International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 04 Issue-17
December 2017

Effects of Developer Experience on Learning and Applying Unit Test-Driven Development

K.Rizwana Khanam & P.Viswanatha Reddy
1 M.Tech Student (SED), Sir Vishveshwaraiah Institute Of Science & Technology Hyderabad
Email: rizwanakhanam.cse@gmail.com
2Sr. Assistant Professor, Department of CSE, Sir Vishveshwaraiah Institute Of Science & Technology Hyderabad

ABSTRACT:

Unit test-driven development (UTDD) is a software development practice where unit test cases are specified iteratively and incrementally before production code. In the last years, researchers have conducted several studies within academia and industry on the effectiveness of this software development practice. They have investigated its utility as compared to other development techniques, focusing mainly on code quality and productivity. This quasi-experiment analyzes the influence of the developers' experience level on the ability to learn and apply UTDD. The ability to apply UTDD is measured in terms of process conformance and development time. From the research point of view, our goal is to evaluate how difficult is learning UTDD by professionals without any prior experience in this technique. From the industrial point of view, the goal is to evaluate the possibility of using this software development practice as an effective solution to take into account in real projects. Our results suggest that skilled developers are able to quickly learn the UTDD concepts and, after practicing them for a short while, become as effective in performing small programming tasks as compared to more traditional test-last development techniques. Junior programmers differ only in their ability to discover the best design, and this translates into a performance penalty since they need to revise their design choices more frequently than senior programmers.

INTRODUCTION:

TEST-DRIVEN development (TDD) is a technique to incrementally develop software where test cases are specified before functional code. Originally, TDD referred to creating unit tests before production code. However, recently, another technique applying a test-driven development strategy at the acceptance test level is gaining attention. In this sense, in the last years, it is usual to distinguish between unit test-driven development (UTDD), which targets unit tests to ensure the system is performing correctly; and acceptance test-driven development (ATDD), a technique focused on the business level. Here we aim our attention at UTDD.Despite its name, UTDD is not a testing technique, but a programming/design practice. In the last years, several studies and experiments have analyzed the influence of this practice on software in terms of software quality and productivity within academia and industry. In literature, UTDD sometimes appears as one of the most efficient software development techniques to write clean and flexible code on time. Nevertheless, these studies report conflicting results (positive, negative and neutral about the use of UTDD) depending on several factors. Thus, no definite conclusions can be drawn, which limits the industrial adoption of this development practice. One of the main problems with UTDD is the difficulty in isolating its effects from other context variables. So, the influence of these context variables must be analyzed in detail to determine the real benefits and cost of adopting UTDD. For example identified seven possible factors limiting the industrial adoption of this development technique. One of these factors is the programmers' experience. Nevertheless, as far as we know, just a few empirical studies investigate directly or indirectly the effect of developer experience on applying UTDD. In the context of these studies, experience (or knowledge) usually refers to the degree of practice or theoretical insights in UTDD, and the qualification of the participants ranges between UTDD novices and UTDD experts, where novices are often students. These studies, for example, analyze the correlation between programming experience and code quality highlight some aspects contributing to the adoption of a TDD strategy in an industrial project or compare the characteristics of experts' and novices' UTDD development process. Results here are conflicting again.

LITURATURE SURVEY:

An Initial Investigation of Test Driven Development in Industry: - Cu wire is increasing in usage in semiconductors due to continuous package cost reduction. Raising bar for reliability requirements, especially customers in automotive industry, has posted numerous challenges for Cu wire in meeting stringent quality requirements. This study is triggered by customer to development Grade 0 temperature cycles (TC) reliability with lower cost. Current package is running with Au wire Grade 1 TC reliability. In order to have better profit margin, internal decision was targeted on bare Cu wire to run with Grade 0 TC reliability at the initial stage. The focus of this paper is development of Cu wire for QFP robust wire necking prevention in Grade 0 temperature cycling (TC) reliability. Wire necking is one the major reliability concern in Grade 0 TC. The first failure of Grade 0 TC is observed with massive open failure after TC500X Grade 0 stress test. Further FA confirmed that wire is rapture due to wire necking. At the same time, the FA expert zooming into the each comer to quantify & classify the failure mode. The failure is localized at north/west of the wire bonded area this area is where the lead frame is not symmetry in design. Other area showed minor or no crack line no broken wire observed. In order to meet the Grade 0 TC, the investigation is streaming into 3 directions: First, process driven weaknesses for bare Cu wire. Second, comprehensive simulation was done in order to foster the development understanding and lastly, Cu wire material understanding. In process driven weaknesses, after series of wire bond & moulding process provocation, only two key indicators showed influence: vibration control during wire bond with different clamper design & symmetry lead frame design influence to stress distribution. Wire bond clamper with reduced vibration on lead finger (spring loaded design) had significantly improved the zero hour on first bond with no micro line or surface dislocation. Despite also improve the second bond wedge peeling significantly. However, after TS 1000X, crack line is observed again on pin 86 & 92 (which is the bad corner). While in symmetry lead frame design, minor crack line is observed after TS 1000X. However, the bad comer effect was deleted as all location observed certain degree of minor crack line.

A Structured Experiment of Test Driven Development: - Utilization solar panels to generate electricity were increased all over the world including Indonesia. Generally, solar panels mounted on permanent structure which the angle of sunlight direction is fixed. This installation method will reduce the efficiency of electrical energy generated by solar panels due the angle of sun direction changes every time. There are needs equipment that used to steer solar panels toward the sun, namely solar tracking. In this study, we proposed development a prototype of two-axis solar tracking using five photodiodes. Overall, solar tracking system consists of two DC motors, five photodiodes, microcontroller system, and mechanical structure. DC motors are used to driving toward azimuth and elevation, photodiode act as sun-sensor functioned to measure the angle of sunlight direction, microcontroller system functioned to process data and give signal to DC motors. The five photodiodes usage are intended to reduce the cost purchasing of sun-sensor which relatively expensive on the market. Solar tracking prototype has been tested using artificial light with an intensity is closer to the sunlight. Experiment result has shown that this solar tracking is able work very well and can follow the sun movement.

SYSTEM ANALYSIS:

Existing system:- The main goal of our study, this is an interesting result that suggests that UTDD performance in the industrial environment could be similar to the performance with other traditional techniques.

Junior programmers in the same way as intermediates and seniors were able to quickly learn and apply the UTDD concepts but in general they had limitations regarding the design decisions they made in each UTDD cycle. Therefore, although they could properly use UTDD, in general they had a worse performance due to the time needed to modify and/or correct the existing code.

Advantage:-The efficiency of the learning process is measured as tradeoffs between the correct application of the UTDD concepts and the additional time for the development due to the learning curves with UTDD. We analyze the evolution during the experiment of the development process conformance to UTDD and the effective development time and the self-training needed

to completely implement a set of requirements. In our analysis, we first verify that the functional code produced during the experiment satisfies the requirements and has the expected behaviour.

Proposed system:- UTDD changes, the changes satisfying that unit tests were written before related pieces of the application code. We include in UTDD changes those cases where the subject does not validate that the test fails after a test code change (weak UTDD change). To identify UTDD changes we apply the same rules proposed By.Refactorings, the changes in the structure of the code that do not alter its observable behaviour.

Disadvantage:-We defined the study to be able to compare from different perspectives the learning curves with UTDD of professional developers with different levels of experience, but all of them inexperienced in test-first practices. The collected data include, between others, all the changes made to the functional and test code during the learning process and the time stamp and the result of each unit test invocation. These data allow us to analyze the effort required for adapting the programmer's mindset to UTDD.

SYSTEM SPECIFICATION:

Hardware Requirements:-

- System : Pentium IV 2.4 GHz.
- Hard Disk : 40 GB.
- Floppy Drive : 1.44 Mb.
- Monitor : 15 VGA Colour.
- Mouse : Logitech.
- RAM : 256 Mb.

Software Requirements:-

- Operating system : Windows 7.
- Front End : Dot net
- Database : SQL SERVER 2008
- Tools : Visual Studio 2010

SYSTEM STUDY:

Feasibility study:- The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential

Three key considerations involved in the feasibility analysis are

- Economical feasibility
- Technical feasibility
- Social feasibility

Economical feasibility:-This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

Technical feasibility:-This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

Social feasibility:-The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some

constructive criticism, which is welcomed, as he is the final user of the system.

SYSTEM TESTING:

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

Types of tests:-

Unit testing:-Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing: - Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields.

Functional test:- Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals. Functional testing is centered on the following items: Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

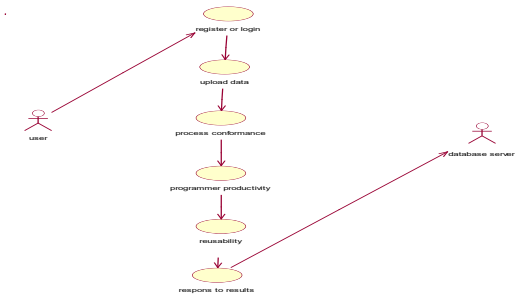
System Test: - System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing:-White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

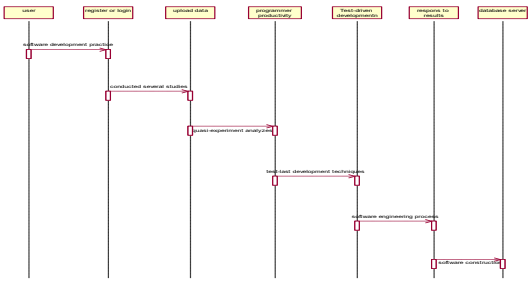
Black Box Testing: - Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works.

UML DIGRAMS:

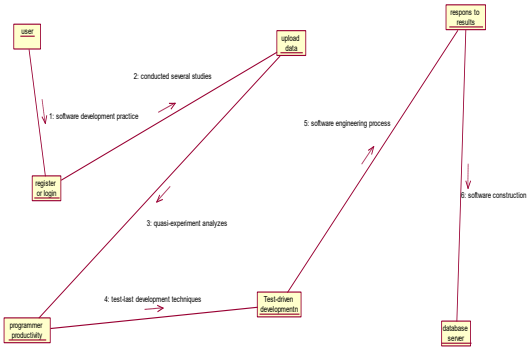
Use Case Diagram:-



Sequence Diagram:-



Collaboration Diagram:-



SOFTWARE ENVIRONMENT:

Features OF. Net:- Microsoft .NET is a set of Microsoft software technologies for rapidly building and integrating XML Web services, Microsoft Windows-based applications, and Web solutions. The .NET Framework is a language-neutral platform for writing programs that can easily and securely interoperate. There's no language barrier with .NET: there are numerous languages available to the developer including Managed C++, C#, Visual Basic and Java Script. The .NET framework provides the foundation for components to interact seamlessly, whether locally or remotely on different platforms. It standardizes common data types and communications protocols so that components created in different languages can easily interoperate. ".NET" is also the collective name given to various software components built upon the .NET platform. These will be both products (Visual Studio.NET and Windows.NET Server, for instance) and services (like Passport, .NET My Services, and so on).

The .net framework:-The .NET Framework has two main parts:

1. The Common Language Runtime (CLR).

2. A hierarchical set of class libraries.

The CLR is described as the "execution engine" of .NET. It provides the environment within which programs run.

<table>
<tr><td colspan="2">ASP.NET
XML WEB SERVICES</td><td>Windows Forms</td></tr>
<tr><td colspan="3">Base Class Libraries</td></tr>
<tr><td colspan="3">Common Language Runtime</td></tr>
<tr><td colspan="3">Operating System</td></tr>
</table>

.Net Framework

Languages supported by .net:- The multi-language capability of the .NET Framework and Visual Studio .NET enables developers to use their existing programming skills to build all types of applications and XML Web services. The .NET framework supports new versions of Microsoft's old favourites Visual Basic and C++ (as VB.NET and Managed C++), but there are also a number of new additions to the family. Visual Basic .NET has been updated to include many new and improved language features that make it a powerful object-oriented programming language. These features include inheritance, interfaces, and overloading, among others. Visual Basic also now supports structured exception handling, custom attributes and also supports multi-threading. Visual Basic .NET is also CLS compliant, which means that any CLS-compliant language can use the classes, objects, and components you create in Visual Basic .NET. Managed Extensions for C++ and attributed programming are just some of the enhancements made to the C++ language. Managed Extensions simplify the task of migrating existing C++ applications to the new .NET Framework's# is Microsoft's new language. It's a C-style language that is essentially "C++ for Rapid Application Development". Unlike other languages, its specification is just the grammar of the language. It has no standard library of its own, and instead has been designed with the intention of using the .NET libraries as its own. Microsoft Visual J#

.NET provides the easiest transition for Java-language developers into the world of XML Web Services and dramatically improves the interoperability of Java-language programs with existing software written in a variety of other programming languages. Active State has created Visual Perl and Visual Python, which enable .NET-aware applications to be built in either Perl or Python. Both products can be integrated into the Visual Studio .NET environment. Visual Perl includes support for Active State's Perl Dev Kit.

Other languages for which .NET compilers are available include

- FORTRAN
- COBOL
- EIFFEL

Features of SQL-SERVERT:-he OLAP Services feature available in SQL Server version 7.0 is now called SQL Server 2000 Analysis Services. The term OLAP Services has been replaced with the term Analysis Services. Analysis Services also includes a new data mining component. The Repository component available in SQL Server version 7.0 is now called Microsoft SQL Server 2000 Meta Data Services. References to the component now use the term Meta Data Services. The term repository is used only in reference to the repository engine within Meta Data Services-SERVER database consist of six type of objects, they are,

- 1. TABLE
- 2. QUERY
- 3. FORM
- 4. REPORT
- 5. MACRO

MODULE DESCRIPTION:

Test-driven development:-The study were industrial developers with computing degrees and several years of experience using traditional methodologies based on test-last development strategies. However, since our main goal was to compare their learning curves with UTDD, not having a previous experience with test-driven development strategies was mandatory. The subjects belonged to three different companies and voluntarily participated in the study because of a personal invitation. In an experiment with students classified the difficulties the subjects find with the adoption of UTDD in three categories: UTDD approach difficulties, designing of test problems and technical difficulties.

Software engineering process:-The ratios of changes tended to stabilize in a nearly constant value as the study progressed. the ratio of UTDD changes grew; while the ratio of other changes dropped. This result confirms that self-training during the development of the initial sets of requirements translated into a better knowledge in UTDD. In average, the number of UTDD changes was larger and similar for intermediate and senior programmers than for junior developers. The ratio of refactoring for seniors was nearly constant during the whole experiment; but, at the end, juniors made more refactoring.

Software construction:-We evaluate the subjects' efficiency during the learning process in order to investigate if the learning curves generated additional time for the development. Performance is analyzed based on the effective time taken to develop a group of three requirements. The possible performance penalties may be a consequence of self-training and/or of the developer's learning curve with UTDD. To quantitatively assess the real effort required by the subjects to properly use this development technique, we compare their performance with UTDD and a non-UTDD development strategy.

Processconformance: Developmentprocessconformance, programmingefficiency mainly depended on the programmer's experience level. Both intermediate and senior programmers were able to efficiently use UTDD with a similar performance to more traditional test-last techniques with just a little practice. In these cases, the use of UTDD had a minimal impact on productivity. Although this is not the main goal of our study, this is an interesting result that suggests that UTDD performance in the industrial environment could be similar to the performance with other traditional techniques.

CONCLUSION:

The principal conclusion drawn from our study is that skilled developers with the appropriate knowledge and without any prior experience in test-first development

can quickly learn the UTDD rules and, after practicing them for a short while, properly apply them in small programming tasks. Interestingly, there even existed some subjects whose development process conformed to UTDD from the beginning of the experiment. In other words, they did not need training or learning period to apply UTDD in the right way. Furthermore, it also seems that the research subjects were able to retain the UTDD knowledge and even use it in their companies within the industrial environment.

REFERENCES

[1] K. Beck, Test Driven Development: By Example. Addison-Wesley Professional, 2003.

[2] D. Astels, Test Driven Development: A Practical Guide. Prentice Hall Professional Technical Reference, 2003.

[3] H. Erdogmus, G. Melnik, and R. Jeffries, "Test-Driven Development,"Encyclopedia of Software Engineering, pp. 1211-1229,Taylor & Francis, http://www.informatik.unitrier.de/~%20ley/db/reference/se/se2010.html, 2010.

[4] L. Koskela, Test Driven: Practical Tdd and Acceptance Tdd for Java Developers. Manning Publications Co., 2007.

[5] E. Hendrickson's, "Acceptance Test Driven Development (Atdd):An Overview," Proc. Seventh SoftwareTestingAustralia/NewZealand(STANZ).Wellington, 2008.

[6] K. Beck, "Aim, Fire," IEEE Software, vol. 18, no. 5, pp. 87-89, Sept./Oct. 2001.

[7] D. Janzen and H. Saiedian, "Test-Driven Development: Concepts, Taxonomy, and Future Direction," Computer, vol. 38, no. 9, pp. 43-
50, Sept. 2005.

[8] D. Janzen and H. Saiedian, "Does Test-Driven Development Really Improve Software Design Quality?" IEEE Software, vol. 25,no. 2, pp. 77-84, Mar./Apr. 2008.

[9] E.M. Maxim lien and L.A. Williams, "Assessing Test-Driven Development at IBM," Proc. 25th Int'l Conf. Software Eng. (ICSE),pp. 564-569, 2003.

[10] H. Erdogmus, M. Morisio, and M. Torchiano, "On the Effectiveness of the Test-First Approach to Programming," IEEE Trans.Software Eng., vol. 31, no. 3, pp. 226-237, Mar. 2005.

[11] T. Bhat and N. Nagappan, "Evaluating the Efficacy of Test-Driven Development: Industrial Case Studies," Proc. ACM/IEEE Int'l Symp. Empirical Software Eng. (ISESE '06), pp. 356-
363, 2006.

[12] N. Nagappan, E.M. Maxim lien, T. Bhat, and L. Williams, "Realizing Quality Improvement through Test Driven Development:
Results and Experiences of Four Industrial Teams," Empirical Software Eng., vol. 13, no. 3, pp. 289-302, June. 2008.

[13] B. George and L. Williams, "An Initial Investigation of Test Driven Development in Industry," Proc. ACM Symp. Applied Computing
(SAC '03), pp. 1135-1139, 2003.

[14] B. George and L. Williams, "A Structured Experiment of Test-Driven Development," Information and Software Technology, vol. 46,
No. 5, pp. 337-342, 2004.

[15] L. Crispin, "Driving Software Quality: How Test-Driven Development Impacts Software Quality," IEEE Software, vol. 23, no. 6,pp. 70-71, Nov./Dec. 2006.

[16] R. Jeffries and G. Melnik, "Guest Editors' Introduction: Tdd–theArt of Fearless Programming," IEEE Software, vol. 24, no. 3,pp. 24-30, May 2007.

[17] A. Causerie, D. Sundmark, and S. Punnekkat, "Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review," Proc. IEEE Fourth Int'l Conf. Software Testing, Verification
And Validation (ICST), pp. 337-346, 2011.

[18] M. M€uller and F. Padberg, "An Empirical Study About the Feelgood
Factor in Pair Programming," Proc. 10th Int'l Symp. SoftwareMetrics, pp. 151-158, 2004.

[19] R. Latorre, "A Successful Application of a Test-Driven Development Strategy in the Industrial Environment," Empirical Software.