Test Driven Development

Arnav Gupta

April 19, 2024

Contents

1 TDD

Algorithm is

1. write a failing test case
2. get it to compile
3. make it pass
4. remove duplication (refactor)
5. repeat

2 Reflection

Look at requirements first. Full control over the pace of writing production code. Get quick feedback, testable code, and fedback about design.

TDD gives you rhythm to follow and a reminder to review code often. Forces using code developed from the beginning.

When writing tests after code, challenge is making sure time between writing code and testing is small enough to provide developers with timely feedback.

TDD is good for:

- when no clear idea of how to design, architect, or implement a specific requirement
- when dealing with a complex problem or a problem with lack of expertise to solve

TDD is not good for when the problem and solution are fully known.

3 Problems

Distracts from writing code (must think negatively first).

Takes a lot of disclipline.

Must write and maintain many simple, useless tests.

Architecture is designed around making it easier to test.

Need many mock objects to decouple from DB and UI.

Maintenance tests are not the same as development tests, so sometimes better to consolidate unit tests and remove low value tests.

Can compromise by ensuring every commit has a test associated, so no need to write test first. If test not added at commit, it is unlikely to ever be added.

4 Legacy Code

Algorithm:

1. get the class to change under test
2. write a failling test case
3. get it to compile
4. make it pass (try not to change existing code)
5. remove duplication (refactor)
6. repeat

TDD allows concentration on either writing code or refactoring, never both at once. This is valuable in legacy code since new code is written independent of old code. After writing new code, can refactor to remove any duplication between it and old code.