I J C T A, 10(8), 2017, pp. 613-618 ISSN: 0974-5572

Test Lifecycle in Behavior Driven Development

G.S. Mahalakshmi* and V. Vani**

ABSTRACT

Testing is essential in modern day software development. With traditional development approaches slowly fading away, software testing has been emerging as the most dominant stage in product development. Test driven development (TDD) appeared as a light house with emphasis on test-first approach. However, it is slowly transforming to take the new avatar of Behavior Driven Development (BDD) as a measure to incorporate the touch of artificial intelligence into product development methodologies. This paper proposes the test life cycle of BDD.

Keywords: STLC, TDD, BDD, software testing

1. INTRODUCTION

Software testing plays a crucial role in ensuring software quality. During the early stages of software development, requirements or the software product to be developed are prescribed in order to explain the intended usage for the product. As the system grows, it should confine itself to these requirements. Generating test cases early helps test engineer to find ambiguities and inconsistencies in the requirements specification and design documents. This is the very nature of Behavior Driven Development (BDD). The aim is to reduce the cost of building the software systems as errors are eliminated early during the life cycle. In other words, every stage thereafter requirements gathering will comply with the theoretical test cases (we name it so..) generated out of BDD requirements specification. Therefore generation of test cases as early in the cycle would solve and prevent the software from multiple fractures. Test Driven Development (TDD) handles the same from a different perspective. TDD insists on test-first approach where before start of coding, the developers design test cases, called pseudo test cases and then attempts to build the code around them. The software testing life cycle (STLC) (Figure 1.1) also has to be different than general STLC.

Behavior Driven Development (BDD) is much better than TDD, in which the objective is not only to write early test cases but also to emphasize the system behavior in the test cases itself. i.e. a function named as cust ( ) in TDD test case generated as early in the requirements phase shall be named appropriately as customer_lookUp ($param_0$, $param_1 \ldots param_n$) in BDD so as to inculcate the very nature of the function and class objective as tied up with the name of the class or function itself. This practice will avoid regression errors creeping into the test cases during TDD refactoring. However, the design engineer is yet allowed full freedom over designing the customer lookup function and the respective classes and their hierarchies and, but only restricted to those. Therefore BDD helps to implement TDD in a cleaner and bug free manner and avoids future surprises in the form of refactoring the code. One strong foundation to implement BDD is generation of automated test cases from use cases because use cases actually carry the essence of customer requirements from functional perspective.

* Department of Computer Science & Engineering, Anna University, Chennai, Tamilnadu, INDIA, *Email: gsmaha@annauniv.edu*

** Department of Information Science & Technology, Eswari Engineering College, Chennai, Tamilnadu, INDIA, *Email: vanimecse@gmail.com*

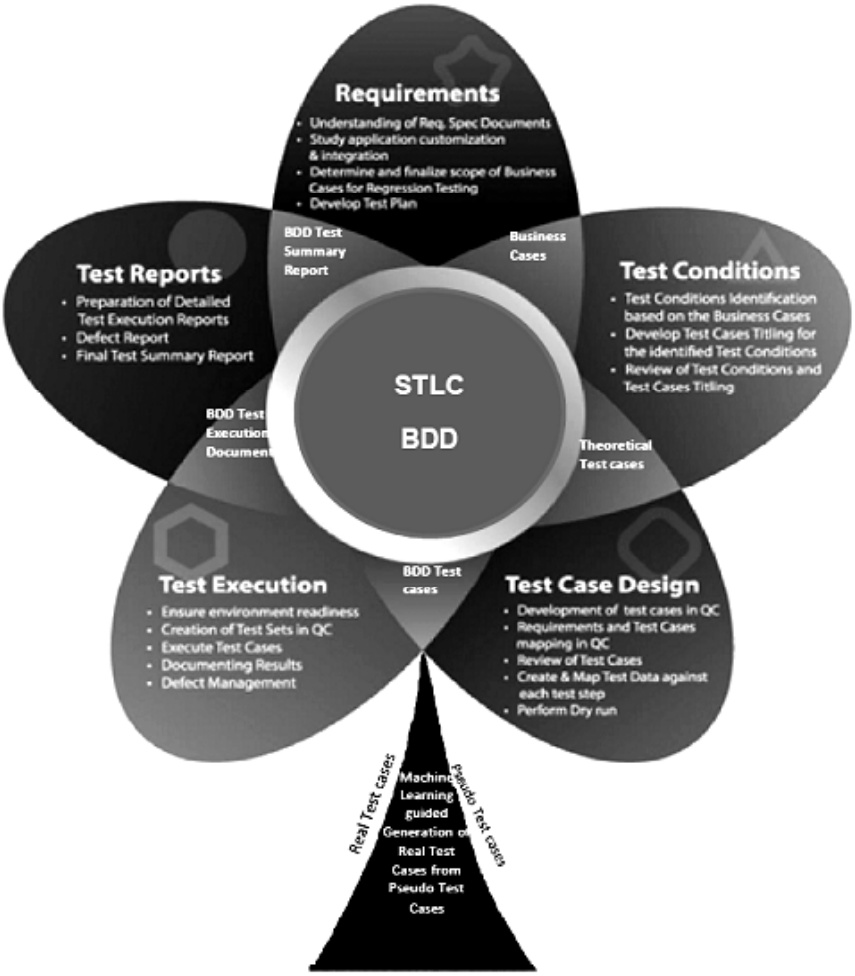

Figure 1: Stages in Software Test Life Cycle in BDD

The test conditions are generated from business cases translated into design model during TDD, whereas, in BDD the business cases are observed from behavioral specifications like narrative stories, acceptance criteria or user scenarios (Figure 1). Therefore the designed test cases are theoretical in nature and have to be trace run for functional mappings during early iterations of STLC. Later on, as the process pass to the later stages of product development, the test cases evolve into pseudo test cases and then towards the end of coding, turn as real test cases.

2. RELATED WORK

The traditional approach model driven development was quite convincing in terms of understanding but not for practice as it involved manual intervention in especially the key region of translation of requirements. Models for involving requirements engineer in agile based development [1] have been proposed. Few works on automation of black box test case generation is attempted upon customer requirements [2], at times, surpassing use cases [3]. Aspectual use cases are also examined for automated test case generation [4]. Tool support like Requirements Use Case Tool (RUT) [5] is built to support customer – developer interactions. Using such tools would reduce [6] testing costs enormously. Behavior trees [7] are also applied in translation of requirements to use cases. Rule based approaches to observe visible errors from requirement specifications [8][9] also exist. Construction of requirements specifications from execution patterns could also be applied for automatic SRS development [10][11]. Classification of requirements as good and bad [12][13], design classification [14][15] and test inputs [16][17] provide role models for inclusion of AI techniques [18][19] and derived methodologies into SE practice.

In addition, work related to requirements miss-estimation and project completion times [20], [21] predictive models for determining software quality [22],[23],[24], [25] are also proposed. Canfora [26] proposes methodologies for identifying work products which could be impacted by change in requirements or fixing of a critical bug. In particular, the software engineering community has utilized the following broad disciplines [27] from AI: 1. Search and optimization 2. Fuzzy and probabilistic reasoning 3. Classification, Clustering, Learning and Prediction

3. MOTIVATION

Automated test case generation from use cases is quite challenging and requires the involvement of artificial intelligence in various forms. Right from the interpretation of natural language use cases, interpreting the functional requirements, alternate flows, etc. needs enormous amount of text processing. One should remember that there is very high human intervention in writing SRS (Software Requirement Specifications) which encompasses the high level and detailed use cases. BDD focuses on generation of generic pseudo-test cases out of requirements. To compliment these procedures, we propose methodologies for automated test case generation from use cases using various artificial intelligence approaches.

Our motivation is to enable BDD developers with the proposed theoretical pseudo test case (we name it so...) which facilitate the development of pseudo-test cases for TDD. Therefore, utilising machine learning approaches and Natural Language Processing (NLP) Techniques for automated development of BDD test cases would serve as a compelling objective for the proposed area of research.

4. PREVIOUS WORK

In this context, we have earlier proposed few approaches [28][29][30][31] to generate test cases from use cases with the view of application of artificial intelligent techniques to automated generation of theoretical pseudo test cases. They are: 1. BDD approach to Software development, 2. Interpretation of Business Cases from SRS, 3.Generation of Theoretical test cases from Use cases, 4.Generation of BDD pseudo Test cases from Theoretical Test Cases, 5. Machine learning based semi-automated approaches to generate BDD Real Test Cases from BDD pseudo Test Cases, 6. Pattern based approach to generation of test cases, 7. Generation of unique test case patterns 8. Using Scenario diagrams for Generation of test case patterns, 9. Application of Named Entity Recognition (NER) for identification of Actor names, 10. Use of Maximum Entropy Model (MEM) Named Entity Recognition (NER) for generation of named entities from use cases, 11.Use case Scenario Matrix Generation using Named Entities, 12. Test Case Generation based on use case scenario matrix, 13. Coverage Analysis for generated test cases, 14. Automatic depiction of use cases into UDGs (Use case dependency graphs), 15. Graph based approaches to test case generation using NEs, 16. Theoretical Verification of Automated test cases and 17. Backward Process: Generation of intermediate use case elements from automated test cases.

Our earlier attempts have proved that it is possible to incorporate AI techniques to transform TDD into a working BDD. This paper discusses the generic testing lifecycle of this transformation.

5. PROPOSED WORK: STLC OF BDD

Though the figure 2 is self-explanatory, the interpretation goes as follows. The transformation of TDD is starting well-ahead right from the stages of requirements engineering. The lifecycle traverses three ideal stages, requirements, design and implementation (aka coding & testing). With detailed use cases now as part of software requirement specifications (SRS), almost every business scenario is depicted as use cases. By interpreting the dependencies between user scenarios, user scenario matrix could easily be constructed. This facilitates the generation of BDD test cases which are purely written as natural language text segments. These we refer as TTC or Theoretical test cases. TTC has to go undergo the test of pass/fail criteria with

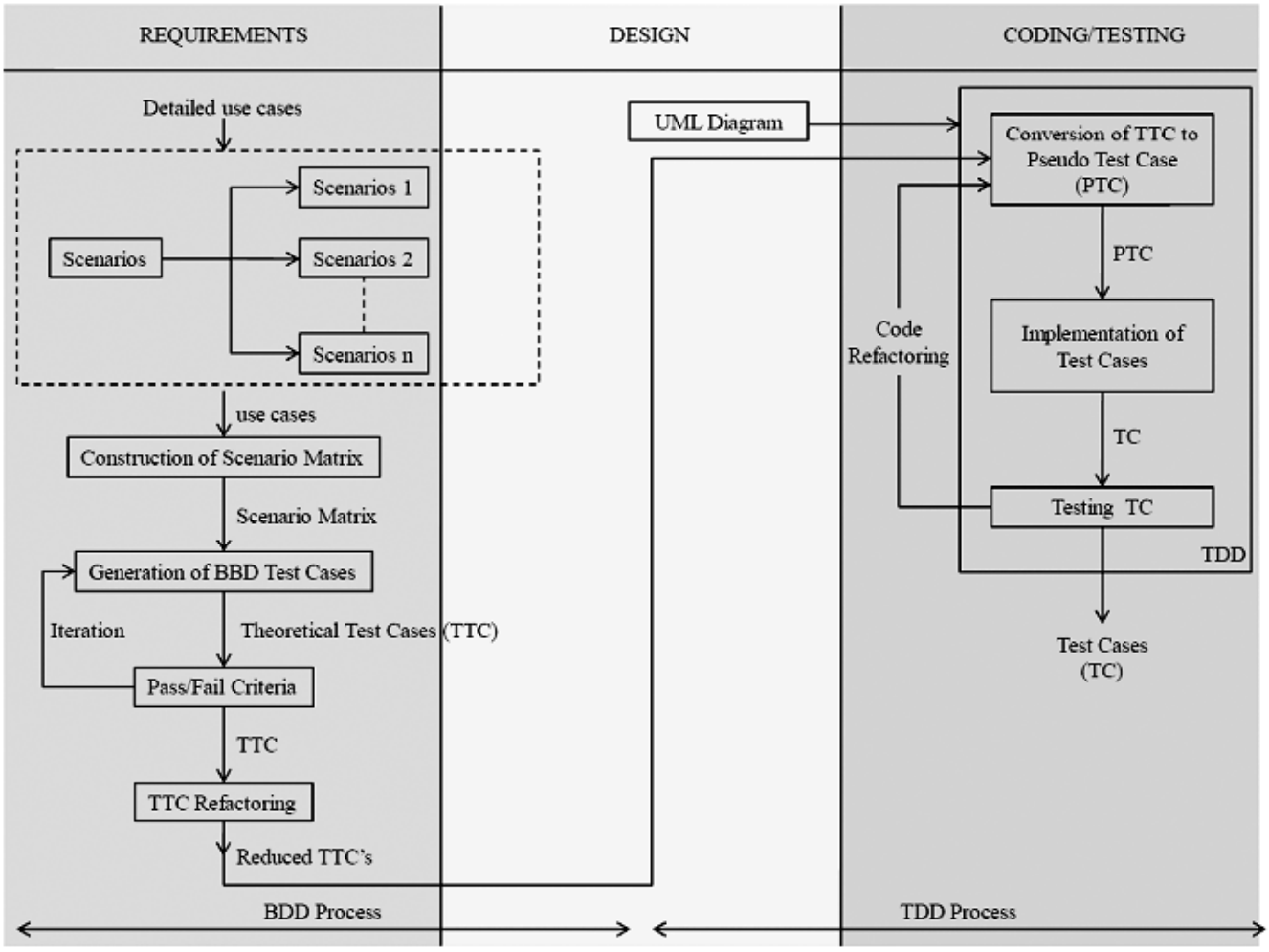

Figure 2: STLC in BDD – operating view

respect to stated requirements; if not, there is still enough room for the customer to intervene and explain their viewpoint. Since the TTCs are in natural language text, involving text mining and machine learning algorithms are essential to validate any viewpoint at this juncture.

With every fail criteria, TTCs take a new avatar and undergo iterations until they pass convincingly. Gradually the TTCs would have grown complex and therefore, refactoring the idea may be required as well at this point. With enough refactoring in place, the TTCs now get shrinked in size but retain the same power effect after having passed the customer requirements criteria. These reduced TTCs are absorbed into UML diagrams and the design is said to complete. It is in this design stage that, the BDD process is over and TDD is slowly emerging. We cannot identify any strong demarcation of the borderline; however, once the design parts are getting confirmed to be passed to subsequent stages, we say BDD process fades to give way for original TDD.

The implementation stage gets both reduced TTCs and the design diagrams so as to cross-check for any error. If correspondence is satisfied, next step is to convert the reduced TTC into pseudo test case (or PTC). PTCs are used for implementation purposes with the same test-first approach and the real TC (Test case) evolves. This undergoes code refactoring if required in iterations and the final Test case is obtained, and eventually the coding would have been finished.

6. CONCLUSION

This paper discussed the interesting aspect of testing life cycle of BDD. This is a breakthrough work in the field of modern software engineering. However, emphasis is only given to text mining aspect of AI.

Interpretation of UML diagrams and other SRS business diagrams would serve a meaningful role in taking BDD totally under the control of AI.

REFERENCES

[1] Cunning,SJ &Rozenblit,JW 1999, 'Automatic Test Case Generation from Requirements Specifications for Real-time Embedded Systems', in Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, pp.784-789.

[2] AjithaRajan 2006, 'Automated requirements-based Test case Generation', In ACM SIGSOFT Software Engineering Notes,vol. 31, no.6, pp.1-2.

[3] Praditwong, Harman,M & Yao,X 2011, 'Software module clustering as a multi-objective search problem', In IEEE Transactions on Software Engineering, vol. 37, no. 2, pp. 264-282.

[4] Chunhui Wang, FabrizioPastore, ArdaGoknil, Lionel Briand &ZohaibIqbal 2015, 'Automatic generation of system test cases from use case specifications', In Proceedings of the 2015 International Symposium on Software Testing and Analysis (ISSTA 2015), ACM, New York, NY, USA, pp. 385-396.

[5] Harman,M2011, 'Software engineering meets evolutionary computation', IEEE Computer, vol. 44, no. 10, pp. 31-39.

[6] Ghashghaei, M, Bagheri, E, Cuzzola, J., Ghorbani, AA &Noorian, Z 2016, 'Semantic Disambiguation and Linking of Quantitative Mentions in Textual Content'. International Journal of Semantic Computing, vol. 10, no. 01, pp.121-142.

[7] Rayadurgam,S &Heimdahl,MP 2001, 'Coverage based test-case generation using model checkers', In Proceedings of the 8th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, pp.83-91.

[8] Raiha, O 2010, 'A survey on search–based software design',In Computer Science Review-Elsevier, vol. 4, no. 4, pp. 203-249.

[9] Prowell,SJ 2005, 'Using markov chain usage models to test complex systems',In Proceedings of the 38th Annual Hawaii International Conference on System Sciences,Washington, DC, USA: IEEE Computer Society, pp. 318.3.

[10] Challagulla,VUB,Bastani,FB,Yen,IL & Paul,RA 2008, 'Empirical assessment of machine learning based software defect prediction techniques', In International Journal on Artificial Intelligence Tools, vol. 17, no. 2, pp. 389-400.

[11] Bertrand Meyer, Arno Fiva, IlincaCiupa, Andreas Leitner, Yi Wei & Emmanuel Stapf 2009, 'Programs That Test Themselves', In IEEE Computer Society, vol.42, no. 9, pp. 46-55.

[12] Harman, MS, Mansouri,A &Zhang, Y 2012, 'Search-based software engineering: Trends, techniques and applications', ACM Comput. Surv.45, vol. 1, Article 11, pp. 61.

[13] Sarma, M & Mall, R 2007, 'Automatic Test Case Generation from UML Models', In 10th International Conference on Information Technology, IEEE computer society, pp. 196-201.

[14] Knuth Donald, Morris James, H& Pratt Vaughan 1977, 'Fast pattern matching in strings', In SIAM Journal on Computing, vol. 6, no. 2, pp. 323-350.

[15] Livshits, B & Zimmermann, T2005, 'Dynamine: finding common error patterns by mining software revision histories', ACM SIGSOFT Software Engineering Notes, vol. 30, no. 5, pp.296-305.

[16] Dickinson, W, Leon, D &Podgurski, A 2001, 'Finding failures by cluster analysis of execution profiles', In Proceedings of the 23rd international conference on Software engineering, IEEE Computer Society pp.339-348..

[17] Fenton,E, Neil,M,Marsh,W,Hearty,P,Radlinski,L & Krause,P 2008, 'On the effectiveness of early life cycle defect prediction with Bayesian Nets',In Empirical Software Engineering, vol. 13, no. 5, pp. 499-537.

[18] De Millo, RA& Offutt, AJ 1991, 'Constraint-based automatic test data generation', IEEE Transactions on Software Engineering, vol.17,no.9, pp. 900-910.

[19] Menon, PR 2016, 'Behavior-Driven Development Using Specification by Example: An Approach for Delivering the Right Software built in Right Way', Emerging Innovations in Agile Software Development, pp.237.

[20] Antoniol, G, Gu´eh´eneuc, YG, Merlo, E &Tonella, P 2007, 'Mining the lexicon used by programmers during software evolution', in Proceedings of the IEEE International Conference on Software Maintenance, pp.14-23.

[21] Fraser, G &Arcuri,A 2013, 'Whole Test Suite Generation', In IEEE Transactions on Software Engineering, vol. 39, no. 2, pp. 276-291.

[22] Bird, C, Gourley, A, Devanbu, P, Gertz, M &Swaminathan, A 2006, 'Mining email social networks', in Proceedings of the International Workshop on Mining Software Repositories, pp.137-143.

[23] Clerissi, D, Leotta, M, Reggio, G &Ricca, F 2016, 'A Lightweight Semi-automated Acceptance Test-Driven Development Approach for Web Applications', In International Conference on Web Engineering, Springer International Publishing, pp. 593-597.

[24] Hu, Y, Chen, J, Rong, Z, Mei, L &Xie, K 2006, 'A Neural Networks Approach for Software Risk Analysis', In Proceedings of the Sixth IEEE International Conference on Data Mining Workshops, Hong Kong. Washington DC: IEEE Computer Society, pp. 722-725.

[25] Anvik, J, Hiew, L & Murphy, GC 2006, 'Who should fix this bug?', in Proceedings of the 28th International Conference on Software Engineering, pp.361-370.

[26] Bouktif,S,Sahraoui,H &Antoniol,G 2006, 'Simulated annealing for improving software quality prediction',In Proceedings of the 8th annual conference on Genetic and evolutionary computation, Seattle, Washington, USA: ACM Press,vol. 2, pp. 1893–1900.

[27] Fraser, G &Arcuri,A 2011, 'Evolutionary generation of whole test suites', in 11th International Conference on Quality Software QSIC, M. Nu´nez, ˜ R. M. Hierons, and M. G. Merayo, Eds. Madrid, Spain: IEEE Computer Society, pp. 31-40.

[28] Mahalakshmi, GS,Vani, V&Betina Antony, J 2018, 'Named Entity Recognition for Automated Test case Generation',International Arab Journal of Information Technology,vol.15,no.4(in press).

[29] Vani,V, Mahalakshmi, GS&Betina Antony, J 2015,'Generation of Patterns for Test cases',International Journal of Science and Technology,vol. 8,no.24.

[30] Vani, V, Mahalakshmi, GS&Betina Antony, J 2015, 'Graph Based Test Case Generation',International Journal of Applied Engineering and Research,vol.10,no.18,pp.3991-39100.

[31] Vani, V&Mahalakshmi, GS 2016, 'Theoretical Verification of Test Cases for Behavior Driven Development', in proceedings of Second International Conference on Recent Trends and Challenges in Computational Models (ICRTCCM'17), organized by Department of Computer Science and Engineering, University College of Engineering Tindivanam, 2017 (accepted)