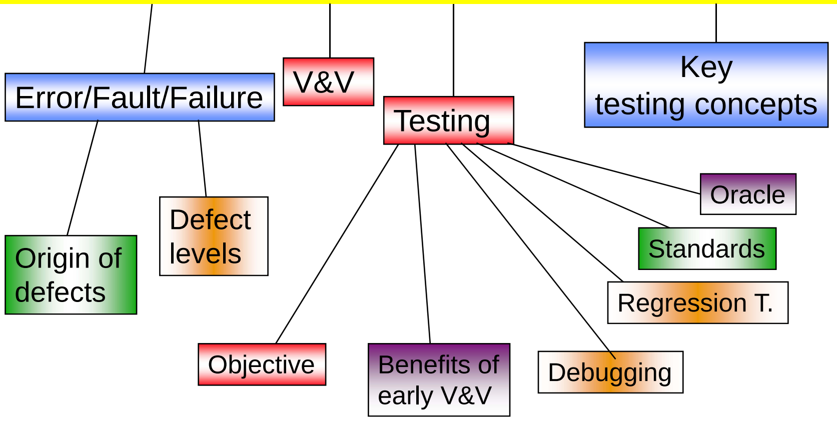
Course Structure

1. Software Quality Assurance
2. **Testing Fundamentals – part I**
3. Code-based Techniques
4. Specification-based Techniques
5. Inspection Technique
6. Test Tools
7. Measuring Software Quality
8. TDD

Fundamentals of Testing



Error/Fault/Failure

Origin of defects

Defect levels

V&V

Testing

Objective

Benefits of early V&V

Debugging

Regression T.

Standards

Oracle

Key testing concepts

"Too little testing is a crime, too much testing is a sin"

Learning Objectives

- Learn **error, fault** and **failure** and their relationship
- Know **verification** and **validation**, and understand their uses
- Learn key aspects of **testing:** test objectives, economic of early testing, comparison to debugging, regression testing, testing standards, and test oracle
- Understand **key testing concepts:** white-box vs. black-box testing, static vs. dynamic testing, manual vs. automated testing, and positive vs. negative testing,

Important Technology

▶ **Introduction**

Error, Fault, Failure

V&V

Testing

Key Concepts

Successful Projects

- ⌘ Accurate software measurement
- ⌘ Early use of estimating tools
- ⌘ Continuous use of planning tools
- ⌘ Formal progress reporting
- ⌘ Formal development methods
- ⌘ Formal design reviews
- ⌘ Formal code inspections
- ⌘ Formal risk management
- ⌘ **Formal testing methods**
- ⌘ Automated configuration control

Unsuccessful Projects

- ⌘ No historical software measurement data
- ⌘ Failure to use estimating tools
- ⌘ Don't use automated planning tools
- ⌘ Failure to monitor progress
- ⌘ Don't use effective dev. method
- ⌘ Don't use design reviews
- ⌘ Don't use code inspections
- ⌘ Don't use risk management
- ⌘ **Informal, inadequate testing**
- ⌘ Don't use formal configuration control

Reference: Jones, Capers, *Patterns of Systems Failure and Success*, International Thomson press, 1996.

Important Social Factors

Knowledge of Testing is key to Success!

Successful Projects

⌘ Realistic schedule expectations
⌘ Executive understanding of estimates
⌘ Excellent team communications
⌘ Experienced senior management
⌘ Capable project management
⌘ Capable technical staff
⌘ **Specialists** used for:
 ➔ Quality assurance
 ➔ **Testing**
 ➔ Planning
 ➔ Estimating

Unsuccessful Projects

⌘ Excessive schedule pressure
⌘ Executive rejection of estimates
⌘ Poor team communications
⌘ Naive senior management
⌘ Project management malpractice
⌘ Unqualified technical staff
⌘ **Generalists** used for:
 ➔ Quality assurance
 ➔ **Testing**
 ➔ Planning
 ➔ Estimating

Errors

Introduction

▸ **Error, Fault, Failure**

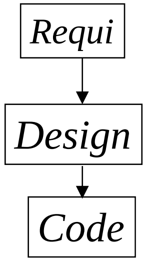
V&V

Testing

Key Concepts

Human make errors in their thoughts, actions and making decisions.
Errors are a part of our daily life.

Software production can be seen as a series of imperfect translation processes. Each translation produces a work product or deliverable.
Errors are introduced when there is a failure to <u>completely and accurately translate</u> one representation to another, or to fully match the solution to the problem.



Errors, Faults, Failures

Introduction

▸ **Error, Fault, Failure**

V&V

Testing

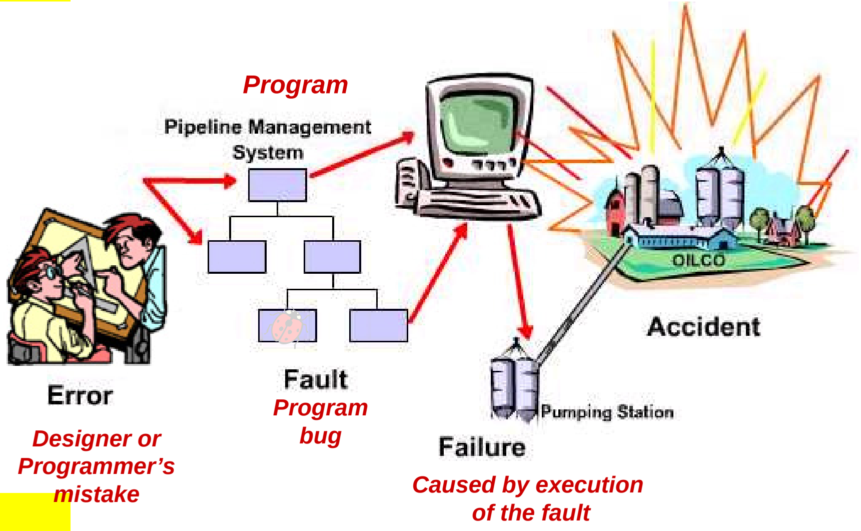
Key Concepts

Error: a mistake in the design or programming; made by a human
Fault: a mistake in the <u>code</u>; the result of an error
Failure: the occurrence of a software fault

Users don't observe errors or faults. They observe <u>execution failures</u>.

H. Mills

An example of Error, Fault, Failure


Program
Pipeline Management System
OILCO
Accident
Pumping Station
Error
Designer or Programmer's mistake
Fault
Program bug
Failure
Caused by execution of the fault

Example: A program to calculate salary

Introduction
Error, Fault, Failure
▶ Example
F & F
Failure Levels
V&V
Testing
Key Concepts

Design:

Salary = Number of Days Worked x DailyRate

Code:

```
Read (DayWorked)
Read(DailyRate)
Salary = DayWorked x DailyRate
Output (Salary)
```

Suppose, Tom has worked 21 days

His Rate is $1000/day.

He also worked one day overtime.

He received only $21,000!

The output is wrong!!

He should also receive payment for the overtime work!

Example

Introduction
Error, Fault, Failure
▶ Example
F & F
Failure Levels
V&V
Testing
Key Concepts

In this example, Tom, the user, sees the **failure** (wrong output)

Where is the **fault**?

In the *program statement*:

Salary = DayWorked x DailyRate

It should be

Salary = DayWorked x DailyRate + OverTimePay

Why the **fault**?

Because there is a design mistake – a human error!

2 Types of Faults

Introduction
Error, Fault, Failure
Example
▶ F & F
Failure Levels
V&V
Testing
Key Concepts

Fault of omission: occur

(1) when there is something missing which should be in the code (e.g., missing a statement to initialize the variable), or

(2) when we fail to enter correct information (e.g., a function has not been implemented, or a missing value for a variable).

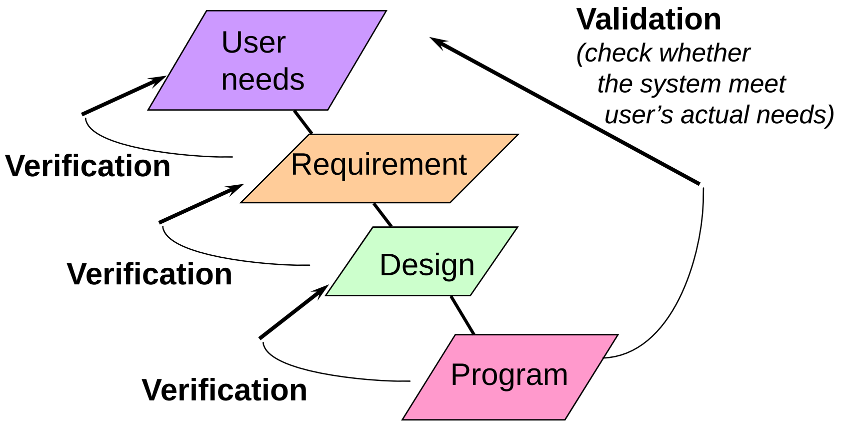
Fault of commission: occur when we enter something into the code that is incorrect (wrong code).

A failure occurs when a fault executes.

How to prevent failure?

- Don’t make mistake
- Ask someone do checking for us
- Use more tools (automation)
- Do V&V

Verification & Validation



User needs

Requirement

Design

Program

Validation
(check whether the system meet user's actual needs)

Verification

Verification

Verification

V & V

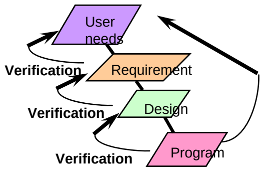
Focus: V&V focus on either the software development process or the products

- ⌘ Include static analysis (e.g., inspection, review)
- ⌘ Include testing
- ⌘ Analyse development products (e.g., design document, test plan, not just code)
- ⌘ Best perform throughout the development cycle

Verification

"The process of comparing 2 levels of work product for proper correspondence"

Compare: spec. to requirement
requirement to design
design to program



User needs

Requirement

Design

Program

Verification

Verification

Verification

ISO: "The process of evaluating the products of a given phase to ensure correctness and consistency with respect to the products and standards provided as input to that phase"

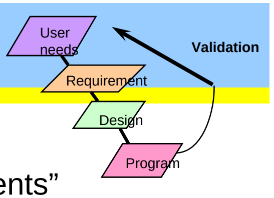
Are we building the product right?

Verification Tasks (from ISO 12207)

- ⌘ Contract verification
- ⌘ Process verification
- ⌘ Requirement verification
- ⌘ Design verification
- ⌘ Code verification (e.g., unit testing)
- ⌘ Integration verification (e.g., integration testing, system testing)
- ⌘ Documentation verification

- ⌘ technical reviews, walkthroughs and inspections
- ⌘ checking that software requirements are traceable to user requirements
- ⌘ checking that design components are traceable to software requirements
- ⌘ check formal proofs and algorithms
- ⌘ acceptance testing
- ⌘ audits

Validation



Introduction
Error, Fault, Failure
V&V
Verification
▸ Validation
Testing
Key Concepts

"The process of assuring that the final product satisfies the system requirements"

ISO: "The process of evaluating software to ensure compliance with specified requirements from users"

Are we building the right product?

Examples

- In design **<u>verification</u>**, we check the design against the system requirements to see whether the design will <u>meet the requirements</u>. E.g., can complete the transaction in 10 seconds.
- In design **<u>validation</u>**, we check with ***the user*** to see whether the system to be developed based on the design will <u>meet the user's needs</u>. E.g., can do exactly what the user wants, whenever he wants.

Summary of V&V

Introduction
Error, Fault, Failure
V&V
Verification
Validation
Testing
Key Concepts

- Both V&V use the <u>same techniques</u> (such as testing, review, inspection, etc.)
- The key difference is what is used as the **"reference for correctness"**
- In verification, we use **<u>the output from the previous step</u>** as the reference of correctness (example: when we verify a code, we check it against the design)
- In validation, we always use the **<u>user needs</u>** as the reference of correctness (example: when we validate a code, we check it against the *user needs and expectation*)

Different Objectives of Verification & Validation

Introduction
Error, Fault, Failure
V&V
Verification
Validation
Testing
Key Concepts

Validation ensures we are ***doing the right things***.

- It demonstrates that the system is <u>fit for use</u> for the set of specified requirements.
- Exit criterion: the complete coverage of requirements.
- A smarter validation process would also demonstrate that the system does not possess undesirable behavior by designing some tests for typical undesirable behavior.

Verification ensures that we are ***doing things right***.

Unit testing, integration testing, static code analysis, design and code inspections are all common verification techniques.

What is Testing?

Introduction
Error, Fault, Failure
V&V
▸ Testing
Key Concepts

- Testing is the **<u>process</u>** of executing software to answer:

  "Does the software behave as specified?"

- This implies that we have a specification,
- Or we have some property that we wish to test for <u>independently of the specification</u>, e.g., "all statements in the code have been executed."

  - Testing provides insight into the quality of the software
  - Based on this insight, organizations can make informed decisions about whether to release the software for operation.

Typical Problems with Testing

- Too many defects are found and reported by customers.
- Testing takes too long and delays delivery.
- Testing is very expensive. (Testing activities may consume 50% development effort.)
- It is difficulty to find volunteer users to test.

- It is impossible to completely test any nontrivial module or any system – **Why?**

1. Theoretical limitations: Halting problem
2. Practical limitations: not enough time and cost

Why Some Errors Go Undetected?

Fail to look

- developers believe the system works
- ensure outcome match beliefs

Fail to see

- developers are familiar with system as specified, and therefore cannot see problems

Inattentional blindness

- Humans (often) don't see what they don't pay attention to.
- We paid attention to the wrong conditions.
- But we can't pay attention to all the conditions

We need independent Testing!

Why Exhaustive Testing (*test everything*) is Impractical?

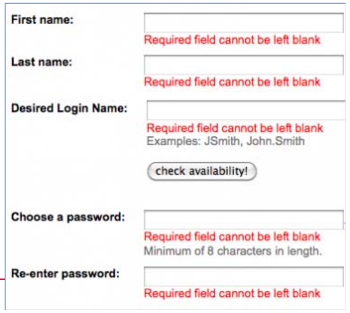
1. Too many combinations

Software system supports variable input

- Input such as integers, real numbers, character strings has many possible values that could be tested!!
- Example: Name can have hundreds or thousand versions.

Order of inputs

- Order of inputs can be arranged into an infinite number of combinations



First name:
Required field cannot be left blank

Last name:
Required field cannot be left blank

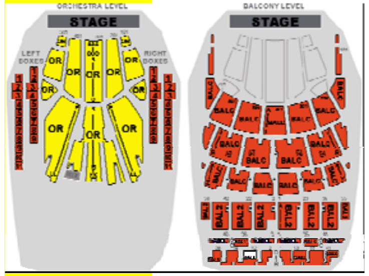
Desired Login Name:
Required field cannot be left blank
Examples: JSmith, John.Smith

Choose a password:
Required field cannot be left blank
Minimum of 8 characters in length.

Re-enter password:
Required field cannot be left blank

Impossible to do complete testing!

Example: Ticket reservation system



Accepts 4 different kinds of inputs:

- Seat location (5 areas)
- Class of customer (senior, adult, children, VIP)
- Date (weekday day, weekday evening, weekend day, weekend evening, holiday day, holiday evening)
- Type of performance (8 types)

Exhaustive testing (trying all combinations) would require 5 x 4 x 6 x 8 = 960 test cases.

Question:

How many variables in your applications (e.g., CRM)?

Can you test all their combinations?

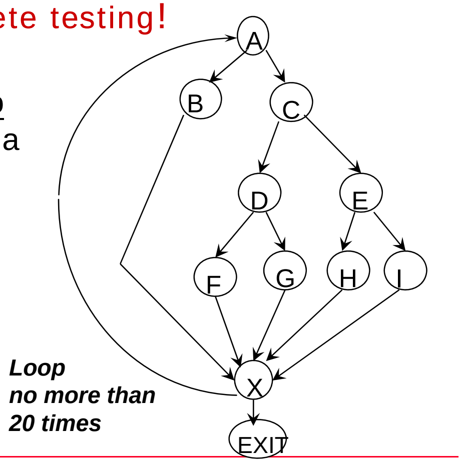
Why Exhaustive Testing is Impractical?

2. **Many sequences of execution** (in the software)

Impossible to do complete testing!

An example that shows too many paths to test in even a fairly simple program.

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



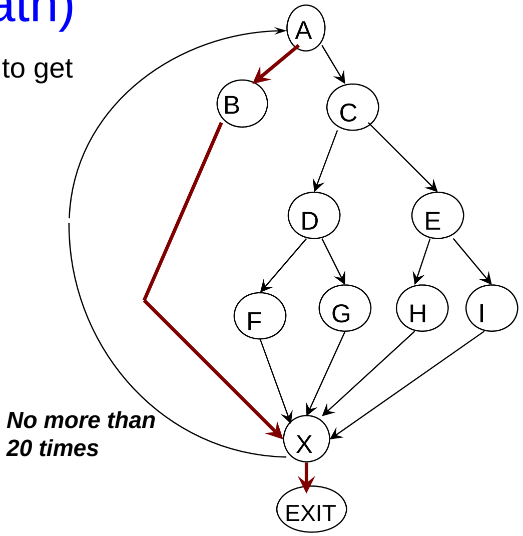
One Execution Sequence (or program path)

There are 5 ways to get from A to X.

One of them is

ABX--EXIT

A program path

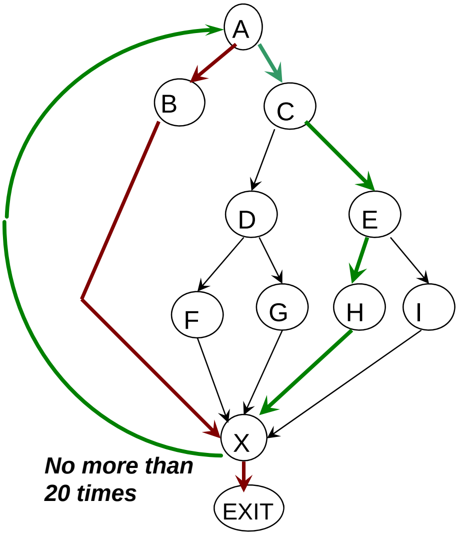


If we go through the loop 2 times:

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 for reaching EXIT by passing through X twice.

Total number of Sequences:

- At most 20 times thru the loop:
- There are $5^1 + 5^2 + ... + 5^{19} + 5^{20}$ = 10^{14} = 100 trillion paths through the program.
- It takes a **billion** years to test all paths, if we could write, execute, verify a test case every 5 minutes.



NO Guarantee

Testing is really sampling from an *infinite input population*.

Test engineers can only test a small subset of input space before product release.

Users have available to them the entire infinite input space.

Therefore, **software producers cannot certify that their software contains no faults.**

What should be the Objective of Testing?

Testing Objectives (from Hetzel)

Some possible objectives of testing:

- Checking programs against specifications
- Finding bugs in programs
- Determining user acceptability
- Insuring that a system is ready for use
- Gaining confidence that it works
- Showing that a system performs correctly
- Demonstrating that errors are not present
- Understanding the limits of performance
- Learning what a system is not able to do
- Evaluating the capabilities of a system
- Verifying documentation

Practical Testing Objective

'Test to break'

Executing a program with the intention of finding errors (make the program fail).

A successful test is one that finds a fault.

Random Error

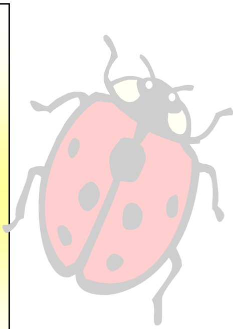
Windows 95 has detected a random error. This error occurs every once in a while. Please wait.

Error

Some sort of error

Origin of Software Defects

IBM (OS/360)	Design errors	45%
	Coding errors	25%
	Bad fixes	20%
	Documentation errors	5%
	Admin errors	5%
TRW Corp.	Design errors	60%
	Coding errors	40%
Mitre Corp.	Design errors	64%
	Coding errors	36%



Most errors occur in Design!!

Similar results in your organization?

Defect Propagation

Introduction

Error, Fault, Failure

V&V

Testing

Objectives

▶ **Early V&V**

Debugging

Regression Test

Test Standards

Oracle

Key Concepts

- ⌘ 1 defect in requirements can result in 3-15 defects in design
- ⌘ 1 defect in design can result in 2-10 defects in code

Worst case: 1 defect in requirements causes 15 defects in design and they in turn create 10 defects in the code.

Total -> **150 code defects**!

How many can you find in testing?

Origin of Defects

Introduction

Error, Fault, Failure

V&V

Testing

Objectives

▶ **Early V&V**

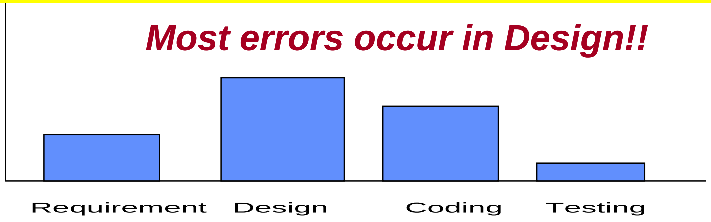
Debugging

Regression Test

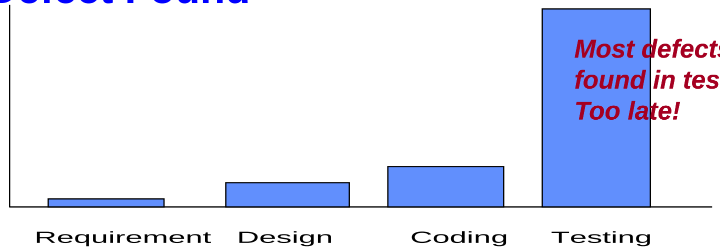
Test Standards

Oracle

Key Concepts


Most errors occur in Design!!

Defect Found


Most defects are found in testing. Too late!

Cost to fix Defects

Introduction

Error, Fault, Failure

V&V

Testing

Objectives

▶ **Early V&V**

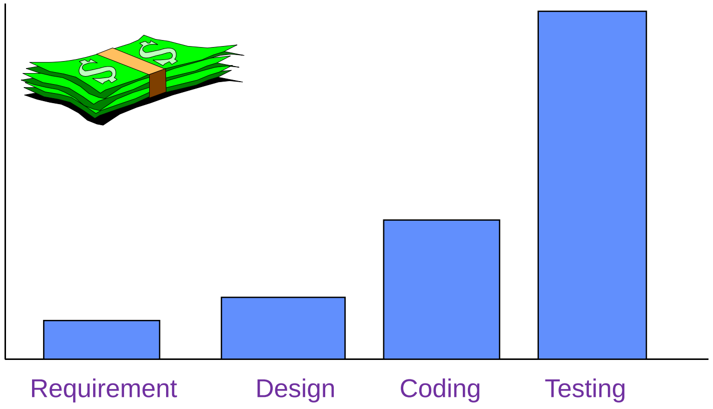
Debugging

Regression Test

Test Standards

Oracle

Key Concepts



- Many errors are **made in early phases** (requirement and design)
- But, these errors are discovered late
- Repairing those errors is costly ⇒ It pays off to start testing early

We should detect and fix defects as early as possible.

Example from Cisco (2004)

Introduction

Error, Fault, Failure

V&V

Testing

Objectives

▶ **Early V&V**

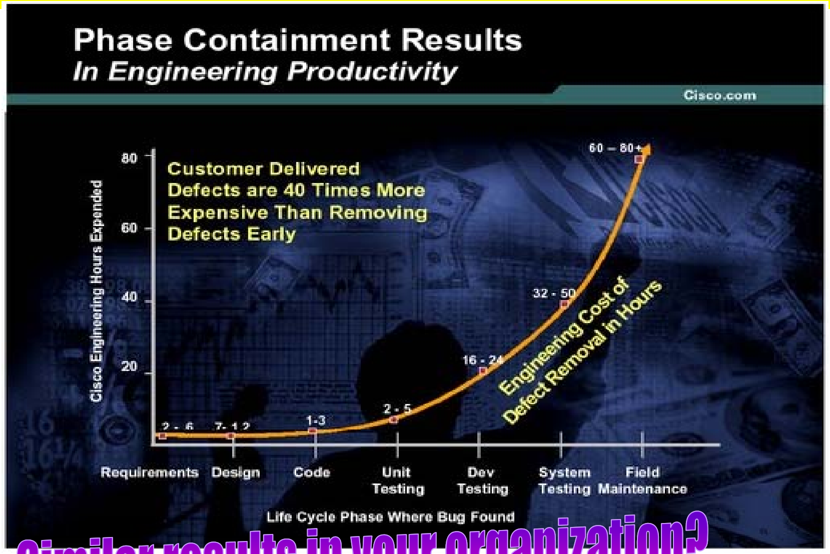
Debugging

Regression Test

Test Standards

Oracle

Key Concepts


Phase Containment Results
In Engineering Productivity

Customer Delivered Defects are 40 Times More Expensive Than Removing Defects Early

Similar results in your organization?

Defect Removal Effectiveness

Introduction
Error, Fault, Failure
V&V
Testing
Objectives
▶ Early V&V
Debugging
Regression Test
Test Standards
Oracle
Key Concepts

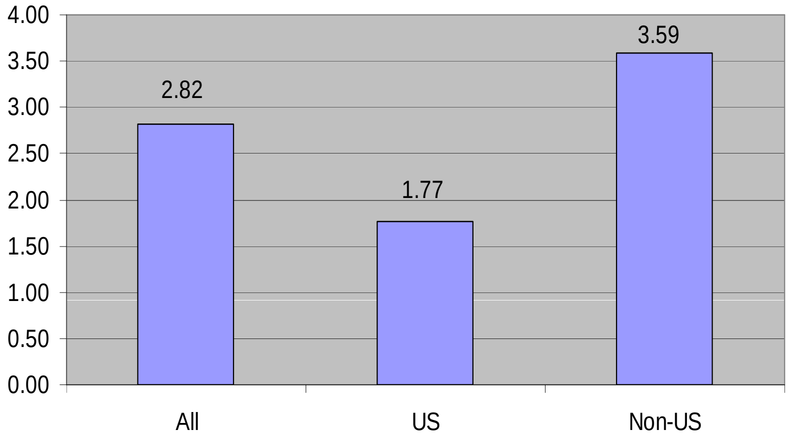
DRE

Activity	Low (%)	High (%)
Informal design review	25	40
Design inspection	45	65
Informal code review	20	35
Code inspection	45	70
Unit test	15	50
Regression test	15	30
Integration test	25	40
System test	25	55
Test of new function	20	35

Reference: Jones, Capers, "Software defect-removal efficiency", *IEEE Computer.* 5/96, pp. 94-96.

Defect Rate per KLOC at Release

Introduction
Error, Fault, Failure
V&V
Testing
Objectives
▶ Early V&V
Debugging
Regression Test
Test Standards
Oracle
Key Concepts



Reference: Rubin, H., *Worldwide IT trends and benchmark report 2001*, www.metricnet.com/analysis2001.pdf

Where do you stand?

Debugging ≠ Testing

Introduction
Error, Fault, Failure
V&V
Testing
Objectives
Early V&V
▶ Debugging
Regression Test
Test Standards
Oracle
Key Concepts

What is debugging?

- ⌘ It is the process of locating the exact cause of a defect (**fault**), and **removing the fault**.
- ⌘ Debugging can only be effectively performed by developers or analysts. Only they have the knowledge of and experience with both the software and environment.
- ⌘ Debugging can eat up 60-70% of the overall development effort.

Testing detect defects

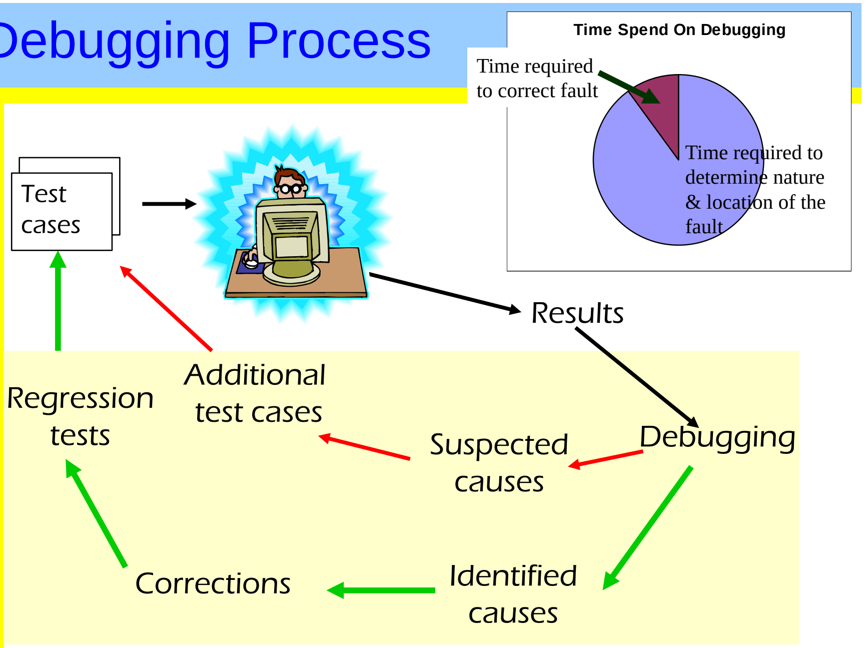
Debugging ≠ Testing

Introduction
Error, Fault, Failure
V&V
Testing
Objectives
Early V&V
▶ Debugging
Regression Test
Test Standards
Oracle
Key Concepts

	Debugging	Testing
Purpose	Eliminate bugs	Evaluate quality risks and guarantee an expected quality
Subject	Program	System (including software)
Work	Correct faults	Identify faults
Worker	Programmer	Test engineer

> ***"Act in haste and repent at leisure; code too soon and debug forever."*** *Raymond Kennington*

Debugging Process



Debugging steps

After a bug is found, a tester must:
- Record the bug into the **defect tracking system**

After the bug is fixed, a tester do regression testing:
a. Verify that the bug was really fixed
b. Check to see if new bugs have been introduced during bug fixing.

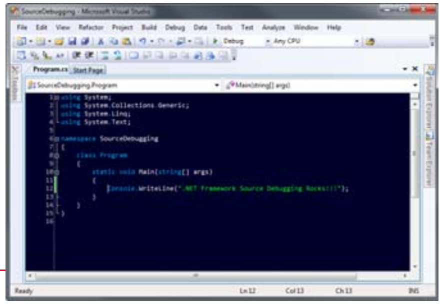
Priority is the magnitude of a bug's impact on the company's business.

- Severity refers to the **technical** aspect of a bug.
- Priority refers to the **business** aspect of the bug.

Severity uses technical criteria to grade the bug, while Priority uses business criteria. It's almost always clear which severity to assign to the bug, while the Priority of a bug is often the subject of arguments and political reasons.

Why Is Debugging So Difficult?

- Symptom & cause may be geographically separated
- Symptom may disappear (temporarily) when another error is corrected
- Symptom may be caused by human error that is not easily traced
- Symptom may be a result of timing problems, but not processing problems


SourceDebugging - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Data Tools Test Analyze Window Help
Debug
Any CPU
Program.cs
Start Page
SourceDebugging.Program
Main(string[] args)
Ready
Ln 12
Col 13
Ch 13
INS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SourceDebugging
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(".NET Framework Source Debugging Rocks!!!");
        }
    }
}
```

Debugging Approaches

- **Brute force testing**
  - Take memory dumps, invoke run-time traces, add many WRITE statements
- **Backtracking**
  - From where a symptom has been found, trace the source code backward
- **Cause elimination**
  - Devise a cause hypothesis, and use data to prove or disapprove hypothesis, i.e., use induction, deduction, binary partitioning

Defect Categories / Consequences

Introduction
Error, Fault, Failure
V&V
Testing
Objectives
Early V&V
▶ Debugging
Regression Test
Test Standards
Oracle
Key Concepts

Categories:

- Function-related
- System-related
- Data
- Coding
- Design
- Documentation
- Standards violations
- Logical
- UI

Consequences (severity):

- ❖ Fatal
- ❖ Serious
- ❖ Normal
- ❖ Minor

A classification of Failures from an IBM lab

Severity levels	Description
1 *(Fatal)*	Customer is unable to use the program, which has a critical impact on his/her operation. Require immediate solution
2 *(Serious)*	Customer is able to use the program, but his/her operations are severely restricted by the problem
3 *(Normal)*	Customer is able to use the program with some restrictions on the functions that he/she can use. No critical impact on operation
4 *(Minor)*	The problem causes little or no impact.

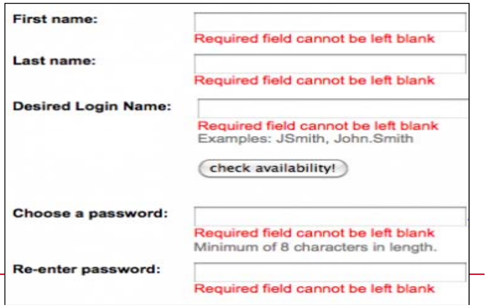
UI defects

Introduction
Error, Fault, Failure
V&V
Testing
Objectives
Early V&V
▶ Debugging
Regression Test
Test Standards
Oracle
Key Concepts

- A UI defect is a defect in **how the software *presents* the information**.
- UI defects range from **visual problems** like
  - *a link on the Web page has a wrong color*

  to **interactive problems** like
  - *it's hard for a user to figure out how to use a function.*

If users cannot figure out how to put a book into the shopping cart, then the perfectly working code of the application doesn't matter.



First name:
Required field cannot be left blank

Last name:
Required field cannot be left blank

Desired Login Name:
Required field cannot be left blank
Examples: JSmith, John.Smith
check availability!

Choose a password:
Required field cannot be left blank
Minimum of 8 characters in length.

Re-enter password:
Required field cannot be left blank

Logical defects

Introduction
Error, Fault, Failure
V&V
Testing
Objectives
Early V&V
▶ Debugging
Regression Test
Test Standards
Oracle
Key Concepts

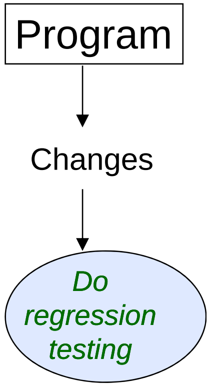
- A logical defect is a defect in **how the software *processes* information.**
- **Logical defects are the primary focus of software testers** because
  - harder to find logical defects than UI defects;
  - The consequences of releasing logical defects are much more severe than the consequences of releasing UI defects.

A favorite developer's expression, "It's not a defect, it's a feature", in human language sounds like, "That something is not a problem with my code. It works (and/or looks) exactly as I want."

Regression Testing

Introduction
Error, Fault, Failure
V&V
Testing
Objectives
Early V&V
Debugging
▶ Regression Test
Test Standards
Oracle
Key Concepts

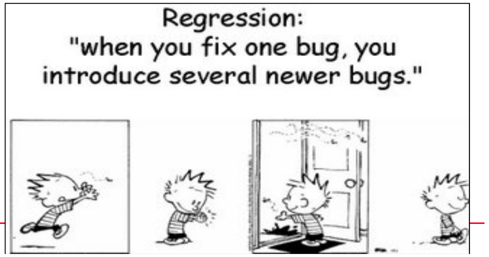
- Whenever a fault is detected and fixed, the system should be re-tested to ensure that the original fault has been successfully removed.
- Also perform RT when the environment is changed.
- Studies show that compared to new code, changed programs are <u>10 times more likely</u> to contain defects.
- RT attempts to verify that modifications have not caused <u>unintended adverse side effects</u> in the unchanged system and that the modified system still meets its requirements.



Regression Testing

- The tests that are re-run are called regression tests
- It is too expensive to re-run every single test case every time a change occurs. Only a **subset** of the previously-successful test cases is actually re-run.
- RT should be automated.
- We should eliminate redundant test cases in the regression test suite.

  As the system evolves, some test cases may not be effective in detecting problems and become obsolete if their requirements have been changed.

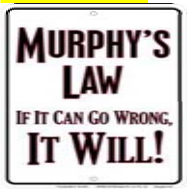


Why is Regression Testing important?

Because: Many Long Live Systems

Example:

- B52 airplane enters services in 1955.
- Expected to last till 2045! (90 years old)
- We sometimes forget the life span of the hardware that our software drives.
- Suppose you are working on avionics software today, can you imagine someone trying to regression test your software in 2096?



"Bugs will appear in one part of a working program immediately after an 'unrelated' part of the program is modified." *Murphy's Law*

Regression Testing vs Retest

Regression testing

- Execute test cases on modified system (from bug fixing or adding new functions) to ensure bug fixes work and did not cause side effects, or new functionalities work.
- May include new test cases

Retest

- Execute the test cases which have identified a bug.
- Try to see whether the bug has been fixed or not.
- Involve old test cases.

Regression Testing vs Retest

First

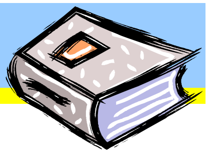
Second

Example

- 2 push buttons: button1 and button2. On clicking button1, the text of the textbox test1 should be 'First', and on clicking buttom2, the text of the textbox test2 should be 'second'
- A bug is detected when button1 is clicked, 'second' is shown.
- The tester reported this bug, and the code was changed.

- Now if the tester clicks only the button1 and checks whether the text of the textbox text1 is 'First', then it is **retesting**.
- If after the tester had checked button1, he also clicks button2 to check both buttons, then it is **regression testing**. Because the tester is checking whether changes made in one part of the system has not affected other part of the system.

Standards on Testing



IEEE Test Standard: http://www.ieee.org/

a) Std. 829-1983 'Software Test Documentation';
b) Std. 1008-1987 'Software Unit Testing';
c) Std. 1012-1986 'Software Verification and Validation Plans';
d) Std. 730-1984 'Software Quality Assurance Plans';
e) Std. 983-1984 'Software Quality Assurance Planning'

Test standards and policies are essential for improving testing and to ensure consistency in testing. Guidelines on test techniques and strategies are defined in the test standard.

Standards on Testing

ISO 9001 on Testing, http://www.iso.org/

a) Software testing includes test plan preparation and review, test data preparation and review, and review of test results.
b) Corrective actions to fix causes of defects.
c) Reassessment of tools, techniques and methodologies used in software production.
d) Programming standards (including testing standards) that describe approved practice and list any prohibited practice.
e) Evaluation of customer-supplied software products and purchased products.
f) Check that test processes are adhered to.
g) Test processes improved where required.

Standards on Testing

SW-CMM on Testing, http://www.sei.cmu.edu/

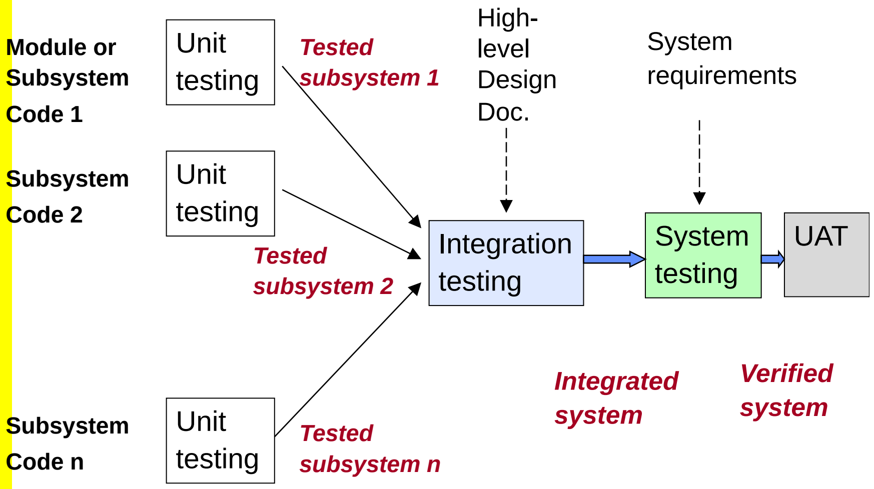
See 'Software Product Engineering' KPA, and 'Training Program', 'Technology Change Management', 'Process Change Management' KPA.

a) Require 4 testing levels: unit, integration, system and acceptance testing.
b) Regression testing should be done to ensure changes are correct.
c) The test plan should be reviewed.
d) Proper training of testers should be conducted for better job performance.
e) Software process standards must be maintained.
f) A system test group should be responsible for performing an independent system test.
g) Test processes should be continually improved.

Summary: Standards on Testing

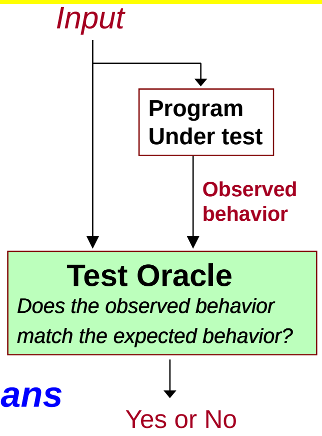
Test Standards		**IEEE**	**CMM**	**ISO9001**
Category	*Sub-category*			
Test phases	unit test	R	R	
	integration test	R	R	
	system test	R	R	
	user acceptance test	R	R	
Test activities	test planning	R	R	R
	test scheduling	R	R	
	test case dev.	R	R	R
	test execution	R	R	R
	test result reporting	R	R	R
	regression test	R	R	
Test Management	test plan review	R	R	R
	test staff training	R	R	R
	test documentation	R	R	

Testing Phases (Levels)



Test Oracles

- Test Oracle helps us deciding whether a test case succeeds or fails (i.e., the program output or behavior is correct or not)



"The system works" really means

"...it appeared to meet some requirement to some degree."

Example

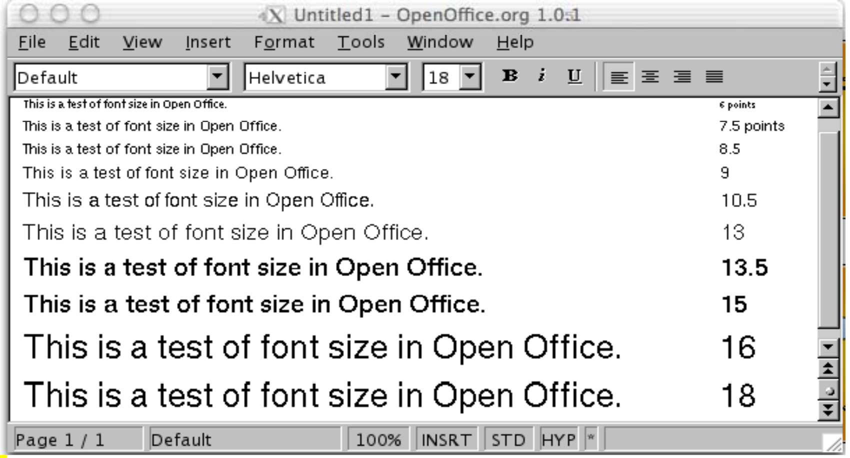
Suppose the Test output of cos(0.5) =0.8775825619

How do we decide whether this answer is correct?

We need an oracle, such as:

- Look up cosine of 0.5 in a book
- Check the answer with a calculator
- Compute the value using Taylor series expansion.

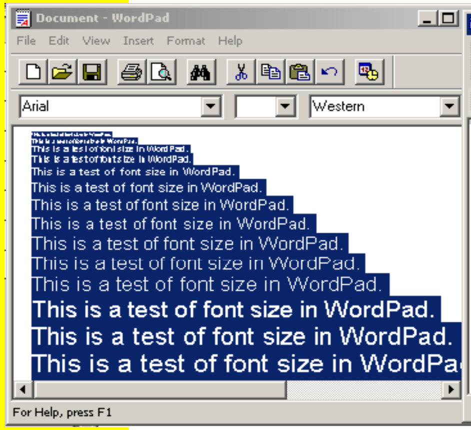
Example: Does font size work in Open Office? *What can be used for the oracle?*


Untitled1 – OpenOffice.org 1.0.1
File Edit View Insert Format Tools Window Help
Default
Helvetica
18
B i U
Page 1 / 1
Default
100%
INSRT
STD
HYP

This is a test of font size in Open Office.	6 points
This is a test of font size in Open Office.	7.5 points
This is a test of font size in Open Office.	8.5
This is a test of font size in Open Office.	9
This is a test of font size in Open Office.	10.5
This is a test of font size in Open Office.	13
This is a test of font size in Open Office.	**13.5**
This is a test of font size in Open Office.	**15**
This is a test of font size in Open Office.	16
This is a test of font size in Open Office.	18

Oracle: Consistent with comparable product
Check against MS Word

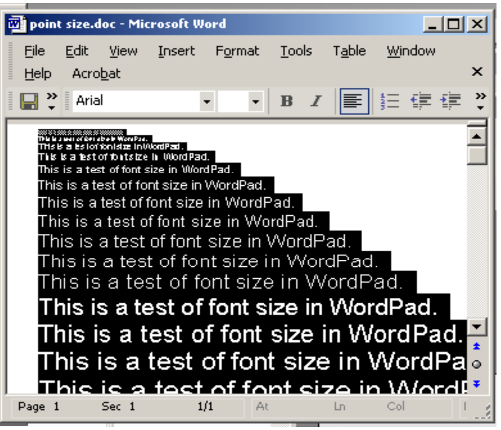
WordPad



This is a test of font size in WordPad.
This is a test of font size in WordPad.
This is a test of font size in WordPad.
This is a test of font size in WordPad.
This is a test of font size in WordPad.
This is a test of font size in WordPad.
This is a test of font size in WordPad.
This is a test of font size in WordPad.
This is a test of font size in WordPad.
This is a test of font size in WordPad.
This is a test of font size in WordPad.
This is a test of font size in WordPa

For Help, press F1

Word


point size.doc - Microsoft Word
File Edit View Insert Format Tools Table Window
Help Acrobat
Arial
B I
Page 1 Sec 1 1/1 At Ln Col

This is a test of font size in WordPad.
This is a test of font size in WordPad.
This is a test of font size in WordPad.
This is a test of font size in WordPad.
This is a test of font size in WordPad.
This is a test of font size in WordPad.
This is a test of font size in WordPad.
This is a test of font size in WordPad.
This is a test of font size in WordPad.
This is a test of font size in WordPad.
This is a test of font size in WordPad.
This is a test of font size in WordPa
This is a test of font size in WordP

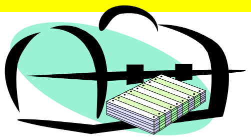
Other useful oracles

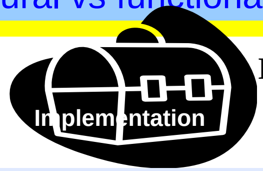
- **Consistent within Product:** Function behavior is consistent with behavior of comparable functions or functional patterns within the product.
- **Consistent with History**: Present behavior is consistent with past behavior.
- **Consistent with our Image:** Behavior is consistent with an image that the organization wants to project.
- **Consistent with Claims:** Behavior consistent with documentation or ads.
- **Consistent with Specifications or Regulations:** Behavior is consistent with claims that must be met.
- **Consistent with User's Expectations:** Behavior is consistent with what we think users want.
- **Consistent with Purpose:** Behavior is consistent with purpose.

Concept 1: Structural vs functional testing

Introduction
Error, Fault, Failure
V&V
Testing
▶ Key Concepts

White -box





Black-box

Structural testing (ST)

⌘ test cases derived from the program's **internal structure**

⌘ ST ensures sufficient testing of the **implementation**

⌘ ST primarily used during the coding phase

Functional testing (FT)

⌘ test cases derived from the program's **function**

⌘ FT not concerned with how processing occurs, but with the results of the processing

⌘ FT ensures that the requirements are properly satisfied.

Concept 2: Dynamic vs static testing

Introduction
Error, Fault, Failure
V&V
Testing
▶ Key Concepts

Dynamic analysis

➔ looks at the behavior of software while it is executing, to provide info. such as execution traces, timing profiles, and test coverage information.

➔ dynamic tests used in the test phase

Static analysis

➔ does not involve actual program execution; involve the analysis (reading) of the system representations, e.g, requirement, design documents, program listing, looking for problems and gathering metrics
Examples: syntax checking, inspections, walkthroughs, analysis and formal verifications

➔ Generally, static tests are used in the requirements and design phase

We will study these techniques

	Black box (functional)	White box (structural)
Dynamic (testing)	Decision table, BV Equivalence Partition OA FSM Cause-Effect Graph	Dataflow testing Domain testing Path-based testing Basis path testing Mutation analysis
Static	Specification proving	Code Walkthroughs Inspections Program proving Symbolic execution Anomaly analysis

When to use What Technique?

Introduction

Error, Fault, Failure

V&V

Testing

▸ **Key Concepts**

Development stage	V&V technique
Requirement	review/ inspect of requirements
Design	review/ inspect of design doc.
Coding	review/ inspect of source code
Unit test	mainly white-box testing, also some black-box
Integration test	white-box and black-box testing
System test	black-box testing
User acceptance test	black-box testing

Concept 3: Manual vs automated testing

Introduction

Error, Fault, Failure

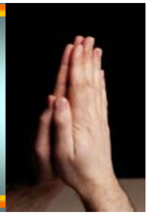
V&V

Testing

▸ **Key Concepts**

- ⌘ Manual techniques are performed by **people**; e.g., code inspection
- ⌘ Automated techniques by the **computer** (use test tools)
- ⌘ The more automated the developmental process, the easier it becomes to automate the test process.

"Software and cathedrals are much the same - first we build them, then we pray." *Samuel T. Redwine, Jr.*



Concept 4: Positive vs Negative Testing

Introduction

Error, Fault, Failure

V&V

Testing

▸ **Key Concepts**

- ⌘ **Positive testing**: test cases are designed to test that the software does what it is supposed to do. The software is used in a normal, error-free way, and the system is assumed to be working fine.
- ⌘ **Negative testing**: test that the software does not do things that it is not supposed to do.
  - ➔ Check situations that involve user error and/or system failure
  - ➔ Try to design test cases which subject each state to each input in the total set of inputs, not just the legal input for that state.
  - ➔ E.g. *press many keys at the same time*.

We should always apply both positive and negative testing!

Positive Testing

⌘ Exercise the application with valid input and verify the outputs are correct.

Example: A word processing application

A positive test for the printing function:

print a document containing both text and graphics to a printer that is online, filled with paper and for which the correct drivers are installed.

Negative Testing

⌘ Exercise the application using some invalid inputs, unexpected operating conditions, erroneous initial conditions, abnormal scenarios.

A negative test for the printing function:

disconnect the printer from the computer while a document is printing.

What can happen?

1. An message appears, informing the user about the situation.
2. The application crashes because the 'abnormal' loss of communications with the printer.

Negative Testing

Introduction

Error, Fault, Failure

V&V

Testing

▸ **Key Concepts**

1. **Negative testing finds more bugs**. Why?

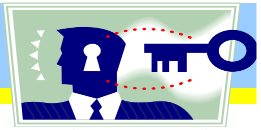
a. Errors and failures can take many shapes and forms, so the programmer might not predict some of them, and thus the code will not be ready to handle certain abnormal situations.

b. When writing and developing features, it is natural to concentrate on the normal usage and normal functioning of the software.

2. Negative testing involves more creativity and puzzle-solving than positive testing. Errors and failures can take many shapes and forms.

3. Both negative and positive tests must be performed as a part of functional testing, but we must execute positive tests **first**. Why? If functionality doesn't work during normal usage, it doesn't really make sense to check if it works with abnormal usage. It's like checking to see if a corpse has chicken pox.

Summary



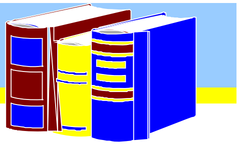
The Power of Testing

- We are blind to our own errors (need independent V &V).
- Any defect removal process only finds some, not all defects.
- Good testing can never save a poor quality program.
- All tests should be traceable to customer requirements.
- Faults tend to cluster together.

80% of defects are traceable to 20% of the modules (Pareto principle - **Error prone modules**: regions of the software with very high error densities).

- Verification checks whether we have built the product right.
- Validation checks whether we have built the right product.
- Testing can be used for both verification and validation.

References



- Black, R., Pragmatic Software Testing: Becoming An Effective and Efficient Test Professional, Wiley, 2007.
- IEEE Std. 829, 1008, 1012, 1059, 1028, 1044
- Whittaker, J. A., *How to break software*, Addison-Wesley, 2003.
- How Google do their testing? http://googletesting.blogspot.com/
- Software Testing and Quality Bookshelf http://www.soft.com/Institute/QualitySource/name.list.html
- Software QA Testing and Test Tool Resources, http://www.aptest.com/resources.html
- http://www.mtsu.edu/~storm/

Supplementary Notes

Definitions

Test Phases: A ‘test phase’ is a level of testing in the software life cycle where certain components of the system are tested to check for compliance with requirements.

Test Activities: Test planning involves specifying the general approach, objectives, scope, resources and schedule of testing, as well as identifying functions and features to be tested, test tasks, and personnel for each task.

The **test result reporting** records the defects detected during the test execution and other data for analysis. Also, it reports on test comprehensiveness and produces a summary of the testing activities.

Test Management:

Test management consists of activities such as

- test plan review,
- training of test staff,
- establishing a test standard,
- ensuring the test documentation follows a standard format and
- organizing an independent test group.

Some Statistics

⌘ More than 50% of the global software population is engaged in modifying existing applications rather than writing new applications.

⌘ Same in automobile industry. There are more automobile mechanics in US who repair automobiles than there are personnel employed in building new automobiles!

⌘ Applications continue to grow and add new features at a rate of 5% - 10% per calendar year, due either to changes in business needs, to new laws and regulations.

⌘ The combination of defect repairs and enhancements tends to gradually degrade the structure and increase the complexity of the application. The term for this increase in **complexity** over time is called “entropy”. The average rate at which software entropy increases is about 1-3% per calendar year.

⌘ Roughly 7% of all defect repairs will contain a new defect that was not there before.

⌘ For very complex and poorly structured applications, these bad-fix injections have topped 20% (from Jones 1995)

The Twelve Bugs of Christmas

For the first bug of Christmas, my manager said to me
See if they can do it again.
For the second bug of Christmas, my manager said to me
Ask them how they did it and
See if they can do it again.
For the third bug of Christmas, my manager said to me
Try to reproduce it
Ask them how they did it and
See if they can do it again.
For the fourth bug of Christmas, my manager said to me
Run with the debugger
Try to reproduce it
Ask them how they did it and
See if they can do it again.
For the fifth bug of Christmas, my manager said to me
Ask for a dump
Run with the debugger
Try to reproduce it
Ask them how they did it and
See if they can do it again.

For the sixth bug of Christmas, my manager said to me
Reinstall the software
Ask for a dump
Run with the debugger
Try to reproduce it
Ask them how they did it and
See if they can do it again.
For the seventh bug of Christmas, my manager said to me
Say they need an upgrade
Reinstall the software
Ask for a dump
Run with the debugger
Try to reproduce it
Ask them how they did it and
See if they can do it again.
For the eighth bug of Christmas, my manager said to me
Find a way around it
Say they need an upgrade
Reinstall the software
Ask for a dump
. . .

For the ninth bug of Christmas, my manager said to me
Blame it on the hardware
Find a way around it
. . .
For the tenth bug of Christmas, my manager said to me
Change the documentation
Blame it on the hardware
. . .
For the eleventh bug of Christmas, my manager said to me
Say it's not supported
Change the documentation
Blame it on the hardware
. . .
For the twelfth bug of Christmas, my manager said to me
Tell them it's a feature
Say it's not supported
Change the documentation
Blame it on the hardware
. . .

Review Questions

1. List 15 things that can be used to create test cases.
2. Who does V&V?
3. What are the advantages of independent V&V?

IV&V has been a mandatory practice in the aeronautics, space, railway, and defense industries, where reliability, security, and safety are of paramount importance.