ESP-JETA
ESP Journal of Engineering & Technology Advancements
ISSN: 2583-2646 / Volume 4 Issue 1 March 2024 / Page No: 150-157
Paper Id: JETA-V4I1P121/ Doi: 10.56472/25832646/JETA-V4I1P121

Original Article

Generative AI Approaches to Automated Unit Test Case Generation in Large-Scale Software Projects

Satyadhar Kumar Chintagunta
Independent Researcher

Received Date: 23 January 2024 Revised Date: 25 February 2024 Accepted Date: 23 March 2024

Abstract: *Automated test case generation is also supposed to complement software testing by reducing the level of manual effort required in test case generation. The SDLCs need experienced professionals who have knowledge of the domain at each level. Skill determines the quality of output and efficiency of every phase. Software testing is a process that belongs to the software development life cycle (SDLC) and has a role of testing the accuracy, reliability, and performance of the product. Traditional forms of testing may be tedious, resource-consuming, and give results only of a section of the system which is especially problematic with multifaceted and dynamic systems. Generative AI (GenAI) and artificial intelligence (AI) offer solutions that are game changers in software testing by automation of the process of test case creation, the discovery of problems, and the coverage of edge cases. GenAI models, including autoencoders, VAEs, GANs, and LSTM-based networks, allow intelligent and adaptable testing and scalable, as well as reduced human error and bias. The paper reviews AI-based testing techniques, the use of GenAI to enhance the quality of the code, and discusses the problems, limitations and opportunities of applying AI-based testing in the modern software development.*

Keywords: *Software Testing, Artificial Intelligence (AI), Generative AI (GenAI), Test Automation, Software Quality Assurance.*

II. INTRODUCTION

Software testing is critical in the design of effective software systems. Testing aids the realization of quality gaps, uncertainty, and overall dependability but without direct improvement of software quality [1]. Overall, testing may be manual or automated and may be applied to open-source and commercial software. The two most common ways that software testing is categorized are static testing and dynamic testing. Static testing makes use of methods like symbolic execution to examine the code and documentation without actually running the program [2]. Dynamic testing is one of the techniques used in testing in modern development, and it consists of executing programs using input data and observing their behavior during execution.

Dynamic testing involves unit testing. It tests the simplest, testable independent unit of software which often is a single class or function. Unit tests are often written in an executable programming language such as Python and a test framework such as pytest which enables a developer to execute them numerous times as the system evolves [3][4]. This practice supports the use of TDD with tests being written prior to the implementation, such that low-level functionality should be fast, reliable and verifiable at all times. However, as software project designs become more complex, unit tests are resource intensive and prone to errors to be designed and maintained manually [5]. Automated testing systems are also welcome, but are highly reliant on prewritten scripts and are therefore not as adaptable towards agile-like frequent code modifications.

Artificial intelligence has also become a significant system in this instance, which supports software development over the past few years, providing the new approaches to the process automatization. The AI implementation in this area is in software testing [6]. Generative AI model is a radical solution. Software engineering, software development, software law, and machine learning are just a few of the many fields that are already benefiting from artificial intelligence thanks to HPC, deep learning, and machine learning. Software testing is made more efficient, accurate, and comprehensive with the use of AI. NLP and code creation are two areas where generative AI models like GPT, VAEs, and GANs have demonstrated exceptional skills [7]. These models can analyze source code, learn patterns, and autonomously generate unit test cases that adapt dynamically to code changes [8]. Unlike conventional automated testing, Generative AI does not rely on static test scripts but continuously creates and evolves tests, identifying edge cases and predicting potential software failures.

A. Structure of the Paper

There are six primary sections to the paper. Section I introduces software testing, and the need for automation. Section II discusses software testing and artificial intelligence. Section III focuses on automated unit test generation in software projects, Section IV reviews generative ai models for test case generation. The conclusion and plans for the future are presented in Section V.

II. SOFTWARE TESTING AND ARTIFICIAL INTELLIGENCE

The goal of software testing is to identify and fix any issues with the program. While flawless software performance is essential, it is inevitable that software contain bugs or faults. Software maintenance, feature addition, code rewriting, bug resolution, and development are all potential sources of bugs [9]. This means the program has to be tested in several environments before the development team can hand it over to the client. Software testing is approached with a variety of approaches and methodologies. Which software testing approach is best chosen is dependent on the software's characteristics? Because manual software testing procedures are so time-consuming, automation has emerged as a viable solution [10]. This section explains how AI can automate software testing and why this method is gaining more popularity than others. ML is a major and extensively used subfield within AI, which is a large area with numerous subareas. It is possible to use the acronym "ML" to describe both DL and ML. The scope of AI software testing, shown in Figure 1, includes a wide range of AI features and capabilities.

Figure 1 : AI Software Testing Scope

A. AI Testing Approaches and Services

The methods illustrated in Figure 2 could be used to test AI software.

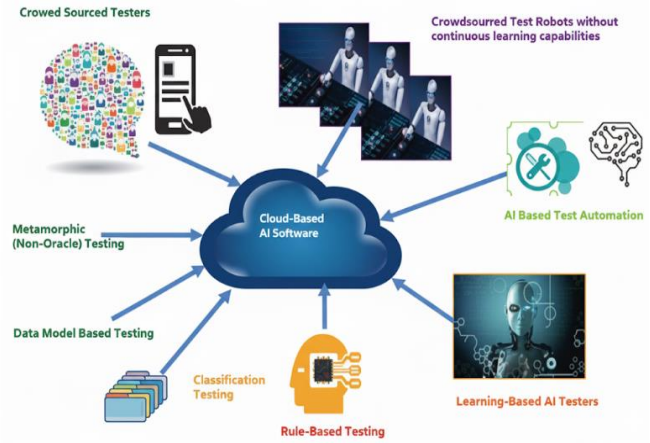

Figure 2 : Different Testing Approaches for AI Software

- Software testing for AI that is rule-based relies on preexisting rules developed and validated by experts.
- A classification-based strategy to testing AI software ensures comprehensive testing across all domains by covering a large variety of input data classes, categorizing situations and settings, and using many different types of classifications [11].
- Creating testable and traceable AI test models from pre-existing machine learning models is the first step in model-based AI software testing. This helps with AI software testing as well as evaluating the quality of training and test data.
- AI software testing relies on data-driven methods and AI models to streamline and enhance AI software testing from several angles.

- The problems with test oracles and test case generation are tackled by the property-based method of software testing known as metamorphic (non-Oracle) testing.
- Software testing robots for artificial intelligence, wherein automated software test robots are programmed to mimic the actions of human testers and acquire new skills, allowing them to do user-oriented testing operations with the use of test data and scenarios.

The test adequacy criteria, or the standard for what makes a test acceptable, are defined by the various facets of testing that have been the subject of research and practice [12]. Many criteria of this kind have been considered and studied, with a lot of study attempting to back up the usage of certain criteria. Next, we'll go over some test adequacy requirements.

a) Testing Types:

Two main categories of testing are:

i) *Manual Testing*

Testers conducting manual testing work directly with test cases rather than relying on automated scripts or tools. Assuming the role of a real user, the tester runs the program through its paces to find any unforeseen behaviour or bugs.

ii) *Automated Testing*

This method of software testing involves creating test cases and then running them using built-in software tools. Common names for the programs used for automated testing include test automation frameworks and tools [13].

B. Testing Techniques:

A total of three primary methods of evaluation has been established.

a) Black-box Testing

Black-box testing, sometimes called functional testing, is a way to evaluate software that focusses on the program's surface behaviour rather than its fundamental logic [14]. Foundational to black-box testing are the software's inputs and outputs.

b) White-box Testing

White-box testing, also known as structural testing, constructs test cases based on the SUT's implementation. As the test suite runs, it checks to see that all SUT structures have been used.

c) Gray-box Testing

The "grey box testing" method involves assessing software without having a complete understanding of how it is designed and implemented. Finding and identifying the problematic areas is the main goal of grey box testing [15].

III. AUTOMATED UNIT TEST GENERATION SOFTWARE PROJECTS

Software testing automation is creating a program in any scripting or programming language that uses an external automation tool to mimic the steps of creating a manual test case. To verify the source code that has already been implemented, toolkits are developed [16]. Its purpose is to facilitate better testing process automation. Both the composing of test scripts and the development of the program are considered development tasks. The former pertains to the application itself, while the latter pertains to the scripts that employed to test the application. Tests that allow for automation:

- Database entry, file logging, and background processes are all instances of inaccessible parts of the system.
- Payment systems, registrations, and similar features are frequently used but have a high error rate. Because thorough functional tests usually take several minutes, automation ensures that faults made quickly.
- The purpose of load testing is to determine how well a system handles a large influx of requests.
- Processes involving templates, including data searches, multi-field form entry, and preservation verification.
- Validation messages: Enter incorrect information and double-check the validation.
- long-term, end-to-end scenarios.
- Accurate mathematical calculations, including data verification, are used in fields like accounting and analytical processing.
- Assuring precision in the data search.

A. Automation Testing Tools

The section delves into many advanced automated testing tools that can generate test cases. There were many automated testing tools that emerged. So, the tester needs to research existing testing tools and weigh their pros and downsides as well as any limitations in terms of time, money, productivity, and speed before settling on a suitable tool [17]. Its primary goal should be to cover as many test cases as possible while keeping the number of available test pathways to a minimum. Automated testing solutions are utilized to decrease the need for human intervention in routine and unskilled work. Developers are capable of making the crucial tool selections needed to construct software systems. Below are some examples of commonly used UML tools.

Several well-known open-source UML modelling tools include Umbrello, Astade, Fujaba, Agro UML, and coral. It can use UML diagrams with these modelling tools: Metricview, VP-UML, object domain, Selenium, Watir, Visual UML, Enterprise Architect, and Visual UML All in One.

B. Challenges and Limitations of AI in Unit Testing

Some challenges and limitations of Artificial intelligence in unit testing are as follows:

- Accuracy and Thoroughness: The tests generated by the AI may not always reflect real-world usage at scale. It is important to review and update test scenarios [18].
- Complexity and Maintenance: The AI generates a lot of complex repetitive code many times to test the features which is impractical and is hard to maintain without documentation.
- Data Privacy and Security: Institutions must be sensitive to training AI on proprietary code. Powerful protocols are essential, such that sensitive logic or user information gets kept in a confidential and safe way.
- Integration Overhead: Unless AI-driven testing is properly planned, its implementation may cause disruptions to the traditional workflow.

IV. GENERATIVE AI MODELS FOR TEST CASE GENERATION

Automation in Software Testing: GenAI is a new technique in software testing. GenAI makes software testing much more efficient and effective because with its assistance, machine learning models, that is, deep neural networks and LLMs, generate test cases by itself and detect problems [19]. Complex tasks such as software behaviour prediction and failure detection fall within GenAI capabilities due to its data size training capabilities; both on large units of data such as code bases, error reports, and previous defect history.

- Test Case Generation: Great application of Generative AI in software testing is automation of test case generation. When testing a software, it is essential to generate test cases to cover all possible use cases, including edge cases and boundary conditions. GenAI has the ability to generate a broad test-space using the learned patterns of valid input, edge cases and unexpected behavior in software. This enable the more detailed testing coverage as it can generate test cases which would be missed by human testers which enhance the strength of testing process.
- Defect Detection: GenAI is also important in defect detection besides production of test cases. GenAI is able to detect any possible flaws, vulnerabilities and performance problems by studying the code of the software and historical defect data, thus preventing them from occurring throughout the execution. GenAI models have the capability of identifying trends in error logs and past software actions to estimate the occurrence of new defects in the current version of the software.
- Automation Scalability: The main benefit of including GenAI in software testing is its scalability. With software development teams moving towards Agile methodology and CI/CD systems, rapid and effective testing is of high importance. GenAI permit the expansion of the extent of automation through rapid-writing test cases and detection routines and through testing the software at all times through the lifecycle. This allows the teams to keep pace with the frequent code changes and deployments and guarantee high quality of software.

A. Generative AI Model

Generative models can generate fresh samples with high similarity to the training samples by learning from the underlying data distribution [20]. These models are capable of producing realistic results that are comparable to the input data, and the latter is typically done by VAEs, autoencoders, and GANs.

- Autoencoders: Neuronal networks can be trained to do things like autoencode data into a smaller representation (the encoder) before decoding it back into its original form (the decoder). Achieving this goal requires reducing the data's dimensionality while preserving its vital properties. While autoencoders are most commonly used for denoising data, they also have the ability to produce new data points that closely resemble their training data by eliminating irrelevant and noisy information.
- Variational Autoencoders (VAEs): VAEs improve upon autoencoders by adding a probabilistic step to the encoding process. Instead of producing a singular compact representation, VAEs during training generate a probability distribution in latent space. This gives them a lot of leeway in terms of what they can accomplish with a given input, and it also adds an element of creativity and chance to the data they generate.
- Generative Adversarial Networks (GANs): Two neural networks, the generator and the discriminator, make up a GAN. Generating hypothetical data samples is the generator's responsibility, while discriminating between real and fake data is the discriminators. Training involves the generator deceiving the discriminator into believing it is receiving real data when, in reality, it is only training the generator. This process is antagonistic and causes the generator to produce more compelling results.
- RNNs and LSTM: The use of RNNs and LSTM networks allows for the generation of sequential data, which includes text and music. These models are able to take into account past inputs when generating future outputs because of feedback

loops. They are able to produce new sequences that are contextually appropriate and coherent by examining patterns in the training data.

B. Test Coverage Limitations

Particularly with complicated systems, ensuring full coverage of tests is a big challenge. The extent to which the test cases simulate the application's code pathways, features, and user interactions is called test coverage. There are however some problems with the attainment of comprehensive coverage:

- Complexity of Systems: Complex systems with many moving parts, interactions, and dependencies make it hard to design exhaustive test cases. The complexity grows exponentially as more features or changes are added, and it is difficult to guarantee that all the paths have been tried.
- Dynamic Nature of Software: Software is continuously updated, bugs eliminated, and new features added; therefore, comprehensive and current test coverage becomes an ongoing process [21]. Any modifications on the application may negate some test cases or may also present new situations that require to be addressed making testing more challenging.
- Edge Cases and Rare Scenarios: Edge cases or uncommon scenarios may be quite difficult to identify and develop test cases. Such conditions may not be apparent in preliminary test design and their omission may lead to a failure to cover and their presence go unnoticed.
- Resource Constraints: The big testing is highly consumptive of both time, expertise and computer power. When there are scarce resources, it might not be possible to construct and implement sufficient test cases to give complete coverage.
- Human Limitations: The test designers are unable to cover all areas due to cognitive or experience limitation leading to some gaps covered by the test. The manual test case creation has little knowledge and attention.
- Automation Challenges: Automatic testing tools could be used to enhance coverage but they are limited. Automated tests might find it difficult to handle highly dynamic or complex user interactions and need careful planning and updating to reflect all the relevant cases.

C. Role of Generative AI in Software

Testing Generative AI is gradually becoming popular in software testing as software development and testing industries are getting progressively more automated. This is a new method that helps companies to transcend the boundaries of conventional automated software testing. This is in contrast with the testing systems that are only able to follow set steps, whereas the DevOps services and solutions based on general AI are capable of generating more innovative and valuable outputs on their own [22]. In addition, the scope and extent of usage of artificial intelligence by QA (quality assurance) is enormous and hence it is important software testers understand this paradigm shift to produce high quality software without errors that could cause failure post release.

D. Enhancing Software Quality with Generative AI

Generative AI introduces intelligent mechanisms that strengthen software development practices by improving accuracy, reliability, and adaptability [23]. Its integration into the testing and development lifecycle helps teams overcome traditional limitations and ensures higher standards of code quality.

a) Improving Code Accuracy and Consistency

Generative AI significantly improves code accuracy. It's impossible in large projects with many contributors to keep code standards manually. AI helps to solve this problem by comparing new submissions to vast datasets. It flags syntax errors, suspicious patterns and potential logical missteps. The feedback of the model would strengthen the style guides, best practices and consistent documentation over time with repeated exposure.

b) Enhancing Code Coverage and Testing Depth

The biggest problem that any software team faces is the awareness that they possess sufficient tests to detect the critical issues. Generative AI would be the first choice when investigating edge cases which a human may not necessarily take into account especially when the state is infrequent and can be catastrophic to ignore.

c) Reducing Human Error and Bias

Human engineers, no matter how skilled they are, are capable of errors and oversights or conditioned behavior of the code as to how it ought to act. Generative AI offers a non-partisan view and completes logic checks in order to identify aberrations [24].

d) Supporting Continuous Integration and Delivery (CI/CD)

Generative AI is a reduction of feedback loops, which is a mainstay of that of current CI/CD pipelines. When new code merges with the main branch, the AI automatically creates or updates its related tests. Issues are detected early, avoiding broken code getting to production. This fast feedback helps large teams that work on several modules in parallel.

V. LITERATURE REVIEW

Table I provides a synopsis of recent studies on Generative AI models for software automated unit test case creation. These studies cover the methodology, primary results, problems, and recommendations for prospective future research.

Table 1 : Summary of Literature Review Based on Automated Unit Test Case Generation in Software

References	Study On	Approach	Key Findings	Challenges	Future Work
Aleti (2023)	Challenges and opportunities in testing GenAI systems	Explores limitations of traditional testing for GenAI and suggests new directions	Highlights inadequacy of existing techniques for GenAI testing	Non-determinism, emergent behaviors, inadequacy of traditional QA methods	Develop advanced testing frameworks to safeguard GenAI and ensure quality
Missaoui, Gerasimou & Matragkas (2023)	Data augmentation and robustness testing of DL models	Presents GENFUZZER, a fuzzing method for coverage-guided data augmentation	Achieved up to 26% increase in coverage criteria; generated more informative datasets	Existing augmentation lacks diversity; robustness remains underexplored	Broaden adoption of generative fuzzing for testing DL models
Fan et al. (2023)	Comprehensive Review of LLMs for Software Development	Reviews LLM applications and identifies open challenges	LLMs show creativity in coding, design, repair, documentation, etc.	Hallucinations, unreliability, lack of verification	Develop methods to filter incorrect outputs and ensure reliability
de Santiago Júnior (2022)	Testing scientific software using AI-based oracles	Proposes TOrC using CNNs (up to 161 layers) as test oracles with transfer learning	CNNs effective in verifying non-trivial outputs like images	Non-deterministic behaviors and complex outputs of scientific software	Expand AI-based oracle techniques and improve reliability in scientific domains
Nigar et al. (2022)	Software Project Scheduling (SPS) optimization	Proposes a dynamic multi-objective SPS model incorporating employee experience evolution	Captures effect of employee learning on large-scale project outcomes	Managing multi-team dynamics and resource allocation	Refine SPS models with real-world datasets and adaptive optimization techniques
Ebadi et al. (2021)	Testing autonomous driving systems (Baidu Apollo)	Uses evolutionary automated test generation in SVL simulator	Identified failure-revealing scenarios using multi-criteria safety heuristics	Complex input modeling, ensuring real-world applicability	Enhance automated test generation for broader autonomous driving platforms

Aleti (2023) delve into the difficulties presented by GenAI systems and talk about possible openings for future testing-related inquiry. Traditional testing methodologies are inadequate or inadequate due to the special properties of GenAI systems, which I shall touch on briefly. In order to better understand how to secure GenAI and to facilitate better quality assurance in this dynamic field, it is necessary to tackle these difficulties and do additional research [25].

Missaoui, Gerasimou and Matragkas (2023) Datasets lacking diversity are a result of current data augmentation methods, which mostly center on basic geometric and color space modifications such as noise, flipping, and scaling. In most cases, the obtained results from testing Deep Learning models on the supplemented dataset do not provide useful information regarding the models' robustness. To fill this void, provide GENFUZZER, an innovative fuzzing method for Deep Learning models supported by generative AI that uses coverage-guided data augmentation and shows how method works by utilizing popular image classification datasets and models shows how it generates informative datasets that lead to a 26% increase in popular coverage criteria [26].

Fan et al. (2023) examines the new field of SE using LLMs. Additionally, it lays down unanswered questions regarding how software developers might use LLMs to solve technical problems. Coding, design, requirements, repair, refactoring, documentation, analytics, performance enhancement, and the whole gamut of Software Engineering tasks can benefit from LLMs' emergent qualities, which inject creativity and new ideas. On the other hand, there are substantial technological obstacles posed by these emergent qualities; specifically, require methods that can consistently exclude erroneous answers, such hallucinations [27].

Junior (2022) The non-determinism and non-trivial nature of the outputs (such as graphics) produced by scientific software makes testing it a difficult task. AI is a new reality that is helping software testing grow overall. Evaluate seven different DNNs as test oracle procedures for evaluating scientific models in this article. These networks are deep CNNs with up to 161 layers. The TOrC approach combines second-order mutations and combinatorial interaction testing on the original codes to generate validation, test, and training picture datasets. Traditional procedures like transfer learning, an approach suggested for DNNs, are also part of TOrC. After that, made sure the oracles (CNNs) were working properly [28].

Nigar et al. (2022) Software project scheduling (SPS) is a scheduling issue that arises in multi-team environments when limited human resources are assigned assignments. Completing large-scale projects on schedule and under budget is directly impacted by personnel' experience progression, among other dynamic events. Incorporating workers' experiences with learning ability progression over time, this research develops a novel SPS model as a dynamic multi-objective optimization problem [29].

Ebadi et al. (2021) provides studies on assessing the pedestrian detection and emergency braking system of the Baidu Apollo autonomous driving platform using the SVL simulator. The proposed method of evolutionary automated test generation generates SVL scenarios that reveal Apollo failures. Approach models the input space using a generic and flexible data structure. The optimization of the objective function is then applied to a multi-criteria safety-based heuristic [30].

VI. CONCLUSION AND FUTURE WORK

This paper presents a generative artificial intelligence-based software testing architecture that demonstrates a remarkable enhancement in the automation of software quality assurance. Large Language Models (LLMs) are leveraged to achieve significant improvements in congruence, accuracy, and test case development through advanced methods such as Retrieval-Augmented Development, quick tuning, and hallucination filtering. The introduction of AI and GenAI in software testing marks a radical shift in the future of software engineering. AI-based testing technologies, including rule-based, classification-based, and model-based testing, have already improved scalability and efficiency. Generative AI further enhances this by autonomously generating test cases, identifying defects, and covering rare cases that may not be captured through manual or automated rule-based testing. Its flexibility allows continuous improvement, enhances code quality, reduces human bias, and provides immense testing depth. Overall, AI and GenAI not only automate the testing process but also significantly enhance software quality, making them critical to modern software engineering.

Future research should focus on adaptive GenAI models that evolve with software changes to ensure real-time coverage. Explainable frameworks will increase trust, domain-specific models will enhance reliability, CI/CD integration will enable autonomous testing, and privacy-conscious solutions will ensure safe, human-AI collaboration.

VII. REFERENCES

[1] M. Tufano, D. Drain, A. Svyatkovskiy, S. K. Deng, and N. Sundaresan, “Unit Test Case Generation with Transformers and Focal Context,” May 2021, doi: 10.48550/arXiv.2009.05617.

[2] X. Guo, H. Okamura, and T. Dohi, “Automated Software Test Data Generation With Generative Adversarial Networks,” IEEE Access, vol. 10, pp. 20690–20700, 2022, doi: 10.1109/ACCESS.2022.3153347.

[3] A. Fontes, G. Gay, F. G. de O. Neto, and R. Feldt, “Automated Support for Unit Test Generation: A Tutorial Book Chapter,” pp. 1–29, Oct. 2021, doi: 10.48550/arXiv.2110.13575.

[4] V. S. Thokala, “Utilizing Docker Containers for Reproducible Builds and Scalable Web Application Deployments,” Int. J. Curr. Eng. Technol., vol. 11, no. 6, pp. 661–668, 2021, doi: 10.14741/ijcet/v.11.6.10.

[5] A. Mustafa et al., “Automated Test Case Generation from Requirements: A Systematic Literature Review,” Comput. Mater. Contin., vol. 67, no. 2, pp. 1819–1833, 2021, doi: 10.32604/cmc.2021.014391.

[6] D. Thakur, A. Mehra, R. Choudhary, and M. Sarker, “Generative AI in Software Engineering: Revolutionizing Test Case Generation and Validation Techniques,” Iconic Res. Eng. Journals, vol. 7, no. 5, pp. 281–293, 2023.

[7] S. Wang, N. Shrestha, A. K. Subburaman, J. Wang, M. Wei, and N. Nagappan, “Automatic Unit Test Generation for Machine Learning Libraries: How Far Are We?,” in 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), IEEE, May 2021, pp. 1548–1560. doi: 10.1109/ICSE43902.2021.00138.

[8] H. P. Kapadia and K. B. Thakkar, “Generative AI for Real-Time Customer Support Content Creation,” J. Emerg. Technol. Innov. Res., vol. 10, no. 12, pp. i36–i43, 2023.

[9] M. Islam, F. Khan, S. Alam, and M. Hasan, “Artificial Intelligence in Software Testing: A Systematic Review,” in TENCON 2023 - 2023

IEEE Region 10 Conference (TENCON), IEEE, Oct. 2023, pp. 524–529. doi: 10.1109/TENCON58879.2023.10322349.
[10] G. Modalavalasa, "The Role of DevOps in Streamlining Software Delivery: Key Practices for Seamless CI/CD," Int. J. Adv. Res. Sci. Commun. Technol., vol. 1, no. 12, pp. 258–267, Jan. 2021, doi: 10.48175/IJARSCT-8978C.
[11] J. Gao, C. Tao, D. Jie, and S. Lu, "Invited Paper: What is AI Software Testing? and Why," in 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE), IEEE, Apr. 2019, pp. 27–2709. doi: 10.1109/SOSE.2019.00015.
[12] Z. Khaliq, S. U. Farooq, and D. Khan, "Artificial Intelligence in Software Testing : Impact, Problems, Challenges and Prospect," arXiv, 2022, doi: 10.48550/arXiv.2201.05371.
[13] A. S. Verma, A. Choudhary, and S. Tiwari, "Software Test Case Generation Tools and Techniques: A Review," Int. J. Math. Eng. Manag. Sci., vol. 8, no. 2, pp. 293–315, Apr. 2023, doi: 10.33889/IJMEMS.2023.8.2.018.
[14] E. Daka and G. Fraser, "A Survey on Unit Testing Practices and Problems," in 2014 IEEE 25th International Symposium on Software Reliability Engineering, IEEE, Nov. 2014, pp. 201–211. doi: 10.1109/ISSRE.2014.11.
[15] V. S. Thokala, "A Comparative Study of Data Integrity and Redundancy in Distributed Databases for Web Applications," Int. J. Res. Anal. Rev., vol. 8, no. 4, pp. 383–389, 2021.
[16] D. S. N, S. D. S, D. Vijayasree, N. S. Roopa, and A. Arun, "A Review on the Process of Automated Software Testing," no. September, 2022, doi: 10.48550/arXiv.2209.03069.
[17] V. Maheshwari and M. Prasanna, "Generation of Test Case using Automation in Software Systems – A Review," Indian J. Sci. Technol., vol. 8, no. 35, Dec. 2015, doi: 10.17485/ijst/2015/v8i35/72881.
[18] H. S. Chandu, "Advancements in Asynchronous FIFO Design: A Review of Recent Innovations and Trends," Int. J. Res. Anal. Rev., vol. 10, no. 02, pp. 573–580, 2023.
[19] L. Barrio, "Leveraging Generative AI for Scalable Automated Test Case Generation and Defect Detection." 2023. doi: 10.13140/RG.2.2.22882.54726.
[20] C. Ebert and P. Louridas, "Generative AI for Software Practitioners," IEEE Softw., vol. 40, no. 4, pp. 30–38, Jul. 2023, doi: 10.1109/MS.2023.3265877.
[21] A. Goyal, "Driving Continuous Improvement in Engineering Projects with AI-Enhanced Agile Testing and Machine Learning," Int. J. Adv. Res. Sci. Commun. Technol., vol. 3, no. 3, pp. 1320–1331, Jul. 2023, doi: 10.48175/IJARSCT-14000T.
[22] A. Bandi, P. V. S. R. Adapa, and Y. E. V. P. K. Kuchi, "The Power of Generative AI: A Review of Requirements, Models, Input–Output Formats, Evaluation Metrics, and Challenges," Futur. Internet, vol. 15, no. 8, Jul. 2023, doi: 10.3390/fi15080260.
[23] A. R. Chan, "Generative AI : Automating Code and Unit Testing for Faster , High-Quality Development," ICONIC Res. Eng. JOURNALS, vol. 6, no. 7, pp. 527–537, 2023.
[24] H. P. Kapadia, "Generative AI for Real Time Conversational Agents," Int. J. Curr. Sci., vol. 13, no. 3, pp. 201–208, 2023.
[25] A. Aleti, "Software Testing of Generative AI Systems: Challenges and Opportunities," in 2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE), IEEE, May 2023, pp. 4–14. doi: 10.1109/ICSE-FoSE59343.2023.00009.
[26] S. Missaoui, S. Gerasimou, and N. Matragkas, "Semantic Data Augmentation for Deep Learning Testing Using Generative AI," in 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, Sep. 2023, pp. 1694–1698. doi: 10.1109/ASE56229.2023.00194.
[27] A. Fan et al., "Large Language Models for Software Engineering: Survey and Open Problems," in 2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE), IEEE, May 2023, pp. 31–53. doi: 10.1109/ICSE-FoSE59343.2023.00008.
[28] V. A. de S. Junior, "A method and experiment to evaluate deep neural networks as test oracles for scientific software," in Proceedings of the 3rd ACM/IEEE International Conference on Automation of Software Test, New York, NY, USA, NY, USA: ACM, May 2022, pp. 40–51. doi: 10.1145/3524481.3527232.
[29] N. Nigar, M. K. Shahzad, S. Islam, S. Kumar, and A. Jaleel, "Modeling Human Resource Experience Evolution for Multiobjective Project Scheduling in Large Scale Software Projects," IEEE Access, vol. 10, pp. 44677–44690, 2022, doi: 10.1109/ACCESS.2022.3169596.
[30] H. Ebadi, M. H. Moghadam, M. Borg, G. Gay, A. Fontes, and K. Socha, "Efficient and Effective Generation of Test Cases for Pedestrian Detection - Search-based Software Testing of Baidu Apollo in SVL," in 2021 IEEE International Conference on Artificial Intelligence Testing (AITest), IEEE, Aug. 2021, pp. 103–110. doi: 10.1109/AITEST52744.2021.00030.