DZone

CHECKLIST: ✓ TEST-DRIVEN DEVELOPMENT

by Giorgio Sironi

Test-Driven Development (TDD) is described by a basic red-green-refactor cycle, constantly repeated to add new features or fix bugs. This checklist is written in the form of questions we should ask ourselves while going through the different phases of the cycle, and that are often overlooked because the cycle is apparently so simple.

1 PHASE RED

✓ **The development of every new feature should start with a failing test.**

- ◯ Have you checked the code into your remote or local repository? In case the code breaks, a revert is faster than a rewrite.
- ◯ Have you already written some production code? If so, comment it or (best) delete it so to not be implicitly tied to an API while writing the test.
- ◯ Have you chosen the right unit to expand? The modified class should be the one that remains more cohesive after the change, and often new classes should be introduced instead of accommodating functionalities in existing ones.
- ◯ Does the test fail? If not, rewrite the test to expose the lack of functionality.
- ◯ Does a subset of the test already fail? If so, you can remove the surplus part of the test, avoiding verbosity; it can come back in different test methods.
- ◯ Does the test prescribe overly specific assertions or expectations? If so, lessen the mock expectations by not checking method calls order or how many times a method is called; improve the assertions by substituting equality matches with matches over properties of the result object.
- ◯ Does the test name describe its intent? Make sure it is not tied to implementation details and works as low-level documentation.
- ◯ How much can you change in a hypothetical implementation without breaking the test (making it brittle)?
- ◯ Is the failure message expressive about what is broken? Make sure it describes where the failing functionality resides, highlighting the right location if it breaks in the future.
- ◯ Are magic numbers and strings expressed as constants? Is there repeated code? Test code refactoring is easy when done early and while a test fails, since in the TDD paradigm it is more important to keep the test failing than to keep it passing.

DZone

by Giorgio Sironi

CHECKLIST: ✓ TEST-DRIVEN DEVELOPMENT

2 PHASE GREEN

✓ **Enough production code should be written to make the test pass.**

- ○ Does the production code make the test pass?
- ○ Does a subset of the production code make the test pass? If so, you can comment or (best) remove the unnecessary production code. Any more lines you write are untested lines you'll have to read and maintain in the future.

Every other specific action will be taken in the Refactor phase.

3 PHASE REFACTOR

✓ **Improve the structure of the code to ease future changes and maintenance.**

- ○ Does repeated code exist in the current class?
- ○ Is the name of the class under test appropriate?
- ○ Do the public and protected method names describe their intent? Are they readable? **Rename** refactorings are among the most powerful.
- ○ Does repeated code exist in different classes? Is there a missing domain concept? You can extract abstract classes or refactor towards composition. At this high level, the refactoring also should be applied to the unit tests.

ABOUT THE AUTHOR

Giorgio Sironi is a programmer and architect (the kind that writes code) with a focus on testing and open source; he maintains the PHPUnit_Selenium project. Giorgio believes that programming is one of the hardest and most beautiful jobs in the world.

DZone, Inc.
150 Preston Executive Dr.
Suite 201
Cary, NC 27513
888.678.0399
919.678.0300

Sponsorship Opportunities
sales@dzone.com