International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Development of Functional Test Cases Using FSM and UML Activity Diagrams for MDT

[1]Dr. Vinod H. Patil, [2]A. Deepak, [3]Dr. Himanshu Sharma, [4]Lavanaya Vaishnavi D. A., [5]Upendra Singh Aswal, [6]Mr. K. K. Bajaj, [7]Dr. Anurag Shrivastava

Submitted: 06/12/2023 **Revised**: 17/01/2024 **Accepted**: 27/01/2024

Abstract: This study's testing of The SDLC's software phase is crucial. This stage does more than only evaluate the software's quality. Model-driven testing and code-driven testing are the two categories that make up this testing procedure. Code-driven testing is based on testing the entire program, line by line. A control flow and data flow that is sequential drives code-driven testing. The model-driven testing method focuses on the executable module rather than each line of code since it supports third-party modules, APIs, and components. One need not be an expert in every field because the tester will focus on how well each component works individually. The combination of an extended finite state machine with a sequence diagram is covered in this essay. One of the most important areas that must be effectively handled to provide effective testing of the given project is model-driven testing. The system that has been implemented combines UML with FSM to cover every situation and all potential outcomes. This situation inspires us to establish a framework for generating test cases automatically, covering every potential pathway (leveraging UML activity diagrams to account for all pathways) and situations (utilizing Finite State Machine to describe various scenarios). The Finite Machine also operates based on triggers, where scenarios are established, and if these criteria are met, the subsequent action is carried out. Model-driven testing is created by considering all scenarios, pathways, and situations.

Keywords: *Extended File System, Model-Driven Testing, Finite State Machine, Coverage of paths and conditions, Activity Diagram.*

1. Introduction

Software testing is an important phase in the software development life cycle since it allows the product's quality to be verified. This stage enables evaluation of the program's suitability for the user's demands. Several testing methods can be used to achieve this, with test case development being the most successful. All components or features will be examined in these test scenarios. All performance-related parameters for these components or capabilities are anticipated to meet the benchmark. The chosen software development life cycle, such as the waterfall model, spiral model, and others, has an impact on the testing approaches used. The particular stage emphasized, hereafter referred to as a phase, dictates the chosen testing approach. Testing stands out as the single most critical phase in the software creation process. The field of software testing is continually advancing. Recent frameworks like TestNG prove highly effective for data-driven testing, particularly when dealing with substantial volumes of data. Among the most popular technologies available today is the Selenium suite, comprising Selenium Web Driver, Selenium Grid, and Selenium RC tools for remote control and distributed computing. QTP, a Windows-based testing tool, is employed to validate software running on the Windows operating system. Industry standards necessitate an agile and adaptable approach. All of this promotes a methodology where

1Department of E&TC Engineering, Bharati Vidyapeeth (Deemed to be University) College of Engineering, Pune
vhpatil@bvucoep.edu.in
2Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences, Saveetha University, Chennai, Tamilnadu
**deepakarun@saveetha.com*
3Associate Professor, Department of Computer Engineering and Applications, GLA University, Mathura
himanshu.sharma@gla.ac.in
4Assistant Professor, Dept of ECE., R L JALAPPA INSTITUTE OF TECHNOLOGY, DODDABALLAPUR, KARNATAKA
lavanyavaishnavi@gmail.com
5Associate Professor, Department of Computer Science and Engineering, Graphic Era Deemed to be University, Dehradun, Uttarakhand
upendrasinghaswal@geu.ac.in
6RNB Global University, Bikaner
vc.kkb@rnbglobal.edu.in
7Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences,
Chennai, Tamilnadu
Anuragshri76@gmail.com

each component can be assessed for compliance with specifications. The fields of electronics, mechanical engineering, civil engineering, and bioinformatics have all witnessed substantial automation growth. This underscores the importance of software quality checks across these domains. At the heart of the software testing process lie unit and functionality tests encompassing all scenarios. The rigour and approach employed in testing both impact the software's quality. This form of quality assessment can be conducted in conjunction with UML diagrams, which allow for the examination of diverse software functionalities and dimensions.

2. Literature Survey

The selection of research papers focusing on Model Driven Testing involved an examination of various scholarly works. Several important methodologies were considered to define the problem statement and provide guidelines for addressing existing issues.

A method known as ESG4WS, developed by Andr'e Takeshi Endo et al. [1], combines Model-Driven Testing with structural testing for web services. To guarantee that the programme fulfils the quality requirements listed in the software requirement papers, structural testing is used [10] [11] [12] [14]. With this method, testing can end when good findings are obtained. To gain a comprehensive understanding of the software's behaviour, particularly when the application's coverage and scope become evident, an Event Sequence Graph of the application under test is constructed. The authors emphasized both data flow and control flow analysis. Control flow [13] is employed to assess the coverage of all nodes and edges, while data flow [7] addresses data usage and potential usage. There is room for improvement in the system's ability to detect faults.

Using a Finite State Machine (FSM) is an alternate approach for determining conditions [2]. In this approach, each step includes condition checks, and the upcoming route to be walked is decided based on these evaluations. Rules are formulated to enhance the utility's performance, and these rules are enforced during traversal to ensure that the system examines all potential conditions in the form of Boolean values [6][9]. However, a limitation of this approach lies in its reliance on Boolean values, which provide only two condition options. This limitation may be problematic when dealing with data scenarios that involve more than two conditions.

The Finite State Machine (FSM) functions by taking into account multiple conditions at each node and determining the subsequent course of action based on desired parameter outcomes.

The FSM process can be divided into three segments:

- E-block: Evaluates triggers for all states.
- FSM-block: Calculates the next state following the current state and a signal that governs the A-block.
- A-block: To execute the necessary data operations and manage data movement.[5]

There is potential for enhancement since the system is focused on web services and exclusively supports utilities developed in Java[3].

Metamodel Transformation [4] has introduced five transformations that are detailed about five distinct metamodels. Metamodels provide insights into functionality, with the initial two metamodels outlining functional requirements, the third specifying test scenarios for evaluation, the fourth dealing with associated values, and the fifth metamodel amalgamating all inputs into organized test case formats [7][8].

After a thorough examination of these methodologies, addressing the challenge of creating a system capable of comprehensively covering all paths and rigorously evaluating all conditions at each node may be a formidable task.

3. Mathematical Modelling

Input: Finite State Machine (FSM), XML representation of an activity diagram

Process: FSM⊕ ACTIVITY

Output: Generate test cases encompassing all pathways, prioritize them, and eliminate any redundancies.

Data Structure: Table No. 1

Serial Number	Variable	Meaning
1.	F	Functionalities
2.	S	States
3.	C	Conditions
4.	E	Edges
5.	N	Nodes

6.	T	test cases
7.	S	serial number
8.	Se	Sender
9.	Re	Receiver

```
SN- 0
   For (i=0; i<=#F;i++)
{
        For (j=0; j<=#C;j++)
        {
                For (k=0; k<=#E;k<=#N;k++)
                {
                        T – (SN, condition, Cs, Se, Re) SN++; T++;
                                                        Display T;
                }

        }
}
```

4 . System Architecture

4.1 Modules

1.Input Activity Diagram:

Activity diagrams are accepted as input for the proposed task. Every activity diagram is theoretically capable of producing its own ADT, which entails having all the necessary information to modify the model and examine the capabilities and functions of every activity diagram. Technically, the ADT can then produce the ADG. For all necessary and practical test techniques, ADG is accessed through DFS. To have the best test cases, the whole major points are added to every checked path using the ADT. Each activity diagram needs to be run through each of the four modules to create the best possible collection of really affordable test cases.

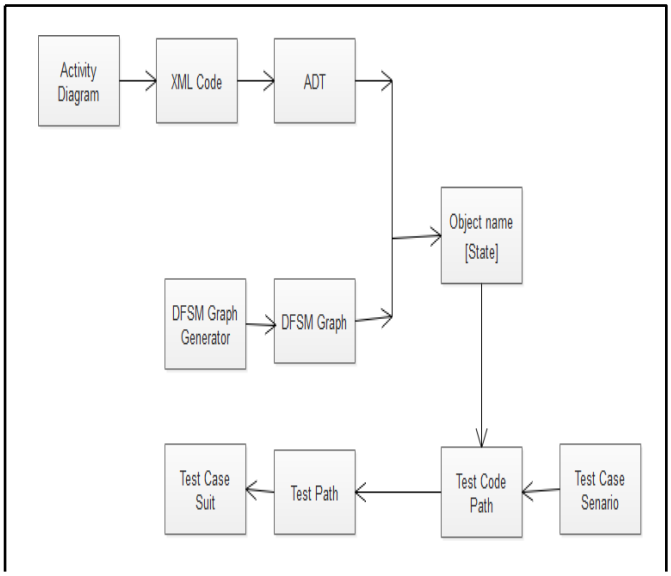

Fig 1: System Architecture of MDT using ADT &FSM

2. Activity Dependency Table Generation

Every activity diagram is essentially employed for the creation of an ADT (Activity Dependency Table), iterations, synchronization, and procedures that depict the activities of tasks. This scheme delineates the tasks earmarked for transfer to various entities that can aid in system, integration, and regression testing. Furthermore, it encompasses the input data alongside

the expected output values for each system operation. Activity Dependency Table makes every interaction between activities very evident. Every activity has a unique symbol that makes it simple to refer to it within relevant dependencies and to use it within various concerned system components. Permanent activity is dispersed completely inside one picture, reducing the searches of the produced ADG (which will be explained more in this section)., Permanent activities are divided solely within one image as opposed to having numerous symbols for a single activity.

3. DFSM Graph Generator

This module elaborates on the specifics of the controlling criteria that are used to regulate the generation of test cases. The DFSM generator addresses the situation where only a singular output is generated.

4. Test Suit Generation

• Round Trip - The path's total round journeys are covered and reported.

• Sequence - The total number of input sequences that this sequence covers.

All acts in the path must be visited a minimum of once.

• Event - Every event in the test cases' path must be visited.

• State - Every state needs to be considered at least once throughout the life cycle.

• Transition - It refers to all of the transitions that are included in the coverage.

5. Implementation and Result

The experiment was conducted using the PIN change functionality of an ATM as an example. The investigation utilized a distinct methodology, where the route was independently computed using the FSM and Activity diagram (Figure 3: Individual Approach). In the unified approach (Figure 5: Combined Approach), the results from both approaches were merged, and duplications were removed before test case generation.

Subsequently, the generated test cases underwent a thorough check to identify and remove any duplicates. The ultimate result of employing the combined approach in Model-Based Testing (MBT) was the production of a larger set of test cases, as evidenced by the total count of 83 test cases. Simultaneously, unnecessary, redundant test cases were filtered out.

The registration system was selected and presented in a manner designed to facilitate reader comprehension. It's a straightforward system where users input their login credentials. Upon verification, if the user is registered, they can successfully log in and proceed with the authentication process. Access is granted only when the specified criteria are met on both the client and server sides.

User Login System:

Table No. 2 Test Cases

Activity Diagram Inputs	Test Cases
Insert Card Validation	● Wrongly Insert Card ● Card Not Valid ● Valid Card
Language Selection Process	● Language Selected ● Cancel
ATM Pin Validation	● Valid PIN ● Invalid PIN ● Cancel
Selection Menu	● Withdraw ● Transfer ● Balance Enquiry
Balance Detail	● Balance Enquiry ● Print Mini Statement ● Cancel
Transfer	● Possible ● Cancel

Withdraw	• Collect Cash • Not Sufficient Amount in ATM. • Not Sufficient Amount in Account. • Exceed Limit.
Transaction Slip	• Possible • Cancel

Control Flow Standards:

- Every node within the registration process must be accessible at least once.
- It is required that every edge within the registration system is traversed at least once.

Data Flow Standards:

- Each utilization of data should be attainable.
- Every potential data usage should be reached.

Based on the decision tree charted, a successful test case is determined by the adherence to predefined parameters. For instance, to utilize the login system, a successful login attempt must satisfy Rules 1, 2, 3, and 4, with Rule 3 being optional. The validity of the results is confirmed through the application of positive and negative values.

In the context of the registration system's login model, achieving a user's visitation of 70% of all nodes is deemed acceptable.

Table No.3 Verification of the results login module of the registration system

	All Nodes	All Edges	All Uses	All pot uses
Positive	70	80	78	90
Negative	30	20	22	10

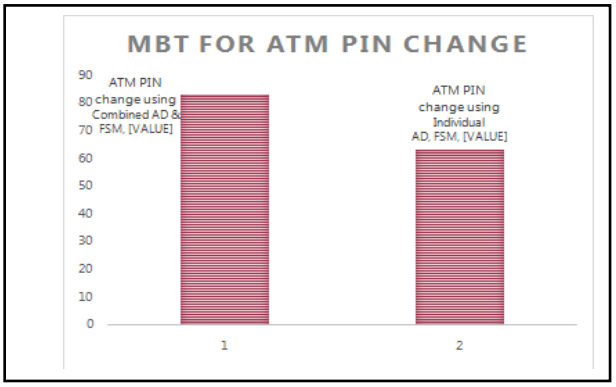

Fig 2: MDT for ATM PIN change

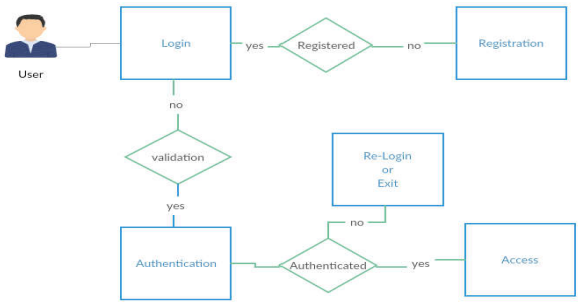

Fig 3: User Login

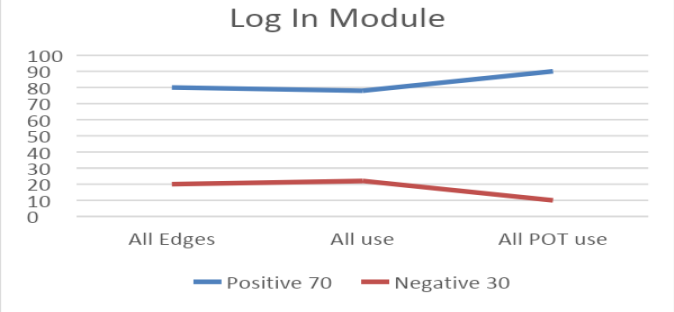

Fig 4: Verification of results login module of registration system

Combination of the Extended Finite State Machine (EFSM) and UML Sequence Diagram:

• Trigger conditions and the essential information are included in the Extended Finite Machine diagram.

Three parts may be identified in the EFSM process:

• E-block: To evaluate all condition triggers.

• FSM-block: Determine the subsequent state using the current state and an A-block control signal.

• A-block: To carry out the necessary data moves and activities.

In addition to the state diagram, a sequence diagram is appended because when the focus is on functionality, dependencies need to be checked. This aspect is crucial because coding is not explored in Model-Based Testing (MBT), and a dependency check is required to ensure the proper invocation of the embedded entity. Table No. 3 provides information on path coverage for the Extended Finite State Machine and Sequential Test Model. For each step, it includes the associated message, the number of states, and transitions.

The total for path coverage, state coverage, and transition coverage reflect the cumulative result of covered paths. As an illustration, the overall number of tests executed for the state is equivalent to 24 by multiplying 2, 1, 2, 3, 2 and 1. All these parameters offer insights into the total number of tests within their respective domains.

Table No.4 Path coverage EFSM_SeTM

Service Name	Message	#State	#Transition
Authentication	Verify ()	2	2
Validation	Validate ()	1	3
Registration	Signup ()	2	2
All Path	P1*p2*…*Pn	3	1
All State	S1*S2*…*Sn	2	1
All Transitions	T1*T2*…*Tn	1	2

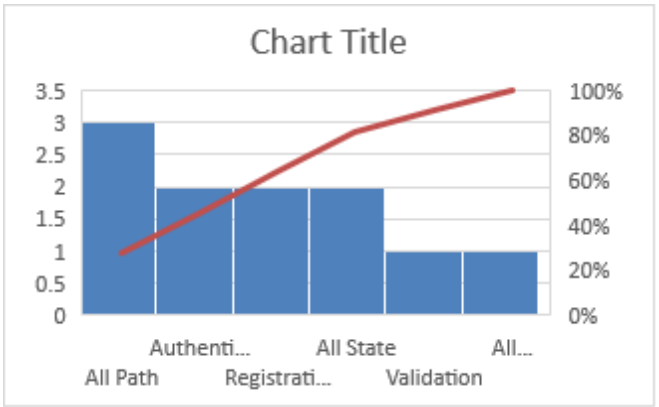

Fig 5: Path coverage by EFSM_SeTM

Opportunities for enhancement in EFSM_SeTM lie in the system's compatibility, which is currently limited to web services specifically crafted using Java.

5. Conclusion

The system is capable of producing a suitable result, as evidenced by the fact that the elimination of redundancy drastically decreased the number of steps required to verify all requirements and cover all paths from 83 to 63. To verify every last detail of the system's functioning, the number of test cases must be increased. Redundancy in the test cases must also be eliminated to avoid testing superfluous functionality. Since each of these systems depends heavily on functionality, the system may suffer if the functionality input is improperly collected. It is crucial to use a model-driven testing methodology that includes complete testing coverage, consideration of events, and inter-module communication. It is also required to correctly design a template or system to run test cases. Then, utilising metrics, these test cases should be thoroughly examined.

Future Scope:

Automation and Tool Development: As technology continues to evolve, there's potential for more sophisticated tools and frameworks to be developed that automate the process of generating functional test cases from FSMs and UML Activity Diagrams. These tools could utilize advanced AI techniques to identify test scenarios, boundary conditions, and edge cases, making the testing process more efficient and thorough.

7. References

[1] A. Bandyopadhyay and S. Ghosh, "Test Input Generation Using UML Sequence and State Machines Models," 2009 International Conference on Software Testing Verification and Validation, Denver, CO, USA, 2009, pp. 121-130, doi: 10.1109/ICST.2009.23.

[2] Vikas Panthi, Durga Prasad Mohapatra, “Automatic Test Case Generation using Sequence Diagram”, International Journal of Applied Information Systems (IJAIS) – ISSN: 2249-0868 Foundation of Computer Science FCS, New York, USA Volume 2– No.4, May 2012 – www.ijais.org

[3] Md Azaharuddin Ali et al. “Test Case Generation using UML State Diagram and OCL Expression”, International Journal of Computer Applications (0975 – 8887) Volume 95– No. 12, June 2014

[4] S. ShanmugaPriya et.al, “ Test Path Generation Using UML Sequence Diagram”, Volume 3, Issue 4, April 2013 ISSN: 2277 128X International Journal of Advanced Research in Computer Science and Software Engineering

[5] Ching-Seh Wu, Chi-Hsin Huang," The Web Services Composition Testing Driven on Extended Finite State Machine and UML Model", 2013 Fifth International Conference on Service Science and Innovation.

[6] M. Benjamin, D. Geist, A. Hartman, Y. Wolfsthal, G. Mas and R. Smeets, "A study in coverage-driven test generation", In Proc. of the 36th Conference on Design Automation Conference, pp. 970-975, 1999.

[7] M. Born, I. Schieferdecker, H.-G. Gross, and P. Santos. “Model-Driven Development and Testing – A Case Study”. In Proc. of the 1st European Workshop on Model Driven Architecture with Emphasis on Industrial Application, pp. 97-104, 2004

[8] C. Crichton, A. Cavarra, and J. Davies, “Using UML for Automatic Test Generation”, In Proc. of the Automation of Software Testing, 2007.

[9] S. R. Ganov, C. Killmar, S. Khurshid, and D. E. Perry. “Test Generation for Graphical User Interfaces Driven on Symbolic Execution”. In Proc. Proc. of the 3rd International Workshop on Automation of Software Test, pp. 33-40, 2008.

[10] H. Garavel, F. Lang, R. Mateescu, and W. Serwe, "CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes", In Proc. of the 19th International Conference on Computer Aided Verification, pp. 158-163, 2007.

[11] J. R. Calame, “Specification-Driven Test Generation with TGV”, Technical Report SEN-R0508, Centrum voor WiskundeenInformatica, 2005.

[12] Benjamin, M., D. Geist, A. Hartman, Y. Wolfsthal, G. Mas and R. Smeets, "A study in coverage-driven test generation", In Proc. of the 36th Conference on Design Automation Conference, pp. 970-975, 1999.

[13] Born, M., I. Schieferdecker, H.-G. Gross, and P. Santos. “Model-Driven Development and Testing – A Case Study”. In Proc. of the 1st European Workshop on Model Driven Architecture with Emphasis on Industrial Application, pp. 97-104, 2004

[14] Bouquet, F., C. Grandpierre, B. Legeard, and F. Peureux, ”A Test Generation Solution to Automate Software Testing”, In Proc. of the 3rd international workshop on Automation of software test, pp. 45-48, 2008. 51828

[15] Bouquet, F., C. Grandpierre, B. Legeard, F. Peureux, N. Vacelet, and M. Utting, “A subset of precise UML for Model-based Testing”, In Proc. of the 3rd International Workshop Advances in Model Based Testing (AMOST), pp. 95-104,

2007. Calame, J. R. 2005.
[16] The Web Services Composition Testing Based on Extended Finite State Machine and UML Model", 2013 Fifth International Conference on Service Science and Innovation Crichton, C., A. Cavarra, and J. Davies,
[17] "Using UML for Automatic Test Generation", In Proc. of the Automation of Software Testing, 2007. Farooq, Q., M. Z. Z. Iqbal, Z. I. Malik and A. Nadeem,
[18] Garavel, H., F. Lang, R. Mateescu, and W. Serwe, "CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes", In Proc. of the 19th International Conference on Computer Aided Verification, pp. 158-163, 2007.
[19] MdAzaharuddin Ali et al. "Test Case Generation using UML State Diagram and OCL Expression", International Journal of Computer Applications (0975 – 8887) Volume 95– No. 12, June 2014
[20] ShanmugaPriya S. et al, " Test Path Generation Using UML Sequence Diagram", Volume 3, Issue 4, April 2013 ISSN: 2277 128X International Journal of Advanced Research in Computer Science and Software Engineering
[21] VikasPanthi, Durga Prasad Mohapatra, 2012. "Automatic Test Case Generation using Sequence Diagram", International Journal of Applied Information Systems (IJAIS) – ISSN: 2249-0868 Foundation of Computer Science FCS, New York, USA Volume 2– No.4, www.ijais.org.
[22] P. Mohagheghi, W. Gilani, A. Stefanescu, M. Fernandez, An empirical study of the state of the practice and acceptance of model-based engineering in four industrial cases, Empirical Software Engineering (2012) 1–28.
[23] Asaithambi SPR, Jarzabek S. Pragmatic Approach to Test Case Reuse-A Case Study in Android OS BiDiTests Library. Software Reuse for Dynamic Systems in the Cloud and Beyond. Springer; 2014. p.122–38.
[24] Ke Z, Bo J, Chan WK. Prioritizing Test Cases for Regression Testing of Location-Based Services: Metrics, Techniques, and Case Study. IEEE Transactions on Services Computing.. 2014; 7(1):54–67.
[25] Papadakis M, Malevris N. Mutation-based test case generation via a path selection strategy. Information and Software Technology. 2012; 54(9):915–32.
[26] Zhang C, Groce A, Alipour MA, editors. Using test case reduction and prioritization to improve symbolic execution. Proceedings of the 2014 International Symposium on Software Testing and Analysis. ACM; 2014.
[27] Mondal SK, Tahbildar H. Regression Test Cases Minimization for Object Oriented Programming using New Optimal Page Replacement Algorithm. International Journal of Software Engineering and Its Applications. 2014; 8(6):253–64
[28] Zhang W, Zhao D. Reuse-Oriented Test Case Management Framework. International Conference on Computer Sciences and Applications (CSA).IEEE; 2013.
[29] Asaithambi S, Jarzabek S. Towards Test Case Reuse: A Study of Redundancies in Android Platform Test Libraries, Berlin Heidelberg. Springer; 2013. p. 49–64.
[30] Fowler M. Refactoring: Improving the design of existing code. India: Pearson Education; 1999.
[31] Lashari SA, Ibrahim R, Senan N. Fuzzy Soft Set based Classification for Mammogram Images. International Journal of Computer Information Systems and Industrial Management Applications. 2015; 7:66–73.
[32] Ahmed M, Ibrahim R, Ibrahim N. An Adaptation Model for Android Application Testing with Refactoring. Growth. 2015; 9(10):65–74.
[33] Fowler M. Refactoring: Improving the Design of Existing Code. 1997. Available from: http://www.martinfowler.com/ books/refactoring.html.
[34] Al Dallal J. Identifying refactoring opportunities in object-oriented code: A systematic literature review. Information and Software Technology. 2015; 58:231–49.
[35] Jena SKSAK, Mohapatra DP. A Novel Approach for Test Case Generation from UML Activity Diagram. 2014.
[36] Ibrahim R, Saringat MZ, Ibrahim N, Ismail N. An Automatic Tool for Generating Test Cases from the System's Requirements. 2007;861–6.
[37] Nguyen CD, Marchetto A, Tonella P, editors. Combining model-based and combinatorial testing for effective test case generation. Proceedings of the 2012 International Symposium on Software Testing and Analysis, ACM; 2012.
[38] Swain R, Panthi V, Behera PK, Mohapatra DP. Automatic test case generation from UML state chart diagram. International Journal of Computer Applications. 2012; 42(7):26–36.
[39] Khan SUR, Lee SP, Ahmad RW, Akhunzada A, Chang V. A Survey on Test Suite Reduction Frameworks and Tools. International Journal of

Information Management. 2016; 36(6): Part A, 963–75.

[40] Shrivastava, A., Chakkaravarthy, M., Shah, M.A..A Novel Approach Using Learning Algorithm for Parkinson's Disease Detection with Handwritten Sketches. In Cybernetics and Systems, 2022

[41] Shrivastava, A., Chakkaravarthy, M., Shah, M.A., A new machine learning method for predicting systolic and diastolic blood pressure using clinical characteristics. In *Healthcare Analytics*, 2023, 4, 100219

[42] Shrivastava, A., Chakkaravarthy, M., Shah, M.A.,Health Monitoring based Cognitive IoT using Fast Machine Learning Technique. In *International Journal of Intelligent Systems and Applications in Engineering*, 2023, 11(6s), pp. 720–729

[43] Shrivastava, A., Rajput, N., Rajesh, P., Swarnalatha, S.R., IoT-Based Label Distribution Learning Mechanism for Autism Spectrum Disorder for Healthcare Application. In *Practical Artificial Intelligence for Internet of Medical Things: Emerging Trends, Issues, and Challenges*, 2023, pp. 305–321

[44] Boina, R., Ganage, D., Chincholkar, Y.D., .Chinthamu, N., Shrivastava, A., Enhancing Intelligence Diagnostic Accuracy Based on Machine Learning Disease Classification. In *International Journal of Intelligent Systems and Applications in Engineering*, 2023, 11(6s), pp. 765–774

[45] Shrivastava, A., Pundir, S., Sharma, A., ...Kumar, R., Khan, A.K. Control of A Virtual System with Hand Gestures. In *Proceedings - 2023 3rd International Conference on Pervasive Computing and Social Networking, ICPCSN 2023,* 2023, pp. 1716–1721

[46] Rieger M, Van Rompaey B, Du Bois B, Meijfroidt K, Olivier P. Refactoring for performance: An experience report. Proc. Software Evolution. 2007; 2(9):1–9.

[47] Yoshioka N, Washizaki H, Maruyama K. A survey on security patterns. Progress in informatics. 2008; 5(5):35–47.

[48] Garrido A, Rossi G, Distante D. Refactoring for usability in web applications. IEEE Software. 2011; 28(3):60.

[49] Al Dallal J. Constructing models for predicting extract subclass refactoring opportunities using object-oriented quality metrics. Information and Software Technology. 2012; 54(10):1125–41.

[50] Bavota G, De Lucia A, Marcus A, Oliveto R. Automating extract class refactoring: an improved method and its evaluation. Empirical Software Engineering. 2014; 19(6):1617–64.

[51] Palomba F, Bavota G, Oliveto R, De Lucia A. Anti-Pattern Detection: Methods, Challenges, and Open Issues. Advances in Computers. 2014; 95:201–38.

[52] Silva D, Terra R, Valente MT. JExtract: An Eclipse Plug-in for Recommending Automated Extract Method Refactorings. Federal University of Minas Gerais: Brazil. 2014, pp.1-8.

[53] Fokaefs M, Tsantalis N, Stroulia E, Chatzigeorgiou A. JDeodorant: Identification and application of extract class refactorings. Proceedings of the 33rd International Conference on Software Engineering, Waikiki, Honolulu, HI, USA.ACM; 2011. p. 1037–9.

[54] Yoo S, Harman M. Pareto efficient multi-objective test case selection. Proceedings of the 2007 International Symposium on Software Testing and Analysis, London, United Kingdom. ACM; 2007. P. 140–50.

[55] S. Mohite, R. Phalnikar, M. Joshi, S. D. Joshi, and S. Jadhav, "Requirement and interaction analysis using aspect-oriented modelling," 2014 IEEE International Advance Computing Conference (IACC), 2014, pp. 1448-1453, doi: 10.1109/IAdCC.2014.6779539.

[56] S. Jadhav, S. B. Vanjale and P. B. Mane, "Illegal Access Point detection using clock skews method in wireless LAN," 2014 International Conference on Computing for Sustainable Global Development (INDIACom), 2014, pp. 724-729, doi: 10.1109/IndiaCom.2014.6828057.

[57] S. Mohite, A. Sarda and S. D. Joshi, "Analysis of System Requirements by Aspects-J Methodology," 2021 International Conference on Computing, Communication and Green Engineering (CCGE), 2021, pp. 1-6, doi: 10.1109/CCGE50943.2021.9776384.

[58] Mohite, S., Phalnikar, R., Joshi, S.D.," Requirement and interaction analysis using aspect-oriented modelling", New Trends in Networking, Computing, E-learning, Systems Sciences, and Engineering. Lecture Notes in Electrical Engineering, Vol 312. Springer, 2015, Cham. https://doi.org/10.1007/978-3-319-06764-3_54

[59] P. A. Jadhav, C. Vinotha, S. K. Gupta, B. Dhyani, V. H. Patil, and R. Kumar, "Asset Class Market Investment Portfolio Analysis and Tracking," *2022 5th International Conference on Contemporary Computing and Informatics (IC3I)*, Uttar Pradesh,

India, 2022, pp. 973-981, doi: 10.1109/IC3I56241.2022.10072525.

[60] Patil, V., Kadam, P., Jadhav, P., & Kadam, A. (2022). Intelligent Agricultural System Based on IoT and Machine Learning. *Available at SSRN 4203128*.

[61] Suryawanshi, P. K., Kadam, A. K., Dhotre, P. S. S., & Jadhav, P. A. (2021). A Novel Approach for Women Security with Information Fusion for Multi-Sensory Data. *Turkish Online Journal of Qualitative Inquiry*, *12*(8).

[62] Dr Vinod H Patil, Dr Anurag Shrivastava, Devvret Verma, Dr A L N Rao, Prateek Chaturvedi, Shaik Vaseem Akram, "Smart Agricultural System Based on Machine Learning and IoT Algorithm", 2nd International Conference on Technological Advancements in Computational Sciences (ICTACS), 2022. DOI: DOI: 10.1109/ICTACS56270.2022.9988530.

[63] Dr Vinod H Patil, Dr Pramod A. Jadhav, Dr C. Vinotha, Dr Sushil Kumar Gupta, Bijesh Dhyani, Rohit Kumar," Asset Class Market Investment Portfolio Analysis and Tracking", 5th International Conference on Contemporary Computing and Informatics (IC3I), December 2022. DOI: 10.1109/IC3I56241.2022.10072525.

[64] Dr. Vinod H Patil, Prasad Kadam, Sudhir Bussa, Dr. Narendra Singh Bohra, Dr. ALN Rao, Professor, Kamepalli Dharani," Wireless Communication in Smart Grid using LoRa Technology", 5th International Conference on Contemporary Computing and Informatics (IC3I), December 2022, DOI: 10.1109/IC3I56241.2022.10073338

[65] Vinod H. Patil, Dr. Shruti Oza, Vishal Sharma, Asritha Siripurapu, Tejaswini Patil, "A Testbed Design of Spectrum Management in Cognitive Radio Network using NI USRP and LabVIEW", International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-8 Issue-2S8, August 2019.

[66] Vinod H. Patil, Shruti Oza, "Green Communication for Power Distribution Smart Grid", International Journal of Recent Technology and Engineering™ (IJRTE), ISSN:2277-3878(Online), Reg. No.: C/819981, Volume-8, Issue-1, Page No. 1035-1039, May-19.

[67] Patil, V.H., Oza, S., Sharma, V., Siripurapu, A., Patil, T.," A testbed design of spectrum management in cognitive radio network using NI USRP and LabVIEW", International Journal of Innovative Technology and Exploring Engineering, 2019, 8(9 Special Issue 2), pp. 257–262.

[68] S. Bussa, A. Bodhankar, V. H. Patil, H. . Pal, S. K. . Bunkar, and A. R. . Khan Qureshi, "An Implementation of Machine Learning Algorithm for Fake News Detection", International Journal on Recent and Innovation Trends in Computing and Communication, ISSN: 2321-8169, Volume: 11 Issue: 9s, pp. 392–401, Aug. 2023. DOI: https://doi.org/10.17762/ijritcc.v11i9s.7435

[69] Kadam, A. K., Krishna, K. H., Varshney, N., Deepak, A., Pokhariya, H. S., Hegde, S. K., & Patil, V. H., "Design of Software Reliability Growth Model for Improving Accuracy in the Software Development Life Cycle (SDLC)", International Journal of Intelligent Systems and Applications in Engineering, vol. 12, Issue No. 1s, pp. 38–50, Sep. 2023.
https://ijisae.org/index.php/IJISAE/article/view/3393