iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 12 **Issue:** II **Month of publication:** February 2024

DOI: https://doi.org/10.22214/ijraset.2024.57902

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue II Feb 2024- Available at www.ijraset.com

A Survey Paper Review on Advancements in AI Driven User Interface Testing

Shital Hote[1], Shivam Dhonde[2], Tanmay Jadhav[3], Atharva Joshi[4], Manish Jansari[5]
[1, 2, 3, 4] *Student,* [5]*Assistant Professor Department of Computer Engineering, SCTR's Pune Institute of Computer Technology, Pune, India*

Abstract: ***This survey explores the quality of software engineering and highlights the important role of artificial intelligence (AI) in improving software testing. It emphasizes the importance of software testing to determine the effectiveness and capabilities of software programs. This paper highlights inconsistencies in measurement guidance and the need for automation. It will also provide a better look at the changing ecosystem of automation products driven by roles in the convergence of the artificial intelligence and machine learning (ML) eras. An AI-powered machine is made based on machine learning principles and is known as a tool that performs test models, provides logic, solves problems and performs tasks correctly. The main purpose of this evaluation is to explain the practical use of artificial intelligence in software testing and to conduct an in-depth analysis of its impact on software performance and development, improving agility. In summary, this communication provides a vision for the future by demonstrating the effectiveness of intelligent automation tools in the software testing environment, making the transition to software development reliable and convenient. More generally, this survey paper discusses today's practices of using AI to improve software development and continually unlock problem-solving innovation in software testing and software engineering along with making UI testing more reliable.***

Keywords: BDD, Model driven approach, Slang Specification Language, Process Model, Test Model, Cucumber J, Slang, JBehave.

I. INTRODUCTION

Software engineering is the design and application of sound engineering principles to create suitable software that is both reliable and works on real systems. Creating a good software product requires a good development process. Software development is a human activity that involves many tasks. These activities; Analysis, design, implementation and testing all lead to the creation of the final product. Because these activities occur constantly during development, it takes time to create a working version of the system. Software testing is one of the most important activities in software development used to define and verify software systems. Testing helps software developers ensure that the software they create performs its intended function and determines whether any problems are fixable. As the software development life cycle is a complex process, there is an urgent need to deliver new products within the stipulated time.

In the software industry, automation plays an important role in test development. Various automation tools are available to streamline the testing process. New technologies such as artificial intelligence (AI) and machine learning (ML) are constantly being used to speed up the software development process. With the development of artificial intelligence technology, many businesses are adopting and using artificial intelligence-based software. Artificial Intelligence systems are built on machine learning models and techniques.

Artificial intelligence is used to facilitate automation and reduce the cost of routine work in the experimental phase by applying logic, problem solving and machine learning. The purpose of this article is to introduce the application and impact of artificial intelligence technology in software test automation. The article concludes with conclusions and ideas for future work to increase the usefulness of AI automation tools.

Testing and similar activities continue taking on during the entire development phase and can take a lot of effort to produce the required software [1]. In [3], author has criticized the inadequate infrastructure for software testing. Author encourages the use of AI in software testing and claims that the software quality

Problems are not too much different from other tasks, which has been successfully tackled by artificial intelligence techniques [3]. As a very positive sign, artificial intelligence has been widely investigated and used to automate the process of software testing

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue II Feb 2024- Available at www.ijraset.com

II. LITERATURE REVIEW

A. Towards Behavior Driven Graphical user interfaceTesting

The researchers propose a specification language called Slang that integrates behavior-driven development (BDD) feature descriptions with wireframe models. This allows for fully automated testcase generation. They conducted an experiment comparing specifying automated test cases usingSlang versus JBehave, an existing BDD tool. The experiment found that Slang was more efficient, taking 63% less time on average to create automated test cases. Slang consists of feature descriptions, mapping definitions, and wireframe models. Thefeature descriptions specify the behavior of the user interface using BDD-like sentences. The mapping definitions map wireframe elements to their implementations. The wireframe models represent the user interface using low-fidelity prototypes. The researchers developed generators that combine the feature descriptions, mapping definitions, and wireframe models to automatically generate executable test scripts and page objects.

The researchers argue that Slang addresses three common issues with GUI test cases: they are expensive to write, time consuming to execute, and brittle. By automatically generating most of the test code, Slang decreases the time required to write test cases. In summary, the experiment and case study results show that Slang offers an efficient language for specifying automated GUI test cases, even for users with limited programming skills. The generated test cases reduce the manual effort needed compared to existing BDD tools.

B. Systematic Automation of Scenario Based Testing of UI

The authors propose a model-driven approach to separate the user interface experience from test scripts. Evaluations are determined by the interactive user interface. During test execution, abstract tests are added along with UI data and executed against the system. They added meta models to standard user interfaces for graphical user interfaces such as desktop and web applications. Meta models are designed to reduce modeling effort by allowing parts of the model to be tested separately. This can be expanded with further testing. A multi-layered process for testing adapters is recommended. Each additional layer moves away from the user interface concept until a system interface that is independent of the user interface is created. The test data defined at this stage is mostly abstract. The adapter uses the UI model to create tests and execute them based on the correct tests.

Finally, the author presents a proof of concept used to test the open source Bugzilla application. Although the functional model is shown, the meta model needs to be adapted to specific applications and requires supporting tools to create and manage the user interface model. The main points to be successful are: the model-driven approach aims to separate the UI experience from the testing, the UI meta model is embedded in the GUI model, the multi-layered process for testing adapters is proposed, and the use of the concept of proofs. However, more work is needed to adapt the meta model to specific applications and to develop supporting tools.

C. Utilizing User Interface Models for Automated Instantiation and Execution of System Test

The text discusses an approach for systematically testing the effectiveness and compatibility of interactive applications using task models and scenario-based testing. The proposed approach consists of two phases, ensuring the effectiveness of the application by automatically generating test scenarios from task models and executing them on the application. Any mismatches indicate errors that need to be fixed in either the task model or the application. The goal is to ensure the application allows users to achieve all the tasks specified in the model. Ensuring compatibility between the application and task model by generating not only normative scenarios but also mutated scenarios to capture possible usererrors. Both positive and negative test cases are executed to verify if the application allows more behaviors than described in the task model. Any mismatches are analyzed to determine if the erroneous behavior should be allowed or fixed.The approach is illustrated using an example from aircraft cockpits. Task models are created for the relevant tasks andscenarios are automatically generated from them. The scenarios are then executed on an interactive flight control unit application to test for effectiveness and compatibility. The proposed approach aims to provide a systematic and repeatable way to analyze the effectiveness and compatibility of interactive systems using task models and scenario-based testing. The automatic generation and execution of test scenarios helps achieve better test coverage than manual testing.

D. Automation of GUI Testing Using a Model-driven Approach

This article discusses issues related to system evaluation of applications with user interfaces. Test scripts are often brittle and difficult to manage due to the combination of functional logic and data-specific user interface. This leads to high maintenance costs and test automation is often abandoned. The authors propose a model-driven approach to separate the user interface experience from test scripts.

Evaluations are determined by the interactive user interface. During test execution, abstract tests are added along with UI data and executed against the system. They added meta models to standard user interfaces for graphical user interfaces such as desktop and web applications. Meta models are designed to reduce modeling effort by allowing parts of the model to be tested separately. This can be expanded with further testing. There are several methods to test adapters. Each additional layer moves away from the user interface concept until a system interface that is independent of the user interface is created. The test problems defined at this stage are mostly abstract. The adapter uses the UI model to create tests and execute them counterfactually.
Finally, the author presents a proof of concept used to test the open source Bugzilla application. Although the functional model is shown, the meta model needs to be adapted to specific applications and requires supporting tools to create and manage the user interface model. To summarize, the main points are: the model-driven approach aims to separate UI experience from testing, the UI metamodel is incorporated into the GUI model, various methods for testing adapters have been proposed, and evidence discusses the use of the concept. . However, more work is needed to adapt the metamodel to specific applications and to develop supporting tools.

E. Model based Approach to Assist Test Case Creation, Execution, and Maintenance for Test Automation

This article describes a standards-based test automation approach to provide end-to-end assistance with test documentation, execution, and maintenance. Some key points are that this approach uses the standard structure of the test in testing to inform the test model that drives test automation. Process models include tasks and methods, while test models include screens and fields. Test data is generated by the process in the standard procedure. The function will then be closed to populate the test pattern and generate the test text for each test; The test text can be modified and saved using the user interface to edit the text directly. Evaluation of the results showed that the method was implemented using tools such as Sahi, Selenium and Robot Framework. Use the Excel interface to replace the text and make the working data of the JBilling application to evaluate the method. Users find the process more efficient and time-saving. In conclusion, the main advantage of the model-based test automation approach is that it provides end-to-end support from test data to execution and maintenance when necessary. Leverage knowledge and save time and resources through a user-friendly interface with the ability of this model.

F. Image-based Approaches for Automating GUI Testing of Interactive Web-based Applications:

This article describes methods for testing and optimizing user interfaces (GUIs). Current benchmarks use methods such as image comparison, WebDriver automation and user login management. But there are still gaps in testing interactive interactions. The authors present additional techniques that combine computer vision and machine learning algorithms to analyze screenshots. This includes pattern recognition to identify shapes and lines, pattern matching to find content, and text recognition to read text. This technology integrates WebDriver to simulate user interaction. Three applications are offered: pattern detection to identify lines and shapes, pattern matching to find content, and text recognition to extract text. Use pre-processing technique to create screenshots before analysis. This strategy is evaluated in the 2D tactical map interface. The results show that masking-improved pattern matching runs only 3-9% longer while increasing accuracy by 80% compared to the standard algorithm. After adjusting the technology for specific intersections, readings are up to 100% accurate. In summary, the implementation process can measure user interactions that were previously difficult to test accurately. When combined with existing methods, they increase the number of interfaces that can be fully identified. This technology can be easily integrated into test models and operators..

G. AI in Software test Automation: A Systematic Literature Review

The text discusses the use of artificial intelligence techniques to automate software testing, particularly graphical user interface (GUI) testing. Artificial intelligence has significantly aided the automation of various software processes by reducing costs and improving quality. While AI has helped automate software testing in general, its application to GUI testing has been more limited. Automated software testing has advantages like reducing development cycles, improving test efficiency, and reducing costs. However, it also has limitations like not being able to replace manual testing completely and being dependent on the quality of test cases. AI techniques like genetic algorithms, ant colony optimization, and simulated annealing have shown promise for generating test data and optimizing test cases. However, most research has focused on software testing in general rather than GUI testing specifically. The text argues that AI techniques should be used more for GUI testing and event-driven software testing. Various AI approaches have been proposed for GUI testing like generating tests based on models, automated test case generation, and using AI planning techniques. However, GUI testing poses unique challenges due to the huge number of states that a GUI can have. While initial results have been promising, the benefits of AI for GUI testing have not been as significant as for software testing in general. The use of AI techniques could help make GUI testing more efficient and effective.

H. Automation of UI Design Testing using ML

The paper discusses various methods to automate the testing of user interface designs using machine learning techniques. Manually testing UI elements is time consuming, tedious and prone to errors. Automating the process can improve efficiency and reduce testing cycles. The key challenges in automating UI testing are detecting dynamic elements in UI designs that change based on device type and protocols and extracting regions of interest like graphics and text boxes from the UI images. Thepaper proposes a methodology that Classifies the UI image based on protocols like Hart, WiHART, Profibus etc. using a CNN model for data augmentation, preprocesses the image using grayscale, thresholding and dilation, extracts rectangular regions of interest from contours, detects text within the regions using an English language dictionary, matches regions with standard graphics using key points matching, executes test cases to detect design flaws that do not follow guidelines, stores results in an excel sheet for the development team.

In conclusion, automating the extraction of variable UIelements using machine learning can make the testing process more efficient and reduce delivery cycles. The proposed approach demonstrates how this can be achieved. The methodology can potentially be extended to other UI designs and protocols. The paper explores automating user interface design testing with machine learning to enhance efficiency and reduce testing time. It addresses challenges like dynamic elements and region extraction. The proposed methodology involves image classification, preprocessing, region extraction, text and graphics detection, and test case execution, providing a potential solution for more efficient testing across various UI designs and protocols. The key challenges in automating UI testing are detecting dynamic elements in UI designs that change based on device type and protocols and extracting regions of interest like graphics and text boxes from the UI images.

Table I: Summary Study on AI based UI testing techniques

Sr No.	Reference	Domain	Parameter study
1.	Filippo Ricca , Alessandro Marchetto , Andrea Stocco Towards BDD	AI	Introduced a language called Slang that combines BDD feature descriptions with wireframe models to enable automated test case generation. It takes 63% less time to create automated test cases thanJBehave.
2.	Systematic automation of scenario based testing of UI	ML	Uses task models and scenario-based testing to systematically assess the effectiveness and compatibility of interactive applications. It involves two phases: confirming task completion and testing for both normative and user error scenarios. Automated test scenario generation enhances coverage compared to manual testing, exemplified with aircraft cockpit applications.
3.	Utilizing User Interface Models for Automated Instantiation and Execution of System Test	AI Techniques	It discusses the problem of maintaining test scripts for UI-based applications and proposes a model-driven approach to separate UI knowledge from tests. They introduce a UI meta-model and a multi- layered test adapter architecture, demonstrating their approach with aBugzilla test case. The authors acknowledge the need for further adaptation and tool support.
4.	Automation of GUI Testing Using a Model- driven Approach	AI in Testing	This outlines an approach to automate graphical user interface (GUI) testing using UML models. It involves employing UML use cases and activity diagrams, enriched with test data requirements, to define and generate test cases. The method aims to enhance test effectiveness, manage the number of test cases, and improve data coverage while offering ease of test maintenance compared to manual scripting.

Sr No.	Reference	Domain	Parameter study
5.	Model based Approach to Assist Test Case Creation, Execution, and Maintenance for Test Automation	NLP	Presents a model-based test automation approach that simplifies test case creation, execution, and maintenance. It offers a user-friendly interface for modifying test scripts, reducing the need for technical expertise, and a case study on the JBilling application found it to be efficient and time-saving
6.	Image-based Approaches for Automating GUI Testing of Interactive Web-based Applications	GUI test	The paper introduces techniques that use computer vision and machine learning to automate the testing of complex graphical user interfaces (GUIs). These techniques include pattern detection, template matching, and text recognition, resulting in improved accuracy. Evaluations on a 2D tactical map interface reveal substantial improvements in accuracy and show promise for exhaustively testing previously challenging UIs, making them easilyintegrals into testing frameworks..
7.	AI in Software test automation: A systematic literature review	Cucumber	It discusses two key automated testing tools: Selenium, known for web application testing, and Cucumber, a behavior-driven development tool. It emphasizes the significance of studying these tools, highlights their features and methodologies, and stresses the importance of selecting the right tool based on various factors. Theultimate aim is to develop a new testing tool surpassing existing options like Cucumber.
8.	Automation of UI design testing using ML	ML	The paper explores automating user interface design testing with machine learning to enhance efficiency and reduce testing time. It addresses challenges like dynamic elements and region extraction. The proposed methodology involves image classification, preprocessing, region extraction, text and graphics detection, and test case execution, providing a potential solution for more efficient testing across various UI designs and protocols.
9.	Model based AI driven test generation system.	AI, CRT	The paper explores how to make web component classification using CRT(Computed Render Tree). Also, explores how image classification and object detection can be leveraged to aid in software testing.
10.	Artificial Intelligence Applied to Software testing		The goal of our study is to understand how AI has been applied to support ST. In this paper it focus on *Testing Activities* whose automation has been supported by different AI techniques and synthesize the main purpose for which each AI technique has been used for

III. CONCLUSION

In conclusion, this survey paper has explored the different techniques to automate the UI testing. We've examined how AI, can be used for the User Interactiontesting, how it useful for more correct and accurate results.

As we move forward, it's essential to acknowledge the challenges and complexities that come with these technologies, such as scalability, interoperability, andregulatory alignment. By harnessing artificial intelligence, machine learning and computer vision, our project seeks to automate and enhance the testing process for the dynamic Sarvatra eHub frontend.

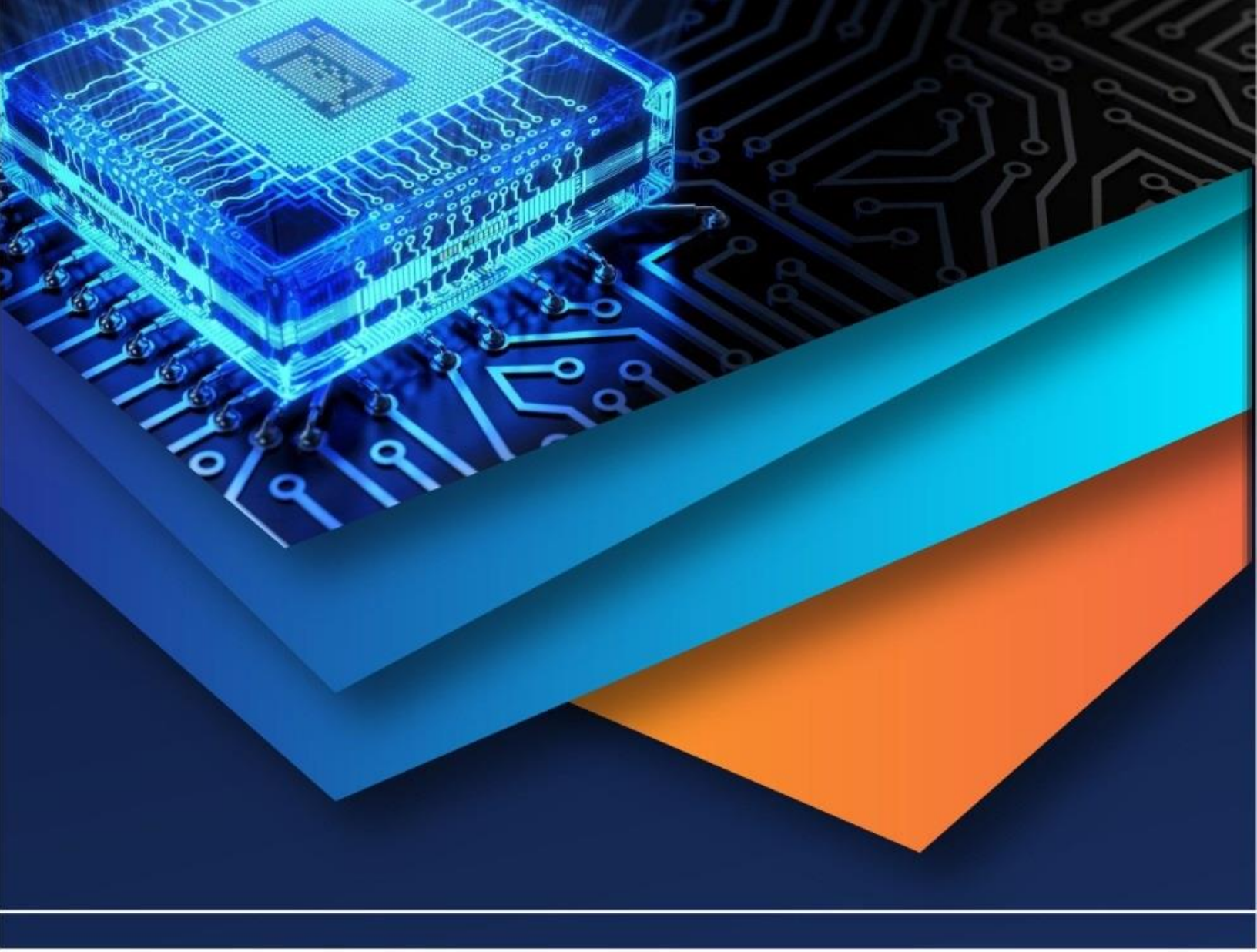
The project's scope includes dynamic test case generation, defect detection, promising comprehensive coverage and adaptability. The feasibility analysis highlights resource allocation and data availability as key factors for success. Ultimately, our mission is to ensure the Sarvatra eHub's reliability and functionality, setting new standards for UI testing in the digital age.

IV. ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our guide Mr. Manish Jansari for their valuable guidance and support throughout the course of our research project. Their expertise and mentorship have been instrumental in shaping the success of this survey paper. We are deeply thankful for their contributionsto our academic journey.

REFERENCES

[1] F. Macchi, P. Rosin, J. M. Mervi and L. Turchet, "Image-based Approaches for Automating GUI Testing of Interactive Web-based Applications," 2021 28th Conference of Open Innovations Association (FRUCT), Moscow, Russia, 2021, pp. 278-285, doi:10.23919/FRUCT50888.2021.9347592.
[2] F. Ricca, M. Leotta, and A. Stocco, "Three open problems in the context of e2e web testing and a vision: Neonate," Advances in Computers, 01 2018.
[3] K. P. Moran, C. Bernal-Cardenas, M. Curcio, R. Bonett, and D. Poshy- ´ vanyk, "Machine learning-based prototyping of graphical user interfaces for mobile apps," IEEE Transactions onSoftware Engineering, 2018.
[4] M. Leotta, A. Stocco, F. Ricca, and P. Tonella, "Automated migration of DOM-based to visual web tests," in Proceedings of 30th Symposium on Applied Computing, ser. SAC 2015. ACM, 2015, pp. 775–782
[5] Using AI to automatically Test GUI https://www.researchgate.net/publication/286851 173_Using_artificial_intelligence_to_automatica lly_test_GUI DOI:10.1109/ICCSE.2014.6926420
[6] M. Leotta, Z. Oliveira, A. Memon, Approaches and tools for automated end-to-end web testing, Adv. Comput. **101** (2016), 193– 237. http://doi.org/10.1016/bs.adcom.2015.11.00 7.
[7] E. Alégroth, Z. Gao, R. Oliveira, A. Memon, Conceptualization and evaluation of component-based testing unified with visual GUI testing: an empirical study, 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), 2015, pp. 1– 10. http://doi.org/10.1109/ICST.2015.7102584.
[8] L. G. Hayes, The automated testing handbook, Software Testing Institute, 2004.
[9] Cucumber Frameworkhttps://www.jetbrains.com/help/webstorm/runnin g-cucumber-js-unit-tests.htm
[10] Xie, X., Xu, B., Nie, c., Shi, L., and Xu, L. 2005. Configuration Strategies for Evolutionary Testing. In Proceedings of the 29th Annual international Computer Software andApplications Conference - Volume 02 (July 26 - 28, 2005). COMPSAC. IEEE Computer Society, Washington, DC.
[11] Benoit Baudry, Automatic Test Case Optimization: A Bacteriologic Algorithm, IEEE SOFTWARE Published by the IEEE ComputerSociety, 2005.
[12] A. Bertolino "Software Testing Forever: Old and New processes and techniques for Validating Today's Applications", Keynote at 9thInternational Conference Product-Focused Software process Improvement (PROFES 2008), Monte Porzio Catone, June 2008, LNCS 5089 , 2008.
[13] V. Mohan, D. Jeya Mala "IntelligenTester -Test Sequence Optimization framework using Multi-Agents", Journal of Computers, June 2008.
[14] Memon, A M., Soffa, M. L. and Pollack, M. E., Coverage criteria for gui testing. ESECIFSE-9: Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering, New York, NY, USA, 2001, ACM Press, pages 256-267
[15] Trudova, Anna et al. "Artificial Intelligence in Software Test Automation: A Systematic Literature Review." *International Conference on Evaluation of Novel Approaches to Software Engineering* (2020).
[16] Zubair Khaliq, Dawood Ashraf Khan, Sheikh Umar Farooq, Using deep learning for selenium web UI functional tests: A case-study with e-commerce applications, Engineering Applications of Artificial Intelligence https://doi.org/10.1016/j.engappai.2022.105446



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY