Explain the practice of test-driven development, and critically evaluate this practice. This evaluation can include your own impressions and experiences, and, for the higher grades, references to evidence from the research literature.

What is test driven development

Test driven development (TDD) is a process carried out in a software development process in which tests drive the development of software code. This is enacted by via the process of creating a test, then developing and refactoring the code it tests so that the test passes before moving onto the next test and code.

Test driven development was made popular thanks to it being a practice in the agile software development methodology, specifically extreme programming. The practice itself has been around for many decades, for example it was used in NASA’s project Mercury in the early 1960s [1]. Many companies which develop software use the agile methodology and as a result test driven development.

TDD is used within various settings outside of software. However, this report will attempt to explain and evaluate it within the context of software development. EXPLAIN WHAT IS EXPLAINED.

Test driven development cycle

Test driven development makes use of the test-driven development cycle. Each time an additional piece of code is to be developed this cycle is performed, consisting of 5 steps.

1. Add a test

The test is written to cover the overall requirements of the function it will be testing.

1. Run all tests and see if the new test fails

This ensures that no other code has been affected since the last development iteration, so that focus can remain on the development of the new function which will be tested. It also checks if the new test builds correctly and does not always pass.

1. Write the code to cause the test to pass

This involves creating the code to make the new test pass, and therefore create a working function. At this stage it does not entirely matter if the developed code is not done in the most efficient or professional way as it will be improved later in step 5 of the cycle.

1. Run all tests

This step is there to ensure the newly developed code is working. If the new test still fails, then the programmer must return to step 3. Running all the tests checks that the newly developed code has not in some way affected other code as well, ensuring integrity of the software.

1. Refactor the code

This step involves refactoring the developed code, making it more efficient, maintainable, and clear. Step including things such as removing duplication, using variable naming conventions, or managing memory.

In general, it is thought that the size of the developed functions should be kept small to provide clarity and reduce debugging effort.

Code Quality

Focuses designers on interfaces of code.

Forces the developer to focus on the specification of the code before writing it.

Test driven development allows for the programmer(s) to think through the requirements of the software in a structured manner before writing the code.

Refactoring as part of its process.

Their results show that the quality of the code developed using TDD increased 2.6–4.2 times when compared to non-TDD developed code. [2]

Lack of Cohesion in Methods. LCOM\* [xxx] 3 projects done, with the one with TDD got lowest chohesion. It is also noticeable that the results are the most consistent in project 3, which probably indicates that the poor cohesion is not likely to be a coincidence. Whether it is due to TDD or the developers' lack of experience remains to be analyzed. However, this result already indicates that TDD does not automatically produce highly cohesive code.

Code Confidence

Tests act as a method of documentation of the system.

Tests tend to be very thorough as a result of covering every path.

Test driven development ensures that all written code is covered by at least one test.

Bhat and Nagappan. The block coverage was 79–88 % at unit test level in projects employing TDD. [2] They also noticed that the tests serve as auto documentation when the code was maintained or used.

TDD fails to ensure confidence in all corners of software development however. Examples of these are user interfaces, applications which make use of databases and those that make use of networking tasks.

Development Management

Tests act as a method of documentation of the system.

Lui and Chan [3] present the results of TDD and software process improvement in China. The authors find that TDD improves task estimation as well as process tracking.

Management heavy, need to ensure that tests are being produced.

Need to find ways of setting up running of tests, like on ci servers. Which takes time and resources.

Time

Alternatively, the project managers estimated that TDD increased the overall development time by 15–35%. Bhat and Nagappan [2]

programmers with an improved program understanding and therefore they are able to use existing methods faster, correctly. Compared against test last development practices.

While more code needs to be written with TDD. Need to mock classes aswell. Large numbers of tests help to limit the number of defects in the code.

Conclusion

[1] G. Larman and V.R. Basili, "Iterative and Incremental Development: A Brief History", IEEE Computer 36(6), IEEE Computer Soc., Los Alamitos, CA, USA, 2003, pp. 47-56.

[2] Bhat, T. and Nagappan, N. Evaluating the efficacy of testdriven development: industrial case studies. In ISESE '06, Rio de Janeiro, Brazil, 2006.

[3] Lui, K. M. and Chan, K.C.C. Test Driven Development and Software Process Improvement in China. In XP 2004, Garmisch-Partenkirchen, Germany, 2004.

[xxx] Henderson-Sellers, B., Object-Oriented Metrics: Measures of Complexity, Prentice Hall, Upper Saddle River, NJ, USA, 1996.

<https://arxiv.org/ftp/arxiv/papers/1711/1711.05082.pdf>

<https://www.researchgate.net/publication/264383320_An_experimental_evaluation_of_test_driven_development_vs_test-last_development_with_industry_professionals>

<https://www.ripublication.com/ijaer17/ijaerv12n18_81.pdf> <-- advantages and pitfalls

<https://www.researchgate.net/publication/323193925_Effectiveness_of_Test-Driven_Development_and_Continuous_Integration_A_Case_Study>

<https://ris.utwente.nl/ws/portalfiles/portal/6943854/TCM_IEEEProf_final.pdf>

Reﬂect on your own experiences of using test-driven development, functional programming, and testing tools for this assignment. This should involve considering what went well and what did not, whether you think the reasons for this were due to the nature of the practice/tools or your own approach/familiarity, and whether and how you would use the practice/tools in future. GIVE NARRATIVE ACCOUNT

How functional programming went.

How using test-driven development went.

What tools I used. How they went. Property-based testing,

While it was useful in some aspects, it also restricted me in terms of testing different sizes of arrays.

Why did it go well, or why did it not go well?