Verification and Validation

Thierry Sans

with slides from Anya Tafliovich

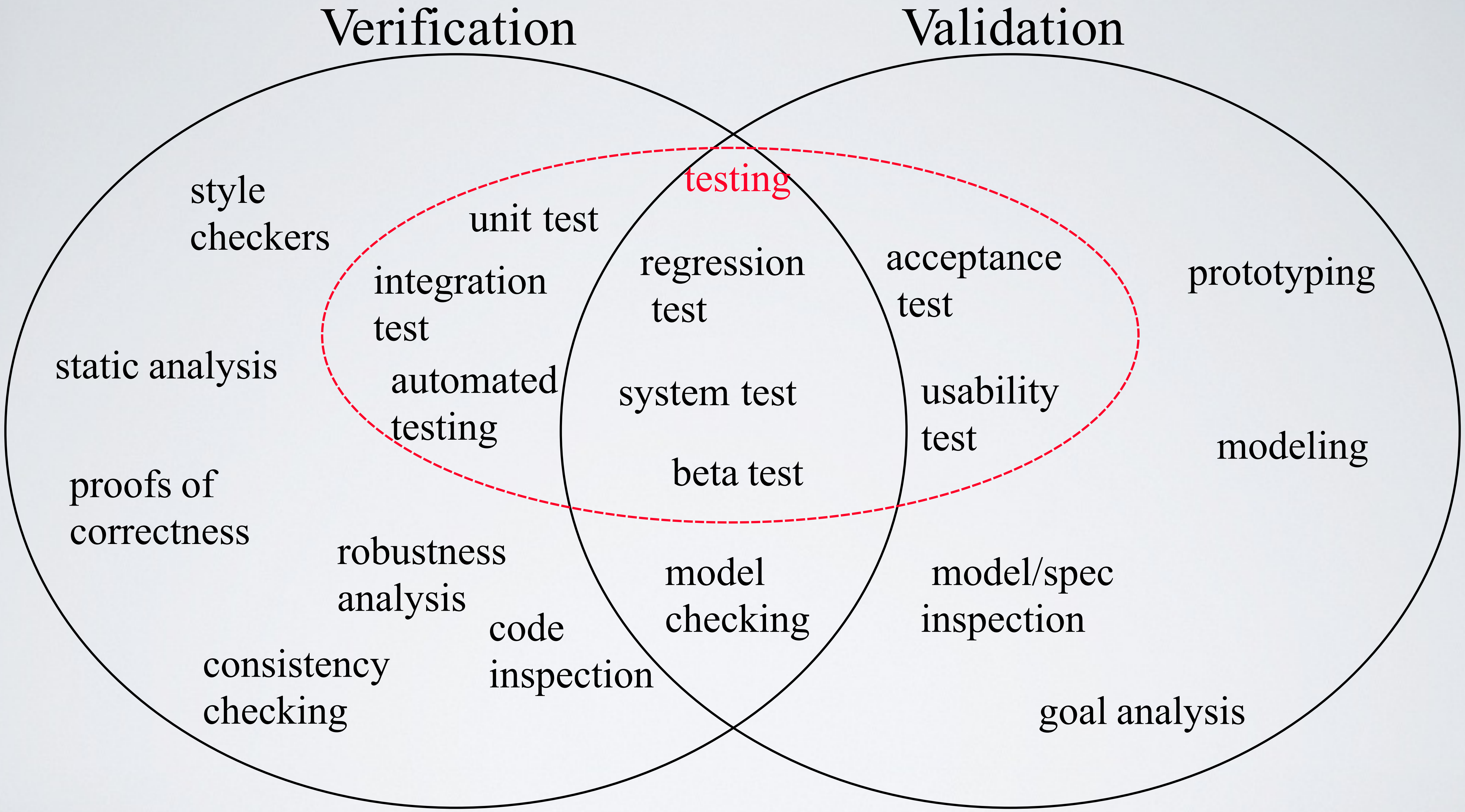
Verification vs Validation
from the Serendipity Blog - Steve Easterbrook

Verification : *Are we building the system right?*

- Does our design meet the spec?
- Does our implementation meet the spec?
- Does the delivered system do what we said it would do?
- Are our requirements consistent with one another?

Validation : *Are we building the right system?*

- Does our problem statement accurately capture the real problem?
- Did we account for the needs of all the stakeholders?

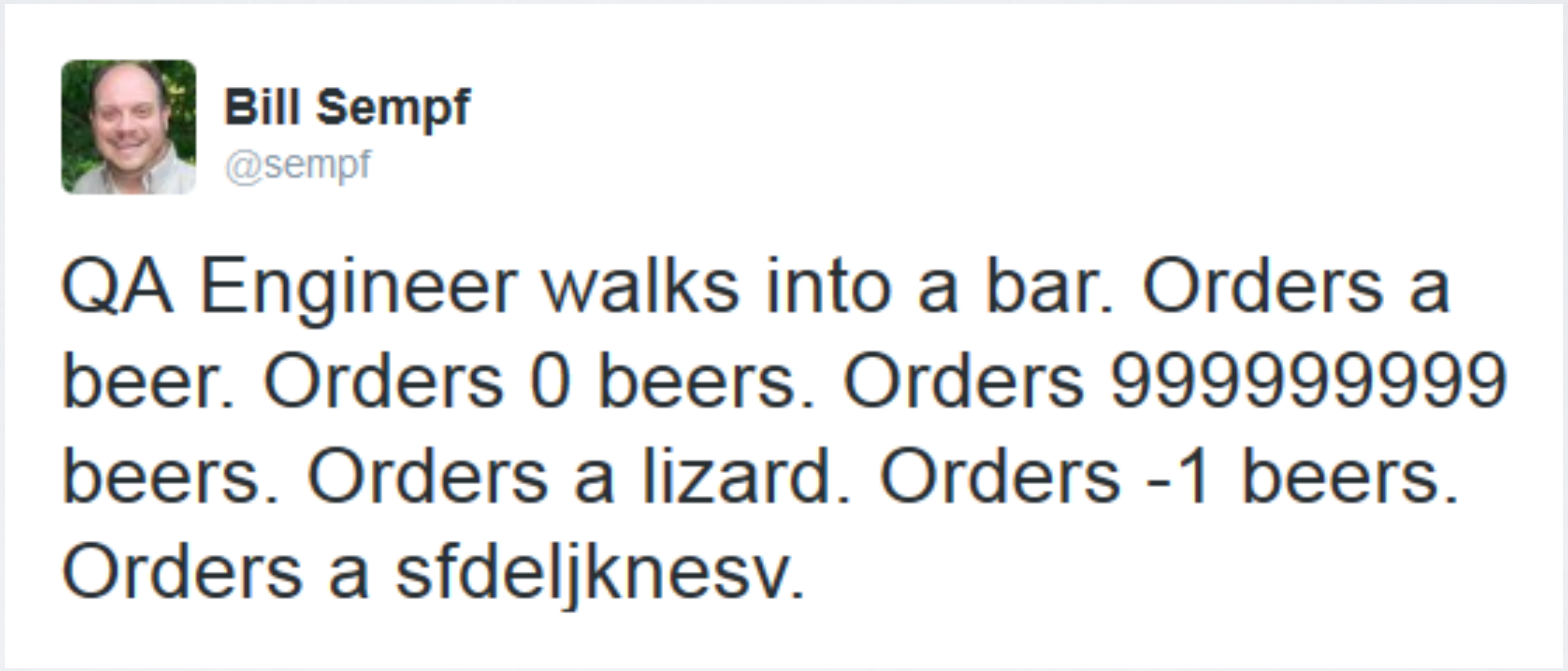


from the Serendipity Blog - Steve Easterbrook

Mainly 3 approaches for verification and validation

- **Test** - experiment with the program
- **Review** - inspect the program and the specs
- **Verify** - reason about the program

Testing


Bill Sempf
@sempf

QA Engineer walks into a bar. Orders a beer. Orders 0 beers. Orders 999999999 beers. Orders a lizard. Orders -1 beers. Orders a sfdeljknesv.

Testing to detect **defects** and **failures**

Many causes of **defects** in software

- Missing requirement
- Specification wrong
- Requirement that was infeasible
- Faulty system design
- Wrong algorithms
- Faulty implementation

Defects (may) lead to failures:

◉ but the failure may show up somewhere else

➡ tracking the failure back to a defect can be hard

Defects

Syntax

- Incorrect use of programming constructs

Algorithmic

- Branching too soon or too late
- Testing for the wrong condition
- Failure to initialize correctly
- Failure to test for exceptions
- Type mismatch

Precision

- Mixed precision, floating point conversion, etc.

Stress

- Overflowing buffers, etc.

Timing

- Processes fail to synchronize
- Wrong order of events

Throughput

- Performance lower than required

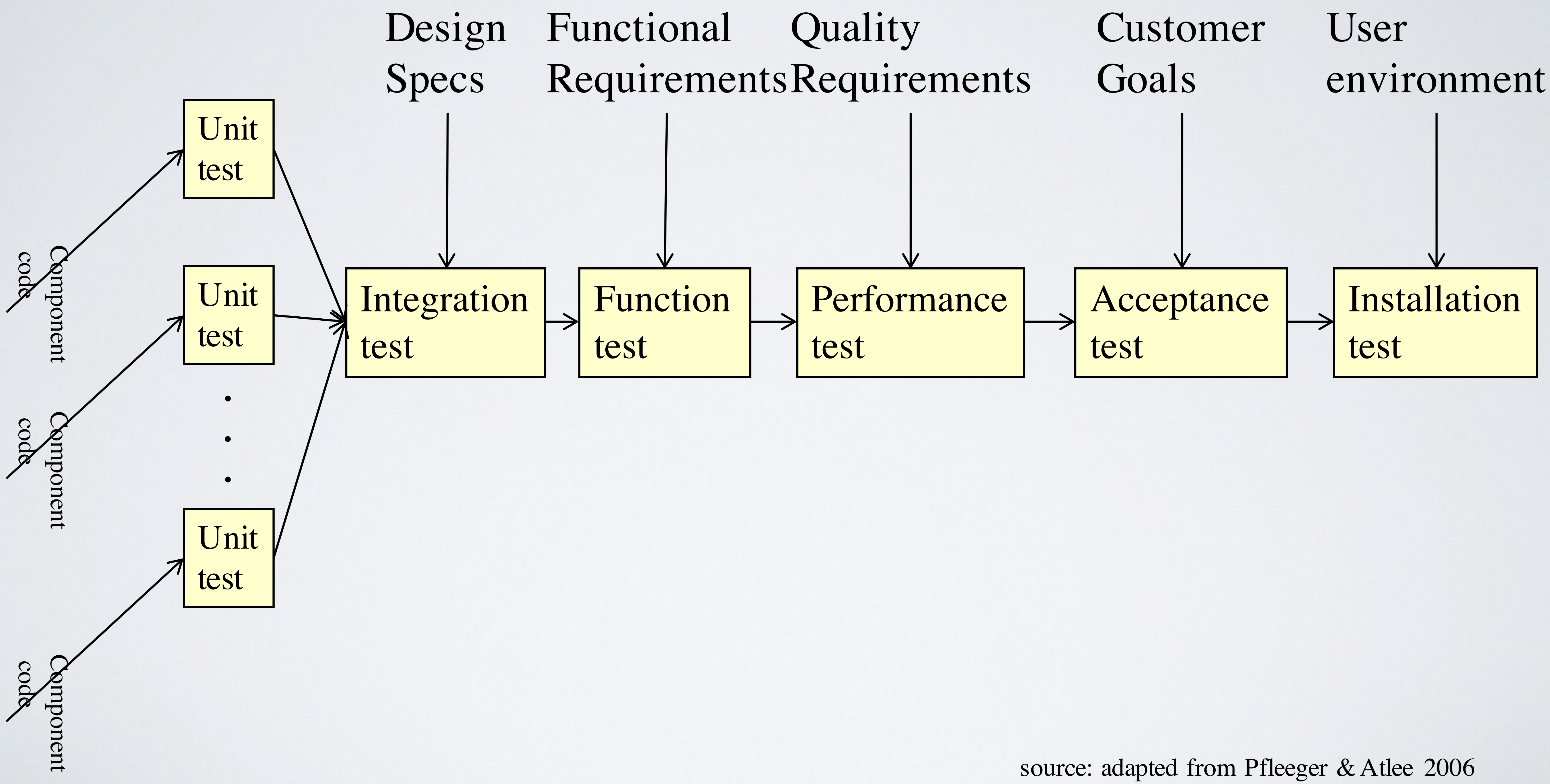
Recovery faults

- Incorrect recovery after another failure

Documentation

- Design docs or user manual is wrong

The testing pipeline



source: adapted from Pfleeger & Atlee 2006

Beta-testing

Customers test for free

➡ Gives test cases representative of customer use

✓ Helps to determine what is most important to the customers

✓ Can do more configuration (environment) testing than in your testing lab

◉ Beta testers might have a particular perspective to the system
may result in not catching diverse system bugs

◉ Beta testers usually will not report usability problems, bugs they do not understand, and bugs that seem obvious

◉ Most beta testers are "techies" who have a higher tolerance of bugs
They do not represent the average customer

◉ Takes much more time and effort to handle a user reported bug

White-Box Testing

Test the structural parts of the software

➡ The tester has explicit knowledge about the internals

◉ Biased, the tester chooses specific paths and determines the appropriate output

✓ Can be applied at the unit, integration and system levels

Black-Box Testing

Test functional requirements of a program

➡ The tester has no prior knowledge to the internals

✓ Unbiased, no programming knowledge needed

✓ Test cases can be made very early on after specs are done

◉ Cannot identify all possible test case

◉ Can be redundant

Example

```
def search(lst, elt):
  '''Return the index of the first
     occurrence of elt in the list lst. If
     elt is not in lst, raise
     NoSuchElementError.
     Pre: lst is sorted in non-decreasing
          order.
  '''
```

Testing in Agile

Testing Principles in Agile

1. Developers defines the unit tests
   - ➡ Driven by the actual implementation
2. Product owner defines the acceptance tests
   - ➡ Driven by user stories
3. Automated Testing is mandated

TDD - Test Driven Development

➡ Create automated tests before writing the code itself

TDD Methodology

1. **Add** a new test to the test suites based on requirements (before implementing the new feature)

2. **Run** new the test along with others previously in the test suite (new one should fail, old ones should succeed)

3. **Write** rough code

4. **Run** all tests and debug code until they all pass

5. **Re-factor** code and keep-on testing

6. **Repeat**