International Journal of Electrical Engineering and Technology (IJEET)
Volume 14, Issue 6, Nov-Dec 2023, pp. 16-30, Article ID: IJEET_14_06_003
Available online at https://iaeme.com/Home/issue/IJEET?Volume=14&Issue=6
ISSN Print: 0976-6545 and ISSN Online: 0976-6553
DOI: https://doi.org/10.34218/IJEET.14.6.003

LEVERAGING AI FOR AUTOMATED FUNCTIONAL COVERAGE AND TEST SEQUENCE GENERATION IN GPU AND AI ACCELERATOR VERIFICATION

Ankit Chandankhede
USA

ABSTRACT

Achieving comprehensive functional coverage is crucial in design verification, particularly for complex architectures like GPUs and AI accelerators. Traditionally, writing functional coverpoints and generating test sequences is a manual and skill-intensive process, often requiring directed test cases to hit complex cross-functional coverpoints. This paper presents an innovative approach utilizing AI to automate the creation of functional coverpoints and the generation of test sequences. By providing AI with transactional and protocol information in a simplified format, the system can generate base and complex sequences to cover intricate scenarios. This methodology is extended to create versatile coverpoints for different protocols, which can be reused across interfaces following similar protocols and packet types. Additionally, the AI script facilitates the creation of a high-level verification plan (HVP) that measures effective coverage across various levels of verification abstraction. The automated process significantly reduces manpower requirements, minimizes the risk of missed datapath coverage, and adapts sequences dynamically based on feedback, ensuring thorough functional coverage and efficient verification.

Keywords: Functional Coverage, Design Verification, Automated Test Sequence Generation, AI in Verification, GPU Verification, AI Accelerator Verification, High-Level Verification Plan (HVP).

Cite this Article: Ankit Chandankhede, Leveraging AI for Automated Functional Coverage and Test Sequence Generation in GPU and AI Accelerator Verification, International Journal of Electrical Engineering and Technology (IJEET). 14(6), 2023, pp. 16-30.
https://iaeme.com/MasterAdmin/Journal_uploads/IJEET/VOLUME_14_ISSUE_6/IJEET_14_06_003.pdf

1. INTRODUCTION

Software testing is highly relevant in the software development process, since around 50% of the time spent (MYERS, BADGETT and SANDLER, 2011) and around 23% of the information technology budget of companies is allocated to testing (CAPGEMINI, 2020).

In view of the importance of testing during the software development process, by applying Artificial Intelligence (AI) in software testing, it is possible to obtain the following benefits indicated by Battina (2019):

- Improved accuracy;
- Testing beyond manual constraints;
- Facilitators for developers and testers;
- Increase in the total number of tests performed;

Reduction in time and money consumed, which speeds up the release of the software to the market.

One of the benefits cited, an increase in the total number of tests carried out, is illustrated in figure 1, which shows how much better AI testing techniques can keep up with the complexity of software when compared to traditional automatic testing techniques.

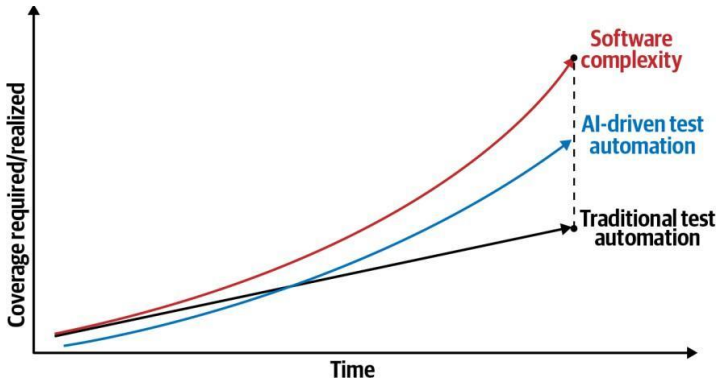

Figure 1. Increasing test coverage with AI-based automation Source: King (2021, n.p.)

Given the importance of testing and the benefits that AI applied to software testing can bring, how can we disseminate and encourage the use of software test automation tools that use AI?

In order to answer this question, the general objective of this work is to provide a demonstration of the benefits of including a tool that uses AI to carry out visual tests in a web application. To meet this general objective, the specific objectives were:

- Study Software Testing, Artificial Intelligence and software test automation tools;
- To indicate the possible benefits of using software test automation tools that use AI;
- Disseminate software test automation tools that use AI;
- Demonstrate the benefits of including AI to perform additional checks;
- Carry out experiments with software test automation tools.

2. THEORETICAL FRAMEWORK

To understand Software Testing, it is first necessary to understand what software quality is, which according to Pressman and Maxim (2021) is quality management with the aim of generating a useful product. It is important for software development because it seeks to prevent defects, evaluate implementations and reduce costs arising from failures.

An important method for achieving quality is through the use of Software Testing, the way to show that the program works according to its requirements and to identify defects before delivery to the customer (SOMMERVILLE, 2019).

Testing is part of the verification and validation (V&V) process (SOMMERVILLE, 2019). This process analyzes the product being built, verifying that it is the right way to build it and validating that it is the right product (BOEHM, 1979).

According to Sommerville (2019), testing is present in all generic software development process models.

Firstly, the waterfall model, which uses them during the implementation and unit testing and integration and system testing phases.

The testing phases of the incremental development model are interspersed with development and are called validation activities. This model is used as a basis by the agile methodology.

Finally, the approaches that use the integration and configuration model, use tests in the application configuration phases for an existing application and for cases where some components are available, apply tests in the component adaptation, component development and system integration phases.

Some software test automation tools apply the concepts of Artificial Intelligence through Machine Learning and Deep Learning algorithms.

According to Norvig and Russel (2013), AI is the field of study of agents that have the ability to function autonomously, learn from changes and create and pursue goals to achieve the best expected result.

One way of applying the concept of AI is Machine Learning (ML), which according to Murphy (2012) is a set of algorithms used to detect patterns in an automated way and use these patterns in attempts to predict future data and aid decisions.

These computer algorithms learn by performing a particular task several times, and their results improve the more they perform the same type of task (MITCHELL, 1997).

According to Bengio, Goodfellow and Courville (2015), a common categorization of these algorithms is unsupervised learning and supervised learning, which indicate the type of experience used during learning.

Unsupervised learning algorithms explore a dataset containing several features to learn useful properties of the dataset structure. This approach is generally used to solve density estimation-type tasks with the role of helping with other tasks (BENGIO; GOODFELLOW; COURVILLE, 2015).

Supervised learning algorithms, on the other hand, are used to explore a data set containing various characteristics, with each example in the set associated with a label. Tasks generally related to this categorization involve regression, classification and structured output problems (BENGIO; GOODFELLOW; COURVILLE, 2015). (AP),

A very powerful approach to applying ML is Deep Learning

[...] which allows computers to learn by experience and to understand the world as a hierarchy of concepts, with each concept defined from simple concepts. By gaining knowledge through experience, this approach avoids the need for human operators to formally specify all the knowledge the computer needs. The concept hierarchy allows the computer to learn complicated concepts by constructing them using simpler ones. If we draw a graph showing how these concepts are built one from the other, the graph is deep, with several layers (BENGIO; GOODFELLOW; COURVILLE, 2015).

Figure 2 illustrates the relationship between Artificial Intelligence, Machine Learning and Deep Learning.

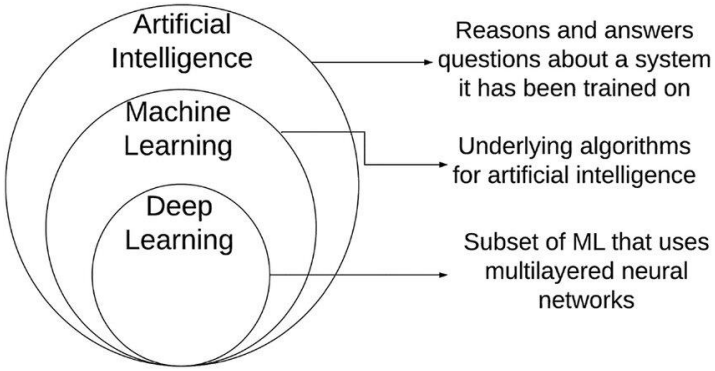

Figure 2. Venn diagram showing how Deep Learning is a type of Machine Learning, which is ultimately a set of Artificial Intelligence algorithms.

Source: author based on Bengio, Goodfellow and Courville (2015, p. 9)

There are several tools with the role of helping software testing by automating this process. The two tools with the greatest presence on the market, according to Enlyft (c2023), a company that aggregates data in real time from companies using various technological solutions, are:

Selenium

This is a web browser automation tool. It is mainly used to automate web applications for testing purposes (SOFTWARE FREENDOM CONSERVANCY, c2023). One of the ways of using such a tool is through the Selenium WebDriver, which refers to two subjects, the first being the code that enables the various programming languages to use Selenium and the second is the implementation of the browser control code (SOFTWARE FREENDOM CONSERVANCY, c2023).

Apache JMeter

This is open source software made 100% in Java for testing the load capacity and functional behavior of applications, and measuring performance. It was originally designed to test web applications, but has been supplemented with other testing functions (APACHE SOFTWARE FOUNDATION, c2022).

In addition to Selenium and Apache JMeter, there are tools that use artificial intelligence to improve the software testing process. Some of them are presented below:

Katalon

It is a quality management platform that uses AI to suggest alternative methods of locating elements when the default option fails; speed up manual checks of user interfaces and error-prone pixel comparisons with methods based on content or structure; and make root cause analysis more efficient by finding similar failures from the results of previous runs (KATALON, c2022).

Of the three plans offered, Free, Premium and Ultimate, only the latter allows use of the AI functionalities.

The promised benefits include:

- Visibility: provides the means to understand the coverage and effectiveness of software quality throughout the organization;
- Scalability: makes it possible to adjust to the needs of the business;
- Speed: makes scalability faster, reduces the time for products to be released onto the market and increases the quality of each version made available;
- Innovation: use of AI and AM to help test faster and more efficiently;
- Affordability: flexible free and paid plans to meet business needs.

Launchable

It is a software development platform focused on continuous integration that uses AM to predict the chance of failure of each test based on previous executions and changes in the code (LAUNCHABLE, n.d.).

The platform offers two types of plans, Moon and Mars, but only Mars enables the use of AI functionality.

The benefits that Launchable promises include:

- Reduced test execution time through the use of predictive test selection that identifies and executes only the 20% of tests that most impact the development process;
- Efficiency throughout the organization, visible from the tool's dashboard;
- Better visibility for projects and teams;
- Visualization of trend data on how the project has evolved.

Applitools

This is a test automation platform that uses a network of visual computing algorithms, made up of various approaches including AP, to analyze application pages and identify differences (APPLITOOLS, c2022).

The platform offers four types of plans, Free, Starter, Enterprise and Ultrafast, all of which allow the use of AI functionality. The main difference is the number of AI tests per month, with the only free plan (Free) being limited to one hundred page tests per month.

The benefits that Applitools promises include:

- Validating entire pages with a single screenshot;
- Managing dynamic or continuously changing content;
- Testing all browsers and devices in seconds;
- Automating revisions and maintenance;
- Fix errors faster with root cause analysis;
- Test for visual accessibility.

3. METHODOLOGY

The methodology was divided into bibliographical research and development. These are detailed below.

3.1. Bibliographical research

This stage was carried out in order to build a theoretical framework with the main themes of the work, thus expanding on what the Software Testing process is and understanding where it is positioned within the software development process. It also defined what Artificial Intelligence is and how Machine Learning and Deep Learning are part of this theme.

The role of software test automation tools was explained, with examples of some that are widely used and others that use Artificial Intelligence.

For the Software Testing section, the materials used were books on Software Engineering by renowned authors such as Pressman, Maxim and Sommerville. When talking about Artificial Intelligence, the materials used were books by Norvig, Russel, Murphy, Bengio, Goodfellow and Courville. These books were chosen because they are formally published works and expand on the parts needed to understand this article. As for software test automation tools, the materials chosen were websites, one from a company that aggregates data in real time from companies that use various technological solutions, and the rest from tools and their documentation. The former was chosen because it shows which tools are the most widely used on the market and the latter was chosen because it details very well what they are and how to use them.

3.2. Development

After exploring the main themes, we looked for tools that offer the availability of functionalities that use AI in free and non-temporary plans, which resulted in the choice of Applitools Eyes.

Once Applitools Eyes had been chosen, its documentation was explored and visual tests integrated with Selenium WebDriver functional tests were found. Selenium was chosen because it accounts for a large share (27.35%) of the software testing tool market according to Enlyft (c2023).

To learn how to use Applitools Eyes in conjunction with Selenium WebDriver, we followed a tutorial, provided by Applitools itself, which teaches how to carry out Selenium WebDriver tests in conjunction with the Jest framework, which uses JavaScript for testing (APPLITOOLS, c2022).

During the learning period, it was noticed that there were four algorithms, called match levels, available to carry out visual checks:

Exact: compares images pixel by pixel, being sensitive to rendering anomalies not visible to the human eye;

Strict: detects changes in text, fonts, color, graphics and the position of elements;

Ignore colors: similar to strict, but ignores color changes;

Layout: identifies page elements such as text, images, buttons and columns and checks that the relative position of the elements is consistent.

After learning about test automation tools, we chose the web application, TMDB Web App, previously developed during an online course.

TMDB Web App is a web application that uses the API of The Movie Database (TMDB), a database of movies and series fed by volunteers (THE MOVIE DATABASE, s.d.), to manage lists of movies. The technologies used to implement this application were HTML, CSS, TypeScript, ReactJS, Axios, Styled Components and Redux Toolkit (CUNHA, 2022).

Table 1 shows the functionalities, detailing the initial state required to use them and the possible final states that can occur.

Table 1. Functionalities of the web application

Functionality	Initial state	Possible final states
Sign in TMDB	User on the Login page..	A. If the user enters valid credentials, they are redirected to the Search page; B. If the user enters invalid credentials, an alert appears warning them of the invalid credentials and they remain on the Login page.
Log out TMDB	User on the Search or Lists pages..	. User redirected to the Login page.
Create list	User on the Lists page..	D. If user enters name and description, a table appears with the list created with name and description; E. If user enters name only, a table appears with the list created with name; F. If the user does not enter a name, an alert appears warning that the Name field is mandatory; G. If the user enters a name from a list that has already been created, an alert appears warning that a list with the same name already exists.
Delete list	User on the Lists page with a list created.	H. List is removed from the list table and if it is the only list, the table also disappears.
Search by movie	User on the Search page.	I. If the movie is present in the database, it shows a table with results; J. If the movie is not in the database, the page does not change.
Add movie to a created list	User on the Search page with a list created.	K. Movie is added to the selected list by enabling the Remove button and disabling the Add button on the Search page for the movie and appearing in the list of movies in the list on the Lists page.
Remove movie from a created list	User on the Search or Lists pages with a list created and a movie added to it.	L. Movie is removed from the selected list by enabling the Add button and disabling the Remove button on the Search page and disappearing from the list of movies in the list on the Lists page.

Source: author

After choosing the web application, the tests to be carried out were planned with the aim of covering the majority of the final states of the application's functionalities, resulting in table 2. The last two columns of this table refer to the first and last columns, respectively, of table 1.

Table 2. Details of the test scenarios

#	Scenario	Functionality	Expected end state
1	With valid credentials you can access the application	Login to TMDB account	A
2	Cannot access the application without valid credentials	Login to TMDB account	B
3	Can create a list with name and description	Create list	D
4	Can create a list with name only	Create list	E
5	Cannot create a list without a name	Create list	F
6	Cannot create a list with the same name as an existing list	Create list	G
7	Searches for a movie in the database	Search for movies	I
8	Searches for a movie not in the database	Search for movies	J
9	Adds a movie to a list	Add movie to a created list	K
10	On the Search page, removes a movie from a list	Remove movie from a created list	L
11	On the Lists page, removes a movie from a list	Remove movie from a created list	L

Source: author

For all the tests, previous and subsequent steps were implemented with the aim of keeping the initial and final state of the application the same and making it easier to implement the tests. With this aim in mind, the functionalities log out of TMDB account and remove list have no scenarios because they were used as subsequent steps for most of the tests.

With the tests planned, we experienced the limitation of the free

Applitools Eyes free plan of only allowing one hundred screenshots during a one-month period.

As each planned test required two captures to record the initial and final page states of each scenario, the author of this article decided to choose a sample of tests to allow for more mistakes during implementation. The tests chosen, detailed in table 2, were 1, 3, 7, 9, 10 and 11, as they demonstrate the successful status of almost all the functionalities.

4. RESULTS

The computer environment used (table 3), its preparation and the performance of the experiments are detailed below.

Table 3. Computing environment

Operating System	*Linux Ubuntu* (22.04.2 *LTS*)
Processor	*Intel Core i5-8250U*
RAM	4 GB
Browser	*Google Chrome* (113.0.5672.63)

Source: author

While preparing the environment for the experiments, tests were implemented in two ways. For the first, using only Selenium code, all the tests in table 2 were implemented. For the second, by including Applitools Eyes in the Selenium tests in order to capture the initial and final states of each test. and final states of each test, only the chosen tests (1, 3, 7, 9, 10 and 11) were implemented.

Part of the implementations involve some configurations, these implemented using as a basis the tutorial code followed to learn how to use Applitools Eyes with Selenium, which resulted in the final configurations represented by figures 3 and 4.

```
let driver;
const getDriver = () => driver; // Permite utilizacao de mesma instancia em outros arquivos

// Configura passos anteriores e posteriores dos testes Selenium
function configureSeleniumSetupAndTeardown() {
    beforeEach(async () => { // Antes de cada teste
        driver = await configureChromeWebDriver();
        await getPage();
    });
    afterEach(async () => { // Apos cada teste
        await driver.quit();// Fecha a instancia do Chrome Web Driver
    });
}

// Cria e configura instancia de Chrome Web Driver.
async function configureChromeWebDriver() {
    const driver = new Builder()                    // Cria instancia
        .withCapabilities({
            browserName: 'chrome',                  // define navegador
            'goog:chromeOptions': { args: [], },    // habilita configuracoes adicionais
        })
        .build();
    // Configura o tempo de espera para cada passo da instancia em 1 segundo
    await driver.manage().setTimeouts({ implicit: 1000 });
    // Configura a janela da instancia
    await driver.manage().window().setRect({
        x: 0, y: 0,                                 // topo da pagina
        width: SCREEN.WIDTH, height: SCREEN.HEIGHT, // resolucao de 1024×768
    });
    return driver;
}

// Acessa localhost na porta 3000 na pagina 'page'
async function getPage(page = '') {
    await getDriver().get(`http://localhost:3000/${page}`);
}
```

Figure 3: Selenium configurations

Source: author based on Applitools (c2022)

```
let eyesConfig;
let eyes;
// Permite utilizacao de mesma instancia em outros arquivos
const getEyesConfig = () => eyesConfig;
const getEyes = () => eyes;

// Configura passos anteriores e posteriores dos testes com Applitools Eyes
function configureApplitoolsEyesSetupAndTeardown() {
    let runner;
    beforeAll(async () => {             // Antes de todos os testes
        eyesConfig = new Configuration();// Cria instancia de configuracao
        // Carrega a chave da API das variaveis de ambiente
        eyesConfig.setApiKey(process.env.APPLITOOLS_API_KEY);
        runner = new ClassicRunner();   // Cria instancia para gerenciar os testes localmente
    });
    beforeEach(async () => {            // Antes de cada teste
        eyes = new Eyes(runner);        // Cria instancia do Eyes, que realiza as comparacoes
        eyes.setConfiguration(eyesConfig);// Define que a instancia Eyes utilize a configuracao atual
        await eyes.open(                                // Abre o Eyes para aceitar testes visuais
            getDriver(),                                // Objeto driver a ser observado
            APPLITOOLS_APP,                             // Nome da aplicacao a ser testada
            expect.getState().currentTestName,          // Nome do teste atual
            new RectangleSize(SCREEN.WIDTH, SCREEN.HEIGHT)// Resolucao a ser testada de 1024×768
        );
    });
    afterEach(async () => {// Apos cada teste
        // Fecha instancia do Eyes para que os testes fiquem disponiveis no painel da Applitools
        await eyes.closeAsync();
    });
    afterAll(async () => {// Apos todos os testes
        // Recupera todos os resultados dos testes
        await runner.getAllTestResults();
    });
}

/*
Para cada pagina da aplicacao a ser testada,
    utilizei a seguinte funcao na secao de passos anteriores a todos os testes da pagina
*/
// Define a qual conjunto de testes o teste atual pertence de acordo com a pagina 'page'
async function setBatchInfoForPage(page) {
    await getEyesConfig().setBatch(new BatchInfo(`${APPLITOOLS_APP} @ ${page}`));
}
```

Figure 4 - Applitools Eyes configurations

Source: author based on Applitools (c2022)

In addition to the configurations shown in figures 3 and 4, configurations were needed so that the Jest testing framework would not reject the tests due to execution time. The time set for the Selenium WebDriver tests was 10 seconds and for the tests with Applitools Eyes was 120 seconds.

The application together with the configurations and tests are available in the GitHub repository (CUNHA, 2023).

After preparing the environment, the tests were run, first with just Selenium WebDriver and then the same tests with Applitools Eyes.

After running the tests, the data obtained and observations on some of the characteristics chosen to analyze the implementation and execution of tests with Selenium WebDriver and with Selenium WebDriver in conjunction with Applitools Eyes were recorded.

These characteristics, detailed in Table 4, were chosen by considering some studies on software test automation tools (SINGH and TARIKA, 2014) (GAMIDO and GAMIDO, 2019).

Table 4. Characteristics of the implementations and executions of the software tests with the tools to be analyzed

Features	Details
Requirements for implementation and execution	What technical knowledge is needed to implement and execute the tests.
Function	What the execution of the tests with the tools validated.
Execution times	How long each test took (in seconds) and the percentage increase in execution time when including Applitools Eyes.
Execution limits	What the limits are when running the tests according to the limitations of each tool.

Source: author

Requirements for implementation and execution

To use Selenium WebDriver, following the technologies used in this experiment, you need to be familiar with writing and debugging JavaScript code, building test suites with Jest and the Selenium WebDriver SDK.

In order to include Applitools Eyes in the tests with Selenium WebDriver, in addition to the previous knowledge, it is necessary to know the Applitools Eyes SDK.

Considering this knowledge, using Applitools Eyes is difficult for someone who is new to the field of software development and quality. On the other hand, those who already have technical knowledge of the Applitools Eyes SDK will have an easier path to enjoying the benefits that Applitools Eyes offers.

Function

The function of the tests, implemented with only Selenium WebDriver, is to verify that it is possible to interact in the expected way with the elements of the application page. When testing the application including Applitools Eyes, the function expands the Selenium WebDriver check by also testing changes to the appearance of the application interface.

Execution times

Below is the data obtained from the test execution times.

Table 1. Execution times (in seconds)

#	***Selenium WebDriver***	***Selenium WebDriver* with *Applitools Eyes***	***Percentage increase in the inclusion of Applitools Eyes***
1	4,06	36,87	808,13%
3	4,10	41,65	915,85%
7	4,31	37,26	764,50%
9	7,27	46,45	538,93%
10	7,13	49,08	588,36%
11	7,64	44,73	485,47%
Total	**34,51**	**256,04**	**641,93%**
Average	**5,75**	**42,67**	**641,93%**

Source: author

As can be seen in table 1, as the inclusion of Applitools Eyes adds the function of performing visual checks in the tests, their execution times increase considerably, by an average of 641.93% per test.

When taking into account the computing environment described in table 3, the web application tested (TMDB Web App), the technologies used (JavaScript and Jest) and the free Applitools plan, these results may change.

Execution limits

Regarding execution limits, when using the Selenium WebDriver tool there is no execution limit, since it is a tool that only has local execution and since it is an open source tool.

Applitools Eyes requires an authentication key linked to execution limits. This is because it is a commercial tool that offers plans for using its functionalities. Considering the plan used in the experiment, the free one, the limit of one hundred screenshots per month is quite strict.

In order to configure the inclusion of Applitools Eyes and carry out a complete execution of the planned tests, 80% of the monthly quota was reached.

5. CONCLUSION

In addition to the features analyzed during the experiments, it was possible to see some of the benefits promised by Applitools (c2022).

By using Applitools Eyes it is possible to validate entire pages with a single screenshot, which means viewing the state of the application for each screenshot taken in the automated tests, checking that the features and Selenium tests are performing as expected. You can also see previews of the other algorithms available for checking the visual state of the application. These are exact, strict, ignore colors and layout. Figure 5 exemplifies this benefit. The highlights in pink are the differences identified between the two versions of the web page, which were caused by the size of the selector content being larger in the more recent version (screenshot of the application on the right), thus moving the buttons to the right. The algorithm used was strict, so it took into account the misalignment of the elements. When applying the preview of the layout algorithm, as exemplified in figure 6, the differences are no longer found since the interface has the same elements.

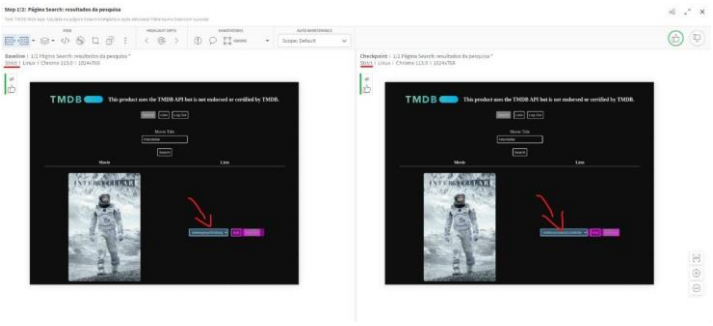

Figure 5 - Example of validating entire pages with a single screenshot Source: authorship

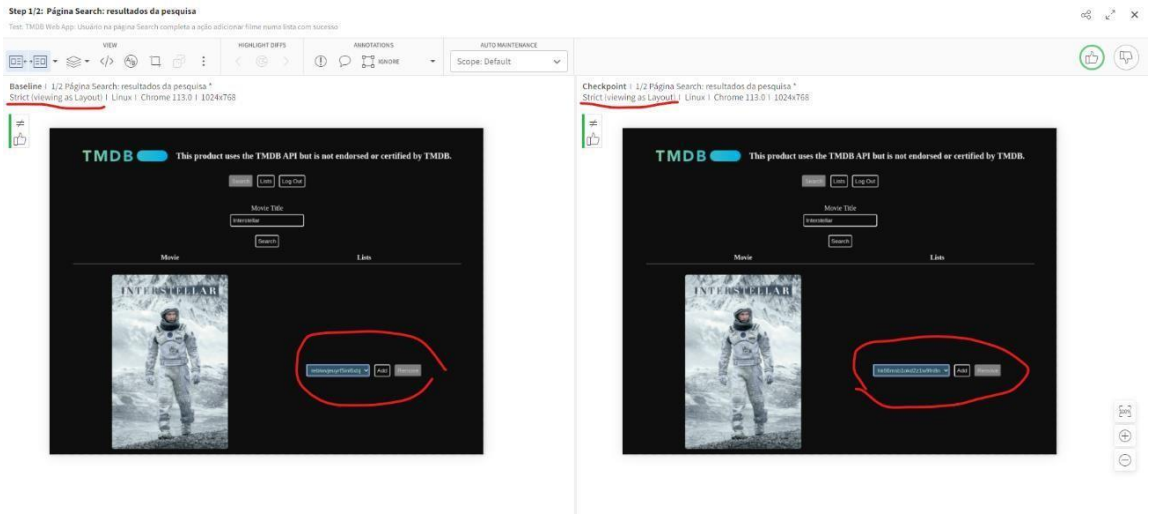

Figure 6 - Example of the preview of different algorithms for validating entire pages with a single screenshot

Source: author

The same algorithms are used to manage dynamic or continuously changing content, especially using the layout algorithm, which checks the structure of the page. With annotations, either by code or in the visualization of test results, it is also possible to use multiple algorithms for different sections of the page. Figure 7 exemplifies this benefit, showing how different algorithms can be added to check sections of the page. The section in gray covering the web application selector shows an annotation to ignore the element and the region with dots that allow resizing demonstrates the addition of the layout algorithm to a check with the strict algorithm.

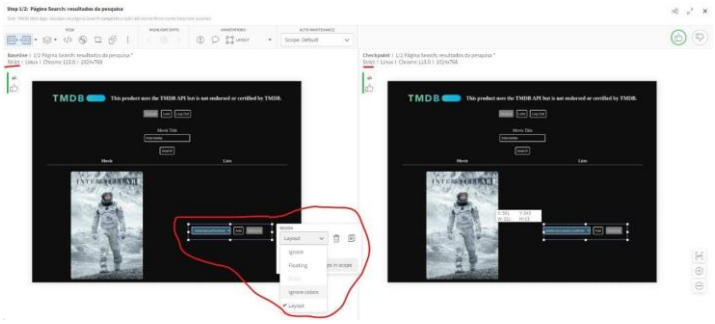

Figure 7. Example of managing dynamic or continuously changing content

Source: author

If necessary, the aforementioned annotations can be used to automate test reviews and maintenance. This means that there is the possibility of running a test via code and, if changes are needed, making changes while viewing the results and allowing these changes to be taken into account for all future tests. By clicking on the Apply to other steps in scope button in figure 8, you can take advantage of this benefit.

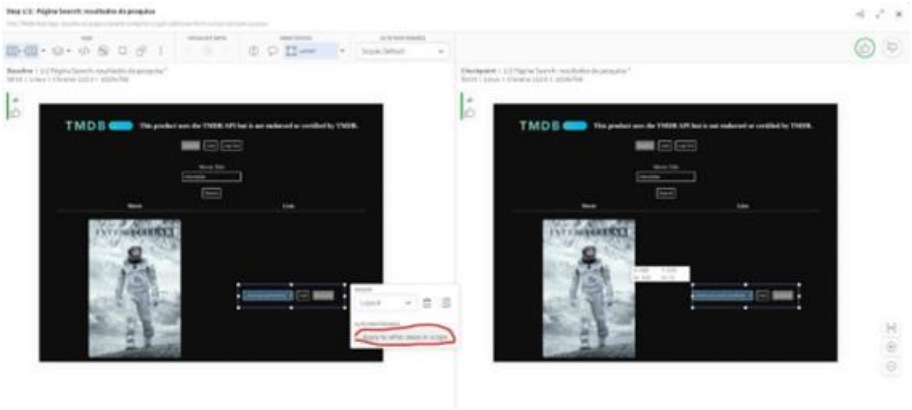

Figure 8. Example of the automation of test reviews and maintenance Source: authorship

By visualizing the result of a test, it is possible to correct errors more quickly with root cause analysis, which shows the reasons why the tests failed according to the algorithm used to check the visual state of the page. Figure 9 exemplifies this benefit: by focusing on the Root cause analysis section, you can see the cause of the pink highlights.

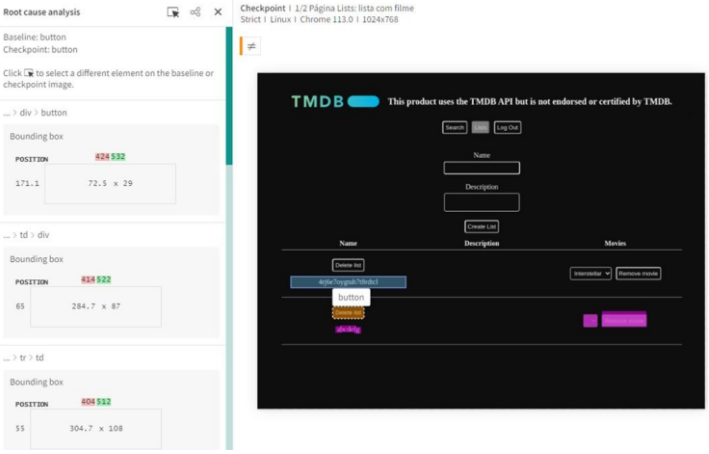

Figure 9. Example of how errors can be corrected more quickly with root cause analysis Source: authorship

During the course of the research, some obstacles were encountered. The first was the fact that the plans for using the first AI tool to be tested, Launchable, were changed and made it impossible to use the AI functionality. This obstacle was overcome by choosing and using another previously researched tool, Applitools Eyes, in conjunction with Selenium WebDriver.

The second obstacle was that difficulties arose in changing the first choice of web application to be tested by the tools, Sharetribe Flex (SHARETRIBE, c2023). This obstacle was overcome by choosing the TMDB Web App.

The third and final obstacle was the limited use of the Applitools Eyes tool, which restricted the tests carried out on the chosen web application.

The expected contributions of this work are to encourage research into Artificial Intelligence applied to Software Testing and to provide information on tools that can improve the software development process.

Future work could include carrying out all the tests proposed and experimenting with all the algorithms available to carry out visual checks. Also, considering the limitations of Applitools Eyes' free plan, tests could be carried out with the different paid plans. Finally, when considering all the software test automation tools that use artificial intelligence, carry out tests with their different plans.

REFERENCES

[1] APACHE SOFTWARE FOUNDATION. Apache JMeter. c2022. Available at: https://jmeter.apache.org/. Accessed on: November 7, 2022.

[2] APPLITOOLS. Applitools. c2022a. Available at: https://applitools.com/. Accessed on: November 7, 2022.

[3] BATTINA, Dhaya Sindhu. Artificial Intelligence in Software Test Automation: A Systematic Literature Review. International Journal of Emerging Technologies and Innovative Research, v. 6, n. 12, p. 1329-1332, dez. 2019. Available at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4004324. Accessed on: November 7, 2022.

[4] BENGIO, Yoshua; GOODFELLOW, Ian; COURVILLE, Aaron. Deep Learning. Cambridge: MIT press, 2015.

[5] BOEHM, Barry W. Software Engineering as it is. Proceedings of the 4th International Conference on Software Engineering. Munich: IEEE Press. 1979. p. 11-21. Available at: https://www.inf.ed.ac.uk/teaching/courses/seoc1/2005_2006/resources/bullet03.pdf. Accessed on: November 7, 2022.

[6] CAPGEMINI. World Quality Report, 2020. Available at: https://www.capgemini.com/insights/research-library/world-quality-report-2019-20/. Accessed on: November 7, 2022.

[7] CUNHA, Gabriel C. da. GitHub: repositório TMDB Web App. 2022. Available at: https://github.com/gccunha015-dio/tmdb_web_app. Accessed on: November 7, 2022.

[8] CUNHA, Gabriel C. da. GitHub: repositório TMDB Web App com testes. 2023. Available at: https://github.com/gccunha015-upm/tmdb_web_app. Accessed on: November 7, 2022.

[9] ENLYFT. Enlyft: Software Testing Tools products. c2023. Available at: https://enlyft.com/tech/software-testing-tools. Accessed on: November 7, 2022.

[10] GAMIDO, Heidilyn V.; GAMIDO, Marlon V. Comparative review of the features of automated software testing tools. International Journal of Electrical and Computer Engineering, v.9, n.5, p.4473-4478, out.2019.

Available at: http://download.garuda.kemdikbud.go.id/article.php?article=1304601&val=146&title=Comparative%20Review%20of%20the%20Features%20of%20Automated%20Software%20Testing%20Tools. Accessed on: November 7, 2022.

[11] KATALON. Katalon. c2022. Available at: https://katalon.com/. Accessed on: November 7, 2022.

[12] KING, Tariq. AI-Driven Testing. Sebastopol, CA, USA: O'Reilly Media, Inc., 2021. Available at:

https://www.oreilly.com/library/view/ai-driven- testing/9781098105983/. Accessed on: November 7, 2022.

[13] LAUNCHABLE. Launchable. s.d. Available at: https://www.launchableinc.com/. Accessed on: November 7, 2022.

[14] MITCHELL, Tom M. Machine Learning. New York, NY, USA: McGraw-Hill, 1997.

[15] MURPHY, Kevin P. Machine Learning: A Probabilistic Perspective. Cambridge, MA, USA: MIT Press, 2012.

[16] MYERS, Glenford J.; BADGETT, Tom; SANDLER, Corey. The Art of Software Testing. 3ª ed. New Jersey: Wiley & Sons, 2011.

[17] NORVIG, Peter; RUSSEL, Stuart. Inteligência Artificial. 3ª ed. Rio de Janeiro: Elsevier Editora, 2013.

[18] PRESSMAN, Roger S.; MAXIM, Bruce R. Engenharia de Software. 9ª ed. [S.l.]: Grupo A, 2021.

[19] SHARETRIBE. Sharetribe. c2023. Available at: https://www.sharetribe.com/. Accessed on: November 7, 2022.

[20] SINGH, Inderjeet; TARIKA, Bindia. Comparative analysis of open source automated software testing tools: Selenium, sikuli and watir. International Journal of Information & Computation Technology, v. 4, n. 15, p. 1507-1518, 2014. Available at: http://www.ripublication.com/irph/ijict_spl/ijictv4n15spl_04.pdf. Accessed on: November 7, 2022.

[21] SOFTWARE FREENDOM CONSERVANCY. Selenium. c2023. Available at: https://www.selenium.dev/. Accessed on: November 7, 2022.

[22] SOMMERVILLE, Ian. Engenharia de Software. 10ª ed. São Paulo: Pearson Education do Brasil, 2019.

[23] THE MOVIE DATABASE. The Movie Database (TMDB). s.d. Available at: https://www.themoviedb.org/. Accessed on: November 7, 2022.

Citation: Ankit Chandankhede, Leveraging AI for Automated Functional Coverage and Test Sequence Generation in GPU and AI Accelerator Verification, International Journal of Electrical Engineering and Technology (IJEET). 14(6), 2023, pp. 16-30

Article Link:
https://iaeme.com/MasterAdmin/Journal_uploads/IJEET/VOLUME_14_ISSUE_6/IJEET_14_06_003.pdf

Abstract Link: https://iaeme.com/Home/article_id/IJEET_14_06_003

✉ **editor@iaeme.com**