Essay 2- Software Engineering (CEN5035)

"Ian Sommerville: My Reasons for Not using TDD"

Name: Vikas Chaubey**, UFID:** 3511 5826**, Email:** vikas.chaubey@ufl.edu

In the book and the blog post, Ian Sommerville has described and presented various arguments to point out the demerits and disadvantages associated with the approach of test-driven development of software programs which he experienced while following TDD for a personal project. **Test driven development** is the software development methodology with the idea that executable tests for program functionality should be written first before actually implementing the program code itself. Once the test cases are written then program code is implemented to pass the test cases. If the program fails to pass the test, then changes in the program code are made, and this process is iterated until the test cases are passed. TDD follows a backward approach compared to the traditional approach of software development where program code is written first and then it is tested for correctness and quality. In main stages of TDD involve gathering of requirements, converting the requirement and feature specification into unit test cases, then writing program code and refactoring it until it passes all test cases. The expected benefits of TDD are that this approach significantly reduces the bugs and defects in the program by increasing code test coverage, it makes specification of the program easy to understand and lead to simpler code design.

According to the author TDD methodology has various limitations and he has provided various reasons for not using TDD. The first argument presented by Ian Somerville in his book and blog post for not using TDD states that this methodology induces psychological reluctance in programmers to make any big scale changes in the already existing code base because psychologically they are aware that introduction of new code might result into test failures for already functioning code modules , Hence they are reluctant to make any big scale changes in the code base. He further argues that TDD approach of software development focusses heavily on test cases hence the whole process psychologically becomes test case centric for programmers, instead of thinking about an effective overall software design to develop a quality software, programmers become mentally limited to write code modules which can just pass test cases. In the response to this argument on the blog post, one user has suggested that getting used to a specific type of software development methodology takes years of practice and hands on experience, the user suggested that since the author has been working on the project and using TDD only for few months, he is not prepared sufficiently to use the development technique effectively. With his own years of experience with TDD the user argues that programmers can learn to overcome the psychological roadblocks suggested by the authors for existing code base ,while implementing new features with more practice and experience and see through the bigger picture of overall software design than just thinking about TDD as a mere code development practice to pass test cases. Another important counter argument made by another user on the blog post is that test-driven development is very beneficial in case of already developed programs which need new feature integration, because TDD makes sure that integration of new code modules does not break already working old code functionality. This way TDD ensures new features are safely integrated with old code base without any failures. TDD helps to maintain the reliability of the software system with integration of new features. These counter arguments sound compelling but various users in blog post agreed with the author on the point made on psychological reluctance and limitations posed by the TDD process during development of software. At the end TDD is a test driven approach with primary focus on writing test cases to develop a bug free code hence it is very obvious that while working with this approach psychologically programmers will tend to think of development in terms of test cases, and even if with enough practice and experience TDD could be used efficiently it still takes many years to learn to use TDD effectively.

The second argument made by the author, states that while using TDD for software program development, he experienced that his software design decisions were majorly influenced by the motive of code modules passing the test cases rather than the actual specification of end user requirements. Since TDD heavily focusses on code test coverage by writing test cases for software features in advance, according to author while writing code he was unconsciously trying to modify and lay out a simpler software design and redefining the actual problem for the program modules in order to make writing test cases easier. According to author, while working on projects using test driven development a software programmer might prefer a simpler software design which is easier to test instead of a complex software design which might be difficult to test but the latter is more robust and more beneficial for overall software performance. However, countering that argument one user on the blog post pointed out that if a code module is hard to test then it is not an efficiently designed module hence its design should be changed. A good design is one which is easily testable. However, sometimes it is unavoidable to incorporate code segments in the software system which are not easily testable hence in that case TDD practices might fail to be effective, to prove that author presents example of graphical user interface ,he explains that even the biggest advocates of TDD approach admit that while TDD is good for developing and testing individual units , testing the whole system is extremely difficult with TDD for example testing GUI system is very difficult using TDD and hence in this case writing test cases in advance might prove to be very difficult process. While presenting arguments author explained that in many cases he deliberately continued with implementation before writing test cases because it was difficult to write test cases in advance.one of the user responded to this argument by explaining that TDD encourages overall simpler software module design, one of the primary advantage of using TDD is that it leads to simpler more structured software design. By employing a simpler design it's easy to understand the functionality of individual code modules in the code base. A simpler software design is easier to test and debug in case of any code failures, also with simpler software design it is easy to integrate new code modules to incorporate new features in the software system so, using TDD might prove to be advantageous in software development process. However, author raises questions whether or not TDD always result into a simpler software design, the author explains that there were various experiments done to test those claims and those experiments were inconclusive. Hence it cannot be claimed that TDD is an effective methodology for achieving a simpler software design. Author states that the main advantage associated with TDD is the regressive testing of the code while its being written to make sure a code unit works as per its desired specification and that is made sure by writing the test cases for the code unit however the actual advantage comes from the automated test which run automatically and test the code unit. hence it does not matter whether these tests are written in advance or after the code is implemented.

In the third argument made by the author for not using TDD, he explains that while developing his own project using TDD he experienced that TDD encourages to focus more on implementation details which deal with passing or failing the test cases rather than the overall structure of the program. However, author strongly advocates the old traditional ways of software development which gives great importance to “think first rather than test first” approach. According to author the requirement specification should be analyzed carefully, and sufficient time should be given to develop an effective design to tackle the proposed problem. This approach encourages to think of the product structure and design as a whole. But in the case of TDD the importance is given to implementation details is so much that developers often forget to focus on overall structure and design of the software system. however a user on the blog post gave an counter intuitive perspective , the user explained that the first stage in test driven development is gathering the software requirement and after analysis these requirement specifications are converted into test cases, so when TDD is followed a programmer is actually working towards meeting the end goal or final requirement specification of the software program. Hence saying that TDD makes a developer focus more on implementation detail rather than actual programming problem as a whole might not be an accurate evaluation. However, this counter example is not accurate either, we know TDD focuses more on passing test cases for bug free code implementation but passing test cases does not guarantee the correctness of code implementation. Author has explained that in many cases in order to simplify test cases, test cases could be written to cover code functionality only partially that means that even if the test case is passed it does not mean that code segment is accurately fully functional. However, in traditional approach advocated by the author in initial phases of software development after analysis of requirements, sufficient time is given to develop effective design and the quality of the software is ensured in the design phase not in test phase which is the case with TDD.

In the fourth argument author explains that the reason behind most of the program failures is the occurrence of unexpected or bad data. Bad data is something which is not expected by the programmer while he is developing the code base but in real time situation this type of scenario arises where the program faces such type of data and has to process it and hence the program fails. Author states that writing test cases for such types of situation for unexpected data is very difficult to reflect the real time situations. Validation checks could be used to reject the bad data and only process the correct data, but it is very difficult to define correct data. Hence TDD poses a big challenge to handle such scenarios. One user in response to this argument stated that in order to handle unexpected data various technological means could be used to write bad data test cases for example while writing applications in java, test cases could be written in Scala, these are inter compatible languages, then a specific library like “Scalacheck” could be used to reflect real time bad data scenarios in test cases. Use of such technological options could help follow TDD effectively. However, availability of such technological means is not guaranteed among all development platforms. Hence in case of TDD writing test cases for bad data is still very challenging.

These arguments presented by the author effectively highlight the demerits and shortfalls associated with the test-driven development approach. TDD bounds programmers psychologically to think of development process in terms of test cases rather than focusing on overall software design, hence it influences programmers to take design decision to pass test cases rather than focusing on the actual specification of end user requirements. it cannot guarantee the correctness of the code implementation. TDD puts more emphasis on implementation details and give less importance to requirement analysis and design phase. In TDD developing tests cases for unexpected data is very challenging. Hence for all those above reasons author concludes that TDD is not an effective software development methodology. However, it is important to note that author has made those observations while working on a personal project, TDD has been proven to be very effective in an industrial situation where software has specific hard and fast specification guidelines and these guidelines have to be met within a given period of time. TDD along with agile methodologies is proven to be effective techniques to develop bug free software systems fast. That’s is why they are used together in software development in industry. TDD could be made more effective by integrating a design phase in the beginning of the development cycle, another approach which is getting traction is “behavior driven development” which focusses more on the software functionality and overall software design. Even with its limitation TDD is still an effective software development tool and it should be used accordingly to develop efficient software solution, where meeting end user requirements is the primary goal not TDD practice itself.

References:

1) https://iansommerville.com/systems-software-and-technology/static/2016/03/17/giving-up-on-test-first-development/
2) https://medium.com/@charleeli/why-tdd-is-bad-and-how-to-improve-your-process-d4b867274255

3) **Engineering Software products: Ian Sommerville**

4) **http://agiledata.org/essays/tdd.html**

5) **https://www.guru99.com/test-driven-development.html**

6) **https://www.projectmanager.com/waterfall-methodology**

7) **https://www.agilealliance.org/glossary/bdd**

8) **https://www.cigniti.com/blog/7-best-practices-for-agile-test-driven-development/**