



# How Novice Testers Perceive and Perform Unit Testing



**Gina Bai**

---

Assistant Professor of the Practice



**VANDERBILT**  
School of Engineering



## Guess:

How much (in USD) does the **poor software quality** cost the United States **in 2022**?

---

**≥ \$2.41 trillion**

<https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2022-report/>

“Most of our new grad hires have limited experience with automated testing, and that’s a daily activity at Google. Every change that you are going to make to the codebase is going to come with unit tests. That is the rule.”

Titus Winters. “The Gap between Industry and CS Edu.” ITiCSE 2022

---

# Purpose of Testing

- ❖ **Level 0** There is no difference between testing and debugging
- ❖ **Level 1** The purpose of testing is to show correctness

# Purpose of Testing

- ❖ **Level 0** There is 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 does not 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.

---

[Ammann & Offutt, Introduction to Software Testing (Edition 2)]

# Representative Questions



Amy

One of my tests failed, is it okay? Should I fix the test to make it pass? Does the failure indicate a bug in the source code? Or in my testing code?

I was wondering how many test cases do I need to write? Do I need to test everything? I've already found one bug in the code. When can I stop testing?



Bob

# Representative Questions

I found some code examples on StackOverflow, but it's giving me a compile error, and I don't know how to fix it. Can I just delete it?



Daniel

I found the bug, but I don't know how to show that in unit tests... Can I just describe it in comments?

# Challenges

- Novices find it challenging to determine **what and how to test**.
- Novices have **no consensus** on good unit tests, and hence
  - Novices find it challenging to determine **when to stop testing**,
  - Novices tend to **only test happy paths**.
- Novices often create test cases that **mismatch** the program specifications.
- Novices face **implementation barriers**



## Test Case Checklist

### Each test case should:

- be executable (i.e., it has an `@Test` annotation and can be run via "Run as JUnit Test")
- have at least one assert statement or assert an exception is thrown. Example assert statements include: `assertTrue`, `assertEquals` ([click for tutorials](#)). For asserting an exception is thrown, there are different approaches: `try{...; fail();} catch {assertThat...;}`, `@Test(expected = exception.class)` in JUnit 4, or `assertThrows` in JUnit 5 ([click for tutorials](#)).
- evaluate/test only one method

### Each test case could:

- be descriptively named and commented
- If there is redundant setup code in multiple test cases, extract it into a common method (e.g., using `@Before`)
- If there are too many assert statements in a single test case (e.g., more than 5), you might split it up so each test evaluate behavior.

## Test Suite Checklist

### The test suite should:

- have at least one test for each requirement
- appropriately use the setup and teardown code (e.g., `@Before`, which runs before each `@Test`)
- contain a fault-revealing test for each bug in the code (i.e., a test that fails)
- For each requirement, contain test cases for:
  - Valid inputs
  - Boundary cases
  - Invalid inputs
  - Expected exceptions

### To improve the test suite, you could:

- measure code coverage using an appropriate tool, such as EclEmma ([installation](#), [tutorial](#)). Inspect uncovered code and w appropriate.

[Bai, Presler-Marshall, Price, Stolee. ITiCSE '22]

# Testing Checklist

- ✓ **Static**
- ✓ **Lightweight**
- ✓ **Transferable**

## Test Case Checklist

### Each test case *should*:

- be executable (i.e., it has an `@Test` annotation and can be run via "Run as JUnit Test")
- have at least one assert statement or assert an exception is thrown. Example assert statements include: `assertTrue`, `assertEquals` ([click for tutorials](#)). For asserting an exception is thrown, there are different approaches: `try{...; fail();} catch{assertThat...;}`, `@Test(expected = exception.class)` in JUnit 4, or `assertThrows` in JUnit 5 ([click for tutorials](#)).
- evaluate/test only one method

### Syntax and tutorials

### Each test case *could*:

- be descriptively named and commented
- If there is redundant setup code in multiple test cases, extract it into a common method (e.g., using `@Before`)
- If there are too many assert statements in a single test case (e.g., more than 5), you might split it up so each test evaluate behavior.

## Test Suite Checklist

### The test suite *should*:

- have at least one test for each requirement
- appropriately use the setup and teardown code (e.g., `@Before`, which runs before each `@Test`)
- contain a fault-revealing test for each bug in the code (i.e., a test that fails)
- For each requirement, contain test cases for:
  - Valid inputs
  - Boundary cases
  - Invalid inputs
  - Expected exceptions

### Test Class Components

### Equivalence Class Partitioning Boundary Value Analysis

### To improve the test suite, you *could*:

- measure code coverage using an appropriate tool, such as EclEmma ([installation](#), [tutorial](#)). Inspect uncovered code and w appropriate.

[Bai, Presler-Marshall, Price, Stolee. ITiCSE '22]

# Testing Checklist

Contains

- ✓ **Testing strategies**
- ✓ **Tutorial information**

## Test Case Checklist

Each test case *should*:

### Syntax Errors

- be executable (i.e., it has an `@Test` annotation and can be run via "Run as JUnit Test")
- have at least one assert statement or assert an exception is thrown. Example assert statements include: `assertTrue`, `assertEquals` ([click for tutorials](#)). For asserting an exception is thrown, there are different approaches: `try{...; fail();} catch{assertThat...;}`, `@Test(expected = exception.class)` in JUnit 4, or `assertThrows` in JUnit 5 ([click for tutorials](#)).
- evaluate/test only one method

### No Assertions

Each test case *could*:

### Bad Naming

- be descriptively named and commented
- If there is redundant setup code in multiple test cases, extract it into a common method (e.g., using `@Before`)
- If there are too many assert statements in a single test case (e.g., more than 5), you might split it up so each test evaluate behavior.

### Assertion Roulette

## Test Suite Checklist

The test suite *should*:

### Poor Requirement Coverage

- have at least one test for each requirement
- appropriately use the setup and teardown code (e.g., `@Before`, which runs before each `@Test`)
- contain a fault-revealing test for each bug in the code (i.e., a test that fails)

### Misinterpretation of Failing Tests

- For each requirement, contain test cases for:
  - Valid inputs
  - Boundary cases
  - Invalid inputs
  - Expected exceptions

### Testing Happy Path Only

To improve the test suite, you *could*:

- measure code coverage using an appropriate tool, such as EclEmma ([installation](#), [tutorial](#)). Inspect uncovered code and w appropriate.

[Bai, Presler-Marshall, Price, Stolee. ITiCSE '22]

# Testing Checklist

Addresses

- ✓ **Common mistakes**
- ✓ **Common test smells**

# Effectiveness

- ❖ Our study shows that...
  - ❖ The lightweight testing checklist is **at least as effective as** a coverage tool, e.g., EclEmma, for writing quality tests.
  - ❖ Novices who have lower prior knowledge of unit testing may benefit more from the checklist

# Takeaways

- ❖ Most novices see no difference between testing and debugging, and many of them believe the goal of testing is to show correctness.
- ❖ Novices face various challenges when performing testing.
- ❖ The tool support does not need to be sophisticated to be effective.