Validation and Verification

Inspections [24.3]

Testing overview [8, 15.2]
- system testing

The V-model

Validation

- Will the product satisfy the customer needs?
- Are we building the right product?

Verification

- Do we satisfy the requirements?
- Are we building the product right?

How much V&V is enough?

- Good enough
- Safe enough
- Competition

- Testing itself does not improve quality
- Costly (maybe impossible!) to increase quality after development

Testing vs. Debugging

The debugging process
Fixing 1 error could create > 1 new error

Testing vs. Inspections

- dynamic: running a program
- static: reviewing a document
Planning

• Determine goals
• Proportion static/dynamic V&V
• Ensure verifiability (req’s, design, code)
• Design tests
• Evaluation criteria - what is good enough?
• Tool support
• Time plan
• Documentation

Inspections [24.3]

+ Apply to all documents:
  no program needed
+ Quality perspective from the start
  - Do not cover emergent properties:
    mostly applies to verification
  - Added cost early in the process =
    + Investment in quality

Inspection goals

• Finding errors
• Checking adherence to standards
• Readability (code, documentation)
• Collecting data
  – Common errors

Inspection pitfalls

• Questioning overall design
  – “This is OK, but I can do better”
• Designing repair during inspection
  – “This is not OK, and I can do better”
• Evaluating people
• Inspection preconditions are not fulfilled

Inspection preconditions

• Precise specification: criteria OK/not OK
• Standards known to team members
• A finished item for inspection
• For code inspections [24.3.2]:
  – Syntactically correct code
  – Checklist of common errors

Figure 24.8. An inspection checklist

<table>
<thead>
<tr>
<th>Fault class</th>
<th>Inspection check</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data faults</td>
<td>Are all program variables initialized before their values are used?</td>
</tr>
<tr>
<td></td>
<td>Have all constants been named?</td>
</tr>
<tr>
<td></td>
<td>Should the upper bound of arrays be equal to the size of the array or Size-1?</td>
</tr>
<tr>
<td></td>
<td>If character strings are used, is a delimiter explicitly assigned?</td>
</tr>
<tr>
<td>Control faults</td>
<td>For each conditional statement, is the condition correct?</td>
</tr>
<tr>
<td></td>
<td>Does each loop get terminated?</td>
</tr>
<tr>
<td></td>
<td>Are compound statements correctly bracketed?</td>
</tr>
<tr>
<td></td>
<td>In case statements, are all possible cases covered?</td>
</tr>
<tr>
<td></td>
<td>Is a case statement differ each case in case statements, has it been included?</td>
</tr>
<tr>
<td>Input/output faults</td>
<td>Are all input variables used?</td>
</tr>
<tr>
<td></td>
<td>Are all output variables assigned a value before they are output?</td>
</tr>
<tr>
<td></td>
<td>Can unexpected inputs cause corruption?</td>
</tr>
<tr>
<td>Interface faults</td>
<td>Do all function and method calls have the correct number of parameters?</td>
</tr>
<tr>
<td></td>
<td>Do formal and actual parameter types match?</td>
</tr>
<tr>
<td></td>
<td>Are the parameters in the right order?</td>
</tr>
<tr>
<td></td>
<td>Are all function calls with shared memory, do they have the same model of the</td>
</tr>
<tr>
<td></td>
<td>shared memory structure?</td>
</tr>
<tr>
<td>Storage management faults</td>
<td>If a linked structure is modified, have all links been correctly reassembled?</td>
</tr>
<tr>
<td></td>
<td>If dynamic storage is used, has space been allocated correctly?</td>
</tr>
<tr>
<td></td>
<td>Is space explicitly deallocated after it is no longer required?</td>
</tr>
<tr>
<td>Exception management faults</td>
<td>Have all possible error conditions been taken into account?</td>
</tr>
</tbody>
</table>
Automated Static Analysis [15.1.3]

```c
#include <stdio.h>

int Anarray;

main()
{
    printf("%d", Anarray);
}
```

<table>
<thead>
<tr>
<th>LINT warnings</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10) c may be used before set</td>
</tr>
<tr>
<td>(10) i may be used before set</td>
</tr>
<tr>
<td>printf: variable # of args: (4) != (10)</td>
</tr>
<tr>
<td>printf: arg.1 used inconsistently (4) != (10)</td>
</tr>
<tr>
<td>printf returns value which is always ignored.</td>
</tr>
</tbody>
</table>

Inspection process

- Planning
- Group preparation
- Orientation
- Finding suspected deviations
- Short meeting
- Error correction
- Improvement
- Group preparation
- Post-review activities

Inspection meeting roles

- Chair (organize)
- Scribe (taking notes)
- Author (fixing ... after meeting)
  - Reader (walk through)
- Inspector(s) (find deviations)
  - viewpoints

Inspections require training!

What is a test?

- A test suite is a set of test cases run together for a single purpose.
- A test case consists of
  - Test data
  - Expected outcome (correct answer)
  - Expected behaviour (e.g. response time)

The oracle problem

What is the correct answer?
1. … at least the program didn’t crash ...
2. Compute by hand and compare
3. Back-to-back testing
4. The answer is "reasonable"
   - Is the list sorted?
   - Is the yellow ball yellow and round?
   - Is the area of the triangle between … and …

Classification of testing

- Classification by goal:
  - finding defects
  - acceptance / validation
  - measurement: reliability, performance, ...
- Classification by level
  - system
  - subsystem
  - module
Acceptance test (system)

- **Factory acceptance test (FAT)**
- **Site acceptance test (SAT)**

Goals:
- is the contract fulfilled? (verification)
- is the product usable? (validation)

Reliability testing

- Requires test data reflecting "normal" operation
- Statistical test [15.2]
  - "random" test

Establish the operational profile.
- from an existing system
- assumptions about use of new system

Construct test data reflecting the operational profile (statistically).

Test: observe the number of failures and the times of these failures.

Compute the reliability after a statistically significant number of failures.

Problems

- Operational profile uncertainty
  - operational profile = real use of the system?
- High costs of test data generation
  - if test data not generated automatically.
- Statistical uncertainty
  - highly reliable systems will rarely fail.
- Recognizing failure
  - conflicting interpretations of a specification.

Performance

- Stress test
  How the system handles increasing / extreme load
  - graceful degrading / total collapse
  - may reveal defects
  - Profiling
    10% of the code takes 90% of the time

Integration / Interface testing

- Top-down vs. Bottom-up
- Needs scaffolding
  stubs for unfinished parts.
- Test for
  - Miscommunication (arguments, ...)
  - Timing (mutex, deadlock)
  - Environmental assumptions
    (available services, memory, etc.)
Reusability

• Back-to-back testing
  – use a previous version of the system (prototype) as the test oracle
• Regression test
  – applies for all kinds of test
  – rerun a test suite for every change in the system
  – goal: did the change break anything?

Test tools

• Automated testing
  – Record, Replay
• Test environment, scaffolding
• Large test suites (stress, statistical test)
• Evaluation
  – Profiling
  – Coverage
• Documentation, traceability