Chapter 6

Testing

(slides from Stephen R. Schach, McGraw Hil)

1

Testing

- "V & V"
  - *Verification*
    - Determine if the workflow was completed correctly
  - *Validation*
    - Determine if the product as a whole satisfies its requirements
    - i.e. is it right
- There are two basic types of testing
  - Execution-based testing
  - Non-execution-based testing

2

6.1 Quality

- Extent to which product satisfies requirements/specifications.
- Software professionals built this in
  - not “added” after by sqa.

- Fault is based on human error.
- Failure is the observed incorrect behavior
- Error is the amount which the result is incorrect

3

6.1.1 Software Quality Assurance

- The members of the SQA group must ensure that the developers are doing high-quality work
  - At the end of each workflow
  - When the product is complete

- In addition, quality assurance must be applied to
  - The process itself
    - Example: Standards

4

6.1.2 Managerial Independence

- There must be managerial independence between
  - The development group
  - The SQA group
- Neither group should have power over the other

5

Managerial Independence (contd)

- More senior management must decide whether to
  - **Deliver the product on time but with faults,** or
  - Test further and deliver the product late
- The decision must take into account the interests of the client and the development organization

6

6.2 Non-Execution Based Testing

- Review review review!
- Underlying principles
  - We should ~~not~~ review our own work
  - We should not be the only one
  - Group synergy

7

6.2.1 Walkthroughs (”informal”)

- A walkthrough team consists of from four to six members
- It includes representatives of
  - The team responsible for the current workflow
  - The team responsible for the next workflow
  - The SQA group
- The walkthrough is preceded by preparation
  - Lists of items
    - Items not understood
    - Items that appear to be incorrect

8

6.2.2 Managing Walkthroughs

- The walkthrough team is chaired by the SQA representative
- In a walkthrough we detect faults, not correct them
  - A correction produced by a committee is likely to be of low quality
  - The cost of a committee correction is too high
  - Not all items flagged are actually incorrect
  - A walkthrough should not last longer than 2 hours
  - There is no time to correct faults as well

9

Managing Walkthroughs

- A walkthrough must be document-driven, rather than participant-driven
- Verbalization leads to fault finding
- A walkthrough should never be used for performance appraisal

10

6.2.3 Inspections (“more formal”)

- An inspection has five formal steps
  - Overview
  - Preparation, aided by statistics of fault types
  - Inspection
  - Rework
  - Follow-up

11

Inspections

- An inspection team has four members
  - Moderator
  - A member of the team performing the current workflow
  - A member of the team performing the next workflow
  - A member of the SQA group
- Special roles are played by the
  - Moderator
  - Reader
  - Recorder

12

Fault Statistics (page 160-161)

- Faults are recorded by severity
  - Example:
    - Major or minor
- Faults are recorded by fault type
  - Examples of design faults:
    - Not all specification items have been addressed
    - Actual and formal arguments do not correspond

13

Fault Statistics (contd)

- For a given workflow, we compare current fault rates with those of previous products
- We take action if there are a disproportionate number of faults in an artifact
  - Redesigning from scratch is a good alternative
- We carry forward fault statistics to the next workflow
  - We may not detect all faults of a particular type in the current inspection

14

Statistics on Inspections

- IBM inspections showed up
  - 82% of all detected faults (1976)
  - 70% of all detected faults (1978)
  - 93% of all detected faults (1986)
- Switching system
  - 90% decrease in the cost of detecting faults (1986)
- JPL
  - Four major faults, 14 minor faults per 2 hours (1990)
  - Savings of $25,000 *per inspection*
  - The number of faults decreased exponentially by phase (1992)

15

Statistics on Inspections

- Warning
- Fault statistics should never be used for performance appraisal
  - "Killing the goose that lays the golden eggs"

16

6.2.4 Comparison of Inspections and Walkthroughs (KNOW THE DIFFERENCE)

- Walkthrough
  - Two-step, informal process
    - Preparation
    - Analysis
- Inspection
  - Five-step, formal process
    - Overview
    - Preparation
    - Inspection
    - Rework
    - Follow-up

17

6.2.5 Strengths and Weaknesses of Reviews

- Reviews can be effective
  - Faults are detected early in the process
- Reviews are less effective if the process is inadequate
  - Large-scale software should consist of smaller, largely independent pieces
  - **The documentation of the previous workflows has to be complete and available online**

18

6.2.6 Metrics for Inspections

- Inspection rate (e.g., design pages inspected per hour)
- Fault density (e.g., faults per KLOC inspected)
- Fault detection rate (e.g., faults detected per hour)
- Fault detection efficiency (e.g., number of major, minor faults detected per hour)
- Why is this important??

19

Metrics for Inspections

- So what do you do if you see a 50% increase in the fault detection rate?

20

6.3 Execution-Based Testing

- Organizations spend up to 50% of their software budget on testing
  - But delivered software is frequently unreliable
- Dijkstra (1972)
  - "Program testing can be a very effective way to show the presence of bugs, **but it is hopelessly inadequate for showing their absence**"

21

6.4 What Should Be Tested?

- Definition of *execution-based testing*
  - "The process of inferring certain behavioral properties of the product based, in part, on the results of executing the product in a known environment with selected inputs"
- This definition has troubling implications

22

6.4 What Should Be Tested?

- "Inference"
  - We have a fault report, the source code, and — often — nothing else
- "Known environment"
  - We never can really know our environment
- "Selected inputs"
  - Sometimes we cannot provide the inputs we want
  - Simulation is needed

23

6.4 What Should Be Tested?

- "Inference"
  - We have a fault report, the source code, and — often — nothing else
- "Known environment"
  - We never can really know our environment
- "Selected inputs"
  - Sometimes we cannot provide the inputs we want
  - Simulation is needed **[AND DON'T FORGET BOUNDARY VALUES]**

24

6.4 What Should Be Tested?

- We need to test correctness (of course), and also
  - Utility
  - Reliability
  - Robustness, and
  - Performance

25

6.4.1 Utility

- The extent to which the product meets the user's needs
  - Examples:
    - Ease of use
    - Useful functions
    - Cost effectiveness

26

6.4.2 Reliability

- A measure of the frequency and criticality of failure
  - Mean time between failures
  - Mean time to repair
  - Time (and cost) to repair the *results* of a failure

27

6.4.3 Robustness

- A function of
  - The range of operating conditions
  - The possibility of unacceptable results with valid input
  - The effect of invalid input

28

6.4.4 Performance

- The extent to which space and time constraints are met
- Real-time software is characterized by *hard* real-time constraints
- If data are lost because the system is too slow
  - There is no way to recover those data

29

6.4.5 Correctness

- A product is correct if it satisfies its specifications

30

Correctness of specifications

- Incorrect specification for a sort:

Input specification:	p : array of n integers, $n > 0$.
Output specification:	q : array of n integers such that $q[0] \leq q[1] \leq \cdots \leq q[n-1]$

Figure 6.1

- Function `trickSort` which satisfies this specification:

```
void trickSort (int p[ ], int q[ ])
{
    int i;
    for (i = 0; i < n; i++)
        q[i] = 0;
}
```

Figure 6.2

31

Correctness of specifications

- Incorrect specification for a sort:

Input specification:	p : array of n integers, $n > 0$.
Output specification:	q : array of n integers such that $q[0] \leq q[1] \leq \cdots \leq q[n-1]$

Figure 6.1 (again)

- Corrected specification for the sort:

Input specification:	p : array of n integers, $n > 0$.
Output specification:	q : array of n integers such that $q[0] \leq q[1] \leq \cdots \leq q[n-1]$ The elements of array q are a permutation of the elements of array p, which are unchanged.

32

Correctness

- Technically, correctness is
- A ~~Product~~ coding artifact is correct if it satisfies its output specifications, independent of its use of computing resources, when operating under permitted conditions. (page 166)

- *Not* necessary
  - Example: C++ compiler (bad example, don't use that compiler)
- *Not* sufficient
  - Example: `trickSort`

33

6.5 Testing versus Correctness Proofs

- A correctness proof is an alternative to execution-based testing

34

6.5.3 Correctness Proofs and Software Engineering

- Three myths of correctness proving (see over)

35

Three Myths of Correctness Proving

- Software engineers do not have enough mathematics for proofs
  - Most computer science majors either know or can learn the mathematics needed for proofs
- Proving is too expensive to be practical
  - Economic viability is determined from cost–benefit analysis
- Proving is too hard
  - Many nontrivial products have been successfully proven
  - Tools like theorem provers can assist us

36

Difficulties with Correctness Proving

- How do we find input–output specifications, loop invariants?
- What if the specifications are wrong?
- We can never be sure that specifications or a verification system are correct

37

Correctness Proofs and Software Engineering (contd)

- Correctness proofs are a vital software engineering tool, *where appropriate*:
  - **When human lives are at stake**
  - When indicated by cost–benefit analysis
  - When the risk of not proving is too great
- Also, informal proofs can improve software quality
  - Use the `assert` statement
- Model checking is a new technology that may eventually take the place of correctness proving (Section 18.11)

38

6.6 Who Should Perform Execution-Based Testing?

- Programming is *con*structive
- Testing is *de*structive
  - A successful test finds a fault
- So, programmers should not test their own code artifacts (should not be the only ones)

39

Who Should Perform Execution-Based Testing? (contd)

- Solution:
  - The programmer does informal (execution-based)testing
  - The SQA group then does systematic testing
  - The programmer debugs the module
- All test cases must be
  - Planned beforehand, including planned input and the expected output, and
  - Retained afterwards
  - Expanded as bugs found

40

6.7 When Testing Stops

- Only when the product has been irrevocably discarded

41

Extra Slide – Reminder (IMPORTANT)

- Black box testing is?
- Can we use ”black box” testing techniques for unit tests?
- Can we use ”black box” testing techniques for functional tests?
- Can we use ”black box” testing techniques for regression tests?
- Can we use ”black box” testing techniques for any kind of testing ever ever ever ever ever ever ever invented??

42

Extra Slide – Reminder (IMPORTANT)

- Black box testing is? Testing to design (or specification)
- Can we use ”black box” testing techniques for unit tests? YES
- Can we use ”black box” testing techniques for functional tests? YES
- Can we use ”black box” testing techniques for regression tests? YES
- Can we use ”black box” testing techniques for any kind of testing ever ever ever ever ever ever ever invented?? YES

43

Extra Slide (IMPORTANT)

- White (glass) box testing is?
- Can we use ”white box” testing techniques for unit tests?
- Can we use ”white box” testing techniques for functional tests?
- Can we use ”white box” testing techniques for regression tests?
- Can we use ”white box” testing techniques for any kind of testing ever ever ever ever ever ever ever ever invented??

44

Extra Slide (IMPORTANT)

- White (glass) box testing is? Testing to code (seeing the code)
- Can we use ”white box” testing techniques for unit tests? Yes
- Can we use ”white box” testing techniques for functional tests? Yes
- Can we use ”white box” testing techniques for regression tests? Yes
- Can we use ”white box” testing techniques for any kind of testing ever ever ever ever ever ever ever ever ever invented?? Yes

45

- End of chapter 6!

46