Software Verification

Bertrand Meyer
Carlo Furia
Sebastian Nanz

Testing basics

Verification techniques

A priori techniques

- Build system for quality; e.g.: process approaches, proof-guided construction, Design by Contract

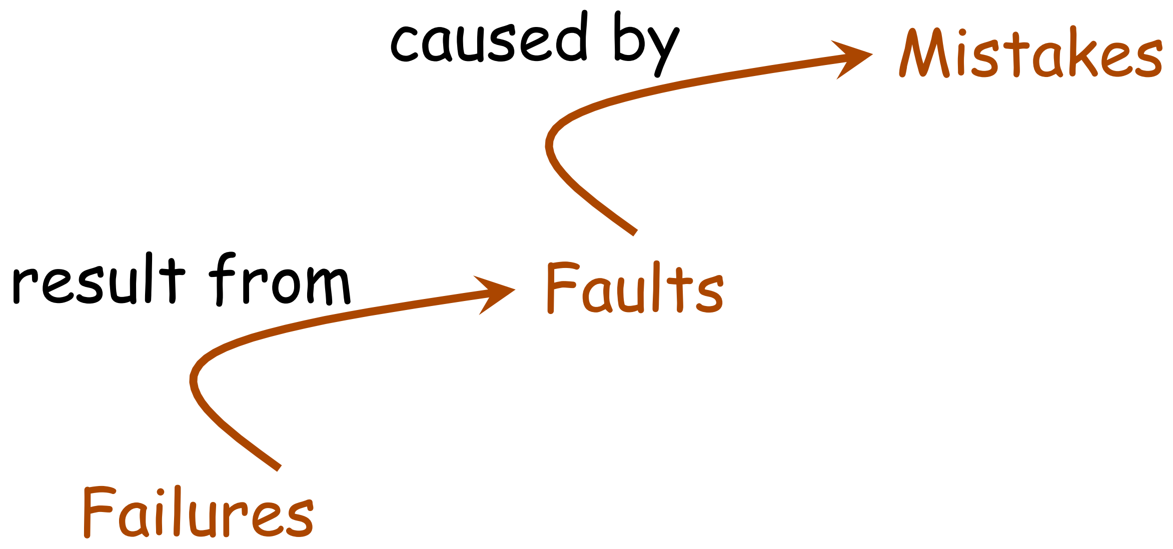
A posteriori techniques

- Static: from software text only
  - Proofs
  - Static analysis
  - Abstract interpretation
- Dynamic: execute software
- Mixed:
  - Symbolic execution
  - Model checking

Failures, faults, mistakes

Quality is the absence of “deficiencies” (or “bugs”).

IEEE standard terminology:



What is a failure?

A failure is any event of system execution that violates a stated quality objective

Typical examples:

- Wrong result
- Crash

Testing basics

Definition: testing

To test a software system is to try to make it fail

Testing is none of:

- Ensuring software quality
- Assessing software quality
- Debugging

"*Ich bin der Geist, der stets verneint*"
Goethe, Faust, *Act I*

Seven principles of software testing

Bertrand Meyer, *Seven Principles of Software Testing*, IEEE *Computer*, August 2008

1. To test a program is to try to make it fail
2. Tests are no substitute for specifications
3. Any failed execution must yield a test case, to remain forever remain part of the regression test base
4. Determining success or failure (oracles) must be automatic

   4': Oracles should be part of the program, as contracts

5. A test suite must include both manual and automated cases
6. Don't believe your testing insights: evaluate any testing strategy through objective criteria
7. The most important criterion is number of faults found against time: fc (t)

Exercise*

Scenario:

- A program reads three integers representing the lengths of a triangle's sides, and prints a message stating whether the triangle is scalene, isosceles or equilateral.

Task:

- Devise inputs to test the program as thoroughly as possible

*After Yuri Gurevich, LASER summer school 2009. Exercise originally from Glen Myers, "*The Art of Software Testing*", Wiley, 1979

Myers: Do you have these?

1. A scalene triangle
2. An isosceles triangle
3. An equilateral triangle
4. 3 permutations of 2
5. A zero-length side
6. A negative-length side
7. Three positive sides, sum of two = third
8. Three permutations of 7
9. Three positive sides, sum of two < third
10. Three permutations of 9
13. (0,0,0)
14. Noninteger values
15. Wrong number of initial values
16. The expected output in each case

- Intermezzo -

Test-Driven Development

"The agile manifesto"

agilemanifesto.org

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- *Individuals and interactions over processes and tools*
- *Working software over comprehensive documentation*
- *Customer collaboration over contract negotiation*
- *Responding to change over following a plan*

That is, while there is value in the items on the right, we value the items on the left more.

Agile methods: basic concepts

Principles:

- Iterative development
- Customer involvement
- Support for change
- Primacy of code
- Self-organizing teams
- Technical excellence
- Search for simplicity

Shunned: "big upfront requirements"; plans; binding documents; diagrams (e.g. UML); non-deliverable products

Practices:

- Evolutionary requirements
- Customer on site
- User stories
- Pair programming
- Design & code standards
- Test-driven development
- Continuous refactoring
- Continuous integration
- Timeboxing
- Risk-driven development
- Daily tracking
- Servant-style manager

Test-Driven Development

Evolutionary approach to development

Combines

- Test-first development
- Refactoring

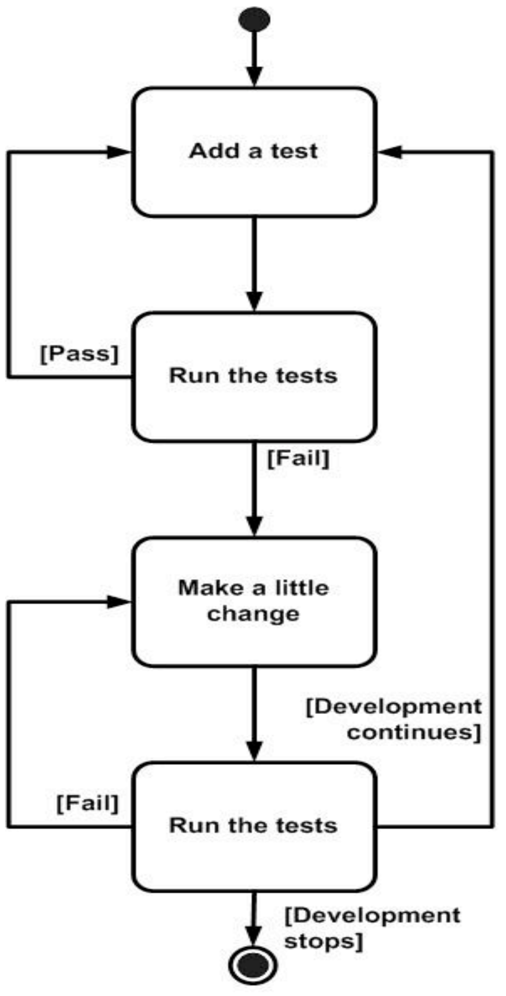
Primarily a method of software design

- Not just a method of testing

TDD1: Test-First Development

After Kent Beck*

1. Add a test
2. Run all tests and check the new one fails
3. Implement code to satisfy functionality
4. Check that new test succeeds
5. Run all tests again to avoid regression
6. Refactor code



** Test Driven Development: By Example, Addison-Wesley*

TDD 2: Refactoring

A change to the system that leaves its behavior unchanged, but enhances some non-functional quality:

- Simplicity
- Understandability
- Performance

Refactoring does not fix bugs or add new functionality.

Examples of refactoring

Change the name of a variable, class, ...

Convert local variable to attribute

Generalize type

Introduce argument

Turn a block of code into a routine

Replace a conditional with polymorphism

Break down large routine

TDD = TFD + Refactoring

Apply test-first development

Refactor whenever you see fit (before next functional modification)

TDD: consequences on unit tests

Developers must learn to write good unit tests:

- Run fast (short setup, run, and tear-down)
- Run in isolation (reordering is possible)
- Use data that makes test cases easy to read
- Use real data when needed
- Each test case is one step towards overall goal

TDD assessment

For:

- Reclaims central role of tests
- Continuous execution: reduce gap between decision and feedback
- Encourage developers to write code that is easily tested
- Yields extensive test repository
- Requires that all tests pass

But:

- Tests are not specs
- Some code difficult to test
- Risk that program pass tests and nothing else

- End of Intermezzo -

Test-Driven Development

What does testing involve?

- Determine system parts & properties to be tested
- Determine appropriate input values
- Determine expected outputs (oracles)
- Run system on selected input values
- Compare results to oracles
- Measure other execution characteristics: time, space…

Components of a test

Test case specifies:

- The state of the implementation under test (IUT) and its environment before test execution
- The test inputs
- The expected result

Oracles define:

- Expected returned values
- Expected messages
- Expected exceptions
- Resulting state of IUT and environment
- Possibly: pass/no pass evaluation

Test execution

Test suite: collection of test cases

Test driver: class or utility program that applies test cases to an IUT

Stub: partial, temporary implementation of a component

- May serve as a placeholder for an incomplete component or implement testing support code

Test harness : a system of test drivers and other tools to support test execution

Types of tests: scope

Unit test

- Scope: program module, e.g. routine, class, cluster

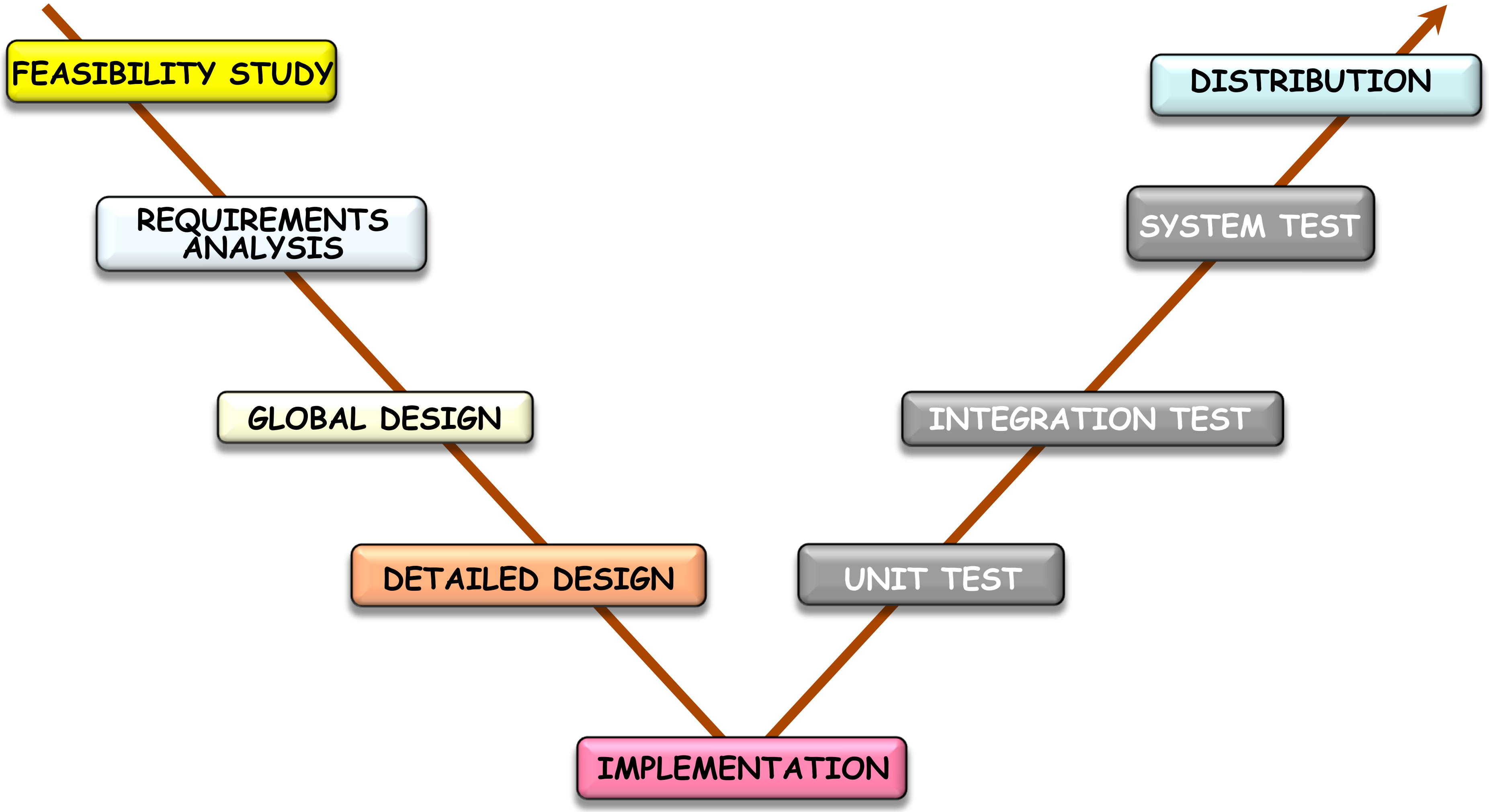
Integration test

- Scope: subsystem or entire system, possibly including hardware
- Exercises interfaces between units to demonstrate that they are collectively operable

System test

- Scope: Complete, integrated application
- Focuses on characteristics that are present only at the level of the entire system
- Categories:
  - Functional
  - Performance
  - Stress or load

V-shaped lifecycle model



Types of tests: intent

Fault-directed testing

- Intent: reveal faults through failures
- Unit and integration testing

Conformance-directed testing

- Intent: demonstrate conformance to required capabilities
- System testing

Acceptance testing

- Intent: enable customer to decide whether to accept software

Types of tests: intent

Regression testing

- After a change., re-test program to find out if change has not introduced, re-introduced or uncovered faults

Mutation testing

- Purposely introducing faults to assess quality of test suite

Black box vs white box testing (1)

Black box testing	**White box testing**
Uses no knowledge of the internals of the SUT	Uses knowledge of the internal structure and implementation of the SUT
Also known as responsibility-based testing and functional testing	Also known as *implementation-based testing* or *structural testing*
Goal: to test how well the SUT conforms to its requirements (Cover all the requirements)	Goal: to test that all paths in the code run correctly (Cover all the code)

Black box vs white box testing (2)

Black box testing	White box testing
Uses no knowledge of the program except its specification	Relies on source code analysis to design test cases
Typically used in integration and system testing	Typically used in unit testing
Can also be done by user	Typically done by programmer