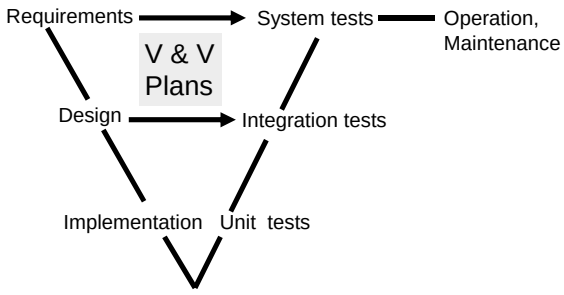
Validation and Verification

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

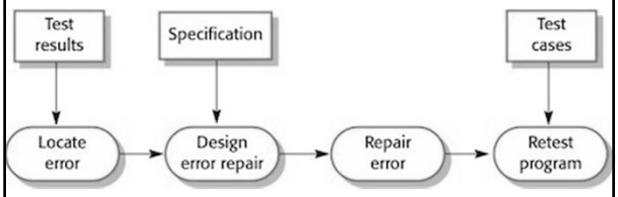
- Do we satisfy the requirements?
- Are we building the product right?

How much V&V is enough?

- Good enough
- Safe enough
- Competition

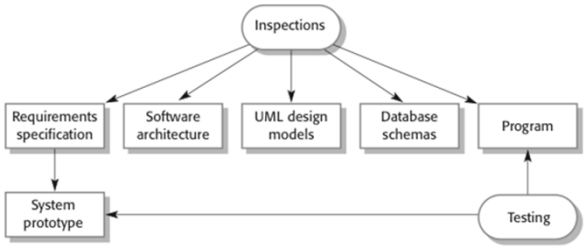
- Testing itself does not improve quality
- Costly (maybe impossible!) to increase quality after development

Testing vs. Debugging



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

Testing vs. Inspections



Testing – dynamic: running a program
Inspections – 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 fullfilled

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

Fault class	Inspection check
Data faults	• Are all program variables initialized before their values are used? • Have all constants been named? • Should the upper bound of arrays be equal to the size of the array or Size -1? • If character strings are used, is a delimiter explicitly assigned? • Is there any possibility of buffer overflow?
Control faults	• For each conditional statement, is the condition correct? • Is each loop certain to terminate? • Are compound statements correctly bracketed? • In case statements, are all possible cases accounted for? • If a break is required after each case in case statements, has it been included?
Input/output faults	• Are all input variables used? • Are all output variables assigned a value before they are output? • Can unexpected inputs cause corruption?
Interface faults	• Do all function and method calls have the correct number of parameters? • Do formal and actual parameter types match? • Are the parameters in the right order? • If components access shared memory, do they have the same model of the shared memory structure?
Storage management faults	• If a linked structure is modified, have all links been correctly reassigned? • If dynamic storage is used, has space been allocated correctly? • Is space explicitly deallocated after it is no longer required?
Exception management faults	• Have all possible error conditions been taken into account?

Automated Static Analysis [15.1.3]

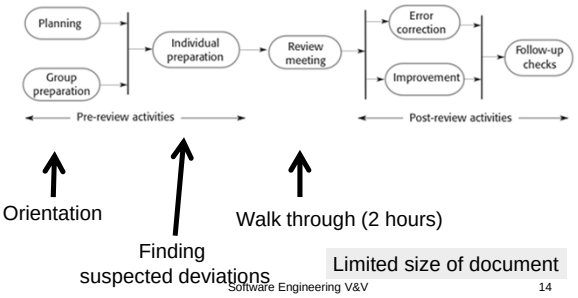
awful, correct C

```
#include <stdio.h>
printarray(Anarray)
  int Anarray;
{
 printf(”%d”,Anarray);
}
main()
{
 int Anarray[5]; int i; char c;
 printarray(Anarray, i, c);
 printarray(Anarray);
}
```

LINT warnings

```
(10) c may be used before set

(10) i may be used before set

printarray: variable # of
args. (4) :: (10)

printarray: arg.1 used
inconsistently (4) :: (10)

printf returns value which is
always ignored.
```


Inspection process



Orientation

Finding suspected deviations

Walk through (2 hours)

Limited size of document

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)*

Installation

- *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 collaps
  - 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

Defect testing

Goals:

- detect as many defects as possible
- detect the most damaging defects
- detect the most likely defects - statistical test!

Black-box testing: the source code is not considered (maybe even not known).

Glass-box testing: the tests are chosen based on the source code.