Jacob Elkins

Doc

Data Structures

March 21, 2018

Test Driven Development

Test driven development is a software development process that uses a short development cycle that is repeated over and over again as each requirement is being implemented. As a developer starts to define the various requirements and features their program requires, each one of these requirements or features is going to turn into a test suite. Test driven development follows a very strict principal of everything being added to the program must be tested. Developers will follow a cycle called the red – green- refactor cycle during development.

The first stage of this cycle is the red stage. This is where you will add a test for a requirement or feature being implemented into the program. The test being written will define exactly what the requirement or feature is specifically doing. If a developer sees that the requirement or feature is doing more than one thing, it would need to be split up into more than one test. By writing a test before any code, it makes the developer focus on the requirement or feature rather than the actual code. Once the test is written, if it is a new test, the test must fail due to the factor there is no code behind the requirement being tested yet. This ensures that the cycle is being followed properly.

The second stage of the cycle is the green stage. This is where a developer will start to write production code. The key is to write just enough code to fulfill the requirement or feature being implemented because anything more could result in untested code. This code does not have to be perfect, the goal is to pass the test. If the test does not pass, the developer must revisit

the test suite and production code to determine where it went wrong to cause the test to fail. Once the test is finally passing, the entire test suite should be rerun to ensure that all tests are still passing with no unanticipated consequences.

The final stage of the cycle is the refactor stage. As a developer goes through the red – green - refactor cycle, it is essential to clean up code regularly after each test and production code is implemented. New code now can be moved from where it was convenient for allowing the test suite to pass to where the developer thinks it logically belongs in the program. During refactoring or programming in general, each method within a program should only do one thing. If a method does more than one thing, it should be refactored accordingly to allow the method to accomplish only one task. Also, through the program, code clones should be eliminated. This will allow for easier maintenance in the long run when developers are revisiting the program. Names of objects, classes, modules, variables and methods should have names that clearly describe what they are or doing and their purpose for being in the program. As the program grows bigger and bigger, names can be longer as needed to improve readability and maintainability. This ensures developers are following a clean code aspect. Once refactoring is complete, rerun all tests again throughout each refactoring stage. This will ensure that the program is still running the way it should be because refactoring can cause other portions of the program to fail. This will allow the developer to be confident that each process is not altering existing functionality. As more and more features and requirements are being implemented into the program, developers will repeat the cycle over and over again until the program is complete. This type of continuous integration of features and requirements helps developers establish revertible checkpoints. These checkpoints are used for when a test or code fails, it allows the developer to easily undo or revert back for debugging to determine the cause of the failure.

When using a development technique such as test-driven development, there are benefits that come along with it. TDD allows the developer to actually think about what the requirements being implemented into the program need to accomplish since they have to write a test around it. This forces you to work in smaller chunks of functionality rather than thinking about the application as a whole. This kind of approach rather than tackling the bigger picture, allows the developer to avoid potential multiple bugs and problems. Because you write a single test for each requirement, the developer must think about the public interface that other code in your application needs to integrate with. By doing this, it will allow for the code to be more readable and make more sense. Throughout the test-driven development process, since the developer continuously reruns the entire test suite, it provides constant feedback that the program is still running and working properly and also provides documentation of what requirements are being implemented into the program. Most importantly, it allows for an easier maintenance process due to clean code. This also allows for more readable code. It is important that if a developer steps away from their code or an outside source comes in and looks at their code, it is essential that they are able to understand what is going on and it's easy to understand. All in all, TDD is a very stream line process with many development benefits.