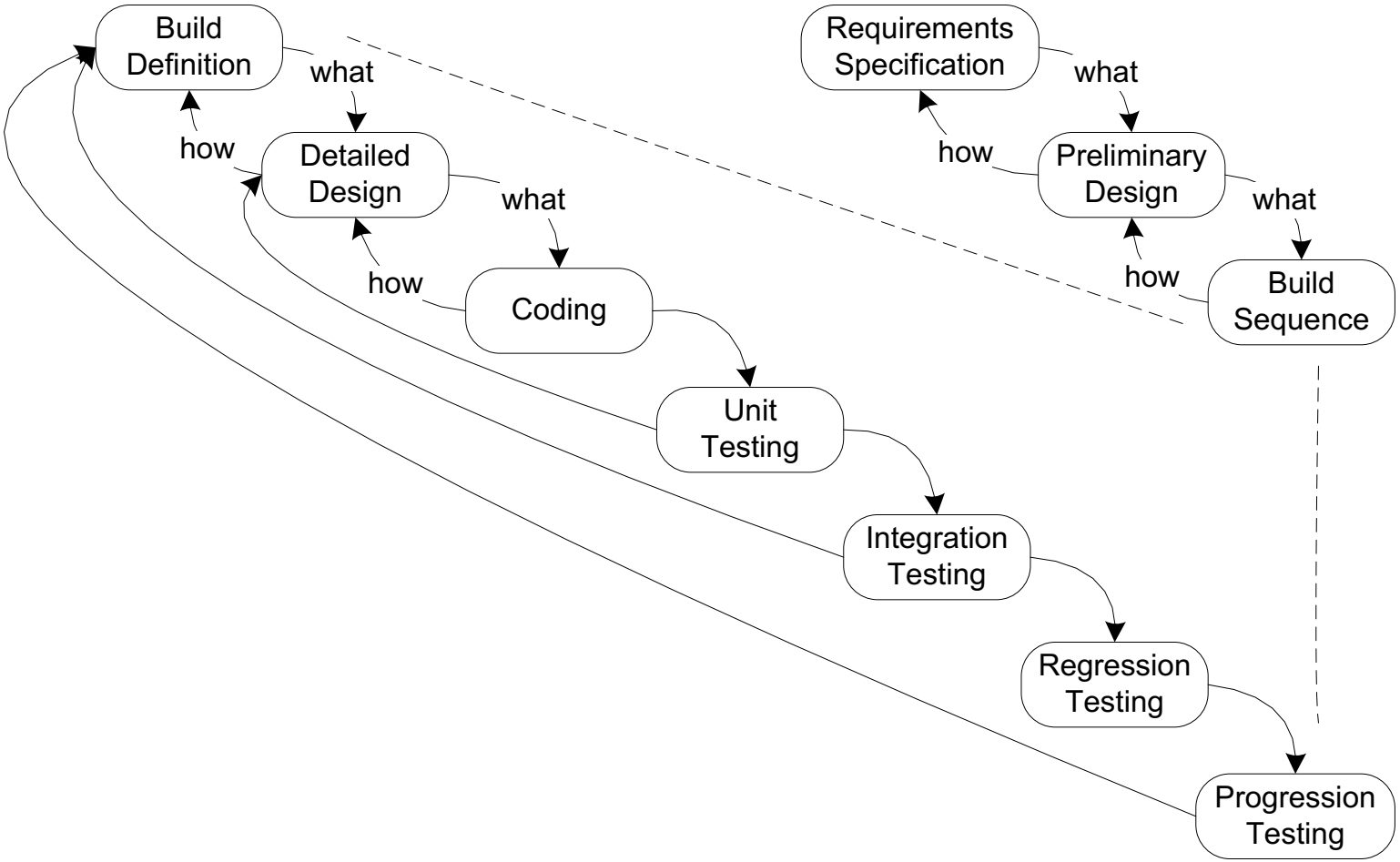
Differences among testing levels



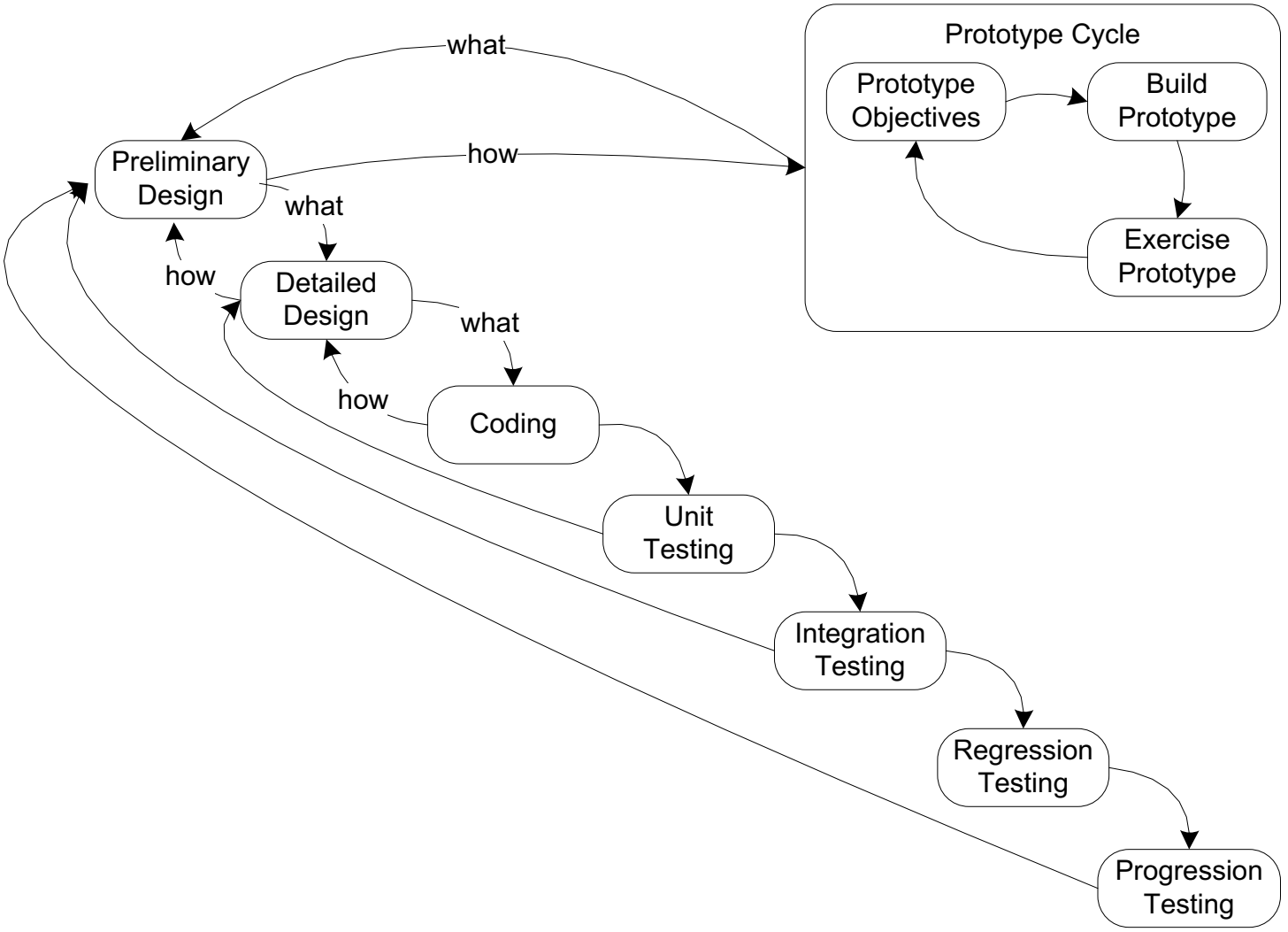
Spin-off Models

- Iterative Development
- Rapid Prototyping
- Executable Specification
- Agile models
  - Scrum
  - eXtreme Programming (XP)
  - Test-Driven Development
- …

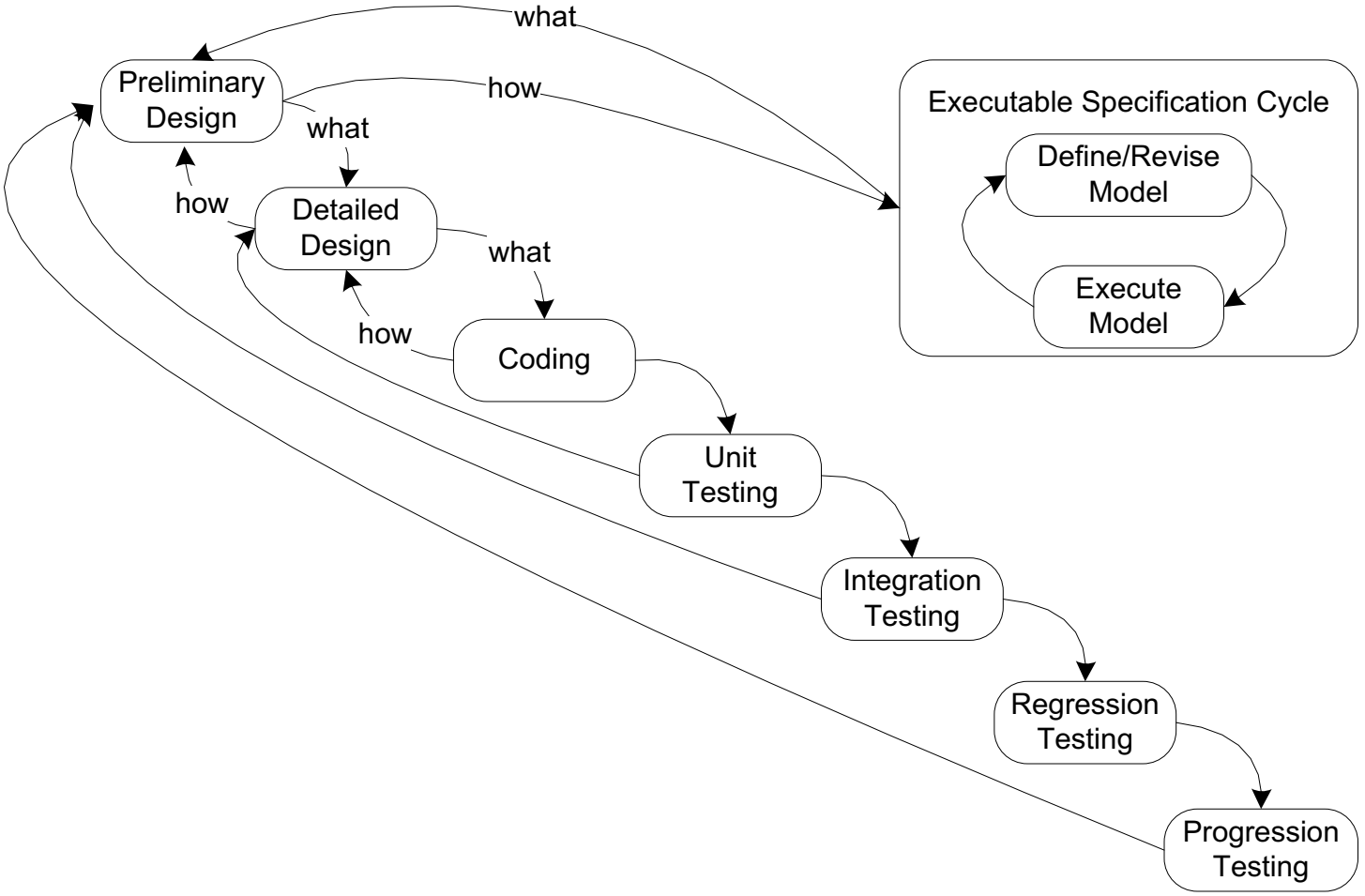
Iterative Development



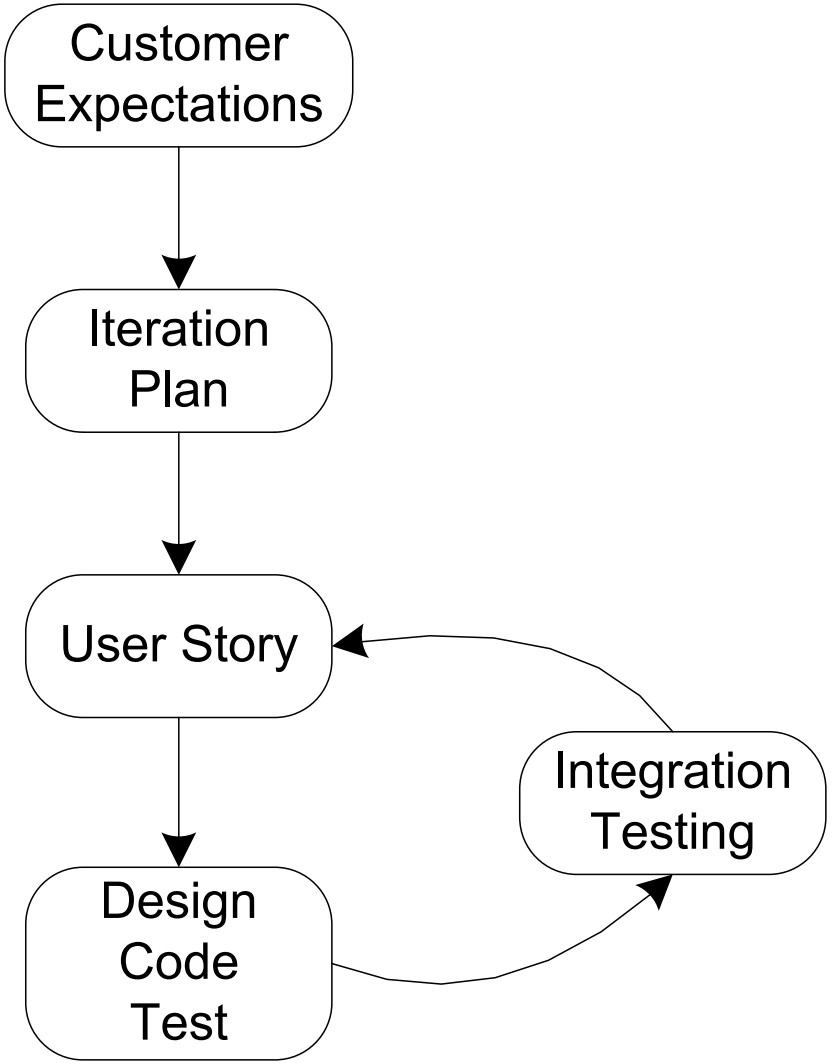
Rapid Prototyping



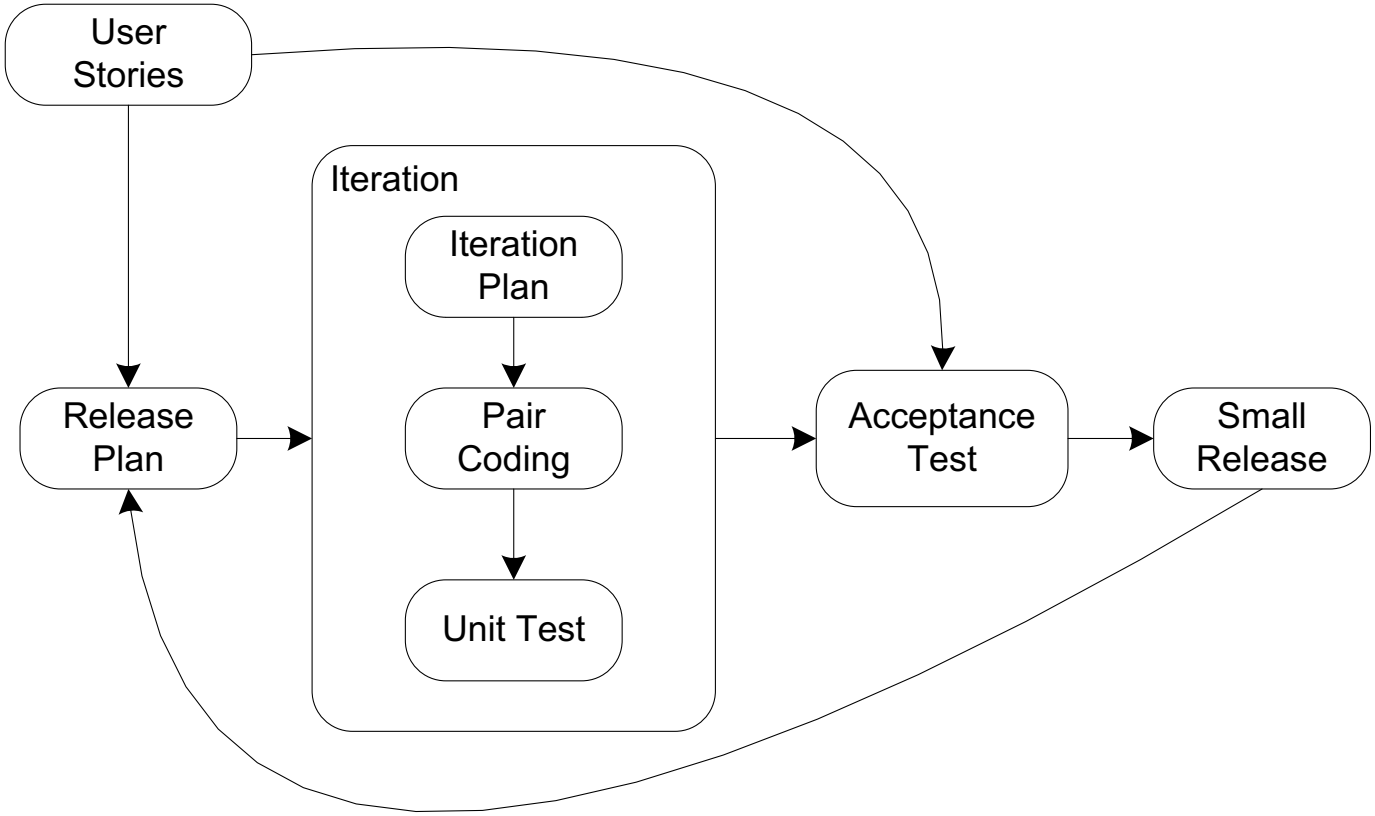
Executable Specification



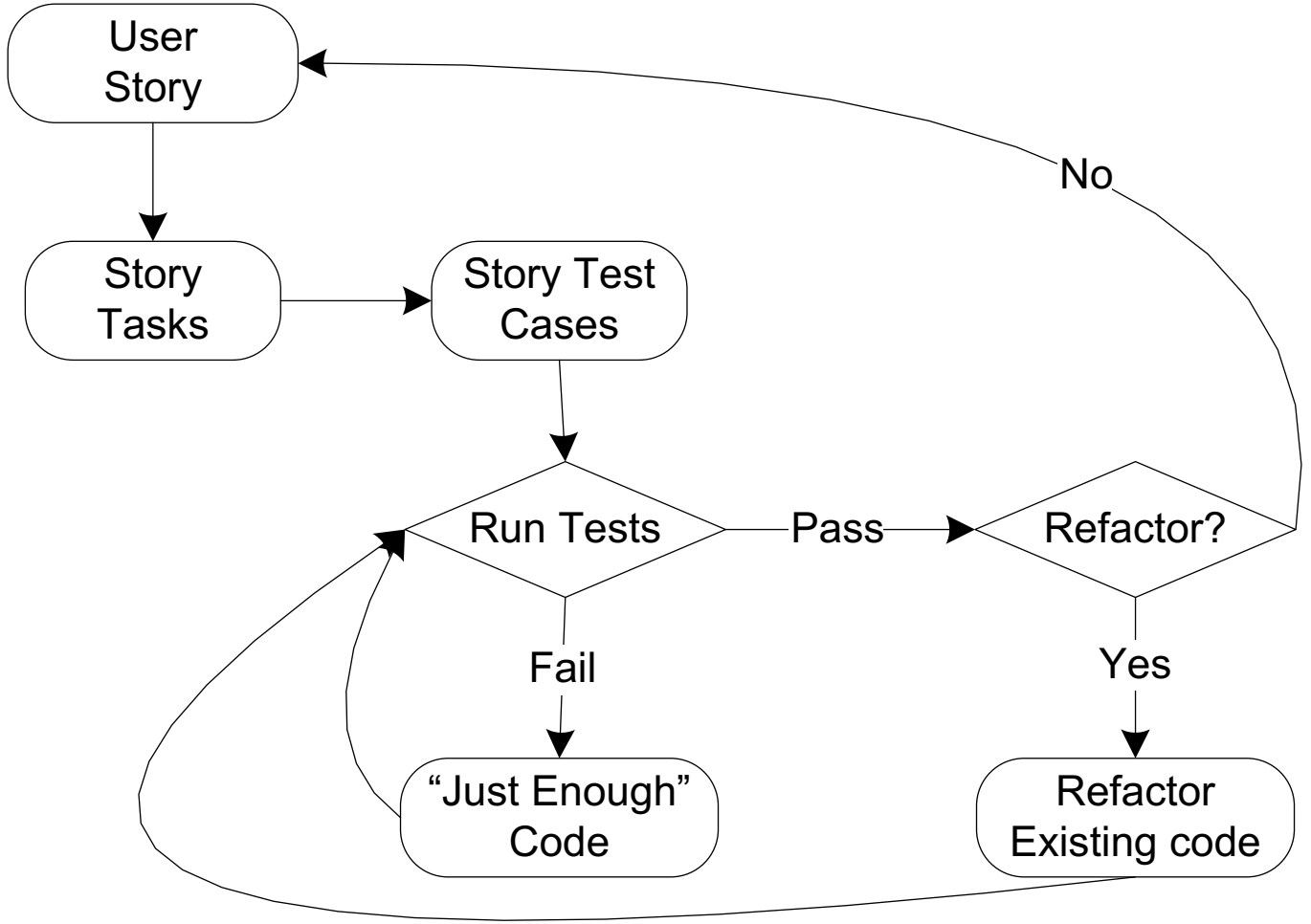
Generic Agile Lifecycle



eXtreme Programming



Test-Driven Development (TDD)



Scrum Lifecycle