Unit Testing Introduction & Discussion

Annette Mechelke
@ Codecinella, June 4th, 2019

@interannette

Outline

- What is unit testing?
- What are the benefits of unit testing?
- What are some challenges associated with unit testing?
- What are some best practices?

What is unit testing?

Unit testing is a type of software testing where individual units of source code are validated to function as expected.

Unit tests are:

- Automated
- White box testing
- Lowest level of testing
- Usually written by developers of the system

What is a unit?

- A unit is a small testable part of the code.
  - In object oriented programming, it's usually some method on a class
  - In imperative or functional programming, it's usually a single function
  - Different teams define "small" differently
- It should have a small number of inputs and a single output to validate.
- It may have dependencies (we'll come back to this)

What does a unit test look like?

- Set up
- Call the code under test
- Assert the results are as expected

```
public class Calculator {
  public int add(int a, int b){
    return a + b;
  }
}

@Test
public void ensure_Negatives() {
  Calculator c = new Calculator();
  int result = c.add(0, -1);
  Assert.assertEquals(-1, result);
}
```

What are the benefits of unit testing?

- Ensures some amount of correctness. (But they can’t ensure you’ll never have any bugs!)
- Improve the design of the code under test
  - If you can’t easily write a unit test, it’s a code smell.
  - Forces the developer to define a contract for the unit
- Can catch bugs earlier in the development cycle. (Where they are cheaper to fix.)
- Increases confidence in refactoring existing code.
- Can serve as a type of documentation.

What are some of the challenges?

- How to define a unit?
- How to handle dependencies?
- Ensuring they are fast
- Ensuring they are not a burden to maintain
- Ensuring they are part of the development process

Best Practices - Test Writing

- Keep each test independent
- Limit assertions to only what you really need
- Name your tests clearly and consistently
- Pick high value to tests
- Cover the edge cases
- Mock external dependencies and state

Best Practices - Test Process

- Use a unit test framework
- Track unit tests in source control, commonly as part of the main codebase
- Run the tests on build; if the tests fail, follow up!
- When you fix a bug, write a test that fails first, then fix the bug
- Recognize unit test headaches as sign you need to revisit the design of the code

Discussion!

- Do you use unit testing?
- What challenges have you run into?
- What benefits have you noticed?

References

- https://martinfowler.com/bliki/UnitTest.html
- http://softwaretestingfundamentals.com/unit-testing
- http://blog.stevensanderson.com/2009/08/24/writing-great-unit-tests-best-and-worst-practises/
- https://stackify.com/unit-testing-basics-best-practices/
- https://en.wikipedia.org/wiki/Unit_testing

Related Topics

- Unit Test Frameworks
  - Mocking Frameworks
  - Assertion Frameworks
- Code Coverage
- Test Driven Development