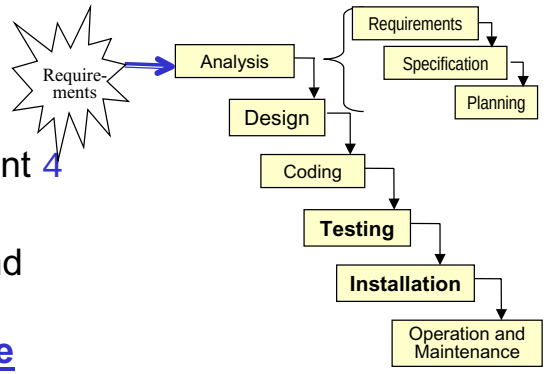
Contents



Quality Assurance

- Introduction
- Testing
  - Testing Phases and Approaches
  - Black-box Testing
  - White-box Testing
  - System Testing

What is Quality Assurance?

QA is the combination of planned and unplanned activities to ensure the fulfillment of predefined quality standards.

- Constructive vs analytic approaches to QA
- Qualitative vs quantitative quality standards
- Measurement
  - Derive qualitative factors from measurable quantitative factors
    - Software Metrics

Approaches to QA

- Constructive Approaches

  Usage of methods, languages, and tools that ensure the fulfillment of some quality factors.
  - Syntax-directed editors
  - Type systems
  - Transformational programming
  - Coding guidelines
  - ...
- Analytic approaches

  Usage of methods, languages, and tools to observe the current quality level.
  - Inspections
  - Static analysis tools (e.g. *lint*)
  - Testing
  - ...

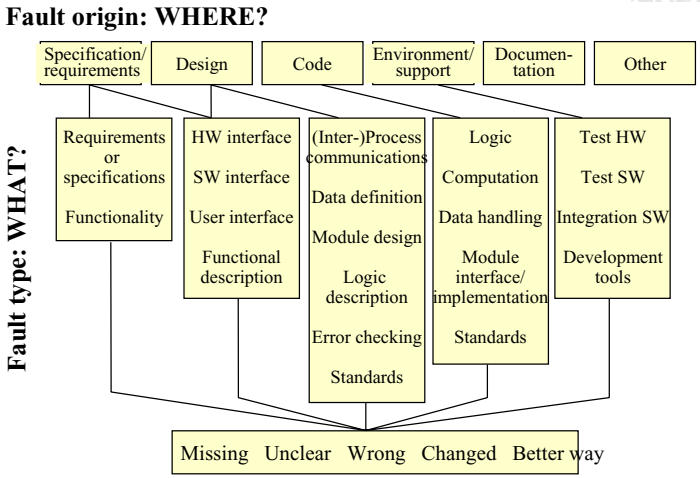
Fault vs Failure

can lead to → can lead to → ?!

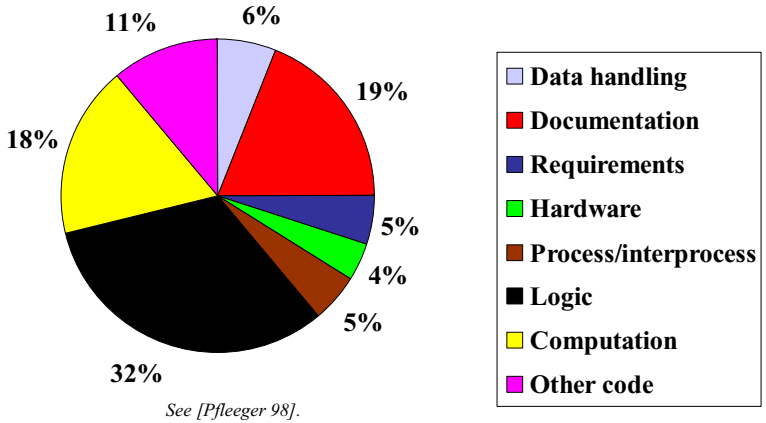
human error **fault** **failure**

- Different types of faults
  - Different identification techniques
  - Different testing techniques
- Fault prevention and -detection strategies should be based on expected fault profile

HP´s Fault Classification

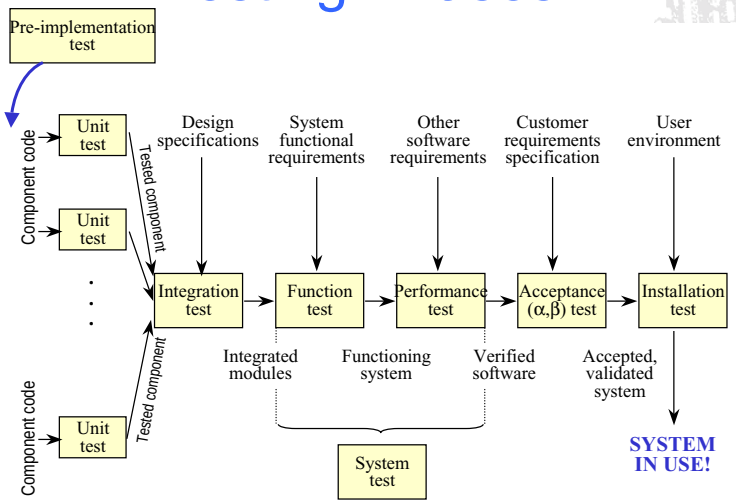


Fault Profile of a HP Division



See [Pfleeger 98].

Testing Phases



Pre-Implementation Testing

- Inspections
  - evaluation of documents and code prior to technical review or testing
- Walkthrough
  - In teams
  - Examine source code/detailed design
- Reviews
  - More informal
  - Often done by document owners
- Advantages
  - Effective
  - High learning effect
  - Distributing system knowledge
- Disadvantages
  - Expensive

Testing vs "Proofing" Correctness

- Verification
  - Check the design/code against the requirements
    - Are we building the product right?
- Validation
  - Check the product against the expectations of the customer
    - Are we building the right product?
- Testing

> Testing is the process in which a (probably unfinished) program is executed with the goal to find errors.
> [Myers 76]

> Testing can only show the presence of errors, never their absence.
> [Dijkstra 69]

  - Testing can neither proof that a program is error free, nor that it is correct

Goals of Testing

- Detect deviations from specifications
  - Debugging
  - Regression testing
- Establish confidence in software
  - Operational testing
- Evaluate properties of software
  - Reliability
  - Performance
  - Memory use/leakage
  - Security
  - Usability

Principles of Software Testing*

- Complete testing is not possible
- Testing is creative and difficult
- A major objective of testing is defect prevention
- Testing must be risk-based
- Testing must be planned
- Testing requires independence

* Hetzel, *The Complete Guide to Software Testing*

Fundamental Steps of Software Testing

- Understand requirements and specifications
- Create the execution environment
- Select test cases
- Execute & evaluate test cases
- Evaluate test progress
- Information feedback & feedforward

Unit Testing

Defn: Tests the smallest individually executable code units.

Objective: Find faults in the units. Assure correct functional behavior of units.

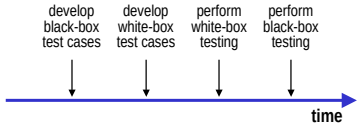
By: Usually programmers.

Tools:

- Test driver/harness
- Coverage evaluator
- Automatic test generator

Test Methods

- Structural testing (white-box, glass-box)
  - Uses code/detailed design is to develop test cases
  - Typically used in unit testing
  - Approaches:
    - Coverage-based testing
    - Symbolic execution
    - Data flow analysis
    - ...



- Functional testing (black-box)
  - Uses function specifications to develop test cases
  - Typically used in system testing
  - Approaches:
    - Equivalence partitioning
    - Border case analysis
    - ...

Test Preparation

- Exhaustive testing is prohibited, because of the combinatorial explosion of test cases
- Choose representative test data

```
for i := 1 to 100 do
   if a = b then
      X
   else
      Y;
```

i	paths to test	#tests
1	X, Y	2
2	XX, XY, YX, YY	4
3	XXX, XXY, ...	8
	...	
100		2^{100}

$2 * 2^{100} - 2 > 2{,}5 * 10^{30}$

With 10^6 tests/sec this would take $8*10^{16}$ years
Choose test data (*test cases*)

Test Case Development

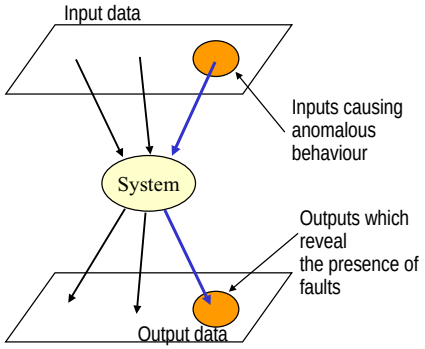
- Problems:
  - Systematic way to develop test cases
  - Find a satisfying set of test cases
- Test case ≠ test data
- Test data: Inputs devised to test the system
- Test case:
  - Situation to test
  - Inputs to test this situation
  - Expected outputs
  - Test are reproducible

Black-box Testing

- Test generation without knowledge of software structure
- Also called *specification-based* or *functional testing*

- Equivalence partitioning
- Boundary-value analysis

Equivalence Partitioning



Input- and output data can be grouped into classes where all members in the class behave in a comparable way.

- Define the classes
- Choose representatives
  - Typical element
  - Borderline cases

```
x ∈ [25 .. 100]   class 1: x < 25
                  class 2: x >= 25 and x <= 100
                  class 3: x > 100
```

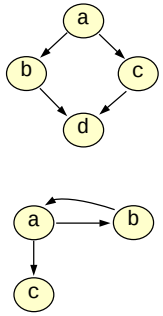

White-box Testing

Methods based on internal structure of code

- Statement coverage
- Branch coverage
- Path coverage
- Data-flow coverage

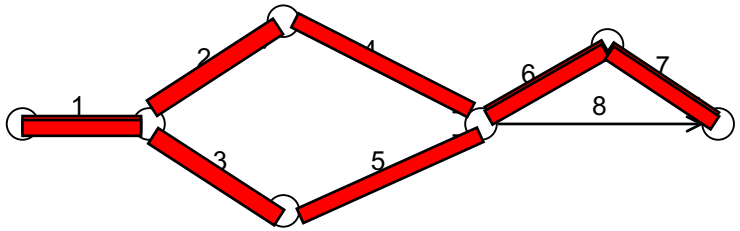
```
if a then
   b
else
   c;
d

while a do
   b;
c
```



Statement coverage

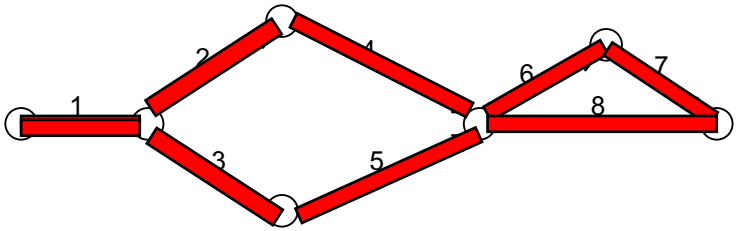
q Every statement is at least executed once in some test



2 test cases: 12467; 13567

Branch Coverage

q For every decision point in the graph, each branch is at least chosen once



2 test cases: 12467; 1358

Condition Coverage

q Test all combinations of conditions in boolean expressions at least once

```
if (X or not (Y and Z) and ... then
   b;
c := (d + e * f - g) div op( h, i, j);
```

Ł Why in boolean expressions only?

Path Coverage

Assure that all paths in the control-flow graph are executed.

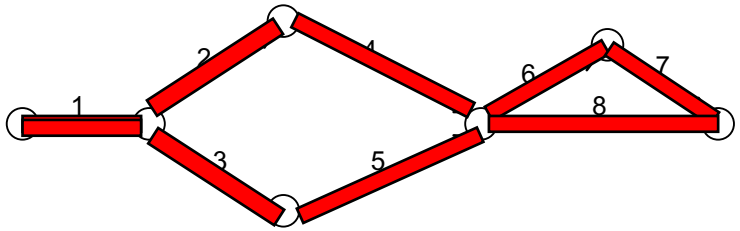
What is the definition of *all paths*?

q All loop-free paths

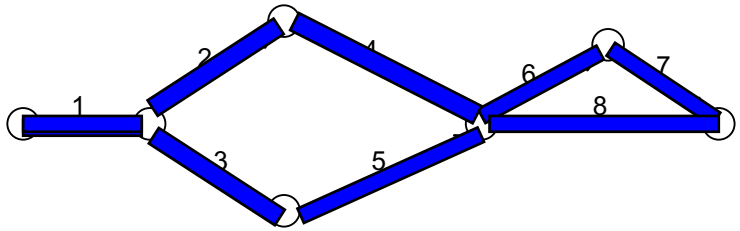
q All loop-free paths, plus all n-iterations of loops

q A set of basis paths for the graph.

Path Coverage



Path Coverage



4 test cases: 12467; 1358; 1248; 13567

Data-flow testing

Def of a variable *v* – assignment of value to *v*

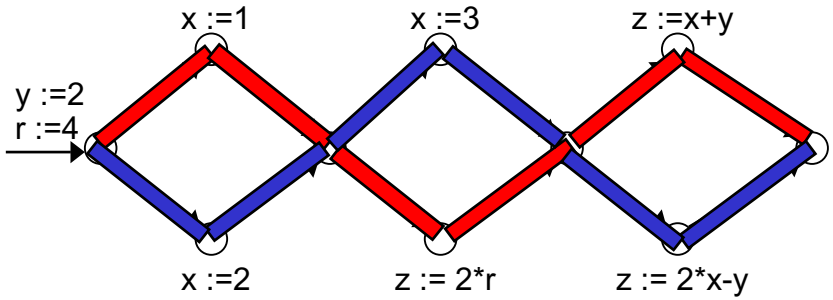
Use of variable *v* – access of the value of *v*

A *def-use association* for variable *v* is a def, a use and a path from the def to the use which contains no intervening definition of *v*.

All-defs – For each def *d,* test at least one path from *d* to some use of *d*

All-uses – For each def *d,* and for each use u_i of *d,* test at least one path from *d* to u_i

Data Flow Coverage
All-uses coverage



Red path covers the defs y :=2; r :=4; x :=1

Blue path covers y :=2; x :=3. Does not cover x :=2

Coverage-based Testing

- Advantages
  - Systematic way to develop test cases
  - Simple model of underlying program
  - Measurable results (the coverage)
  - Extensive tool support
    - Flow graph generators
    - Test data generators
    - Bookkeeping
    - Documentation support
- Disadvantages
  - Code must be available
  - Does not (yet) work well for data-driven programs

Integration Testing

Defn: Testing two or more units or components

Objectives

- Interface errors
- Functionality of combined units; look for problems with functional threads

By: Developers or Testing group

Tools: Interface analysis; call pairs

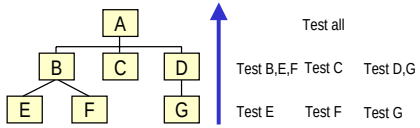
Issues:

- Strategy for combining units
- Assuring compatibility and correctness of externally-supplied components

Integration Testing

How to integrate & test the system

- Top-down
- Bottom-up
- Critical units first
- Functionality-oriented (threads)



Implications of build order

- Top-down => stubs; more thorough top-level
- Bottom-up => drivers; more thorough bottom-level
- Critical => stubs & drivers.

System Testing

Defn: Test the functionality, performance, reliability, security of the entire system.

By: Separate test group.

Objective: Find errors in the overall system behavior. Establish confidence in system functionality. Validate that system achieves its desired non-functional attributes.

Tools: User simulator. Load simulator

Realities of System Testing

- Available time for testing is short
  - Compressing development risks introducing problems
  - Compressing testing risks missing critical problems
- Testers want to start testing early
- System testing requires an available system
- Developers resist testing until system is “ready”

To optimize use of the existing resources, use **risk analysis**.

Acceptance Testing

Defn: Operate system in user environment, with standard user input scenarios.

By: End user

Objective: Evaluate whether system meets customer criteria. Determine if customer will accept system.

Tools: User simulator. Customer test scripts/logs from operation of previous system.

Regression Testing

Defn: Test of modified versions of previously validated system.

By: System or regression test group.

Objective: Assure that changes to system have not introduced new errors.

Tools: Regression test base, capture/replay

Issues: Minimal regression suite, test prioritization

Test Automation

Automation has several meanings

- Test generation: Produce test cases by processing of specifications, code, model.
- Test execution: Run large numbers of test cases/suites without human intervention.
- Test management: Log test cases & results; map tests to requirements & functionality; track test progress & completeness

Issues of Test Automation

Automating Test Execution

- Does the payoff from test automation justify the expense and effort of automation?
- Learning to use the automation tool is non-trivial
- Testers become programmers
- All tests, including automated tests, have a finite lifetime.
- Complete automated execution implies putting the system into the proper state, supplying the inputs, running the test case, collecting the result, verifying the result.

Best uses of automated test execution

- Load/stress tests--Nearly impossible to have 1000 human testers simultaneously accessing a database.
- Regression test suites--Tests collected from previous releases; run to check that updates don't break previously correct operation.
- Sanity tests (smoke tests)--run after every new system build to check for obvious problems.

Test documentation

- Test plan: describes system and plan for exercising all functions and characteristics
- Test specification and evaluation: details each test and defines criteria for evaluating each feature
- Test description: test data and procedures for each test
- Test analysis report: results of each test

The Key Problems of Software Testing

- Selecting or generating the right test cases.
- Knowing when a system has been tested enough.
- Knowing what has been discovered/demonstrated by execution of a test suite.