International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 18 (2017) pp. 7705-7716

Evaluating the Effectiveness of Test Driven Development: Advantages and Pitfalls

Zeba Khanam [1] and Mohammed Najeeb Ahsan[2]

[1] College of Computing and Informatics, Saudi Electronic University, Kingdom of Saudi Arabia.
[1]Orcid Id: 0000-0003-3488-0360

[2] School of Management Studies, Sri Satya Sai University of Technology & Medical Sciences, Sehore (M.P), India.
[2]Orcid Id: 0000-0003-3959-1170

Abstract

This paper examines the impact of Test Driven Development on different software parameters such as software quality, cost effectiveness, speed of development, test quality, refactoring phenomena and its impact, overall effort required and productivity, maintainability and time required. The study is based primarily on the research conducted over the last ten years. This work makes a detailed analysis of the effects of test driven development and its applications and intends to help the researchers get a quick insight into its application areas, advantages and pitfalls with respect to the above mentioned parameters evaluated on academically controlled experiments and industrial case studies.

Keywords: Test Driven Development, Software development management, Extreme Programming paradigm, Agile Software Development, Test last development.

INTRODUCTION

Test driven development (TDD) is one of the common practices of Agile core development. It is acquired from the Agile manifesto principles and Extreme programming. XP has two important test practices: test-driven development (TDD) [17] and customer acceptance testing. It assures stability to improve the confidence of the developer and reach high test coverage for loosely and highly coupled system. It also motivates the explicitness regarding deployment scope. TDD is one of the engineering techniques for developing the software that joins the program design and testing in series of micro-iterations. Therefore it is considered as a union of test-first development in which the unit tests are conducted before deploying the code. Refactoring is also associated with this process and it plays an important role in restructuring the code piece. Further, the tests should succeed to reduce complexity and enhance its understandability, maintainability and clarity.

Generally, the TDD is described with red-green-refactor cycle functioned with many steps. According to the mantra of TDD, red refers to the first implementation of the code which starts with the failing test, green indicates the need to create and refactor. Additionally, TDD has the tendency to create the minor positive effects on the quality but it creates the distinguishable effect on productivity [2]. TDD usage is supposedly contributing to improvements in quality of code [3] and quality attributes. The defects density, coverage of code, complexity of code, size, cohesion, coupling, external quality etc. are some of the extracted attributes and qualities associated with the effects of TDD.

Outside the software industry, the TDD has not always lead to drastically better result when compared to the other methods of development. Many empirical studies on the effect of TDD have already been conducted at different times on different software parameters [18][45][66]. In this paper, we present the impact of TDD practice on software development productivity such as: the effort required, software quality improvements, defect reductions peed, cost effectiveness, time and also activities associated such as testing and test quality, refactoring and maintenance and also the environments where TDD is used. We get a quick overview of advantages and drawbacks of TDD over these parameters.

TEST DRIVEN DEVELOPMENT APPLICATIONS

This section focuses on the significance of applicability of TDD in various areas and its implications.

TDD in Web Applications

According to the conclusions drawn by a web application developer using Spring Framework initially TDD did not seem to be a good fit in for bottom to top workflow. For example if this sequence is followed: [48]

1. Creation of database tables and the domain model objects.
2. Repository layer implementation.
3. Service layer creation.
4. and finally the web layer.

As linking a test case with a business requirement did not seem to be a feasible practice. But often it happened after the applications were built there was an essential piece of information that was missing. In lieu of thinking about the problem before the solution, the developer did the other way around. Analyzing the different web application layers, it was observed that the layer which was "strongly associated to the solved problem is the web layer. This brings the conclusion that the work should begin from this layer instead. Therefore if the workflow is reversed, these are the conclusions drawn by the author:

1. **Each layer specifies the requirements for the layer below it**. This leads to the elimination of unrequired code. But still that doesn't imply that the code is final because there might be changes in the requirements leading to altering the code as well. However, the changes due to unclear requirements are reduced.
2. **We can concentrate on the requirements and transform them into test cases**. The clear understanding of the requirements of a specific component, makes it easier to chalk the required test cases even before writing the code.

Just contrary to the above results another author quotes that it doesn't help resolve large scale design problem where a project might consist of several modules divided into several layers. For this scenario some degree of beforehand designing and preplanning is to be done [44].

Another author [47] who fetched the results from case studies on web applications obtained from big companies like Microsoft and IBM stated the following results:

The Microsoft case study projected that TDD project has almost double code quality but consumed 15% more time for writing tests whereas the IBM Case Study stated that 40% fewer defects were reported whereas there was negligible effect on the team's productivity. Case study conducted by Ericsson gave the result that TDD produces higher quality code and there was an impact of 16% on the team's productivity. [47]

The different types of testing were also projected at both front end and the back end. The tests for Front end browsers were System Tests and Acceptance test, Unit tests for Web Services, testing of Browser/OS/etc with Compatibility, Security and performance testing. The tests for Back-end included creating unit test for Functional Testing of business logic and Reusable Components.

TDD in Embedded Software:

A study done on embedded system tends to establish [4][60], Test-driven development as a reliable embedded software engineering practice. In designing the embedded system testing is mostly adhoc and postponed until after hardware development .The reason being that during the process of designing, the required hardware might not be available, that complicates the testing process. But thorough testing is very important for embedded systems, since there is an exponential growth in the cost of repair once the system is under the production. Thus TDD is facilitated using mocking hardware strategies such as Interface based Mock replacement, inheritance based mock replacement etc. They stated earlier that programming to an interface, instead of to a concrete class itself, isolates the software from the hardware. This is substituted using virtual drivers. Preferably Test-Driven Development would be an ideal choice for the code that's free of any external dependencies. Such projects are well suited for TDD as they don't require a complex setup and can be developed in isolation [43]. But in case of embedded software the embedded environment is complicated. Generally there are four constraints that impact the embedded software development and TDD as well.

TDD AND DEVELOPMENT METHODOLOGIES

The research carried out on TDD implemented by a number software development teams [46] also explored the methodologies used by them. Agile methodology seemed to be the favorite choice of most of the teams and four teams followed Scrum and/or Kanban, two teams reported to have used the Waterfall model but the results found were not satisfactory and one team did not follow any methodology. The implementation process did not use any of the formal testing strategies. Though one team reported to have used Selenium plug-in tests.

TDD and Code Quality:

An investigation is also done [7] on the impact of teaching TDD to novice programmers. Test code, reference tests and solution code are some of the factors that are symbolic in the computation of the outcome metrics on the basis on the student code reference tests. When Test first approach is encouraged, the design seemed simpler and it required fewer code lines [68] to produce the efficient result. In addition to these, from the findings of the research, there was a positive correlation between the adherence to TDD and code quality solution and testing.

The flexibility of design could be achieved [8][9] as the tests represent the documentation for the low level design based on the structure and the system interaction. Sometimes, production code may lose until successfully executing the tests.

Defect reduction is also frequently analyzed [10][11] by different authors that contributes to a good low level designing. It provided the greater predictability in the development performance which assisted in estimating the cost of project successively. Robust design context and the

smoother code integration are some of the practices which have significantly associated with the defect density reduction. Another research study examines about reducing testing effort in the TDD [12]. Similar to that [13], it was stated that frequency of defects and the test cases development time spent in the process of TDD are high when compared to the conventional technique (Test-last).

Significance of TDD over product quality was studied by different researchers [14][15][16]. When TDD is adopted in industries it satisfies the end-clients by quality obtained during the development process. The communication obtained by the TDD has ensured to understand the scope and requirements of the products.

TDD and Class Design

Many practitioners believe that TDD helps in software design and enhances the code quality and that writing tests encourages significant refinement in class design, as a result produce promotes cohesiveness, and loosely coupled classes [1][17] [63][64]. But the exact impact of TDD in actually guiding the developers during class design still remained unclear. Therefore a recent study [62] inspects the developers' views and inclinations on how the practice of TDD correlates with class design. The qualitative study was conducted with 25 participants from 6 different companies in Brazil. Some of the results derived are useful and needed to be further explored:

- Only TDD or writing unit tests won't be beneficial. For better class designing, the practical exposure and previous skills in object-orientation and design principles is what counts more.
- Promotes simpler design as without perceiving the test cases, a developer think about the class design at once, resulting in a more complicated and misunderstood structure than needed.
- Changing the class design was easier and more confident because of refactoring and test suites.
- Constant feedback improves confidence.

TDD in Open Source software

OSS has gained increasing adoption rate but despite its popularity, the quality, credibility, and development process of OSS projects have been researched over since many years. One reason is that the OSS projects do not use a single development practice but varied development practices, this in turn enhances the complexity [55][56].OSS projects have been researched to be suffering from inadequate testing. The research [57][58] focused on the use of TDD in OSS.

The team conducted an empirical study on 2500 OSS projects downloaded from GitHub and analyzed their test files using an automatic script. Different project characteristics were studied with respect to establishing a correlation between tests after to get an understanding and results to establish if certain parameters correlate with the usage of TDD. However, the results found only weak correlations and none of them were statistically symbolic. Thus, conclusion was drawn that the analyzed project characteristics do not have much correlation with the usage of TDD.

TDD and its variants

TDD evolved as a refinement of "Test First" development [66] but there are variants to the TDD as well. For example, Acceptance Test-Driven Development (ATDD)[71] creates acceptance tests before coding begins, based on the team's understanding of the requirements. This requires even more discussion with the requirements' authors, creating deeper collaboration and furthering the understanding of the requirements, by the developers and the authors. ATDD helps developers to transform requirements into test cases and allows verifying the functionality of a system. ATDD supports automation of Acceptance test. Another style of development is Behavior Driven Development (BDD) [67][69][73] that combines unit tests and acceptance tests within specific contexts. BDD is generally quoted to have evolved because of some limitations in TDD and ATDD. Both of them emphasize on system state and verification of the state of the system rather than the desired behavior of the system, and that test code is tightly coupled with the actual systems' implementation [73]. But BDD mainly concentrates on the implementation phase of a software project and provides minimal support to the analysis phase, and almost negligible planning phase [70]. While TDD can be implemented on its own, without the use of ATDD or BDD, the scenario and the preconditions decides which framework most appropriately supports a given team or project**.**

The Advantages, Pitfalls and Complexities of Test-Driven Development

General Discussion

Various studies depicted in the previous section have evaluated and concluded that application of TDD for software in the development phase has many advantages. Though the test-driven development facilitates the incorporation of requirement changes easily, results in better and cleaner code, but still lot of developers believe that test driven development is followed by a number of loop holes. In this section, some of the pitfalls of TDD are highlighted. A comprehensive summary of the benefits and drawbacks of TDD with respect to various parameters have been projected in Table1:

Table 1: A Summary of the advantages and pitfalls of employing Test Driven Development

Parameters	**Advantages**	**Disadvantages**
Defect Intensity	• Quality improvement by fixing bugs at the earliest during development (by means of continuous and rigorous testing and refactoring) [19][27][3][44][10][11] • 42% lower defect density[44][60]	• More defects found in TDD based software.[28] • Even a wide range of test cases does not guarantee that software is free of bugs and errors [58]
Programmer's productivity	Good, Higher [19] academic scenario	• The industrial case studies reported a decrease in productivity [21][24][3]. • Great amount of time and focus may be spent on writing tests rather than adding new functionality.[35] • 19 % extra effort [44]
Time		• Projects reported to have consumed 15% extra time for writing the tests.[20][6] • In case of frequent design change, configuring new test suites every time would eventually consume more time.[25]
Quality(internal and external)	• Significant increase in quality of the code.[17][20] [21][23][2] • Complexity and reuse better[35] • Improvement not exactly attributed to TDD.[72]	• Coupling and cohesion are worse [35]. • External quality improved, but with a perceived less productivity. [61]
Refactoring / Learning Curve	• Enhances code understanding and hence the learning as refactoring calls for regular improvement. • In the long run it may be faster. Refactoring code written long back becomes complex gradually. But if that code is supported by a good suite of unit tests, the whole process is eased up.	• Refactoring knowledge becomes mandatory for the developer. • Needs a good knowledge of the code smells and the respective refactorings. Shouldering double responsibility. • Not all developers may be refactoring experts that may lead to a bad design. • Usually Floss refactoring may impact negatively [39][37][36] • Regression bugs and build breaks.[50]
Limiting the Scope	• Scope creeps (extra code for contingency) are avoided and not needed.[33]	• Not thinking for the future could lead to major change requirements and additional refactoring needs.
Maintainability	• Improvement to an extent[2][5].A study showed that development time actually increased from 15% to 35% but was compensated by reduced maintenance.[46]	
Confidence	• Improves confidence[31]	• But may also create false illusions or sense of security.
Cost	• In project 1 the results depicted that less time was spend on rework and bugs in comparison with a project where no TDD approach was used.[40]	• In the supervised projects the developers took 31% more time for developing the modules and components[41] • If the productivity of TDD is declined say by 10% as compared to the productivity of the conventional project, a 5 % improvement in defect-reduction-efficiency satisfies for TDD to break-even with the traditional process (1.7: 1). If the productivity of TDD is much lower, and recedes to 70 % less, then even an improvement of 80 % defect reduction-efficiency does not suffice to a better result as compared to the conventional process (BCR is 1 : 1.1).[59]

Speed	• TDD improves the speed as developers don't need to spend time in debugging.[31]	• Lot of effort and time is required to design test cases and maintain proper test suites.
Test Quality	• Improvement noted from pilot studies.[35] • Testing effort reduced. [12]. • If tests are designed post implementation then the risk involved is that they may satisfy the implementation, not the requirements[49]	• Study on student project reported that the branch coverage is greater for the TDD, the mutation score indicator measured worse than the score of other students who were engaged with a test-last approach.[26][40] • Writing good test cases and avoiding superfluous cases require expertise otherwise could lead to an extra effort. • Writing unit tests for GUI code is a difficult job. [65]

The study conducted on 25 test driven development case studies [24] in a statistical meta-analysis focused on the feature of productivity and external quality. They were categorized as academic or industrial. The results from the study depicted that there could be improvement in the external quality, but the result varied depending upon which development process was used by the reference group. The results were found to have improved with test-driven development in larger, and longer, industrial projects but the academic experiments did not witness the same outcome. The programmer productivity, was still unclear with indecisive results. Therefore undesirable difference between the academic experiments and industrial case studies could be witnessed than with external quality.

The summary of result reported in [3] included significant positive impact linked with defects along with significant favorable results associated to external quality, complexity, maintainability and size as well. They also reported negative effects that were primarily associated with productivity and effort although in one study test-driven development witnessed a reduction in effort. Whereas many other studies pointed that effort had no effect. A significant conclusion was drawn from this study, regarding effort and productivity; there could be some unknown context parameters that have an influence on the effect of test-driven development.

The other researches carried out by students [27][28] responded slightly better with respect to the number of defects when teams followed test-driven development .In another experiment, they concluded that code inspections fared better and proved more efficacy in detecting defects than test-driven development; that is, more defects still persisted in the code that employed test-driven development than in the code that had undergone code inspection. Maintainability has not been assessed so far in a detailed way as yet by the research community. Though the study [5] had specifically focused on maintainability, the encouraging result that was achieved was, increased maintainability and reduced effort required maintaining code later but at the time of the review there was only a single study. Test coverage is also an important area focused by a lot of researchers. A student based research explored that the students using TDD had higher branch coverage was but, the mutation score indicator was actually worse than the score of other students who were developing their code with a test-last approach [26].

A detailed research done to gather quantitative evidence on the effects of the TDD on internal code quality, external quality, productivity, and test quality was carried out by his team in their book [35]. The evaluation was based on data gathered from 32 trials. The trials were conducted in an academic or industrial setting in the form of controlled experiments, pilot studies, or commercial projects.

Some good results were visible from the trials [35] .The evidence states that TDD does not have a consistent effect on internal quality. Although TDD appears to perform better over the control group for certain types of metrics (complexity and reuse), other metrics (coupling and cohesion) are often worse in the TDD treatment. Another observation from the trial data is that TDD yields production code that is less complex at the method/class level, but more complex at the package/project level. The external quality results still fared better but no proper evidence was reported. About the productivity usually it seemed to have decreased due to an extra amount of effort required from the developer but still there is one line of argument that claims productivity to increase with TDD; reasons include easy context switching from one simple task to another, improved external quality (i.e., there are few errors and errors can be detected quickly) whereas some researchers and engineers argue that TDD will negatively impact productivity because too much time and focus may be spent on authoring tests as opposed to adding new functionality.

Another assessment done over quality and productivity using regression models to assess the impact of TDD [36] reported that Quality and productivity improvements were primarily positively associated with the granularity and uniformity and the order of sequencing of test and production code , had no important impact. Quality was measured by functional correctness. Also refactoring effort was negatively associated with both outcomes.

Author has identified some of the benefits which is associated

with the TDD namely efficiency and feedback, low-level design, reducing the defect injection and test assets. The fine granularity of test and cycle of the code provides continuous feedback to software developer is main advantage among efficiency and feedback of TDD.

Though TDD is termed as a practice of defect-reduction [6] because of its tight feedback loops. With the help of TDD, defects are determined as new code and it is included to the system and also identified the source of problem. It was also found that fault efficiency and decrease in debug time has compensated for the extra time spent in writing a test case and executing. Apart from these, the TDD does not create the detrimental effect on the software developer productivity.

Speed of development has also been assessed in [31] and is concluded that Test-Driven Development assists in developing code faster. They further elaborated that the code that is hard to test often falls short of automated test, which renders the code difficult to refactor.

The evaluation of cost-effectiveness was also done but not by calculating in monetary values, but how in terms of time/effort saved when using TDD. The assessment was carried out by monitoring the time developers spent on rework and errors in a project. Besides working on rework and bugs, there maybe also other factors which could be affected by the introduction of TDD and could affect the cost-effectiveness of the practice. In project 1 we monitored that the average overhead was 31%. This means that developers took 31% more time developing items when using a TDD approach than when they do not use a TDD approach to implement work items [41].Another study on Return of Investment of TDD concludes that there are two factors contribute to the ROI of TDD: the difference between the productivity of conventional process and test-driven development, and the capacity of test-driven development to deliver higher quality code [59].

TDD and Testing Complexities

Practically performing the "real TDD (read: test first with the red, green, refactor steps) involves performing the usage of mocks/stubs, when performing integration tests. ("*Stubs are hand-written classes that mimic the dependency's behavior, but do that with significant shortcuts. For example, a stub for an order repository can store orders in memory and return those orders as a result of search operations instead of querying a real database. Whereas Mocks are dynamic wrappers for dependencies used in tests"*)[42]

The usage of mocks, after a while, is followed by the usage of Dependency Injection (DI) and a Inversion of Control (IoC) container. To accomplish this task the user interfaces is required for everything (Now that is an overhead and finally, it results in having the developer write a lot more code, than doing it in the conventional way.) Therefore the task of merely writing order c lass has to be accompanied by an interface, a mock class, some IoC configuration and a few tests.

The tests also require a good design on top of that. And they too need to be maintained and regenerated. A good test should have a clear readability like any other code and that is a time consuming task. A good test suite works as a regression safety net on bugs: A bug should be handled by a proper test case so it could be projected and removed by changing the production code and all other tests may succeed. All previous bug fixes are verified by each successful run.

Therefore the developers need to carry out all these activities "the right way". It sometimes gets more complex than was perceived. Often the usage of TDD makes the project very difficult to understand. The unit tests not designed properly often test the wrong thing, the wrong way. And to decide what a good test would look like is again debatable.

These results in making the changes to the behavior of the system all the more complicated (opposite to refactoring) and time consuming. TDD method in the process of software development was explored in another research [5] that projected the benefits of implementing TDD similar to automation of test code that is probability to do constant assurance of code quality. This provides immediate feedback to developers regarding their code state and number of faults remaining in the code are identified in later testing as well as at consumer suites. When automated testing is carried out late in cycle of development, it leads to all faults just prior to delivery to consumer. Corrections of identified faults will result in re-testing that delays the products' delivery.

A few studies have investigated other approaches as well to study the cost-effectiveness. One of these is Acceptance Test-Driven Development (ATDD) [43][40]. Acceptance tests are test written (preferably by the customer) to test whether the User Story meets the acceptance criteria. Benefits associated with the ATDD approach are that the developer does not have to define the test scenarios himself. In addition, the developer does not have to worry about mock objects and interfaces, because the tests do not have to be independent from external sources. This could remove a lot of overhead created by the more traditional approach of TDD.

Refactoring and TDD

Another significant development contributed by TDD is the applicability of refactoring techniques during the course of development, that had almost taken a backseat in routine software development and its usually when a software goes into a maintainable state that the developers think of refactoring the code. The refactoring is the core activity in the process of Test-Driven Development. Refactoring forms the core of test driven development , though generally it is most frequently applied in the maintenance of legacy software to

remove the code smell and the most beneficial are improved readability and maintainability [29][30][31][50]. In the same study [50][74],the authors shows that the biggest problem faced by most of the developers during refactoring are regression bugs and build breaks. Refactoring may involve removal of duplication, other design improvements, or replacement of temporary, stub code. But the application of refactoring should ensure behavior preservation, thus each refactoring activity should prove that all the existing tests should pass, in accommodating the design change. However, in practice refactoring is often inter-mixed with other activities, and as such, it sometimes involves adding new production code, engaging novel analytical processes, dealing a code smell in multiple ways [38] as well as changing existing tests and behavior. These practical deviations, however, sometimes tend to act negatively, also potentially nullify or reverse the hypothesized benefit [37]. The conclusions given by the author pointed out that most cycles detected as refactoring cycles could have been associated with floss refactoring [39], a practice that involves other activities as well with refactoring. A typical example could be injecting some extra code for \enhancements or sneaking in production code that implements new functionality while refactoring. The developer who refactors the code may at some point realize that a piece of functionality is missing or may be some change should be incorporated for betterment and mixes in the proper production code, but the code is not covered by any tests, possibly leading to a bug or an anomaly.

In TDD the code that makes the test pass generally is the minimal code required to achieve a task and make the test pass and is mostly not a great-looking code but it gives an indication to the developer that they are proceeding on the right way, though the code may still need refactoring. Though a lot of automation has been already incorporated in the area of refactoring, still the application of correct refactoring need manual intervention and correct understanding of the coding anomaly. Apart from the regular refactorings a lot of other improvements and enhancements such as the usage of Aspect oriented techniques [30][32] along with the object oriented techniques and the procedural techniques for code improvement have been introduced and is being continuously added to the refactoring field. This requires the developer to have a good understanding of the coding practices and the code smells that are already known and how to appropriately apply so as not to introduce behavioral changes in the software.

Beck and Fowler [34] describe this using a metaphor of "two hats": developers can wear an implementation hat or a refactoring hat, but cannot wear both at once. But in TDD refactoring plays an essential role, it implies that the developer apart from the testing technique should also possess a good exposure to refactoring that to an extent increases the responsibility and effort. Halting the refactoring process is endorsed by the developer and he decides to move on to the next task. As had already been stated in the previous section that a team analyzed that there were more defects in TDD based software than in the code that was assessed specifically by a code inspection group. Experience of the developers in coding, testing and refactoring in general seems to impact the ability of the developer to adapt the test-driven development process.

"Is TDD Dead" controversy

Another significant controversy that surrounded TDD was the debate about TDD being dead [51][52] by Hasnen that resulted into a number of online discussions between Kent Beck and Martin Fowler [53][54].He further quoted that overzealous testing can actually damage the design. Their discussion basically revolved around mocking, "Red green refactor" cycle and tests [54].

Hanson disapproves of mocks, he insists that testing the test units separately , introduces additional layers of complexity .The counter by Fowler is that you might end up with the same design and may choose to isolate the classes even with no tests or TDD at all. Beck added to it that there are numerous reasons for isolating and decoupling your classes but the design ideas should be used wisely , to use them every time and to everything may enhance the complications. Hansson further makes a strong point by stating the unrequired changes to design for the purpose of testing only contributes to test–induced damage.

Another point made by Hansson is regarding the red, green, refactor cycle. He calls for the usage of this cycle as an option rather than like a mandate to be compulsive all the time. He insists that there may be common scenarios when a developer might not be able to perceive the test to be written but has a picture of implementation. Another problem was that people tend to under refactor the code due to lack of adequate exposure to refactoring.

He further argues that TDD is achieved with its own bargains and sometimes the developer may have to bear the loss to follow TDD. He quotes an example: "*If you achieve 95% uptime of your system but you want to achieve 99% or even 99.99%, the effort to move to the next step grows exponentially. Of course it can make sense for critical systems to go for the highest possible uptime but we don't build such systems most of the time."*

He further emphasizes that it is an arduous task to find out whether a software development technique is really apt or not due to its dependence on various parameters. He further quotes the reason for the neglection in refactoring could be that the developers are more concerned in running the next test so the design becomes less important and code becomes under–refactored.

CONCLUSION

It has been found by a number of researchers that TDD proves advantageous over TLD with respect to internal and external software quality, but the developers' productivity tends to reduce as compared to TLD [21][20][25]. Experiment results to evaluate the efficacy of TDD in [22] suggests that the statistics result related to McCabe's Cyclomatic complexity, number of lines of code per ,number of acceptance test cases passed, branch coverage, person hours, number of user stories implemented per person hours are statistically inconsequential. Also the study projected that mostly developers are inclined to adopt TLD over TDD, as lesser effort and learning is required to understand and implement TLD than TDD.

The study conducted on Open Source Software as a sample for evaluation explores that Test-Driven Development provides a remarkable upswing in code quality in the categories of cohesion, coupling, and code complexity [23]. There was an approximate improvement of 21% when using TDD over the code developed using the test-last technique. Test-Driven Development proved efficient in decreasing the code complexity, with an improvement of 31% and an improvement of 21% in cohesion metrics but not much difference was witnessed in the coupling metrics. Although the writer claims that the sample size was small so taking a larger sample would better validate the results.

Many developers are of the opinion that adopting TDD may require a rigorous training over testing and refactoring and a steep learning curve may initially impact the productivity in a negative way, there is no well concluded evidence on the lasting impact and enhanced test quality (i.e., there is less probability of introducing fresh errors due to automated tests). A counter argument to this theory by some researchers is that TDD incurs too much overhead and will most likely decrease the productivity because majority of the time is devoted to authoring tests instead of adding new code and functionality.

The benefit of TDD projected by a research team precisely articulated that it may not necessarily be due to the characteristics test-first dynamic, but rather the reason is that TDD-like processes promotes small, fine and steady steps that boosts the focus and flow [36]

Despite the fact that test dependencies lead to complications, the process of writing tests during the development process remained consistent. The major problem with writing unit tests in different projects was formulation of tests for everything that needed to be tested. Although we could not find too many experimental evidences supporting TDD still a few facts can be easily established in support of TDD from the analysis:

- It facilities modular code that is easier to understand and debug. This may eventually help the developers to detect early defects and fix them
- The tests can act as documentation and enhance the understandability of the code.
- Maintenance and refactoring of code becomes easier.
- It might be slow when started but could enhance the speed of development in the long run.
- It supports the developers to think from an end user point-of-view.
- The suite of unit tests facilitates constant feedback system about the state of a component whether it is still working; this in turn boosts the confidence of the developer that the software is working fine.
- TDD can also contribute the process of class designing, but the developer needs to possess specific expertise and skill to achieve that.
- There is a difference between TDD and unit tests. The unit tests only verify the behavior whereas the test in TDD drives the development.
- TDD is also used in OSS and Embedded software.
- If not applying TDD in its original form then there is an option to focus on Behavior Driven Development or ATDD (Acceptance TDD), of course depending on the situation.

It's good not to engage in TDD when:

- The system design is not clear. Sometimes it evolves as the project progresses but this might force the developer to rewrite the test several times which will generate a big time lose. Therefore it is advisable to postpone the unit tests until the design is well sketched in your mind.
- Sometimes unit testing may become more time consuming and inappropriate as not all real life systems can be generalized and simplified for unit testing and a forced unit testing may lead to devoting more time in rectifying the unit tests than performing the real testing.
- With legacy systems and applications not supporting a proper unit-testing framework, it might not always be a feasible option.
- Not all developers might be good testers and refactoring experts.
- It is suited for libraries but does not go well with the GUI applications. Lot of features in a project is hard to test or can't be tested at all like the server errors that require the usage of mock-objects.

The tabular results also give a quick grasp of the advantages and disadvantages of TDD with respect to

different parameters. From the above table we can further conclude that there are very few studies that contribute to assess the programmer's productivity, the time and cost involved in TDD and also the impact on maintainability of the system, so these may serve as future directions of research.

However, for a new project with concrete specifications and a defined design that would be prone to minimal changes, TDD could serve as rapid and economical route to a quality product. Therefore let the TDD fit in the system rather than making the system design fit to TDD.

REFERENCES

[1] Shrivastava D P and Jain R C, "Innovative Engineering Technique of Object Oriented Software Development", *Journal of Engineering Science and Management Education*, Vol-3, pp 41-46. 2010

[2] Kumar P R, Raju G K J and Maruthuperumal S. "An External Quality Supporting Test-Driven Development of Web Service Choreographies", *International Journal of Computer Applications*.2014

[3] Makinen S and Munch J, Effects of Test-Driven Development. "A Comparative Analysis of Empirical Studies", Proceedings, Lecture Notes in Business Information Processing. Springer. 2014

[4] Piet Cordemans , Sille VanLandschoot, Jeroen Boyden s, Eric Steegmans, "Test-Driven Development as a Reliable Embedded Software Engineering Practice". **Chapter** Perspective Volume 520 of the series Studies in Computational Intelligence pp 91-130, Date: 20 November 2013.

[5] Duka D and Hribar L (n.d), "Test Driven Development Method in Software Development Process". 2010.

[6] Williams L, Maximilien EM et al., "Test-driven development as a defect-reduction practice". In: Proceedings of the IEEE International Symposium on Software Reliability Engineering, Denver, CO. IEEE Computer Society, Washington, DC. 2003

[7] Buffardi K and Edwards S H. "Impacts of Teaching Test-Driven Development to Novice Programmers", *International Journal of Information and Computer Science*, 1 (6), pp 135-143. 2012

[8] Markovic D et al. "Test-Driven Development of IEEE 1451 Transducer Services and Applications", *Telfor Journal*, 4 (1). 2012

[9] Guerra E and Aniche M. "Achieving Quality on Software Design through Test-Driven Development".2013

[10] Parsons D, Lal R and Lange M. "Test Driven Development: Advancing Knowledge by Conjecture and Confirmation", *Future Internet*, Vol-3, pp 281-297. 2011

[11] Bulajic A, Sambasivam S and Stojic R. "Overview of the Test Driven Development Research Projects and Experiments", Proceedings of Informing Science & IT Education Conference (InSITE). 2012

[12] Khan N et al, "Reducing Testing Effort in the Test Driven Development", Global *Journal of Computer Science and Technology Software and Data Engineering*, 13 (7). 2013

[13] AlHammad N et al, "Comparison between Test-Driven Development and Conventional Development: A Case Study", *International Research Journal of Electronics and Computer Engineering*, 2 (1). 2016

[14] Sanchez J C, Williams L and Maxmilien E M. "A Longitudinal Study of the Use of a Test-Driven Development Practice in Industry", IBM Corporation and North Carolina State University. 2007

[15] Ress A P, Moraes R O and Salerno M S, "Test-Driven Development as an Innovation Value Chain", *Journal of Technology Management and Innovation*, Vol-8. 2011

[16] A. Causevic, S. Punnekkat, and D. Sundmark, "Quality of Testing in Test Driven Development," in Quality of Information and Communications Technology (QUATIC), 2012 Eight International Conference on the, September 2012

[17] Beck, K. "Test Driven Development -- by Example". Boston, Addison Wesley. 2003(book)

[18] Hans Wasmus "EVALUATION OF TEST-DRIVEN DEVELOPMENT An Industrial Case Study", Report TUD-SERG-2007-014, ISSN 1872-5392. 2007(Thesis)

[19] Madeyski and Lukasz Sza la, "The Impact of Test-Driven Development on Software Development Productivity — An Empirical Study", in Software Process Improvement, ser. Lecture Notes in Computer Science, P. Abrahamsson, N. Baddoo, T. Margaria, R. Messnarz, Eds., LNCS 4764. Springer, 2007, pp.200-211.)

[20] Bhat, T., Nagappan, N. "Evaluating the Efficacy of Test-driven Development: Industrial Case Studies". In: Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering. ISESE '06, New York, NY, USA ACM (2006) 356–363.)(Conference Proceedings)

[21] Wilson,B. etal, "The effects of test driven

development on internal quality, external quality and productivity: A systematic review", Information and Software Technology Volume 74, June 2016, Pages 45–54

[22] Munir.H.,etal., "An experimental evaluation of test driven development vs. test-last development with industry professionals", Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, London, England, United Kingdom — May 13 - 14, 2014 , ACM New York, NY, USA ©2014

[23] Rod Hilton, "Quantitatively Evaluating Test-Driven Development by Applying Object-Oriented Quality Metrics to Open Source Project", December 19, 2009.

[24] Rafique, Y., Misic, V. "The Effects of Test-Driven Development on External Quality and Productivity: A Meta Analysis". IEEE Transactions on Software Engineering 39(6) (2013) 835–856.

[25] Dogša, T., Batič, D. "The Effectiveness of Test-Driven Development: An Industrial Case Study". Software Quality Journal 19 (2011) 643–661

[26] Pančur, M., Ciglarič, M. "Impact of Test-Driven Development on Productivity, Code and Tests: A Controlled Experiment". Information and Software Technology 53(6) (2011) 557–573

[27] Vu, J., Frojd, N., Shenkel-Therolf, C., Janzen, D. "Evaluating Test-Driven Development in an Industry-Sponsored Capstone Project". In: Proceedings of the Sixth International Conference on Information Technology: New Generations. ITNG '09 (April 2009) 229–234

[28] Wilkerson, J., Nunamaker, J.J., Mercer, R. "Comparing the Defect Reduction Benefits of Code Inspection and Test-Driven Development". IEEE Transactions on Software Engineering 38(3) (May-June 2012) 547–560

[29] Rizvi, S., Khanam, Z. "A methodology for refactoring legacy code." In: International Conference on Electronics Computer Technology (ICECT 2011), pp. 198–200 (2011), IEEE Xplore.

[30] Khanam, Z. and Rizvi, S. "Refactoring Catalog for Legacy software using C and Aspect Oriented Language"- the proceedings of SERP 2011, USA.

[31] Shore, J., & Warden, S. (2007). "The art of agile development. Sebastopol, CA:" O'Reilly Media, Inc. 7, 33

[32] Rizvi S, Z Khanam, "A Comparative Study of using Object oriented approach and Aspect oriented approach for the Evolution of Legacy System" International Journal of Computer Application, United States, ISSN, 0975-888.

[33] Beck, K., & Andres, C. "Extreme programming explained." Boston, MA: Addison-Wesley Professional. 5,2004.

[34] Fowler, M., Beck, K., Brant, J., Opdyke, W., & Roberts, D. "Refactoring: Improving the design of existing code." Boston, MA: Addison-Wesley Professional. 12, 33, 34, 67. 1999

[35] B. Turhan, L. Layman, M. Diep, H. Erdogmus, and F. Shull, "How Effective is Test Driven Development?," in Making Software: What Really Works, and Why We Believe It (A. Oram and G. Wilson, eds.), pp. 207–219, Cambridge, MA: O'Reilly, 2010.

[36] Davide Fucci ; Hakan Erdogmus ; Burak Turhan ; Markku Oivo ; Natalia Juristo, "A Dissection of Test-Driven Development: Does It Really Matter to Test-First or to Test-Last?",published in IEEE Transactions on Software Engineering (Volume: PP, Issue: 99), 18 October 2016 .

[37] G. Bavofta, B. De Carluccio, A. De Lucia, M. Di Penta, R. Oliveto, and O. Strollo, "When does a refactoring induce bugs? An empirical study," in Source Code Analysis and Manipulation (SCAM), 2012 IEEE 12th International Working Conference on. IEEE, 2012, pp. 104–11.

[38] Z Khanam, SAM Rizvi. "Aspectual Analysis of Legacy Systems: Code Smells and Transformations in C", International Journal of Modern Education and Computer Science, Volume 5, Issue 11, Page 57, 2013.

[39] Murphy-Hill, C. Parnin, and A. P. Black, "How we refactor, and how we know it," IEEE Transactions on Software Engineering, vol. 38, no. 1, pp. 5–18, 2012

[40] Madeyski, L. "The Impact of Test-First Programming on Branch Coverage and Mutation Score Indicator of Unit Tests: An Experiment." Information and Software Technology 52(2) (2010) 169–184

[41] Dennis de Bode, "Cost-Effectiveness of Test-Driven Development ", Master's Thesis, 2009.

[42] https://msdn.microsoft.com/enus/library/aa730844(v=vs.80).aspx.

[43] Grigori Melnik and Frank Maurer. "Multiple perspectives on executable acceptance test-driven development." In Agile Processes in Software Engineering and Extreme Programming, 8th International Conference, XP 2007, Como, Italy,

June 18-22, 2007, Proceedings, volume 4536 of Lecture Notes in Computer Science, pages 245–249. Springer, 2007.

[44] Weikko Aejmelaeus, "Test-driven development", Thesis, Arcada University of Applied Sciences Helsinki, 2009.

[45] P. Sfetsos, I. Stamelos, Empirical studies on quality in agile practices: "A systematic literature review", 2010. QUATIC '10 Proceedings of the 2010 Seventh International Conference on the Quality of Information and Communications Technology,PP 44-53,ACM Digital Library.

[46] R.Aguilar, "Using test-driven development to improve software development practices", thesis, Reykjavik University, 2016.

[47] https://derickrethans.nl/talks/tdd-pfcongrez09.pdf

[48] https://www.petrikainulainen.net/programming/unit-testing/from-top-to-bottom-tdd-for-web-applications/ (2013)

[49] Elssamadisy, Amr. 2008. Agile Adoption Patterns, A roadmap to Organizational Success. Addison-Wesley. ISBN: 0-321-51452-1.

[50] Kim, M., Zimmermann T., & Nagappan, N. "An empirical study of refactoring challenges and benefits at Microsoft". IEEE transactions on software engineering, Vol. 40:7. 2014.

[51] Hansson, D. H. Tdd is dead. Long live testing. 2014

[52] Hansson, D. H. Test-induced design damage. 2014

[53] HALL, A. Anti-test-driven development arguments and myths.2010

[54] Petr Jasek, Ivan Aaen, "Test–Driven Development: 15 years later", Aalborg University, Sept, 2014.

[55] Deshpande, A., & Riehle, D. "Continuous Integration in Open Source Software Development." In B. Russo, E. Damiani, S. Hissam, B. Lundell, & G. Succi, Open Source Development, Communities and Quality (Vol. 275, pp. 273-280). Springer US.(2008)

[56] Michlmayr, M., "Software Process Maturity and the Success of Free Software Projects". Proceedings of the 2005 conference on Software Engineering: Evolution and Emerging Technologies (pp. 3-14). 2005(IOS Press Amsterdam))

[57] Jari Hanhela, "Usage of Test-Driven Development in Open Source Projects", Master's Thesis Jari Hanhela Spring, 2015.

[58] Kochhar, P. S., Bissyande, T. F., Lo, D., & Jiang, L. "Adoption of Software Testing in Open Source Projects A Preliminary Study on 50,000 Projects". 17th European Conference on Software Maintenance and Reengineering (pp. 353- 356). Genova: IEEE.2013)

[59] 59. M. Müller and F. Padberg. "About the return on investment of test-driven development". In International Workshop on Economics-Driven Software Engineering ResearchEDSER-4, 2003.

[60] J. Boydens, P. Cordemans, E. Steegmans. "Test-driven development of embedded software", in Proceedings of the Fourth European Conference on the Use of Modern Information and Communication Technologies, 2010)

[61] Munir H, Moayyed M, Petersen K. "Considering rigor and relevance when evaluating test driven development: a systematic review". Inf Softw Techno 56(4): 375–394.(2014).

[62] Aniche, M. & Gerosa, M.A. J Braz Comput Soc, 21: 15. doi:10.1186/s13173-015-0034-z.(2015)

[63] Martin R. "Agile principles, patterns, and practices in C#." 1st edition. Prentice Hall, Upper Saddle River.(2006)

[64] e Nat Pryce SF, "Growing object-oriented software, Guided by Tests". 1° edn. Addison-Wesley Professional, Boston, USA.(2009)

[65] Sadeh B., Ørbekk K., Eide M.M., Gjerde N.C.A., Tønnesland T.A., Gopalakrishnan S., "Towards Unit Testing of User Interface Code for Android Mobile Applications". In: Zain J.M., Wan Mohd W.M.., El-Qawasmeh E. (eds) Software Engineering and Computer Systems. ICSECS 2011. Communications in Computer and Information Science, vol 181. Springer, Berlin, Heidelberg.2011

[66] Agile Java: Crafting Code with Test-Driven Development, by Jeff Langr (2005)

[67] The Cucumber Book: Behavior-Driven Development for Testers and Developers, by Matt Wyne and Aslak Hellesoy.2012

[68] Clean Code: A Handbook of Agile Software Craftsmanship, by Robert C. Martin.2008.

[69] D. North, "Introducing BDD", 2006. Available at: http://dannorth.net/introducing-bdd

[70] Carlos Solís and Xiaofeng Wang. "A Study of the Characteristics of Behaviors Driven Development", Published in the proceedings of the 2011 37th EUROMICRO Conference of Software Engineering and Advanced Applications,383-387.ACMDigital Library(2011))

[71] L. Koskela. "Test Driven: TDD and Acceptance TDD for Java Developers", Manning Publications,

2007.

[72] D. Janzen and H. Saiedian. "Does Test-Driven Development Really Improve Software Design Quality?" IEEE Software. vol. 25, no. 2, 2008.

[73] D. Chelimsky, D. Astels, Z. Dennis, A. Hellesoy, D. North. The RSpec book: "Behavior Driven Development with Rspec", cucumber and friends, Pragmatic Bookshelf, 2010.

[74] Z Khanam, SAM Rizvi. "Assessment of the Impact of Aspect Oriented Programming on Refactoring Procedural Software". Computers and Mathematics in Automation and Materials Science, MATHIMA-24,Cambridge,USA.2014