Verification

1. Verification
   1. All changes in the code have to be verified
      1. Refactoring
      2. Actualization
   2. Essential difficulties
      1. Programmers very often produce imperfect work
      2. Defects are called “bugs”
   3. Verification finds bugs
2. Verifications techniques
   1. Many techniques of software verification have been researched and proposed
   2. Current practice
      1. Testing
      2. Code inspection
3. Testing
   1. Tests execute the program or its parts
      1. Specific input to the execution
      2. Compare the outputs of the execution with the previously specified output
      3. Report if there is a deviation
   2. Tests are usually organized into a test suite
      1. Several, often many, test
4. Incompleteness of testing
   1. Testing can demonstrate the presence of the bugs, but not their absence
   2. No matter how much testing has been done, residual bugs still can hide in the code
      1. They have not been reached and revealed by any test
      2. No test suite can guarantee that the program runs without error
5. Turing’s Halting problem
   1. Theoretical reason for testing incompleteness
   2. It is theoretically impossible to create a perfect test suite
   3. The programmers have been trying to do the best under the circumstances
      1. Techniques of the testing cannot guarantee a complete correctness of software
      2. Well-designed tests come close to be adequate
6. New vs. old code tests
   1. Tests of the new code
      1. New tests must be written with the new code
   2. Testing of the old code
      1. The tests make sure the old code is not broken by the change
         1. Regression test
         2. Prevent regression of what was already functioning in the software
            1. Merriam Webster: “regression” = “a trend or shift toward a lower or less perfect state”
   3. Variety of software testing
      1. Setting
         1. Programmer’s workspace
         2. Team’s configuration management
      2. Strategy
         1. Structural
         2. Unit
         3. Functional
      3. Functionality
         1. Old (regression testing)
         2. New
         3. Combined
7. Acceptance tests
   1. Final functional test
      1. Both the new and the old
   2. Done during the phase of change conclusion
      1. Test the complete functionality of the software
      2. Software stakeholders are able to asses the progress of the software project
8. Composition of the test suite
   1. Unit test
      1. Test for a specific class
   2. Functional test
      1. Test a specific functionality of the whole program
         1. User manual or graphics user interface guide the creation of the functional test
         2. All features that are available to the user should be tested
   3. Structural tests
9. Harness (scaffolding)
   1. Test drivers
      1. Implement the support for the test
   2. Stub
      1. Implement the replacement for missing classes and subsystem
   3. Environment simulation
   4. Harness = drivers + stubs + simulators
      1. Production code vs. harness
      2. Developers vs. testers
10. Harness and production code
    1. Production code goes to the user
    2. Harness stays within the programming group
    3. Parallel evolution of both
11. Coverage
    1. We cannot guarantee a complete correctness of the code by testing
    2. We are going to guarantee that each unit of the tested code is executed at least once
       1. This guarantees that at least some of the bugs are discovered
          1. In particular, the bugs that are brought up by this single execution of the unit
12. Granularity of method
    1. calSubTotal(), getPrice(), setPrice()
       1. each of them executed at least once
    2. Seems like a crude approach
       1. Systematic
       2. Guarantees correctness better than random selection of tests
13. Statement coverage
    1. Guarantees that every statement of the program is executes at least once
    2. Minimal test suite does not have any redundant test
       1. Test that cover only statements that are already covered by the other test
    3. Minimal test suite efficiently accomplishes the coverage
       1. Preferred approach
14. Minimal complete test suite
    1. Easy to write the first test
       1. No matter what it covers, it adds to the test suite coverage
    2. Increased number of tests
       1. The statement not covered are fewer
       2. It is increasingly hard to aim new tests at these remaining uncovered statements
       3. Increased coverage is not economic
    3. 70% coverage is considered to be good
15. Unit Testing
    1. Testing of individual modules
       1. Testing classes and methods
    2. Testing of composite functionality
       1. The class is being tested together will all classes that support it
    3. Testing local functionality
16. Testing local responsibility
    1. Driver + Stub
       1. Stub simulates suppliers
          1. Part of harness
    2. Reason: supplier classes
       1. Are not available
       2. Have not been tested
          1. Confidence in them is low
       3. Support a limited contract (limited precondition)
          1. The tested class planned for a wider use with other suppliers
17. Stubbing techniques
    1. Less effective algorithm
       1. Easier to implement
       2. Test becomes less efficient
          1. Developers do testing, acceptable impact
    2. Limited precondition of the stub
       1. Simplifies the code of the stub substantially
          1. Covert the data into a day of the week
             1. The stub does that only for a selected
          2. Inappropriate if the stubbing is to broaden the contract
    3. User intervention
       1. Interrupts the test, the user provides the correct answer
          1. Practical only in if the stub is executed only few times during test
          2. Human user may input incorrect values
    4. Replacement contract
       1. Quick but incorrect post-condition
       2. The most controversial stubbing technique
       3. Still may provide valuable result
18. Functional testing
    1. Tests the functionality of the complete program
    2. Program with GUI: every function
    3. Coverage
       1. Percentage of the requirements tested
19. Regression testing
    1. After a change programmer retest the code
       1. Re-establish the confidence that the former functionalities of the software still work
       2. Change may have inadvertently introduced stray bugs into the intact part
    2. Tests from the past constitute the bulk of the regression test suite
       1. Test suite often grows
       2. Testing is done overnight
20. Test suite evolution
    1. System test --> regression test by delete obsolete tests
    2. Regression test 🡪 system test by adding tests of new functionality
21. Obsolete tests
    1. Broken test cases that cannot run
       1. They do not interface with the software anymore
    2. Tests that do not fulfill their purpose
       1. A test case testing the limits of a range becomes obsolete when the range is changed
    3. Tests that no longer provide a deterministic result
       1. A test case which may now impacted by the mouse
22. Code inspection
    1. Somebody else than the author reads the code
       1. Checks its correctness
       2. Report bugs
    2. Code inspection does not require execution of a system
       1. Can be applied to incomplete code or to other artifacts
          1. Models, documentation
23. Effectiveness of code inspection
    1. Habituaiton
       1. People become blind to their own mistakes
    2. After reading the code several times
       1. Programmers no longer read the code
          1. Recall from the memory what the code should contain
          2. Some errors repeatedly escape their attention
       2. A different reader spots these errors easily and right away
24. Inspections and testing are complementary
    1. Some bugs are easily found by testing
       1. They appear in each test
          1. They are sometimes hard to spot by human
          2. E.g. misspellings of long identifier
    2. Some bugs are hard to find by testing
       1. It is hard to create a test that finds them
          1. Human readers can find them easily
             1. Division by 0
          2. To create a test that causes such situation can be hard
25. Inspection of different artifacts
    1. Inspections can also check whether different artifacts agree with each other
       1. Change request corresponded
       2. UML model
    2. Inspection cannot check non-functional characteristics
       1. Performance and usability
26. Walkthroughs
    1. Structured inspection process
    2. Walkthrough team
       1. At least four members
          1. The author of the inspected code
          2. A moderator
          3. At least two code reviewers
27. Walkthrough process
    1. Preparation
       1. The code or other work products are distributed to inspection team
       2. One participant is selected to inspects the document thoroughly
    2. Walkthrough meeting
       1. All member participate
       2. Whole group walks through the document under direction of the reader
       3. The reader notes the error, omissions and inconsistencies in the code
       4. The moderator chairs the meeting
          1. Note the error
          2. Produces a report