2020

Test Driven Development

HAMID MUJTABA

Contents

Test Driven Development

Test driven development is a style of programming where developers create test cases for every functionality of the program to ensure the code runs as expected. They refactor/change the code if the automated test fails and improve the code until the test passes, which also helps avoid duplication of code. TDD is believed to be an established technique for delivering better software, more rapidly, and more sustainably over time (Mohamed Taman, 2019).

Test driven development is an implementation cycle with 5 main steps.

1. The first step is to understand what the function requirements are and then add a test which defines the function.
2. The second step is to run the tests to see if new tests fail. The tests are expected to fail as there is no code that has been implemented yet.
3. The third step is to write and implement code which meet the requirements so that the tests pass.
4. Fourth step is to run the tests again to see if they pass or not. If it passes, you move to the first step and add a new test. However, if the tests fail, you go back to step 3 to refactor the code and run the tests again; you repeat step 3 and 4 until the tests pass.
5. Step is to repeat this process for each functionality until all your tests pass and your software meets all the requirements.

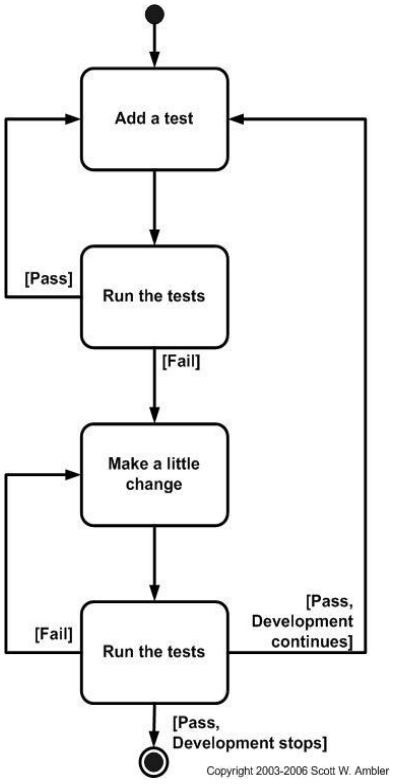

Figure 1: TDD Cycle

Over the last couple of years, test driven development has gained a lot of popularity, however after a lot of programmers trying and failing to successfully make use of it, they concluded “TDD is not worth the effort it requires” (Andrea Koutifaris, 2018). Some of them thought that, although it may be a good practice in theory, it is too time consuming as there is never enough time to use test driven development, and few of them even agreed it is a waste of time. Some even argue that TTD focuses on passing tests, and not necessarily the correctness of code. Few of the well-known framework for test-driven development include, ‘csUnit and NUnit’ for .NET projects, ‘PyUnit and DocTest’ for Python, ‘Junit’ and ‘TestNG’ for Java and finally ‘Rspec’ for Ruby.

Pros vs Cons

There are pros and cons for using test driven development. One of the benefits of test-driven development is that it allows the code to be easily maintained and refactor. It makes it easier for others to edit your code because the tests would inform them if the changes made are causing the tests to fail. This allows collaboration between team members easier. Another benefit is that it makes debugging a lot easier. It can also be proved to be very useful in the long term as it is faster. For example, if you are trying to refactor some code you have written couple of years ago, it will prove to be very difficult; however, if you have already set up good suite of unit tests, it will be a lot easier and efficient.

Some of the cons of test-driven development is that it can be difficult to learn, so there is a difference between executing it and executing it well. Test driven development is also known to be initially slower as it is very time consuming.

Applying Test-driven development approach

To put test-driven development into practise, it was used to develop a Caesar Cipher algorithm using the Haskell programming language.

Algorithm

The Caesar Cipher technique is one of the earliest known method of encryption, named after Julius Caesar, who used this this method to encrypt and send secret messages to his generals. It is a type of substitution cipher where each letter of a given text is moved some fixed number of positions down the alphabet. For example, with a shift of 1, A would be replaced by the letter B and B would become C and C would be D and so on.

To cypher some text, or send an encrypted message between two people, it is important both parties have the ‘key’ to encrypt the message or to decrypt it. The ‘key’ represents an integer value which indicates the number of characters to move down in the cipher alphabet. For example, if I wanted to encrypt the word ‘HELLO’ with the shift (key) of 1. It would look something like this: IFMMP.

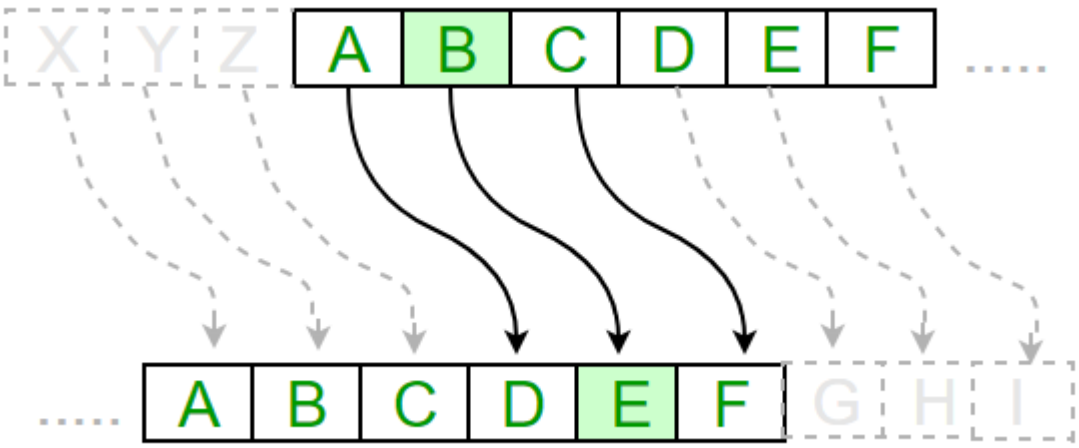

Figure 2: Caesar Cipher example

The mathematical equation for an encryption function would be 'e(x) = (x + k) (mod 26)', and the decryption function would look like 'e(x) = (x - k) (mod 26)'.

Development

For the development process of test-driven development, unit testing tools were used. The unit testing tool chosen for this was QuickCheck, which is a Haskell library that allows for random testing of the functionality in your code. It comes with some pre-defined test cases; however, you can manually create your own test cases depending on what kind of tests need to be performed to ensure the code quality of your code.

Whilst developing the Caesar Cipher algorithm, Unit tests were planned for the main steps of the algorithm which are verifying inputs are valid, encrypting/decrypting single characters and encrypting/decrypting string messages.

Starting with verifying inputs, a test was written to ensure the inputs were either capital letters, lowercase letters or the space character. The next task was to develop the functionality to verify the inputs and ensure it passes the previously mentioned Unit test.

After this, similar Unit tests were written for verifying encrypted and decrypted characters are either capital letters, lowercase letters or the space character. Now that a test was written for encrypting and decrypting characters, it was time to develop the functionality to perform the encryption and decryption.

Finally, now that the functionality for encrypting and decrypting single characters had been written and tested, Unit tests were written for encrypting and decrypting a string of characters. The tests written were confirming encrypted and decrypted messages are the same length as the original input message, encrypted and decrypted messages are different to the original input message and that encrypting and then decrypting a message results in the original input message.

The functionality for encrypting and decrypting strings was developed and tested with the Unit tests written for it.

Reflecting on test driven development experiences

Pros

The test-driven development process made it easy to plan the development of the algorithm by splitting it into different steps and creating Unit tests for each.

It was easy to find the cause of any errors in the algorithm as tests were written for each part of it. This would be much more difficult if simply running the algorithm on its own.

The QuickCheck library saved lots of time that would be needed to think of different test cases and create test data. It was very fast to get a large amount of test data which meant there was less chance of a certain case being missed.

Cons

The lack of experience using the Haskell programming language made the overall development process a lot more difficult and time consuming. This would become less of an issue with practise.

The chosen algorithm was not overly suitable for the test-driven development approach due to its lack of complexity, it resulted in it taking much more time than it would for a different approach. This would not be the case for a more difficult algorithm, and it would likely be very useful.

Future

The author would likely use the test-driven development approach in the future when writing a more complex algorithm as it would be better suited. Further practise using the Haskell programming language would be needed for it to be time effective.

References

Andrea koutifaris. 2018. Test Driven Development: what it is, and what it is not. [Online]. [25 May 2021]. Available from: https://www.freecodecamp.org/news/test-driven-development-what-it-is-and-what-it-is-not-41fa6bca02a2/

Mohamed taman. 2019. Test-Driven Development: Really, It's a Design Technique. [Online]. [25 May 2021]. Available from: https://www.infoq.com/articles/test-driven-design-java/

Quora. 2016. What are the pros and cons of test-driven development?. [Online]. [25 May 2021]. Available from: https://www.quora.com/What-are-the-pros-and-cons-of-test-driven-development

Agile data. c2021. Introduction to Test Driven Development (TDD). [Online]. [28 May 2021]. Available from: http://agiledata.org/essays/tdd.html

Jash unadkat. 2021. What is Test Driven Development (TDD) : Approach & Benefits. [Online]. [28 May 2021]. Available from: https://www.browserstack.com/guide/what-is-test-driven-development

Hackage and cabal. 2020. An introduction to QuickCheck testing. [Online]. [25 May 2021]. Available from: https://www.schoolofhaskell.com/user/pbv/an-introduction-to-quickcheck-testing

Geeksforgeeks. 2021. Caesar Cipher in Cryptography. [Online]. [25 May 2021]. Available from: https://www.geeksforgeeks.org/caesar-cipher-in-cryptography/

Practical cryptography. c2012. Caesar Cipher. [Online]. [25 May 2021]. Available from: http://practicalcryptography.com/ciphers/caesar-cipher/