Introduction

How strange it is to say that testing a program and never having it result in a failure is a problem

What is product quality?

- Quality, simplistically, means that a product should meet its specification
- Software point of view:
  - Meet the requirements
- This is problematical for software systems
  - Some requirements are difficult to specify in an unambiguous way
  - Software specifications are usually incomplete and often inconsistent
  - The quality compromise: we cannot wait for specifications to improve before paying attention to quality, and procedures must be put into place to improve quality in spite of imperfect specification
- More general notion (since specifications are imperfect): fitness for purpose, value to user
- Quality = expectations - delivery

The current status of software quality

- Microsoft Windows XP End-User License Agreement:

  11. LIMITED WARRANTY FOR PRODUCT ACQUIRED IN THE US AND CANADA.

  Microsoft warrants that the Product **will perform substantially** in accordance with the accompanying materials **for a period of ninety days from the date of receipt**.
  (…)
  If an implied warranty or condition is created by your state/jurisdiction and federal or state/provincial law prohibits disclaimer of it, you also have an implied warranty or condition, BUT ONLY AS TO DEFECTS DISCOVERED DURING THE PERIOD OF THIS LIMITED WARRANTY (NINETY DAYS).
  (…)
  Some states/jurisdictions do not allow limitations on how long an implied warranty or condition lasts, so the above limitation may not apply to you.
  (…)
  YOUR EXCLUSIVE REMEDY. Microsoft's and its suppliers' entire liability and your exclusive remedy shall be, at Microsoft's option from time to time exercised subject to applicable law, **(a) return of the price paid** (if any) for the Product, or **(b) repair or replacement of the Product**, that does not meet this Limited Warranty and that is returned to Microsoft with a copy of your receipt.
  (..)
  This Limited Warranty is void if failure of the Product has resulted from accident, abuse, misapplication, abnormal use or a virus.

The current status of software quality – Windows 10

LIMITED WARRANTY

Microsoft warrants that properly licensed software **will perform substantially** as described in any Microsoft materials that accompany the software. This limited warranty does not cover problems that you cause, or that arise when you fail to follow our instructions, or that are caused by events beyond Microsoft's reasonable control. The limited warranty starts when the first user of your copy of the software acquires that copy, **and lasts for one year**. Any supplements, updates, or replacement software that you may receive from Microsoft during that year are also covered, but only for the remainder of that one year period or for 30 days, whichever is longer. …

Investing in software testing

- (Some) project managers view testing as a necessary evil that occurs **at most only at the end** of the project
  - Costs money
  - Doesn't help to build the project
  - Takes too long
  - Creates hostility between test team and the other teams

  $\Rightarrow$ **spend as little as possible**
- However, smart managers invest in testing. Why?
  - It increases the quality of product
  - Reduces the time to develop the product
  - It saves money

Quality cost

- Costs of conformance
  - All costs associated with planning and running tests (and revisions) just one time and activities that promote good quality (code reviews, training, …)
- Costs of nonconformance
  - Costs due to internal failures (before release)
    - Cost of isolating, reporting and regression testing bugs (found before the product is released) to assure that they're fixed
  - Costs due to external failures (after release)
    - If bugs are missed and make it through to the customers, the result will be costly product support calls, possibly fixing, retesting, and releasing the software, and – in a worst case-scenario – a product recall or lawsuits
    - Angry clients

Quality cost - Costs of nonconformance

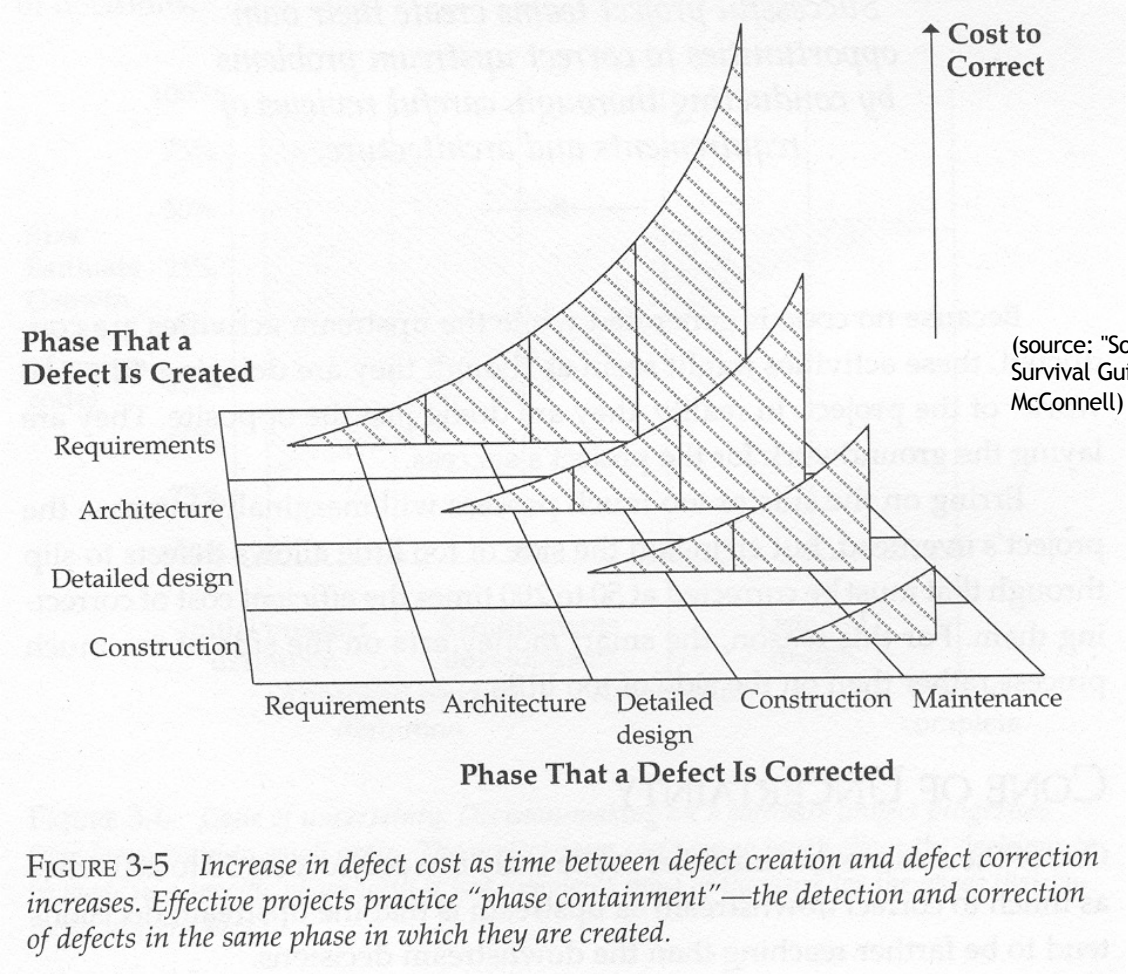
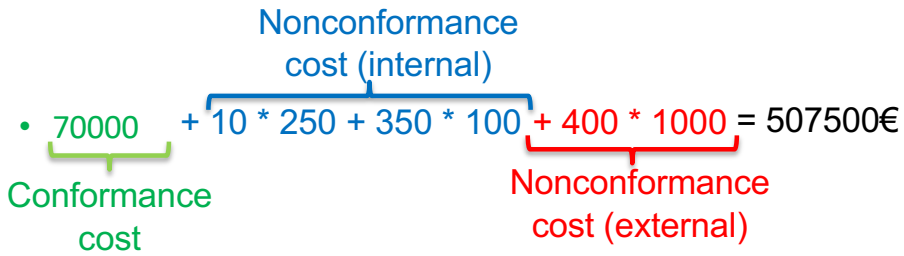


FIGURE 3-5 *Increase in defect cost as time between defect creation and defect correction increases. Effective projects practice "phase containment"—the detection and correction of defects in the same phase in which they are created.*

(source: "Software Project Survival Guide", Steve McConnell)

Quality cost - Quality is free!?

- Development software example:
  - Each release contains 1000 bugs on average
  - Developers found 250 during development
  - Cost of internal bug:10€
  - Cost of external bug: 1000€
- Scenario without test team
  - Cost of quality
    - 10 * 250 + 750 * 1000 = 752500€
- Scenario with test team
  - Test team costs 70000 € and bugs found by test team cost 100 €
  - Testers found 350 bugs and users 400
  - Cost of quality:



How much does a bug cost?

- Study made by IBM in 2008
  - X -> a normalized unit of cost

- **Design - 1X**
- **Implementation - 5X**
- **Integration - 10X**
- **Beta Testing – 15X**
- **After Release – 30X**

A simple object-oriented example

- Input: Three integers, a, b and c, that represent the lengths of the side of a triangle
- Output: The class has methods to show if the specified triangle is scalene, isosceles, equilateral or invalid
- Problem: Devise a test plan to test this class
- Specification - A valid triangle must meet two conditions:
  - Each side greater than 0
  - Each side must be shorter than the sum of the all sides divided by two
    - All sides equal: equilateral triangle
    - Two sides equal: isosceles triangle
    - All sides unequal: scalene triangle

Example implementation

```
class Triangle {
   private int a, b, c;

   public Triangle(int a, int b,int c) throws InvalidArgumentException  {…}
   public boolean isIsosceles() {…}
   public boolean isScalene() {…}
   public boolean isEquilateral() {…}
   public void draw() {…}
   public void erase() {…}
}

public class Example {
 public static int main(String args[]) {
  // read three numbers and draw the rectangle
 }
}
```

Basic test case classes

- Valid scalene, isosceles, equilateral triangle
  - (5, 3, 7); (3, 3, 5); (3, 3, 3)
- All permutations of two equal sides
  - (50;50;25); (25, 50, 50); (50, 25, 50)
- Zero or negative lengths
  - (0, 2, 2); (2, 0, 2); …
- All permutations of a + b < c
- All permutations of a + b = c
  - (7, 5, 12); (5, 7, 12); (12, 5, 7); …
- MAXINT values
  - (MAXINT, MAXINT, MAXINT); (MAXINT, MAXINT, 1); …
- Non-integer inputs
- Missing input a, b and c

Extra Tests

- Is the constructor correct?
- Is only one of the **`is*`** methods true in every case?
- Do results repeat, e.g. when running **`isScalene`** twice or more?
- Results change after **`draw`** or **`erase`**?
- …

Object-oriented terms

Term	Definition
Client	A class (object) that sends a message to another class (object)
Server	A class (object) that answers to a message sent by a client
Cluster	A group of classes that support some common purpose
Member	A method or instance variable
Accessor	A method that does not change the state of an object
Modifier	A method that can change the state of an object
UT **M/C/O/SUT**	Under Test Method/Class/Object/System Under Test

Software Testing - Introduction

- Testing is a 2-step process:

  1. Design tests by analyzing SUT

  2. Execute the tests
     - Manual
     - Automatic

- Manual Testing
  - Low implementation cost
  - Repetitive
  - Boring
  - Error-prone
  - Slow
  - High execution cost

- Automatic Testing
  - High implementation cost
  - Low execution cost
  - Fast

What is software testing?

- A problem in systems engineering
  - Automatic Testing is mainly about the development of an automated system
  - Design and implementation of a special kind of software system
- Often - Software testing = find bugs
  - However, there are tests that do not reveal failures
- Better definition:
  - Software testing is the process of executing a software system to determine whether it matches its specification in its intended environment.
    - Testing $\Rightarrow$ **running** a program
    - A **specification** is crucial: Defines correct behavior so that incorrect behavior is easier to identify
      - Incorrect behavior = software failure
    - Specification should represent the expectations of the client

When does a software bug occur?

- The software doesn't do something that the product specification says it should do
- The software does something that the product specification says it shouldn't do
- The software does something that the product specification doesn't mention
- More?
- The software doesn't do something that the product specification doesn't mention but should
- The software is difficult to understand, hard to use, slow, or – in the software tester's eyes – will be viewed by end users as just plain not right

What are the main sources of bugs?

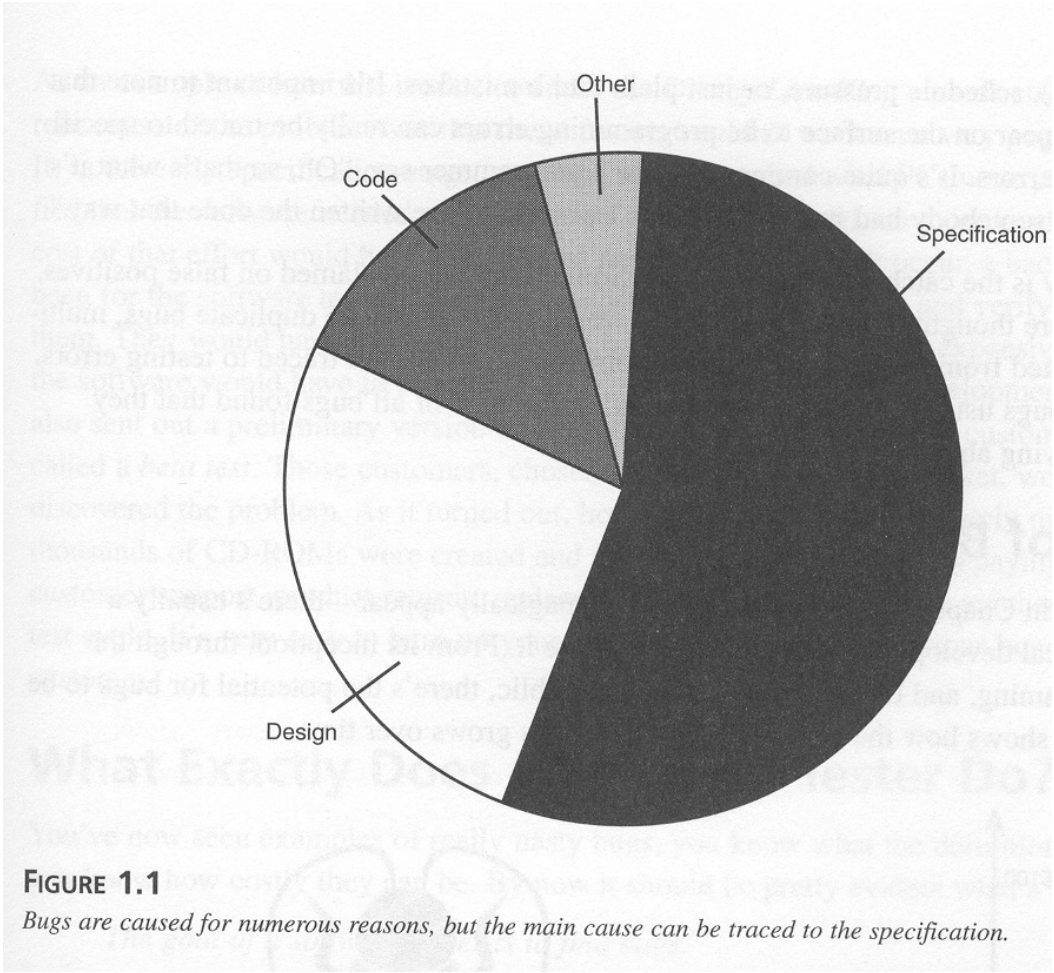


FIGURE 1.1
Bugs are caused for numerous reasons, but the main cause can be traced to the specification.

(source: "Software Testing", Ron Patton)

How did those bugs escape testing?

- Many reasons

- The user executed untested code
  - **Code that is not tested does not work!**
- Different order of execution of the statements
- Untested combination of input values
- The user´s operating system was never tested
- Environment conditions not tested
- …

Types of tests - 1

- Depend on scope and/or goal of testing
- Unit Test
  - Scope: a relatively small executable: a class, a method or a cluster of interdependent classes
  - Goal: to show that the target of the test works has required
- System Test
  - Scope: complete integrated application
  - Focus on the capabilities and characteristics that are present only with the entire system
  - Tests may be functional (input/output), performance and stress or load

Types of tests - 2

- Integration testing
  - Scope: a complete system or subsystem of software
  - Can be difficult:
    - Place objects int the states required by the test
    - Some classes not developped yet
- Regression testing
  - Changes often break working code
  - Rerun tests of version *n – 1* on version *n* before testing anything new
  - Which tests? All of them may take too much time.

The Limits of testing

- Passing a test is not enough to show the absence of bugs
  - If *f(x)* returns a correct result when *x = 123* does not mean that it returns a correct value for all values of *x*.
  - Moreover, if we run *f(123)* in different conditions it can return an incorrect value.
  - Even doing many test for different values of x does not mean that *f(x)* works correctly for all values of *x*

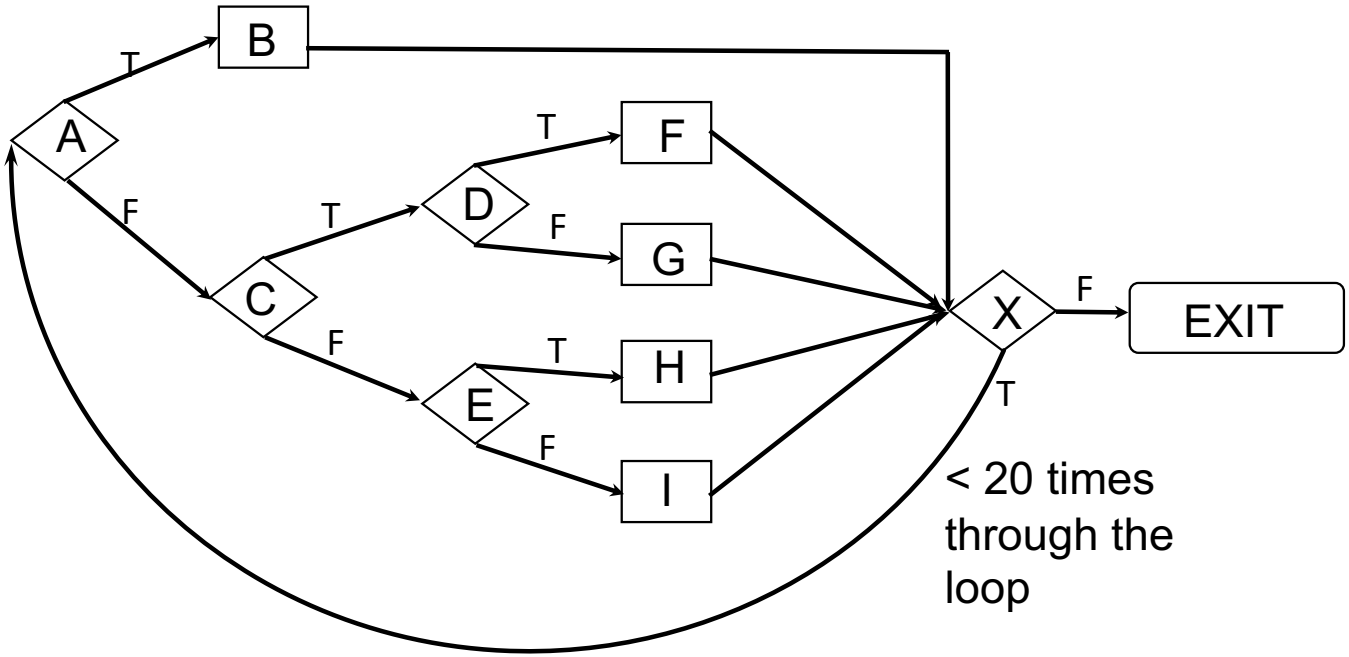
- Proof of correctness is equivalent to exhaustive testing.

The Input/Output space

- Triangle example:
  - Length side in] 0,10000]
  - 10^4 possibilities for a segment size
  - 10^{12} for a triangle.
  - Testing 1000 cases per second, you would need 317 years!

- Number of input and output combinations for trivial programs is surprisingly large
- For typical programs is astronomical
- For typical systems is beyond comprehension

Testing all paths in the system



Source: Myers, *The Art of Software Testing.*

Number of paths

- One path is ABX-Exit. There are 5 ways to get to X and then to the EXIT in one pass.
- Another path is ABXACDFX-Exit. There are 5 ways to get to X the first time, 5 more to get back to X the second time, so there are 5 x 5 = 25 cases like this.
- There are $5^1 + 5^2 + ... + 5^{19} + 5^{20} = 10^{14}$ = 100 trillion paths through the program.
- It would take only a billion years to test every path (if one could write, execute and verify a test case every five minutes).

Fault sensitivity and Coincidental correctness

- Related terms
- Best bugs?
  - Those that cause a failure every time they execute
- However, most bugs are not like that
- Fault sensitivity
  - The ability of code to hide faults from a test suite
    - Executing buggy code does not always expose the bug
- Coincidental correctness
  - Buggy code can produce correct results for some inputs
  - Have **x + x** instead of **x * x**
  - Same result for x = 0 or x = 2

Coincidental correctness example

```
int scale (int j){
    j = j – 1; // should be j = j + 1
    j = j / 3000;
    return j;
}
```

Only 40 out of 65,536 values produce an incorrect result: …, -3000, -2999, 2999, 3000, …

A test suite with a coverage of 99.939% of all input values might not find the problem.

How to design test cases

- How to design test cases in a rational way?
  - Number of places to look for bugs is infinite
- Solution
  - Model the IUT
  - Consider its fault model
    - A fault model identifies relationships and components of the SUT that are more likely to have faults
  - Generate test cases based on the fault model
    - Apply a test strategy
  - Add test cases based on code analysis, suspicions and error-guessing
  - Develop expected results for each test case

Test strategies

- Black box (behavioral) testing
  - Uses expected responsabilities of IUT to design tests
  - Tests are design independently of the language or algorithm used in IUT
- White box testing
  - Also relies on source code analisys to develop tests

- Fault-based testing
  - Purposely introduces faults into code to test
  - Then check if those faults are detected by the test suite

Specification of a test case

- Test Case = a test of something
  - In OO, corresponds to invoke a **given method** on **an object**
- Specification of a test case includes:
  - **The method to test**
  - **The initial state of the system**
    - initial state of the object being invoked (OUT)
    - and maybe some other global variables
  - **Input test values**
    - Includes the parameters of the MUT
  - **The expected output**
    - Includes returned value (if any)
    - expected result state of the invoked object
    - and (maybe) expected state of parameters,
    - And (maybe) expected state of global variables,
    - …

Properties of a test case

- Test a single condition of the IUT
  - Do not try to exercise the same method several times to save implementation time
- Independent
  - Should not depend on the outcome of the previous test case
- Self-cleaning
  - Returns the system's state to initial state
- Documented
  - Test goal should be clear and understandable
    - Document the test method or
    - Use a good test method name
- Simple and clear to understand
  - It should be small
    - No more than 10-15 LOC excluding setup/tear down

Properties of a test case - 2

- Accurate
  - Agrees with documentation
- Reasonable probability of catching a defect
- Repeatable
  - Can be used to perform the test over and over
  - It is completely automated
- Fast

Test execution

- Test execution typically follows several steps:
  1. Establish that the IUT is minimally operational
  2. Execute the test suite(s)
     - If the result of executing a test case
       - Is equal to the expected result → pass
       - Is not equal to the expected value → no pass
         - Reveals a bug and therefore is a **sucessful** test
  3. Use a coverage tool
  4. If necessary, develop aditional test
  5. Stop testing when coverage goal is met and all tests pass
- Test design and execution – better in parallel with aplication, analysis, design and coding

Absolute limitations of testing

- Dijkstra: “Program Testing can be used to show the presence of defects, but never their absence”
  - Proof of correctness is equivalent to exhaustive testing.
- Testing cannot verify requirements
  - Incorrect or incomplete requirements may lead to spurious tests
- We can never be sure that a testing system is correct. Bugs in test design or test drivers are equally hard to find
- Expected output for certain test cases might be hard to determine

Conclusion

- Testing is complex, critical and challenging
- Testing is not debugging
- It is impossible to fully test a software system in a reasonable amount of time or money.
- When is testing complete?
  - When you run out of time or money.
- Testing is more effective when test development begins at the outset of a project