Domain-Specific Software System Testing by Using Intelligent Approaches

Pooja Chauhan, Vijayakumar Balakrishnan

Abstract— As Software industries have grown over the last few years, software testing has been considered a crucial process in the software development lifecycle. In the past, software testing was done manually, and this is a very laborious and costly process that carries errors and requires time, money, and effort. Currently, testers want to perform the testing automatically to save time and get accurate results. In this paper, we propose the development of a domain-specific tool Architecture to automate test case generation across various industries like Automotive, Finance, and healthcare. This will retrieve user requirements and apply Intelligent Techniques to generate test cases. By enhancing the intelligent models' concrete tests with fewer ambiguities can be generated through this tool.

Index Terms— **Software Testing, Test Case Generation, Artificial Intelligence (AI), Natural Language Processing (NLP)**

I. INTRODUCTION

Software testing continues to be a major process in the software development lifecycle to ensure the quality of software and validate if it meets user requirements or not [1]. The need for software testing increases with the advancement of technology and the complexity of the software [2]. Software testing helps to create a system more efficient and reliable which in turn builds confidence in the Developers, management team, and end-users.

Testing identifies systems or program bugs and flaws before it is used by the end user. It aims to find software faults or defects and to ensure that it meets the user's needs [3]. In the traditional approach of software testing testers design, write, and execute test cases based on user requirements. This method has been facing challenges as Technology is scaling. Manual testing is costly, time-consuming, and prone to human error which can compromise the efficiency of the testing process [21].

To overcome these challenges, the use of Artificial Intelligence (AI) and Natural Language Processing (NLP) techniques in software testing has revolutionized the industry by automating the testing process. By using these automated process software testing can be made more accurate, reliable, and less time-consuming. Testing can be automated by using these smart techniques by analyzing the natural language requirement, use cases, and other software artifacts to generate the test cases automatically.AI enhances software quality by improving the quality of testing procedures, minimizing manual work, generating test case and detecting bugs. AI together with software testing helps to speed up the testing process which helps to save lots of time, and cost and handle complex requirements of software engineering [4].

This direction of using AI and NLP with the software testing process has speeded up the process by finding the faults and inconsistencies in the program. In recent work Natural language requirements through text documents, and use cases are taken as input and processed by AI and NLP approaches. This integration used many approaches to address the complex nature of software testing [5]. Some approaches use domain details to structure the testing process. This structured process enhances the efficiency and accuracy of the testing Phase.

With the integration of AI and NLP with the software testing process comes some open issues and challenges that need some enhancement. Firstly, Due to the ambiguity of natural language struggles while interpreting and analyzing the textual artifacts such as requirement documents, user stories, and software specifications. Additionally, as industries are growing day by day software is becoming more complex and most of the companies have different domains which adds more complexity and requires AI and NLP models to optimize and specialize to work on different linguistic patterns and terms. The existing approaches often struggle to cope with different testing cases and the complex nature of software [6].

There are some specific ways that make Intelligent software testing differ from Manual Testing [7].

- Automation: By using AI and NLP one can generate test cases automatically by using document which reduces the need for manual effort [17].
- Coverage: Automated testing techniques ensure to cover all the areas of the software and increase the likelihood of finding the defects.
- Prioritization: By using AI, Testing can prioritize the execution of test cases based on their likelihood of finding buys and allowing testers to focus on important tests [18].

Therefore, Automated software testing can improve quality, efficiency, and reliability by identifying defects that might have been missed while doing manual testing.

2024 International Conference on Computational Intelligence and Network Systems (CINS) | 979-8-3315-0410-6/24/$31.00 ©2024 IEEE | DOI: 10.1109/CINS63881.2024.10864412

In this paper, we proposed an architecture of a tool that can be designed to automate software testing by using intelligent models and AI, and NLP approaches [19]. This architecture explains how tests will be generated automatically for domain-specific software systems by fetching the information from the documents. Our approach aims to connect the tool with domain-specific software and by using AI and NLP natural language natural language requirements, use cases, and other textual information are analyzed. By doing so, the tool will generate accurate test cases to the specific needs of the industries, while improving overall software quality and reducing manual work.

II. OBJECTIVES

The current work proposes an architecture tool that will automate software testing. Leveraging AI and NLP approaches to generate test cases using textual information from domain-specific software makes software testing more efficient.

The specific objectives are listed below:

- Identify the type of issues faced during the manual testing.
- Propose a software architectural design for an application domain.
- Discuss examples and case studies to automate the testing process using intelligent approaches in various industries.

III. MOTIVATION

As modern software is becoming more complex day by day and various challenges are faced with manual testing like time consuming, prone to human error, and very difficult to cover all the scenarios. To overcome these challenges automation of testing is in demand by using an Intelligent model having AI and NLP approaches. Still, there are challenges with these approaches as well. Below are a few :

- Natural language Ambiguity
- Struggle with Domain Specific Software
- The scalability of software makes it difficult for intelligent models to cope.

It is very much essential to design a suitable architecture to address all these challenges

IV. RELATED WORK

This section gives an overview of existing literature related to the architectural design for information retrieval of multimedia data, scalability and adaptability approaches to explore multimedia objects. It also includes a brief outline of search and retrieval operations that are involved in rich media objects.

Automated test case Generation has been a significant enhancement with Natural language Processing (NLP) techniques.This approach [8], focuses on software testing through an automotive knowledge graph by using NLP techniques with the application of automatic test case generation. The methods included in this study are Auto ontology (an ontology that has been developed by analyzing different industries' domain software systems, AutoRe (a relation extraction Re model to fetch triplets from various sentence types found in domains), Autovec, an algorithm for triplet matching and context-based search. The AutoKG pipeline components are Autovec, conference resolution, sentence classification, Name Entity Recognition and chunking Intent Extraction, and auto ontology. Once the Graphs have been extracted further queries are executed to generate the complete test case generation. There are a total of five phases in this work to generate test cases. The application of AutoKG shows good potential to manage the complex requirements of automotive software systems.

Another Approach for automated testing used in [9], is to generate test cases through use case description by using NLP and control flow involves several processes. The framework proposed in this work involves methods first it takes input as text and then it generates the control flow graph, NLP Table, and test path. The proposed algorithm creates a parent node which represents the condition statement and the children node represents the true false statement this algorithm creates an NLP table. Once the table is created it will generate the test paths which will result in generating the test cases. This technique has shown effective results in handling complex linguistic while maintaining accuracy in test generation [16].

Automating test case generation has been focusing on both Model-driven software engineering (MDSE) [10], and NLP to generate the Abstract test cases to test software. MDSE approaches, such as those by Gutiérrez et al. (2015) and Hue et al. (2019), used the model to create test cases by using use cases and Data model, through NLP tools like Stanford CoreNLP extraction of test cases is done through natural language description. Combining MDSE and NLP approach work emphasizes generating Abstract test cases and makes the testing process more structured. This approach reduces the manual effort involved in test case generation and shows its effectiveness through case studies on real-world software systems.

Similarly, in the context of Acceptance testing, which is used to confirm whether it’s fulfilling the end user requirement or not. However, it’s a laborious task to determine the right test case that covers the requirements. In this approach, a tool has been developed CIRA (Conditional in requirement Artifacts) [11], by using NLP which will create a set of required test case from conditional statements in informal requirements. CIRA involves three processes Detection of conditions, Extraction of conditions, and creation of cause-effect Graph. This tool has been demonstrated with three industry partners and compared the manual test cases to check its feasibility and it’s found that out of 578 manually created test cases, 71.8 % can be generated automatically. In addition, it was found that 80 test cases were missed in manual test case design. This tool highlights the ability of NLP to improve test coverage and system reliability.

Further extending the capabilities of NLP-based software testing , [12] proposed the use of T5 and GPT models for generating the test cases. The proposed approach eliminates the need for manual test case generation and allows for the testing for more complex systems. Once user input the Document or text model T5 will convert the Natural language Processing (NLP) problems into a text-to-text format. Once preprocessing is done GPT-3 model will check the output from T5 and it will refine further. This method is effective to cover different scenarios that may be missed during manual test case creation.

In parallel [13], a natural Language Processing-based Approach presents UMTGG, which is a method that automates the generation of test cases from Natural Language requirements. By using the Natural language processing Technique and RUCM(Restricted Use Case Modeling), UMTG extracts test case scenarios from use case documentation and generates formal constraints for test data, It reduces manual effort. This technique has been tested in industries and achieves high accuracy in generating test cases.

In addition to the advancement of NLP approaches [14] aims to enhance the test case generation for NLP programs by using a scatter search strategy [20], through search-based algorithms. Previous algorithms faced issues with generating test cases that cover all possible paths in NLP programs due to specific input variables required but in this proposed strategy it has explored input variables effectively by making sure that even paths requiring specific input are covered.

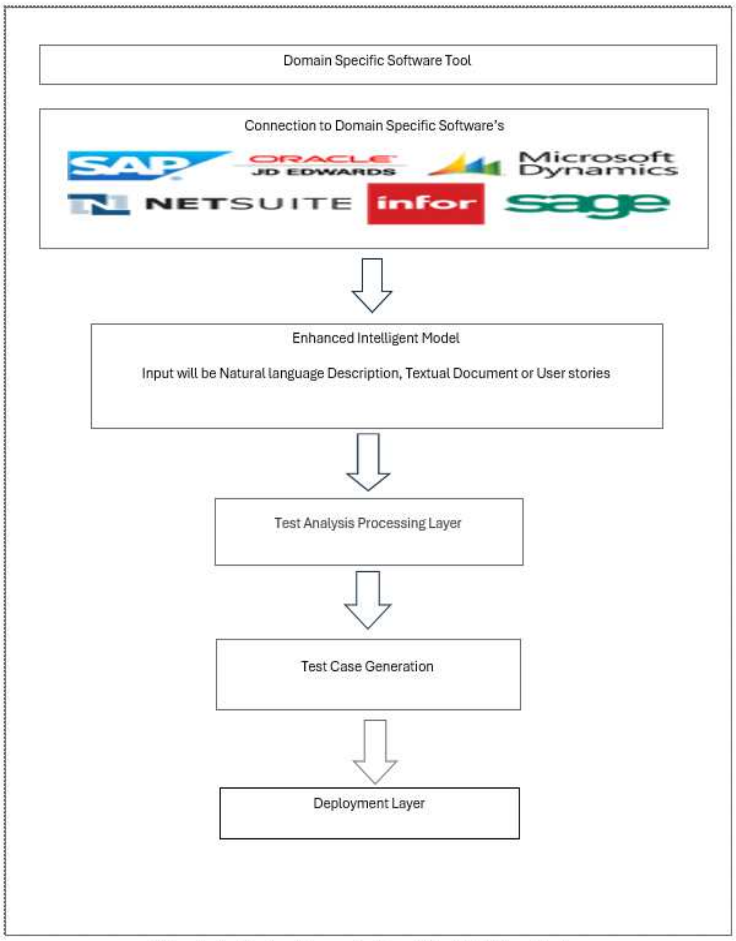
There is structured overview which explains the evolution of Artificial Intelligence (AI) in software testing, and sheds lights on its profound impact to improve efficiency and reliability in this phase. Through a detailed verification of various AI approaches, including Machine learning and Deep Learning [14], this review explains, how AI can make software testing automatic by generating test cases, and defect prediction. The review explains the increasing complexity of software systems, which makes traditional manual testing methods complex, time-consuming, and less practical, and on the other hand AI promises to provide solutions to these challenges. By exploring selected articles from different research databases, the study represents the current trends in AI-powered software testing.

While the Techniques of NLP in software testing have shown substantial promise for generating test cases. A systematic review has been done by [15] highlighting its efficiency in automating the task such as requirement analysis and test case generation. The review by them shows the benefits like reducing the manual effort and improving the test coverage but addressing the challenges faced like ambiguity, and domain-specific adaptation. The authors emphasize the need for further research to develop more fine-tuned algorithms that can be used with different domains and handle large-scale software.

V. PROPOSED ARCHITECTURAL DESIGN FOR DOMAIN-SPECIFIC SOFTWARE SYSTEM TESTING TOOL

The architectural Diagram for the proposed system is shown in Figure 1 to automate the test case generation. Architecture shows step by step process starts with connection to the software then processing the inputs to the enhanced intelligent model and deployment of test cases Below are the Five phases of this structure:

- **Domain-Specific Software tool:** This layer shows the overall tool that communicates with different Software. This layer plays the central platform for the automation of testing.
- **Domain-Specific Software Connection Phase**: This Phase represents the connection between the tool and different software to extract requirements document and user stories. The connection can be made through the Existing API or Data connectors.



Domain Specific Software Tool
Connection to Domain Specific Software's
SAP
ORACLE JD EDWARDS
Microsoft Dynamics
NETSUITE
infor
sage
Enhanced Intelligent Model
Input will be Natural language Description, Textual Document or User stories
Test Analysis Processing Layer
Test Case Generation
Deployment Layer

Figure 1: Architecture Diagram for Domain-Specific Software Tool

- **Enhanced Intelligent Model Phase:** This is the main layer where the tool uses the Intelligent techniques to interpret and analyze the inputs from the software. The model takes inputs from natural language documents, descriptions, or user stories. Enhanced intelligent model will process these inputs and extract the main information like requirements, and conditions.

- **Processing Phase:** Once data has been inserted into text analysis, the Knowledge extraction technique will transform raw data into a structured format.
- **Test Case Generation:** Based on the process data through intelligent model test cases will be generated.
- **Deployment Layer:** Test cases will be deployed and executed into the software development cycle.

The proposed Architecture will Overcome the below challenges:

- **Automation**: This Architecture is designed to automate the whole test case generation [3], process by reducing the manual effort, cost, and time.
- **Natural Language Ambiguities**: Existing Intelligent Models will be enhanced to process the inputs which are mainly in textual documents or user stories. The Enhanced Model will use Natural language processing (NLP) Techniques and AI models (e.g. GPT, T5) to analyze the natural language data. It will reduce the Natural language Ambiguity [15], by extracting the precise data from textual documents or User stories.
- **Handling Frequent changes in Requirement Document:** During the development phase requirement changes frequently and after deployment, as well updates are done frequently [15], so testing for newly input data is very hard. The proposed Architecture will reprocess the new input data and will adjust the test case according to new updates and new requirements.

The proposed Architecture provides solutions to key challenges faced in software testing. By using Enhanced Intelligent approaches this will automate the test case generation. Moving forward this approach can be further extended to support a wide range of industries and more complex requirements.

VI. EXAMPLES AND CASE STUDIES

Several studies and developments have been done to automate the testing process using intelligent approaches in various industries.

For an automotive company that develops embedded systems like Anti-Lock Braking Systems (ABS) for vehicle control. To test the software for embedded systems traditionally domain experts were required to create test cases manually which is very time consuming, and labor-intensive. With the advancement of Natural language Processing (NLP) and Knowledge of Graphs, the testing process can be automated. By using the AutoKG Framework explained in a study [8]. The pipeline will fetch the information from unstructured documents and will build a knowledge graph that will generate test cases based on some preconditions like “If the vehicle speed exceeds a certain frequency, the ABS should activate”.

Another example can be seen in the case of a large online retail platform that faces challenges in manual testing because of its frequent updates in the system like Products added, changes in the customer service module, and payment. As manual testing is time-consuming and prone to human error the company adapted T5 and GPT-3 models [12], to automate the testing process. T5 model processes the technical document whereas GPT-3 summarizes this information and generates the test cases. This automation reduces the time required to generate the test case and improves efficiency.

Additionally, the CIRA tool [11], has been developed to automate the creation of test cases from the requirements using conditions. This tool has been tested in some industries. Leopold Kostal is one of them they used it to test the functionality of their plug interlock system which prevents the electric vehicle plug from disconnection during charging. CIRA identifies the condition statement in the requirement document and maps it into the cause-effect graph to produce necessary test cases. CIRA automatically generated 71.8% of the total test cases.

These Developments show the potential of an automated testing process by reducing manual efforts and improving the efficiency of covering the testing for complex modules. As technology is growing these intelligent approaches play a very important role in optimizing the software testing process.

VII. CONCLUSION

The current work deals with architecture design for automated software testing that will integrate Intelligent Approaches using AI and NLP and make Traditional testing easy. In this approach Techniques will be enhanced to overcome the issues of Natural language ambiguities, and complexity of the software system and to fine-tune the existing automated approaches.

REFERENCES

[1] G. J. Myers, T. Badgett, T. M. Thomas, and C. Sandler, *The Art of SoftwareTesting*, vol. 2. Hoboken, NJ, USA: Wiley, 2004.

[2] Hangensen, T.M. and Kristensen, B.B., 1992. Consistency in software system development: Framework, model, techniques & tools. *ACM SIGSOFT Software Engineering Notes*, *17*(5), pp.58-67.

[3] Bertolino, A., 2007, May. Software testing research: Achievements, challenges, dreams. In *Future of Software Engineering (FOSE'07)* (pp. 85-103). IEEE.

[4] Tahvili, S. and Hatvani, L., 2022. *Artificial Intelligence Methods for Optimization of the Software Testing Process: With Practical Examples and Exercises*. Academic Press.

[5] Konreddy, S.D.R., 2021. The Impact of NLP on Software Testing. *Journal of University of Shanghai for Science and Technology, ISSN*, pp.1007-6735.

[6] Ke, W., Wu, C., Fu, X., Gao, C. and Song, Y., 2020, December. Interpretable test case recommendation based on knowledge graph. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)* (pp. 489-496). IEEE.

[7] Boukhlif, M., Hanine, M. and Kharmoum, N., 2023. A decade of intelligent software testing research: A bibliometric analysis. *Electronics*, *12*(9), p.2109.

[8] Kesri, V., Nayak, A. and Ponnalagu, K., 2021, April. AutoKG-an automotive domain knowledge graph for software testing: a position paper. In *2021 ieee international conference on software testing, verification and validation workshops (icstw)* (pp. 234-238). IEEE.

[9] Lafi, M., Alrawashed, T. and Hammad, A.M., 2021, July. Automated test cases generation from requirements specification. In *2021 International Conference on Information Technology (ICIT)* (pp. 852-857). IEEE.

[10] Allala, S.C., Sotomayor, J.P., Santiago, D., King, T.M. and Clarke, P.J., 2022, December. Generating Abstract Test Cases from User Requirements using MDSE and NLP. In *2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)* (pp. 744-753). IEEE.

[11] Fischbach, J., Frattini, J., Vogelsang, A., Mendez, D., Unterkalmsteiner, M., Wehrle, A., Henao, P.R., Yousefi, P., Juricic, T., Radduenz, J. and Wiecher, C., 2023. Automatic creation of acceptance tests by extracting conditionals from requirements: NLP approach and case study. *Journal of Systems and Software*, *197*, p.111549.

[12] Mathur, A., Pradhan, S., Soni, P., Patel, D. and Regunathan, R., 2023, March. Automated test case generation using t5 and GPT-3. In *2023 9th International Conference on Advanced Computing and Communication Systems (ICACCS)* (Vol. 1, pp. 1986-1992). IEEE.

[13] Wang, C., Pastore, F., Goknil, A. and Briand, L.C., 2020. Automatic generation of acceptance test cases from use case specifications: an NLP-based approach. *IEEE Transactions on Software Engineering*, *48*(2), pp.585-616.

[14] Liu, F., Huang, H., Yang, Z., Hao, Z. and Wang, J., 2019. Search-based algorithm with scatter search strategy for automated test case generation of NLP toolkit. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *5*(3), pp.491-503.

[15] Boukhlif, M., Hanine, M., Khartoum, N., Noriega, A.R., Obeso, D.G. and Ashraf, I., 2024. Natural Language Processing-based Software Testing: A Systematic Literature Review. *IEEE Access*.

[16] Tahvili, S., Hatvani, L., Ramentol, E., Pimentel, R., Afzal, W. and Herrera, F., 2020. A novel methodology to classify test cases using natural language processing and imbalanced learning. *Engineering applications of artificial intelligence*, *95*, p.103878.

[17] Sofian, H., Yunus, N.A.M. and Ahmad, R., 2022. Systematic mapping: Artificial intelligence techniques in software engineering. *IEEE Access*, *10*, pp.51021-51040.

[18] Pan, R., Bagherzadeh, M., Ghaleb, T.A. and Briand, L., 2022. Test case selection and prioritization using machine learning: a systematic literature review. *Empirical Software Engineering*, *27*(2), p.29.

[19] Riccio, V., Jahangirova, G., Stocco, A., Humbatova, N., Weiss, M. and Tonella, P., 2020. Testing machine learning based systems: a systematic mapping. *Empirical Software Engineering*, *25*, pp.5193-5254.

[20] Mukherjee, R. and Patnaik, K.S., 2021. A survey on different approaches for software test case prioritization. *Journal of King Saud University-Computer and Information Sciences*, *33*(9), pp.1041-1054.

[21] Gurcan, F., Dalveren, G.G.M., Cagiltay, N.E., Roman, D. and Soylu, A., 2022. Evolution of software testing strategies and trends: Semantic content analysis of software research corpus of the last 40 years. *IEEE Access*, *10*, pp.106093-106109.