Verification and Validation

Object-Oriented Programming

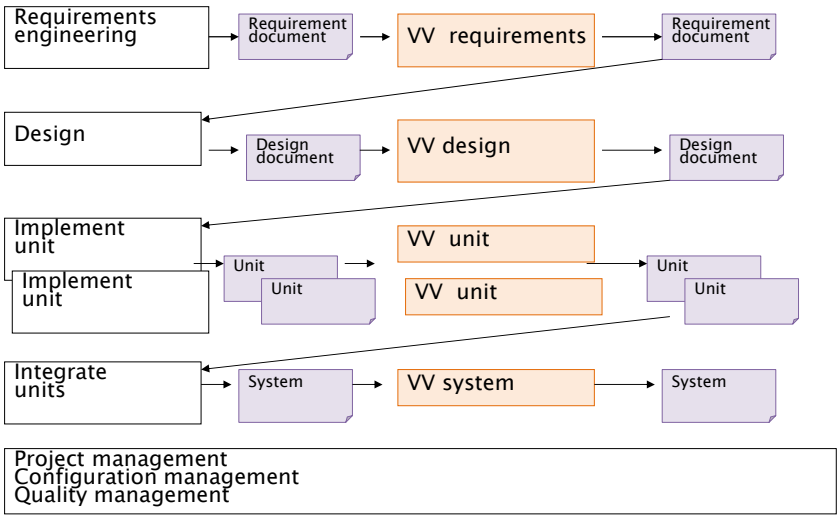
http://softeng.polito.it/courses/09CBI

SoftEng
http://softeng.polito.it

Version 1.2.1 – June 2019

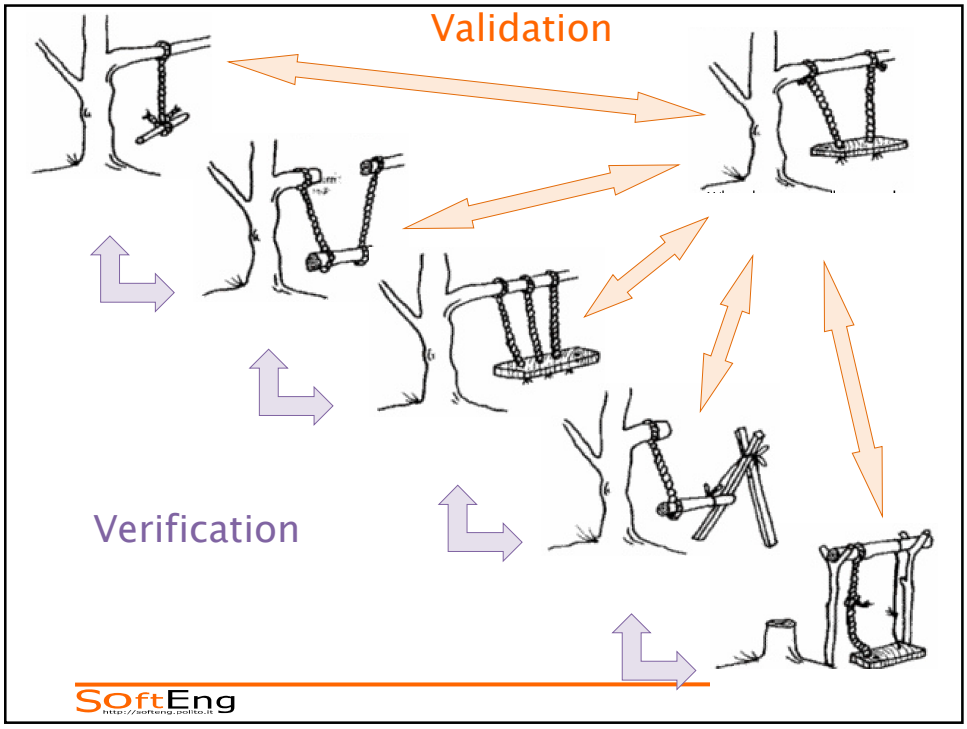



Development process



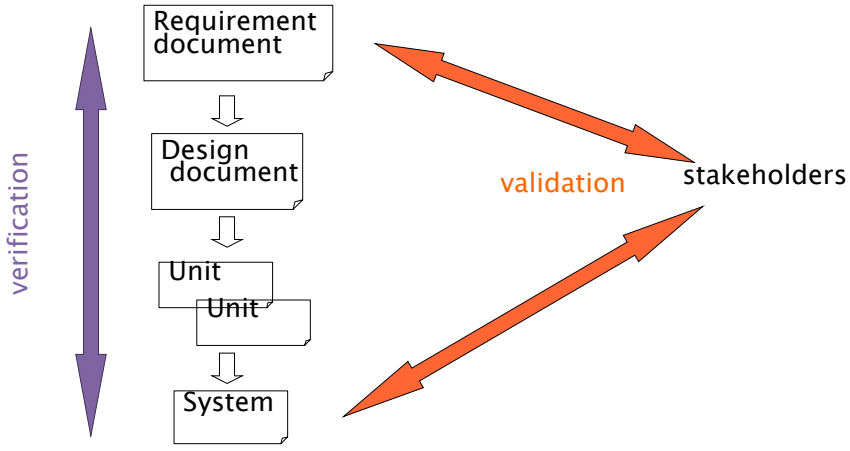
V&V

- Validation
  - is it the right software system?
  - effectiveness
  - external (vs. user)
  - reliability
- Verification
  - is the software system right?
  - efficiency
  - internal (correctness of transformations)
  - correctness



5

V & V



SOftEng

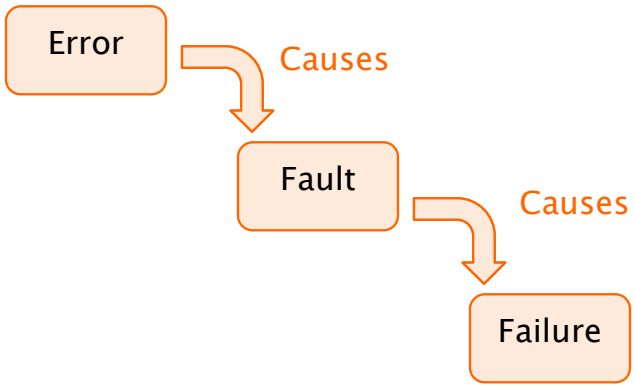
6

TERMINOLOGY

Failure, fault, defect

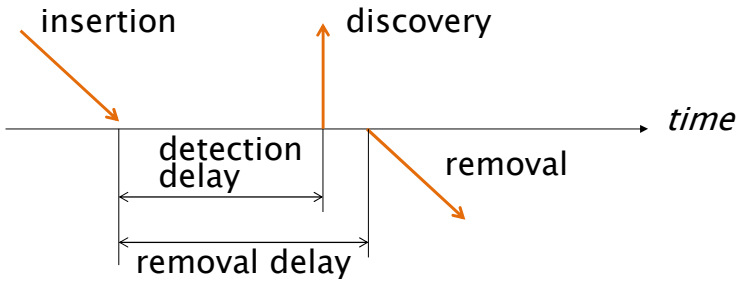
- Error
  - A mistake e.g. committed by a programmer
- Fault (Bug)
  - The feature of software that causes a failure
  - May be due to:
    - An error in software
    - Incomplete/incorrect requirements
- Failure
  - An execution event where the software behaves in an unexpected way
- Defect
  - Typically a fault (sometimes a failure)

Error–Fault–Failure



Insertion / removal

- Defect is characterized by
  - Insertion activity (phase)
  - Discovery
  - Removal activity (phase)



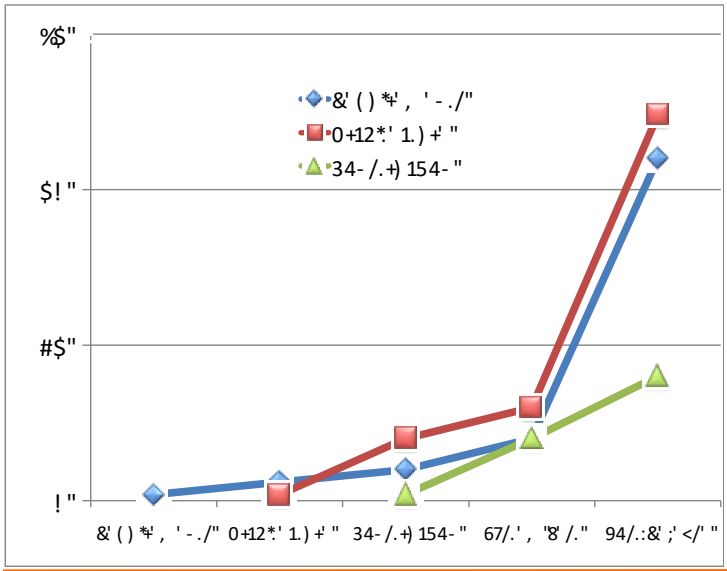
Cost of detection delay

Design and architecture	Implementation	Integration testing	Customer beta test	Postproduct release
1X*	5X	10X	15X	30X

*X is a normalized unit of cost and can be expressed in terms of person-hours, dollars, etc.
Source: National Institute of Standards and Technology (NIST)†

By catching defects as early as possible in the development cycle, you can significantly reduce your development costs.

Cost of detection delay



Basic goals of VV

- Minimize number of defects inserted
  - Cannot be zero due to inherent complexity of software
- Maximize number of defects discovered and removed
  - Cannot prove 100% is achieved
- Minimize detection delay

V&V approaches

- Static
  - inspections
  - source code analysis
- Dynamic
  - testing

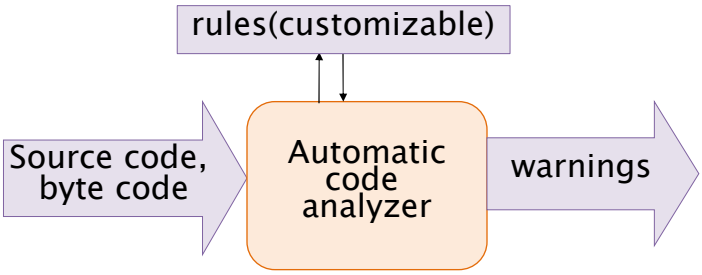
STATIC ANALYSIS

Static analysis techniques

- Compilation static analysis
- Control flow analysis
- Data flow analysis
- Symbolic execution
- Inspections

Automatic code analysis

- It is performed
  - without actually executing programs (at compile time)
  - On source code, or byte code



Code smells

- A code smell is a surface indication that usually corresponds to a deeper problem in the system
- Smells are certain structures in the code that indicate violation of fundamental design principles and negatively impact design quality

Fowler et al., Refactoring, Improving quality of existing code. Addison-Wesley

Technical Debt

- Technical debt reflects the extra development work that arises when code that is easy to implement in the short run is used instead of applying the best overall solution

Technical Debt

- "Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite... The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt."

  [W.Cunningham]

Tools

- Static Analysis
  - SonarQube
  - Cast

TESTING

Definition

The process of

operating a system or component

under specified conditions

observing and recording the results

to detect the differences between actual and expected behavior (i.e. failures)

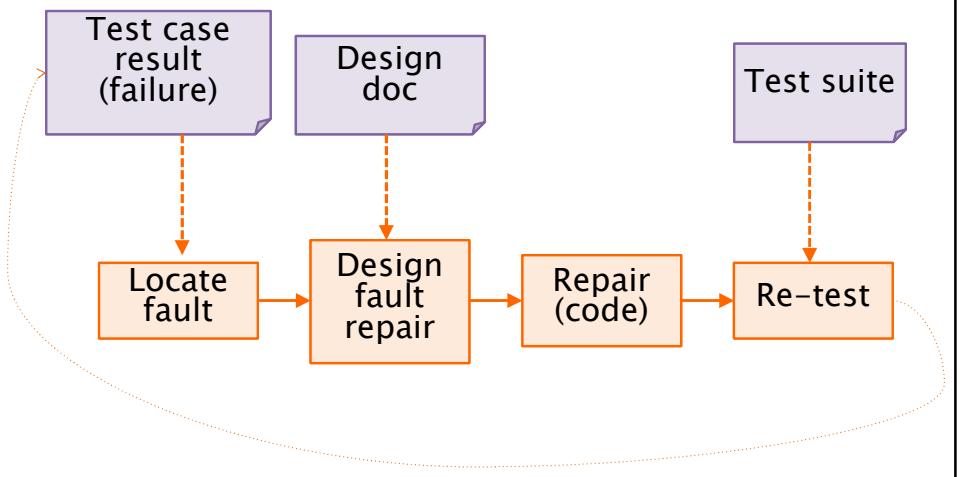
Purpose of test

- The purpose of testing process is to find defects in software products
  - A test process is successful if it is able to detect failures

Testing vs. debugging

- Defect testing and debugging are different activities
  - Often performed by different roles in different times
- Testing tries to detect failures
- Debugging searches for the location of the relative faults and removes them

Debugging



Test case

- A given stimulus applied to executable (system or unit), consists in
  - name
  - input (or sequence of –)
  - expected output
- With defined constraints/context
  - E.g. version and type of OS, DBMS, GUI ..
- Test suite = set of related test cases

Good test case

- Reasonable chance of catching failure
- Does interesting things
- Doesn't do unnecessary things
- Neither too simple nor too complex
- Non redundant w.r.t. other tests
- Makes failures obvious
- Mutually Exclusive
- Collectively Exhaustive

Test case log

- Test case reference

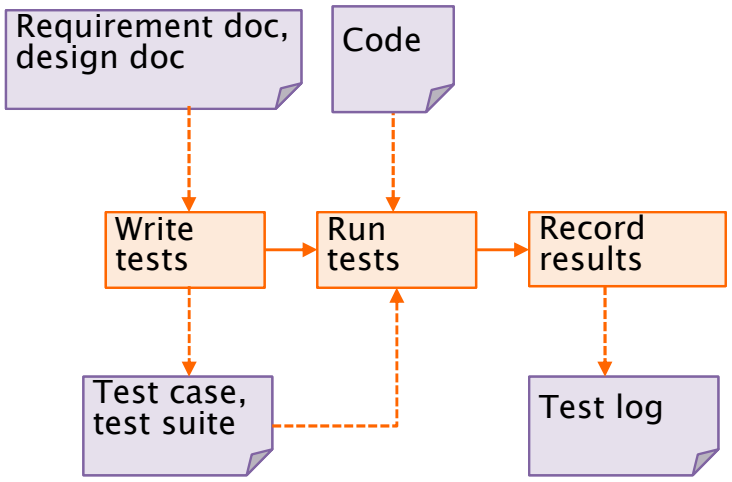
  +

- Time and date of application
- Actual output
- Result (pass / no pass)

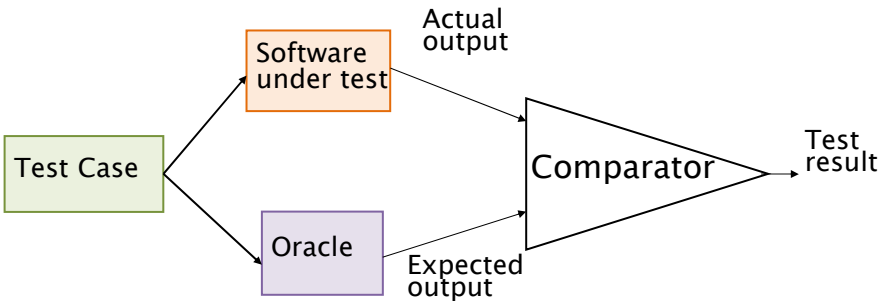
Test cases examples

- Function **`add(int x, int y)`**
- Test case:
  - T1: (1,1; 2)
  - T2: (3,5; 8)
- Test suite
  - TS1: {T1, T2}
- Test log
  - T1, 16-3-2018 9:31, result 2, success
  - T2, 16-3-2018 9:32, result 9, fail

Test activities



Oracle



Oracle

- The ideal condition would be to have an automatic oracle and an automatic comparator
  - The former is very difficult to have
  - The latter is available only in some cases
- A human oracle is subject to errors
- The oracle is based on the program specifications (which can be wrong)

Oracle

- Necessary condition to perform testing:
  - Know the expected behavior of a program for a given test case (oracle)
- Human oracle
  - Based on req. specification or judgment
- Automatic oracle
  - Generated from (formal) req. specification
  - Same software developed by other parties
  - Previous version of the program (regression)

Software peculiarities

- No ageing
  - If function sum(2,3) works, it works forever
    - Supporting microprocessor will eventually fail for age, not the software
- Not linear, not continuous
  - If sum(2,3) works, may be sum(2,4) does not

Exhaustive test

- function: $Y = A + B$
- A and B integers, 32 bit

- Total number of test cases:
  $2^{32} * 2^{32} = 2^{64} \approx 10^{20}$
- 1 ns/test $\Rightarrow$ ~ 3171 years

Exhaustive test

- Exhaustive test is impossible
- Goal of test is finding defects, not demonstrating that systems is defect free
- Final objective of test (and VV in general) is assuring a *good enough* level of quality, confidence in sw

Dijkstra thesis

- *Testing can only reveal the presence of errors, never their absence*

E. W. Dijkstra. Notes on Structured Programming.
In *Structured Programming,* O.-J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, Eds. Academic, New York, 1972, pp. 1–81.

Test classification

- Per phase/granularity level
  - Unit, integration, system
  - Regression
- Per approach
  - Black box (functional)
  - White box (structural)
  - Reliability assessment/prediction
  - Risk based (safety security)

Test per granularity level/phase

- Unit tests
  - Individual modules
- Integration tests
  - Modules when working together
- System tests
  - The system as a whole (usable system)
- Acceptance tests
  - The system by customer

Unit test

- Black box (functional)
  - Random
  - Equivalence classes partitioning
  - Boundary conditions
- White Box (structural)
  - Coverage of structural elements
    - Statement
    - Decision, condition (simple, multiple)
    - Path
    - Loop

Integration test

- Add one unit at a time, test the partial aggregate
  - Defects found, most likely, come by last unit/interaction added
  - Check dependencies on
    - external services
    - Third party components

Stub, driver

- Driver
  - Unit (function or class) developed to pilot another unit
- Stub
  - Unit developed to substitute another unit (fake unit)
- Also called mockups

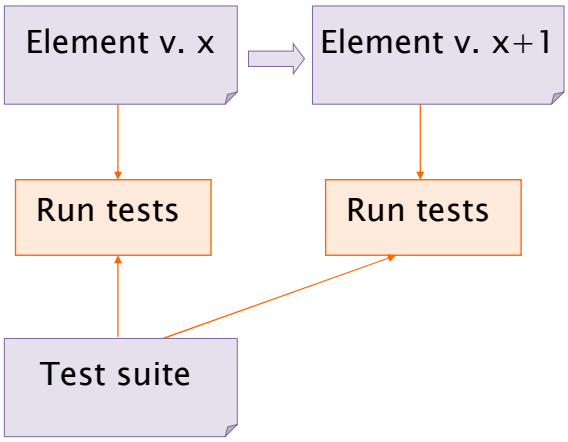
System test

- Is applied to the software system as a whole
  - Aims at verifying the correspondence of the system to the requirements
- Test of functional requirements
  - Coverage of uses cases/scenarios as listed in requirement document
  - End-to-end testing: follow a real use case from input to final result
- Test in conditions as far as possible close to working conditions

Regression testing

- Regression testing
  - Tests previously defined are repeated after a change
  - To assure that the change has not introduced defects
    - Time0
      - Element (unit, system) in v0, test set t0 is defined and applied, all tests pass
    - Time1
      - Element is changed to v1
      - Test set t0 is re-applied, do all tests still pass?

Regression testing



References and Further Readings

- IEEE Std 829-2008: IEEE Standard for Software and System Test Documentation
- ISTQB, Certified Tester Foundation Level Syllabus, 2001
  - http://www.istqb.org/downloads/send/2-foundation-level-documents/3-foundation-level-syllabus-2011.html4
- Fowler et al., Refactoring, Improving quality of existing code. Addison-Wesley
- E. W. Dijkstra. Notes on Structured Programming. In Structured Programming, O.-J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, Eds. Academic, New York, 1972, pp. 1-81.