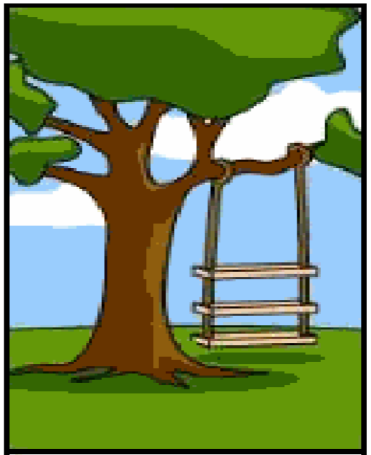
COMP5111 – Fundamentals of Software Analysis

Overview of Software Testing

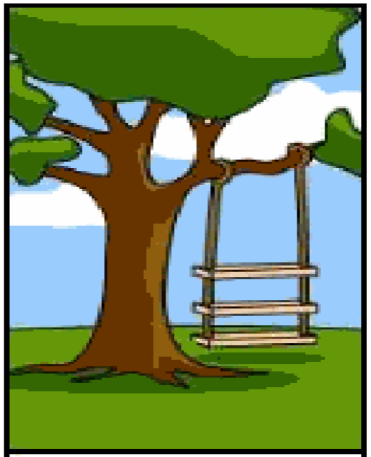
Shing-Chi Cheung

Computer Science & Engineering

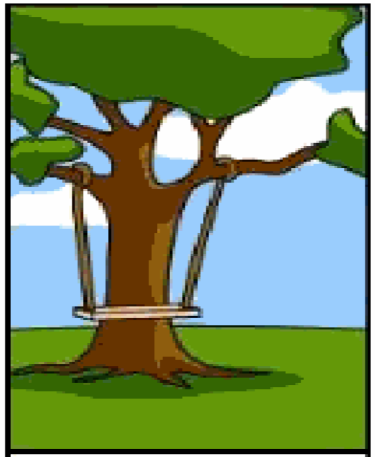
HKUST



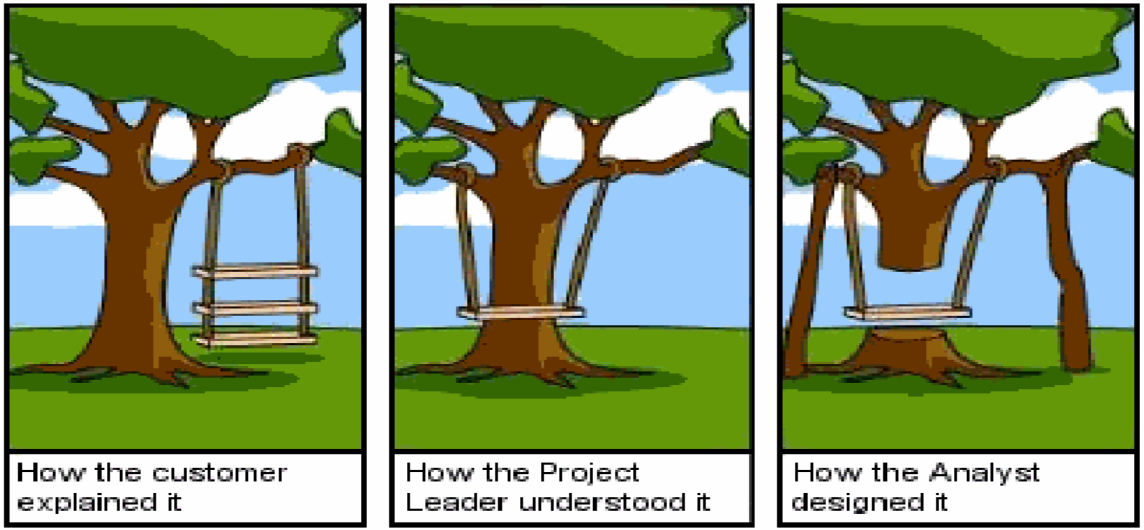
How the customer explained it

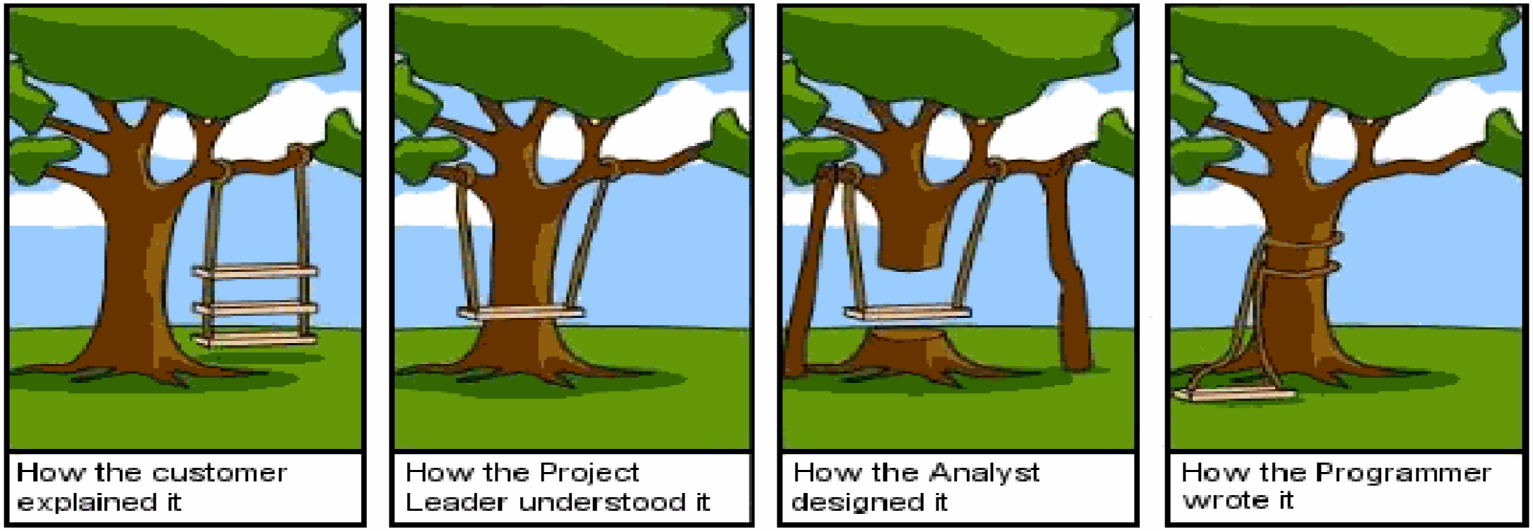


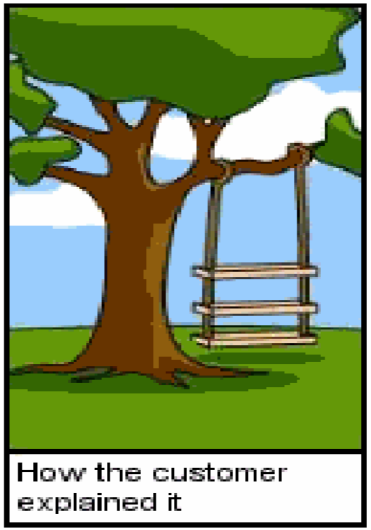
How the customer explained it

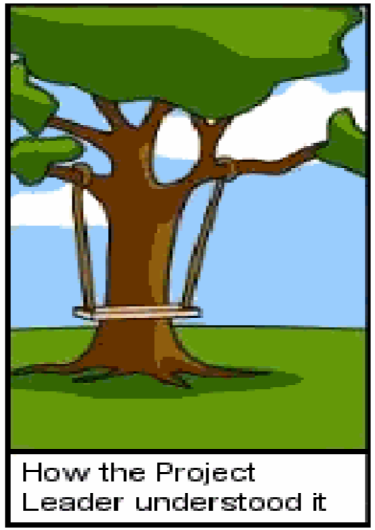


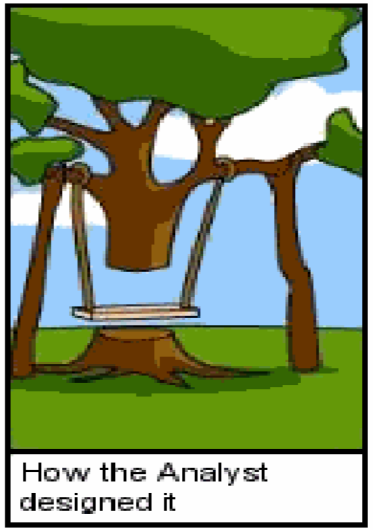
How the Project Leader understood it

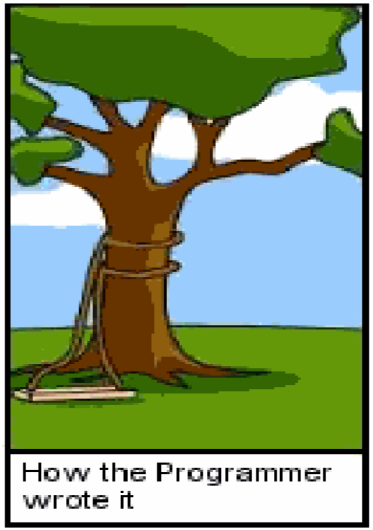

How the customer explained it
How the Project Leader understood it
How the Analyst designed it

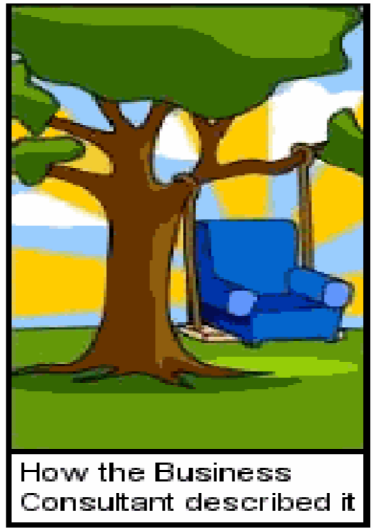

How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it

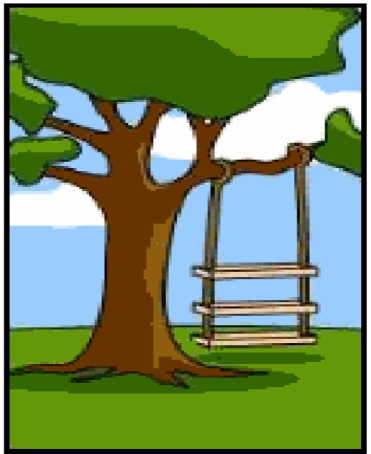

How the customer explained it


How the Project Leader understood it

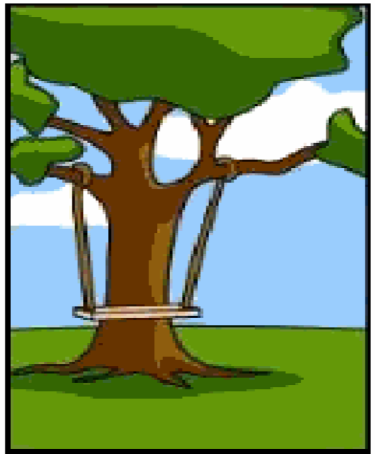

How the Analyst designed it


How the Programmer wrote it

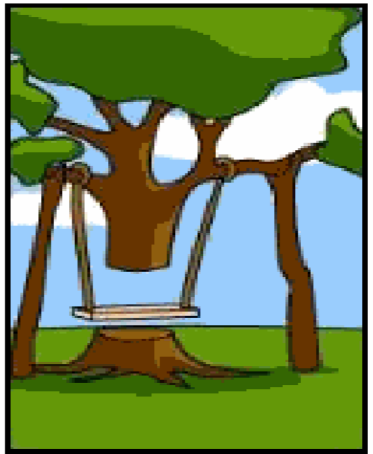

How the Business Consultant described it



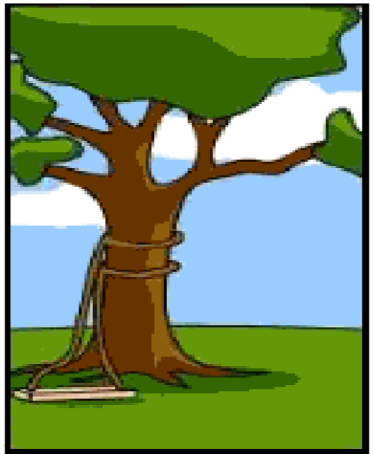
How the customer explained it



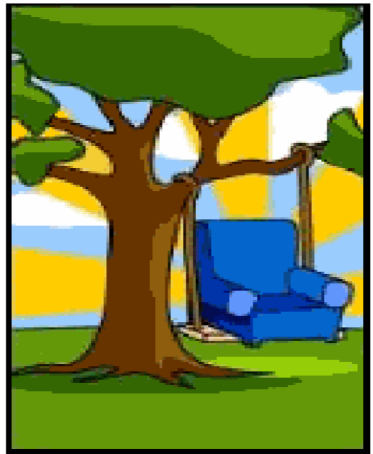
How the Project Leader understood it



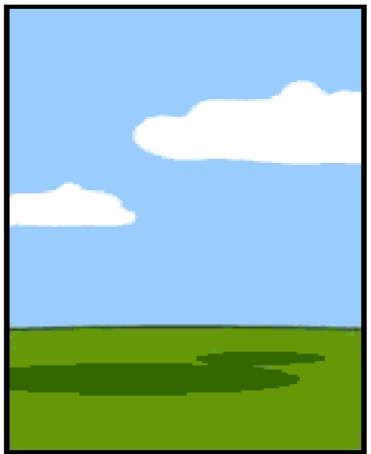
How the Analyst designed it



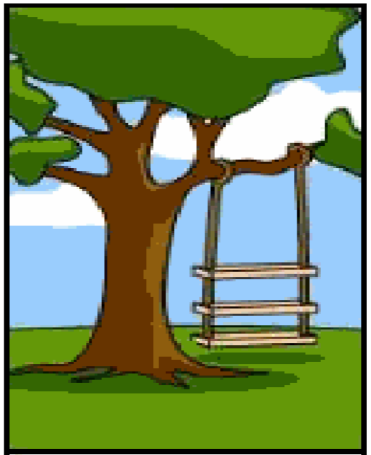
How the Programmer wrote it



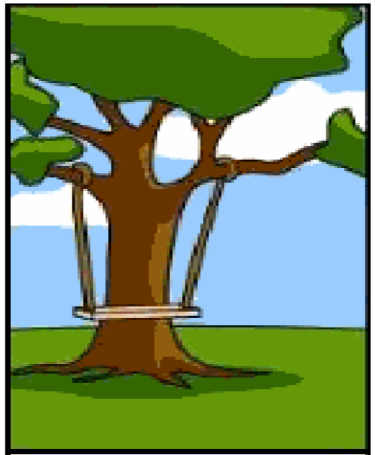
How the Business Consultant described it



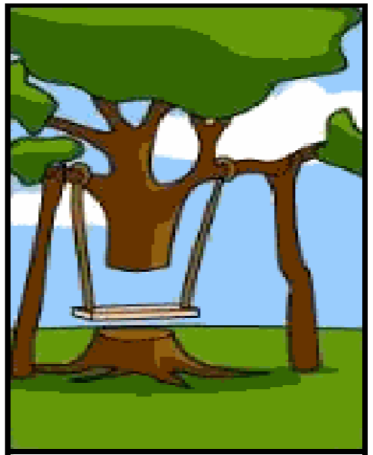
How the project was documented



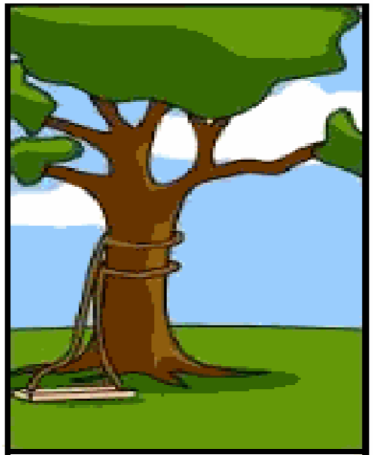
How the customer explained it



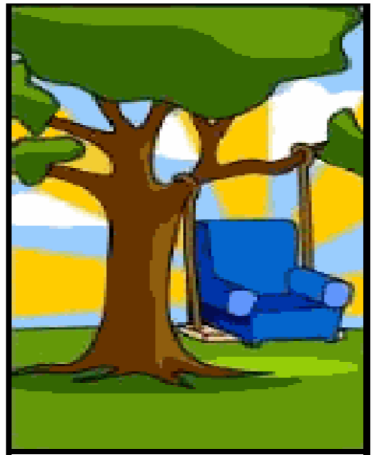
How the Project Leader understood it



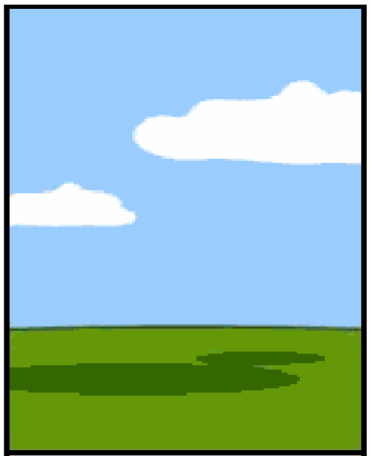
How the Analyst designed it



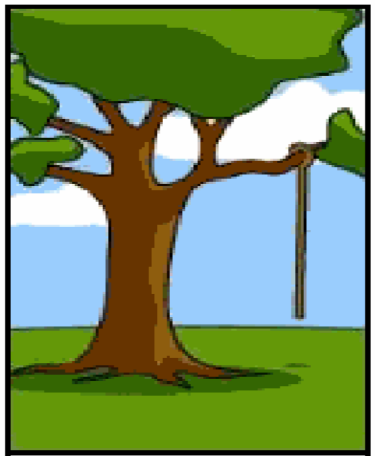
How the Programmer wrote it



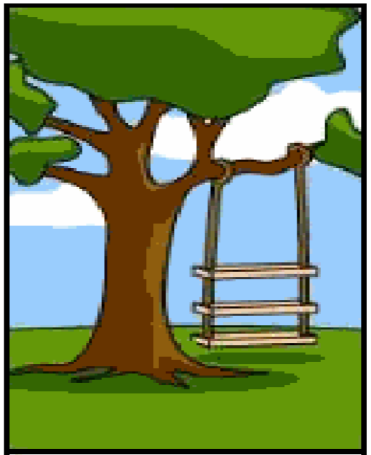
How the Business Consultant described it



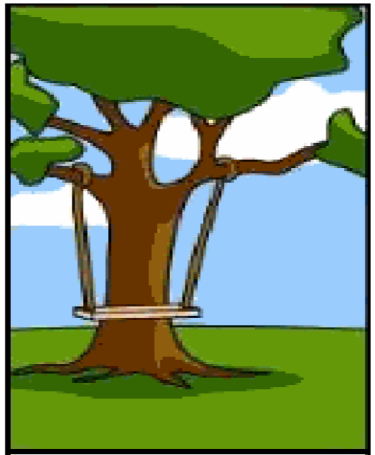
How the project was documented



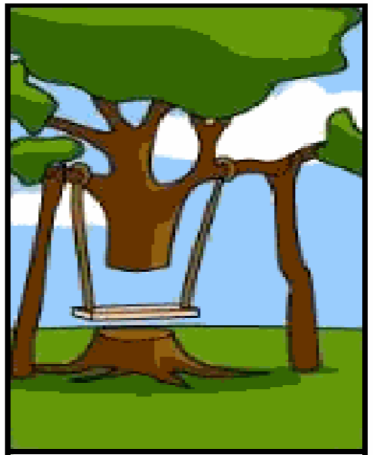
What operations installed



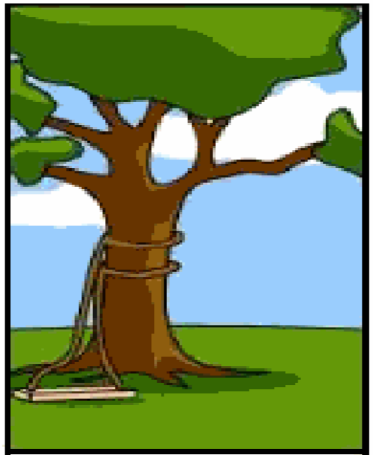
How the customer explained it



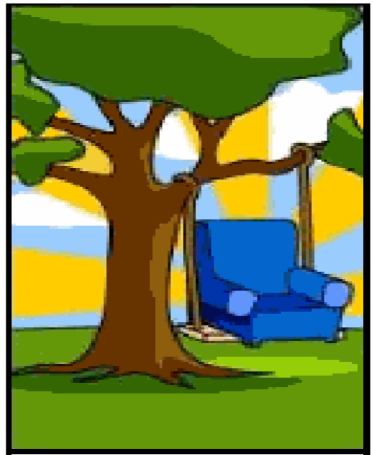
How the Project Leader understood it



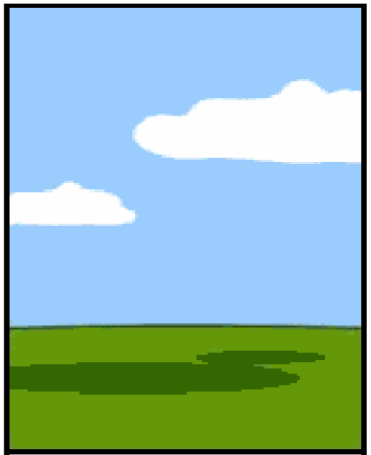
How the Analyst designed it



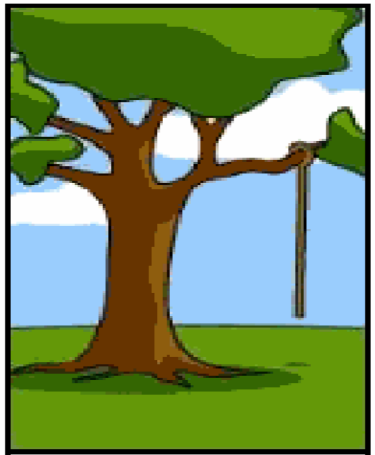
How the Programmer wrote it



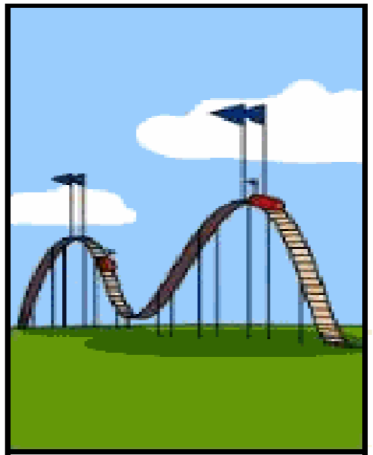
How the Business Consultant described it



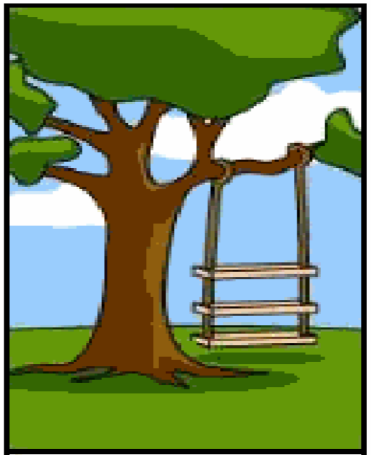
How the project was documented



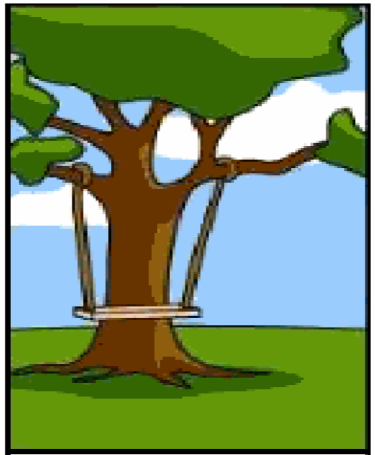
What operations installed



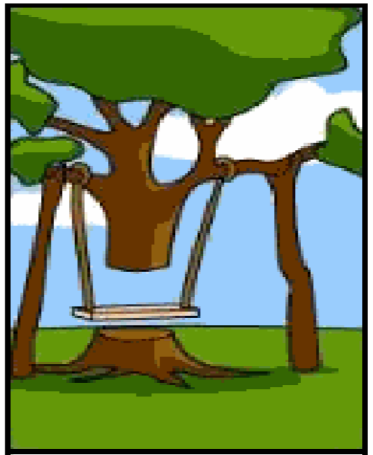
How the customer was billed



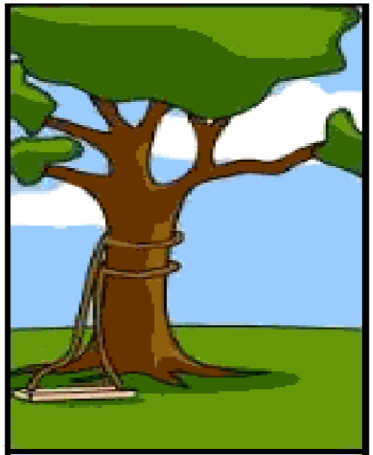
How the customer explained it



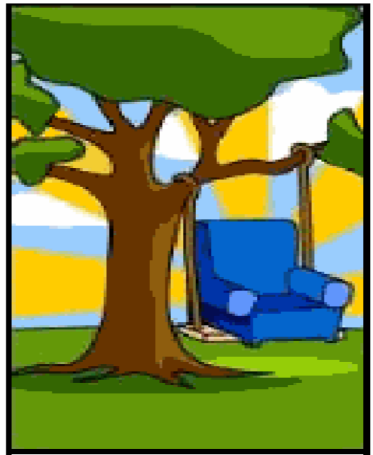
How the Project Leader understood it



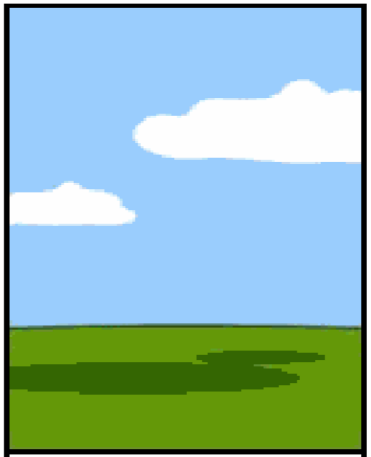
How the Analyst designed it



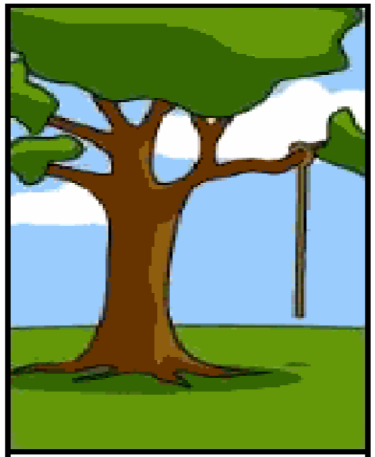
How the Programmer wrote it



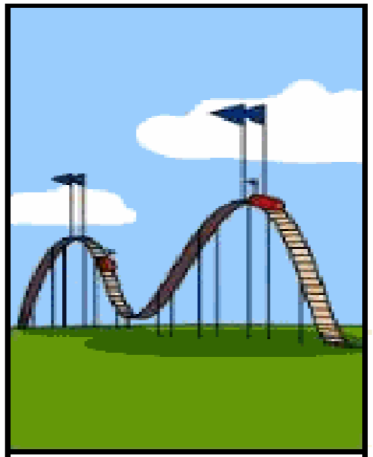
How the Business Consultant described it



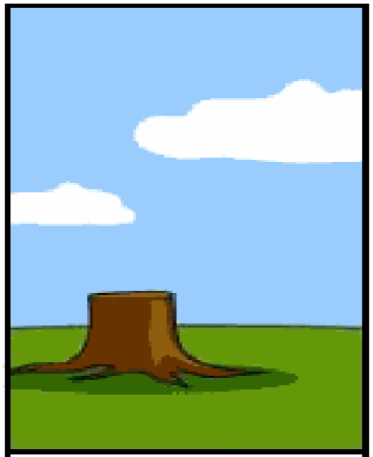
How the project was documented



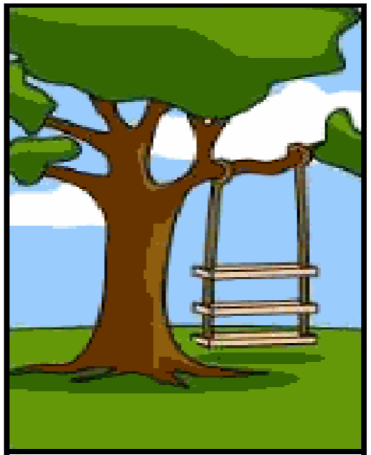
What operations installed



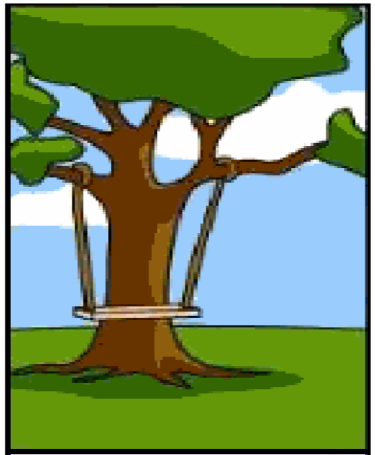
How the customer was billed



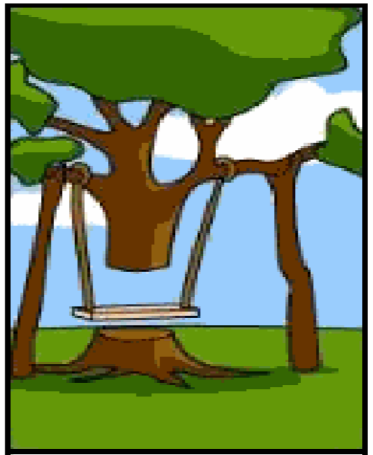
How it was supported



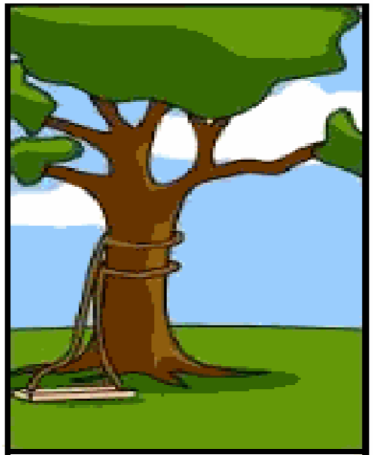
How the customer explained it



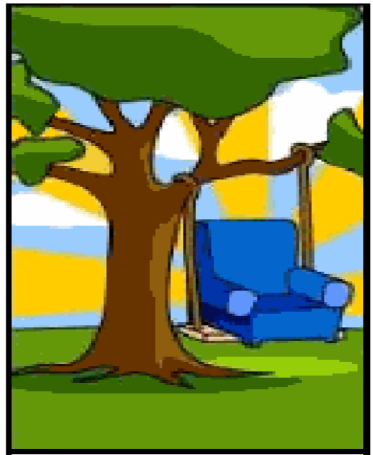
How the Project Leader understood it



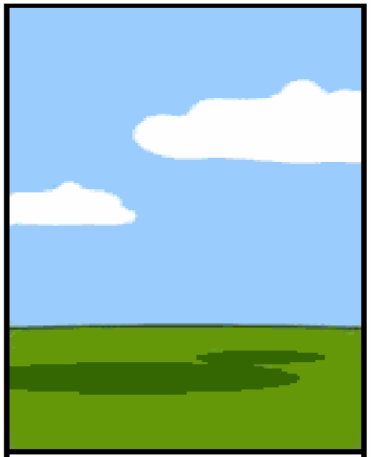
How the Analyst designed it



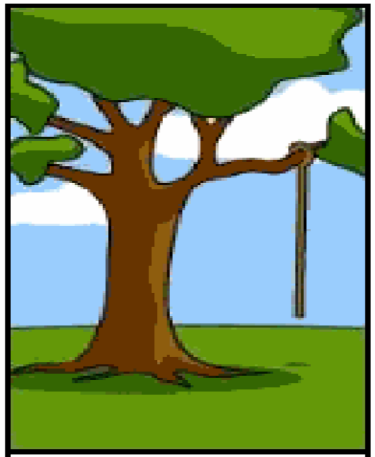
How the Programmer wrote it



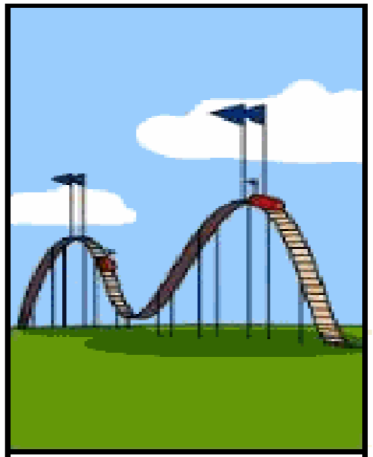
How the Business Consultant described it



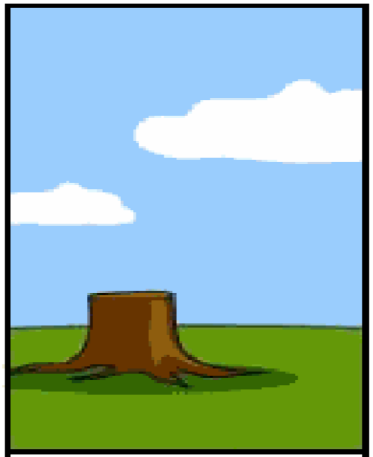
How the project was documented



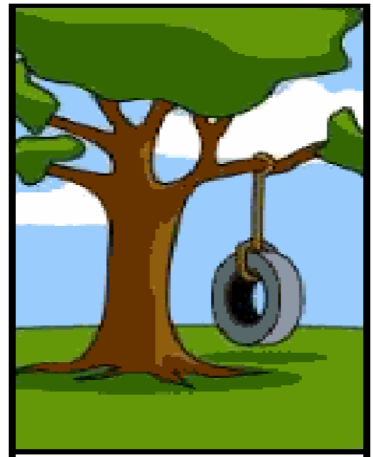
What operations installed



How the customer was billed



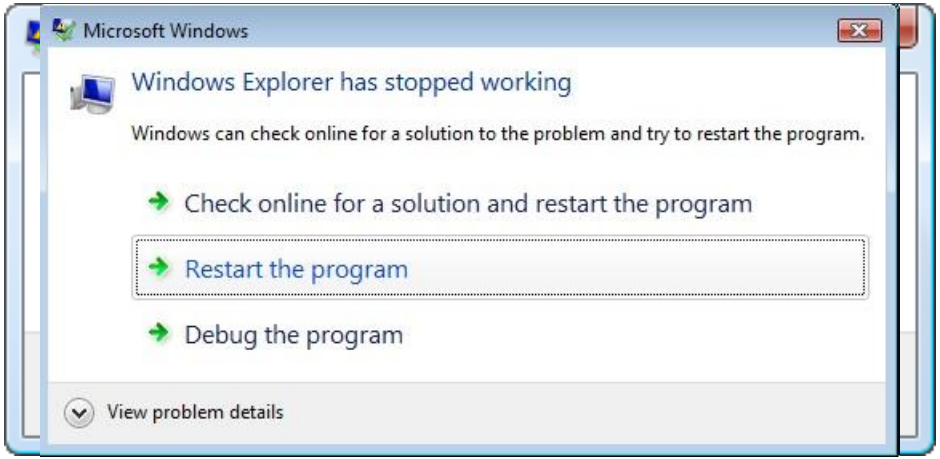
How it was supported

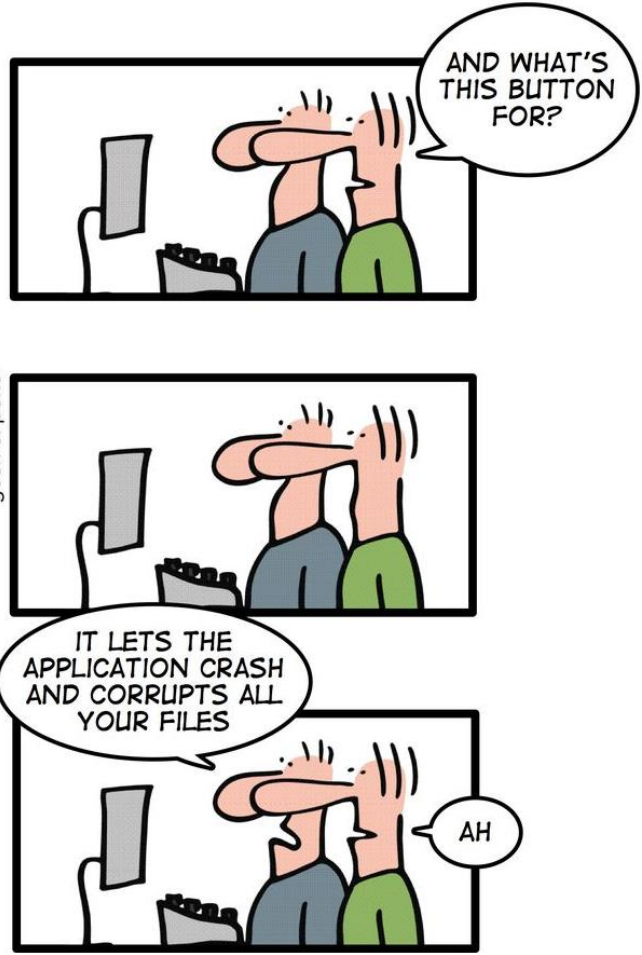


What the customer really needed

What is users' experience

- Sounds familiar when using Windows sometimes?





WE'VE GOTTEN USED TO IT

What? You want to test my code! Why? We haven't got time anyway.

1960s – 1980s
Constraint

OK, maybe you were right about testing. A nasty bug likely exists and customers are complaining.

1990s
Need

Testers! You must work harder! Longer! Faster!

2000+
Asset

Facts in Google as at 2011

- Principles
  - Quality must be owned by engineering
  - Test must be part of engineering
- 15,000 developers run 100+ million tests per day.
- 20+ code changes/minute
- 30K code commits per day (one every 3 seconds)

Software Testing - Facts

expensive

30-90% of development effort

	Cost of inadequate testing on US economy (billions)
developers	$21.2
users	$38.3
TOTAL	**$59.5**

source: NIST 2002

Recall: Lessons Learnt with FindBugs

- Static analysis, at best, might catch 5-10% of software quality problems
- Need to understand which types of bugs matter to your project
- Example: public static final field pointing to an array where anyone can change the content of the array
  - It is likely a big concern if your program run with other untrusted code in the same VM
  - Otherwise, it is likely a minor concern

Recall: Lessons Learnt with FindBugs

- High false positives
  - Only a small portion of warnings are real; most of them are false alarms.
  - Allow users to enable/disable the rules individually.
  - Focus on the new ones not found in the baseline

Software Testing - Facts

expensive

30-90% of development effort

	Cost of inadequate testing on US economy (billions)
developers	$21.2
users	$38.3
TOTAL	**$59.5**

source: NIST 2002

difficult

complex software

› many behaviors to test

large input spaces

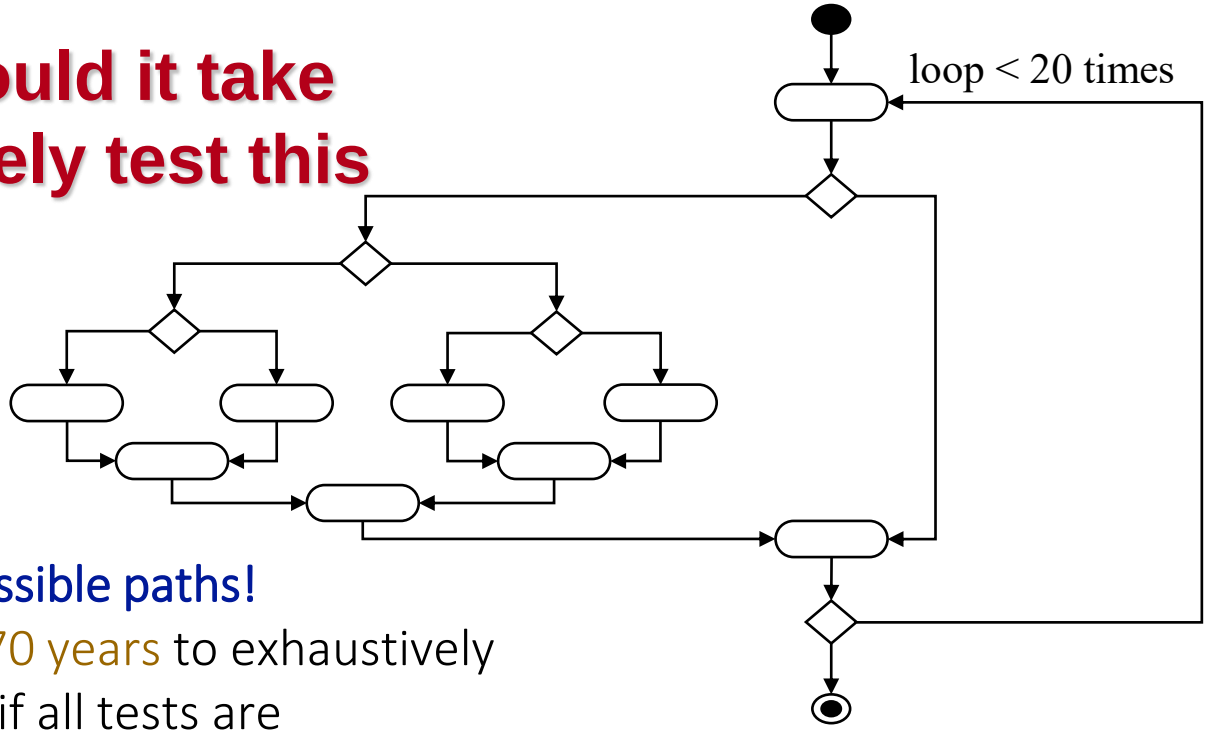
› selecting subset is hard

done mostly by hand

› at Microsoft, ½ of engineers

WHY is Testing Difficult?

How long would it take to exhaustively test this program?



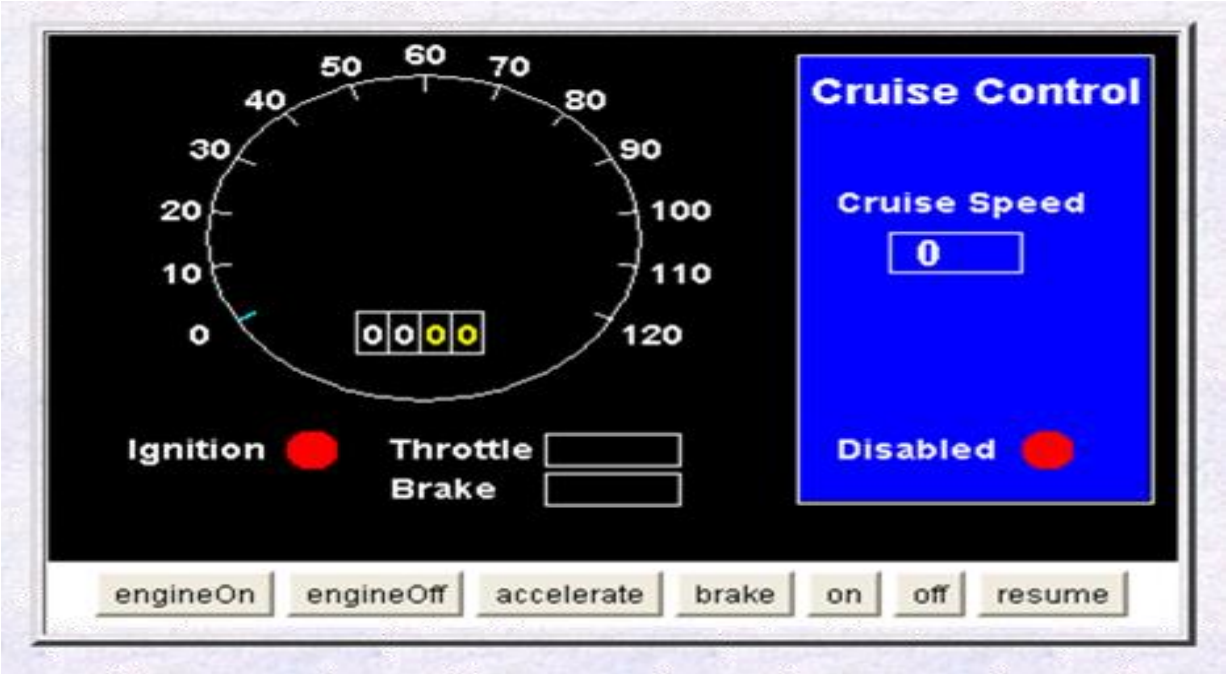
➔ There are 10^{14} possible paths!

It would take 3,170 years to exhaustively test the program if all tests are automated and each test requires 1 ms!!

Recall: Facts of real projects

- Eclipse is known for its high reliability
- Defect density of Eclipse 3.0
  - Scariest defects: 30 per million LOC
  - Scary: 160 per million LOC
  - Troubling: 480 per million LOC
  - Of concern: 6,000 per million LOC

Here! Try it out Yourself!



http://www.cse.ust/hk/~scc/CruiseWithBugs.zip

Software Testing - Facts

expensive

30-90% of development effort

	Cost of inadequate testing on US economy (billions)
developers	$21.2
users	$38.3
TOTAL	**$59.5**

source: NIST 2002

difficult

complex software

› many behaviors to test

large input spaces

› selecting subset is hard

done mostly by hand

› at Microsoft, ½ of engineers

goal: automation

automate test case creation

› a principal testing activity

› a significant portion of cost

Understanding Basic Terminologies

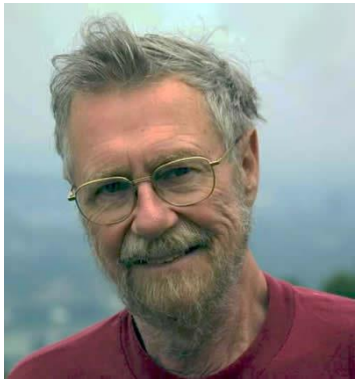
What is Testing?

Testing is the activity of finding out whether a piece of code (a method, class, or program) produces the intended behavior.

taken from “Objects First with Java”

Edsger Dijkstra (1930-2002):

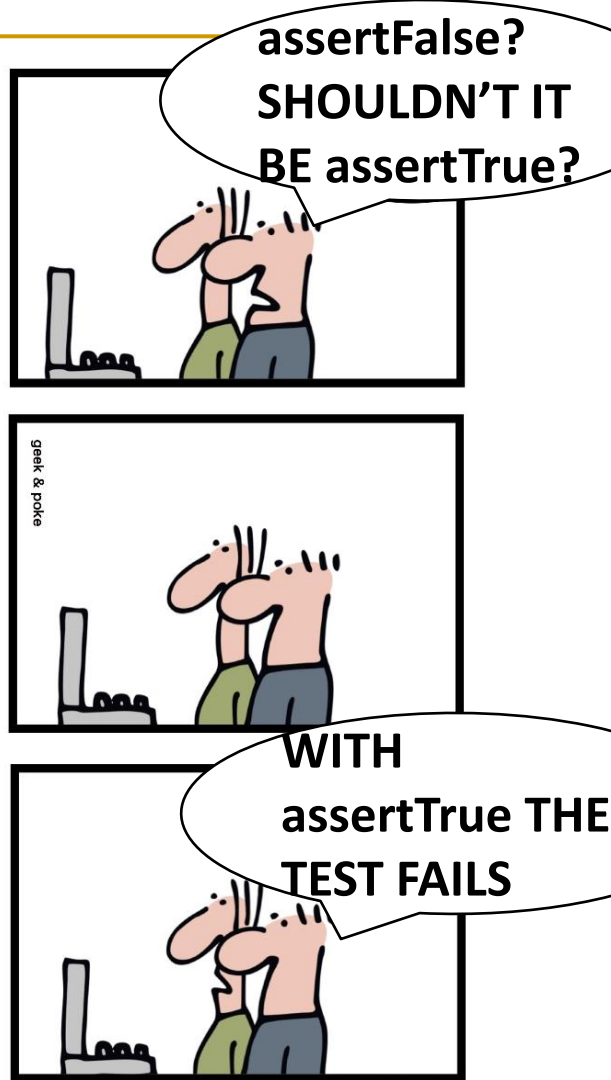
Program testing can be used to show the presence of bugs, but never to show their absence *unless …*



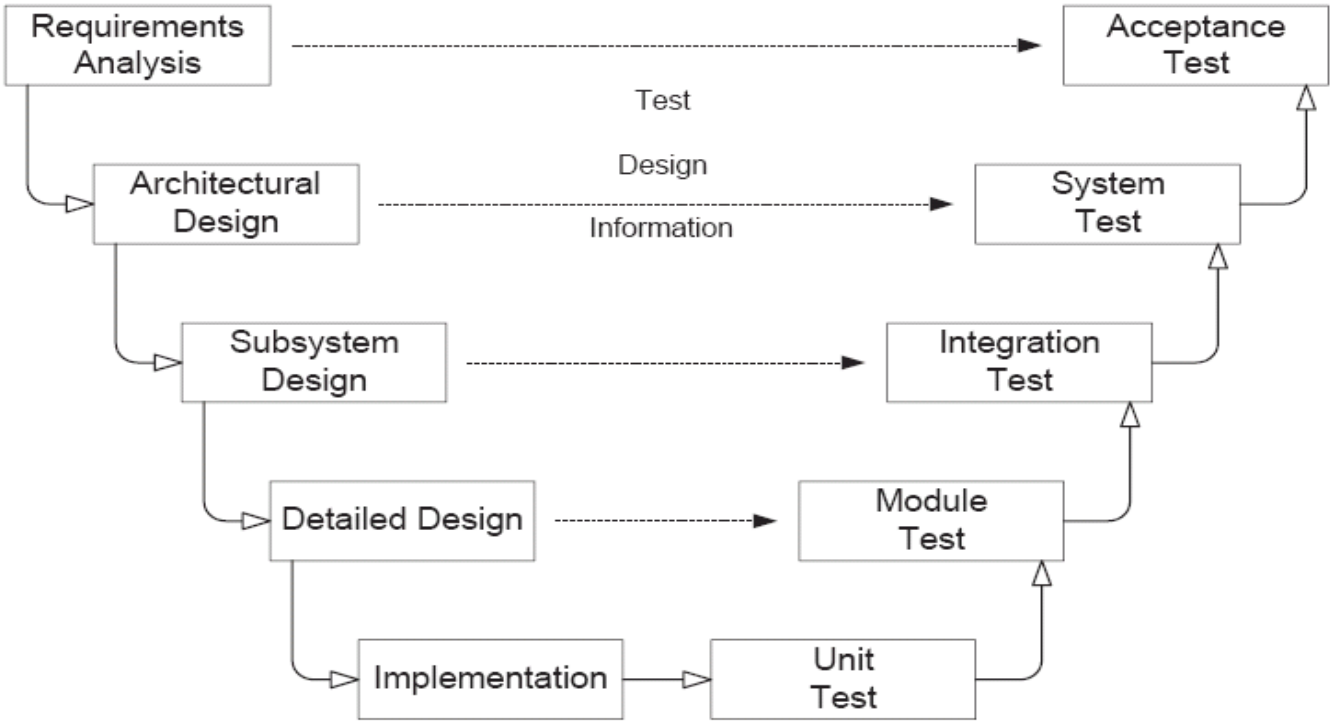
Received the ACM Turing Award at the age of 42 - the most prestigious computer science award

Tests should be designed to reveal execution failures

- not execution successes
- must be augmented with proper assertions



Software development activities and testing levels



Recall: Three Analysis Approaches

- Static Analysis: Validate without executing the program.
  - This include software inspections.
- Dynamic Analysis: Validate by executing the program with real inputs
  - Testing
- Repository Mining: Validate by the patterns/knowledge mined from various project repositories and forums

Software Faults, Errors & Failures

- Software Fault : A static defect in the software

Software faults are coding mistakes and hard to eliminate

- Software Error : An incorrect internal state that is the manifestation of some fault
- Software Failure : Observable incorrect behavior with respect to the requirements or other description of the expected behavior

Illustrative Example

```
public static int numZero (int[] x) {
// Effects: if x == null throw NullPointerException
// else return the number of occurrences of 0 in x
   int count = 0;
   for (int i = 1; i < x.length; i++) {
       if (x[i] == 0) {
             count++;
       }
   }
   return count;
}
```

Illustrative Example – Software Fault

```
public static int numZero (int[] x) {
// Effects: if x == null throw NullPointerException
// else return the number of occurrences of 0 in x
   int count = 0;
   for (int i = 1; i < x.length; i++) {
       if (x[i] == 0) {
             count++;
       }
   }
   return count;
}
```

Illustrative Example – Software Error

```
public static int numZero (int[] x) {
// Effects: if x == null throw NullPointerException
// else return the number of occurrences of 0 in x
   int count = 0;
   for (int i = 1; i < x.length; i++) {
       if (x[i] == 0) {
            count++;
       }
   }
   return count;
}
```

numZero([2, 7, 0])

Any failure occur?
Any error occur?

Illustrative Example – Software Error

```
public static int numZero (int[] x) {
// Effects: if x == null throw NullPointerException
// else return the number of occurrences of 0 in x
   int count = 0;
   for (int i = 1; i < x.length; i++) {
       if (x[i] == 0) {
             count++;
       }
   }
}
```

numZero([2, 7, 0])

Any failure occur?

Any error occur?

State error occurs at the first iteration where the state is (x=[2,7,0], count=0, i=1, PC=if). The correct state should be (x=[2,7,0], count=0, i=0, PC=if).

Illustrative Example – Software Error

```
public static int numZero (int[] x) {
// Effects: if x == null throw NullPointerException
// else return the number of occurrences of 0 in x
   int count = 0;
   for (int i = 1; i < x.length; i++) {
       if (x[i] == 0) {
             count++;
       }
   }
   return count;
}
```

numZero([0, 7, 2])

Any failure occur?
Any error occur?

Can you think of a program example and a test case that triggers a fault but causes no error and failure?

```
int square (int x) {
   int result = x + x;
   return result;
}
```

Test case: square(2)

Coincidental correctness occurs when a test triggers a fault but outputs a correct result.

Testing & Debugging

- Testing : Finding inputs that cause the software to fail

- Debugging : The process of finding a fault given a failure

Fault & Failure Model

Three conditions necessary for a failure to occur

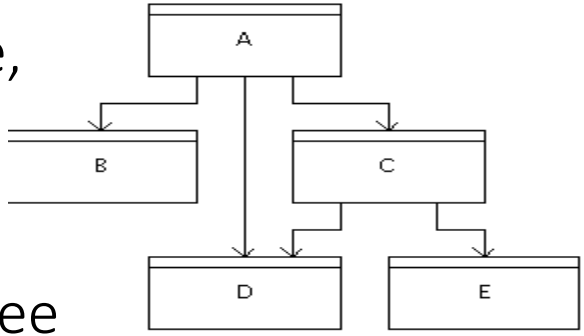
1. Reachability : The location or locations in the program that contain the fault must be reached
2. Infection : The state of the program must be incorrect
3. Propagation : The infected state must propagate to cause some output of the program to be incorrect

Test Case (=Test Case Values + Expected Results)

- Test Case Values (a.k.a. **test inputs**) : The input values necessary to complete the intended program execution of a test case.
- Expected Results : The result that will be produced when executing the test if the program satisfies its intended behavior.
- Test Set (or **test suite**) : A set of test cases.
- Test Script : A test case that is prepared in a form for automatic execution.

Top-Down and Bottom-Up Testing

- Top-Down Testing : Test the main procedure, then go down through procedures it calls



- Bottom-Up Testing : Test the leaves in the tree (procedures that make no calls) and move up to the root.
  - Each procedure is not tested until all of its children have been tested

White-box and Black-box Testing

- Black-box testing : Deriving tests from external descriptions of the software, including specifications, requirements, and design
- White-box testing : Deriving tests from the source code internals of the software, specifically including branches, individual conditions, and statements

A federal agency says an overreliance on Tesla's Autopilot contributed to a fatal crash

The National Transportation Safety Board met on Tuesday to determine the cause of May's fatal Tesla crash.

BY JOHANA BHUIYAN | @JMBOOYAH | SEP 12, 2017, 11:57AM EDT

TWEET SHARE LINKEDIN

The National Transportation Safety Board said that its investigation into a fatal Tesla crash showed that the limitations of the company's automated driver system, Autopilot, played a role in the May 2016 collision.

References

- How Google tests software? (a video by Google Engineering Director)
  - https://vimeo.com/47959189
- Inheritance, polymorphism and testing (a video from Google Tech Talk)
  - http://www.youtube.com/watch?v=4F72VULWFvc&feature=relmfu
- Your tests aren't flaky (a video from Google Tech Talk)
  - https://www.youtube.com/watch?v=hmk1h40shaE