Introduction to Software Testing (Ch. 1)

Why Do We Test Software?

Brittany Johnson

Adapted from slides by Paul Ammann & Jeff Offutt
Testing in the 21st Century

Software defines behavior
- network routers, finance, switching networks, etc.

Today’s software market:
- is much bigger
- is much more competitive
- has more users

Embedded Control Applications
- airplanes - spaceships
- watches - our homes
- cell phones - automobiles

Agile processes put increased pressure on testers
- unit testing critical (with no training or education!)
- Tests are key to functional requirements – but who builds these tests?
Industry is going through a revolution in what testing means to success of software products.
Software is EVERYWHERE!
Software faults, errors, & failures

**Software fault:** A static defect in the software

**Software error:** An incorrect internal state that is the manifestation of some fault

**Software failure:** External, incorrect behavior with respect to the requirements or other description of expected behavior

Faults in software are equivalent to design mistakes in hardware.

Software does not degrade.
Failure, fault, and error example
(non-technical)

A patient gives a doctor a list of symptoms
- Failures
The doctor tries to diagnose the root cause (ailment)
- Fault
The doctor may look for abnormal internal conditions (high blood pressure, irregular heartbeat)
- Errors

Most medical problems result from external attacks (bacteria, viruses) or degradation as we age.
Software faults are put there (or were always there) and do not “appear” when a part gets old or wears out.
A concrete example

```java
public static int numZero (int [] arr)
{
    int count = 0;
    for (int i = 1; i < arr.length; i++)
    {
        if (arr [i] == 0)
        {
            count++;
        }
    }
    return count;
}
```

**Fault:** Should start searching at 0, not 1

**Test 1**

<table>
<thead>
<tr>
<th>Test 1</th>
<th>Expected: 1</th>
<th>Actual: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[2, 7, 0]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Error:** i is 1, not 0, on the first iteration

Failure: none

**Test 2**

<table>
<thead>
<tr>
<th>Test 2</th>
<th>Expected: 1</th>
<th>Actual: 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0, 2, 7]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Error:** i is 1, not 0

Error propagates to the variable count

Failure: count is 0 at the return statement
The term “bug”

“Bug” is used informally - sometimes a fault, sometimes error, sometimes failure

This course will try to avoid using this word so that we understand the precise terminology

Though you’ll probably use or encounter the term bug informally or at work quite often 😊
Spectacular Software Failures

NASA’s Mars lander
September 1999; crashed due to unit integration fault

THERAC-25 radiation machine
1980s; poor testing of safety critical software can cost lives: 3 patients killed
Spectacular Software Failures

Ariane 5 explosion
Millions of $$ lost from exception handling bug

Intel Pentium FDIV fault
public relations nightmare
Spectacular Software Failures

**Boeing A220**
Engines failed after software updated allowed excessive vibrations

**Boeing 737 Max**
Crashed due to overly aggressive software flight overrides
Spectacular Software Failures

Toyota brakes
Dozens dead, thousands of crashes

Heathcare.gov website
Crashed repeatedly on launch – never load tested
We need our software to be dependable.

Testing is one way to assess dependability.
Software testers try to find faults before the faults find users.
Costly Software Failures


- Inadequate software testing cost US alone between $22 and $59 billion annually

Huge losses due to web app failures

- Financial services: $6.5 million per hour (just in US!)
- Credit card sales apps: $2.4 million per hour (in US)

Symantec (2007) says that most security vulnerabilities are due to faulty software
Costly Software Failures

Northeast blackout
2003; 50 million people, $6 billion USD lost because of power overload (alarm system failed)

Amazon BOGO no-go
Dec 2006; amazon.com’s BOGO offer turned into a double discount
World-wide monetary loss due to poor software testing and maintenance is staggering!
Testing in the 21st Century

More **safety critical, real-time** software

**Embedded** software is ubiquitous

**Enterprise** applications means bigger programs, more users

Paradoxically, free software **increases** our expectations

**Security** is now all about software faults

- secure software is **reliable** software

The **web** offers new deployment platform

- Very **competitive** and very **available** to more users
- Web apps are **distributed**
- Web apps must be **highly reliable**
Industry desperately needs our interventions and help!
The true cost of a software failure

Analysis of news articles in 2016 revealed:
  606 reported software failures
  Impacted half the world’s population
  Cost a combined $1.7 trillion US dollars

Poor software is a drag on the world’s economy

Also...super frustrating
So what does this mean?

Software testing is getting more important.

What are we trying to do when we test? What are our goals?
Validation & Verification (IEEE)

**Validation:** The process of evaluating software at the end of software development to ensure compliance with intended usage.

**Verification:** The process of determining whether the products of a given phase of the software development process fulfills the requirements established during the previous phase.

IV&V stands for “independent verification & validation”
Test goals based on test process maturity

Level 0: There’s no difference between testing and debugging

Level 1: The purpose of testing is to show correctness

Level 2: The purpose of testing is to show that the software doesn’t work.

Level 3: The purpose of testing is not to prove anything specific, but to reduce the risk of using the software

Level 4: Testing is a mental discipline that helps all IT professionals develop higher quality software
Level 0 explained

Testing is the same as debugging

Does not distinguish between incorrect behavior and mistakes in the program

Does not help develop software that is reliable and safe

This is what we typically teach undergraduate CS majors.
Level 1 explained

Purpose is to show correctness

Correctness is impossible to achieve

What do we know if no failures?
- Good software or bad/not enough tests?

Test engineers have no:
- Strict goal
- Real stopping rule
- Formal test technique
- Test managers are powerless

This is what hardware engineers often expect.
Level 2 explained

Purpose is to show **failures**

Looking for failures is a **negative** activity

Puts testers and developers into an **adversarial** relationship

What if there are **no failures**?

This describes most software companies.

How can we move to a **team approach**??
Level 3 explained

Testing can only show the **presence of failures**

Whenever we use software, we incur some **risk**

Risk may be **small** and consequences unimportant

Risk may be **great** and consequences catastrophic

Testers and developers cooperate to **reduce risk**

This describes handful of “enlightened” software companies.
Level 4 explained
A mental discipline that increases quality

Testing is only **one way** to increase quality

Test engineers can become **technical leaders** of project

Primary responsibility to **measure and improve** software quality

Their expertise should **help the developers**

This is the way “traditional” engineering works.
Where are you?

Are you at level 0, 1, or 2?

Is your organization at work at level 0, 1, or 2?
Or maybe 3?

We hope to teach you to become “change agents”...
Advocates for level 4 thinking
Tactical goals: why each test?

If you don’t know why you’re conducting each test, it won’t be very helpful.

Written test objectives and requirements must be documented

What are your planned coverage levels?

How much testing is enough?

Common objective = spend the budget ... test until the ship date...

- sometimes called the “date criterion”
Why each test?

If you don’t start planning for each test when the functional requirements are formed, you’ll never know why you’re conducting the test.

1980: "The software shall be easily maintainable:"

Threshold reliability requirements?

What fact does each test try to verify?

Requirements definition teams need testers!
Cost of **not** testing

Poor program managers might say:

“Testing is too expensive.”

Testing is the **most time consuming** and **expensive** part of software development.

Not testing is even **more expensive**

If we have too little testing effort early, the cost **increases**

Planning for testing after development is **prohibitively** expensive.
Cost of late testing

Assume $1000 unit cost, per fault, 100 faults

Requirements - $6K
Design - $13K
Prog / Unit Test - $20K
Integration Test - $100K
System Test - $360K
Post-Deployment - $250K

Software Engineering Institute; Carnegie Mellon University; Handbook CMU/SEI-96-HB-002
Summary:
Why do we test software?

A tester’s goal is to eliminate faults as *early as possible*.

- Improve quality
- Reduce cost
- Preserve customer satisfaction