QUALIZEAL

Role of AI

In functional testing

"AI can help automate repetitive testing tasks, enabling testers to focus on more complex and creative testing activities."

-Gartner

MarketsandMarkets; The AI in software testing market is projected to grow from $284 million in 2020 to $1.4 billion by 2025, at a CAGR of 38.8% during the forecast period.

Functional Testing is a crucial phase of software testing that entails an extensive and meticulous validation methodology to ascertain the adherence of a software system to its functional requirements and specifications. This testing methodology primarily focuses on validating each discrete function of the software application by providing appropriate input and verifying the output against the specified functional requirements.

Functional testing relies predominantly on black box testing techniques and is agnostic of the application's source code. This methodology scrutinizes various facets of the Application Under Test (AUT), such as User Interface (UI), Application Programming Interfaces (APIs), Database, Security, Client/Server communication, and other functional aspects of the software system.

In a study conducted by IBM, it was found that AI-powered functional testing was able to identify defects with an accuracy of 95%, compared to 60% accuracy for manual testing.

The testing process can be executed manually, which is a time-consuming, expensive, and laborious process, or through automation, which leverages a variety of testing frameworks and tools. Automation helps optimize the testing process, improves test coverage, and reduces the overall testing time and cost. However, automation requires considerable investment in infrastructure, tools, and skilled personnel.

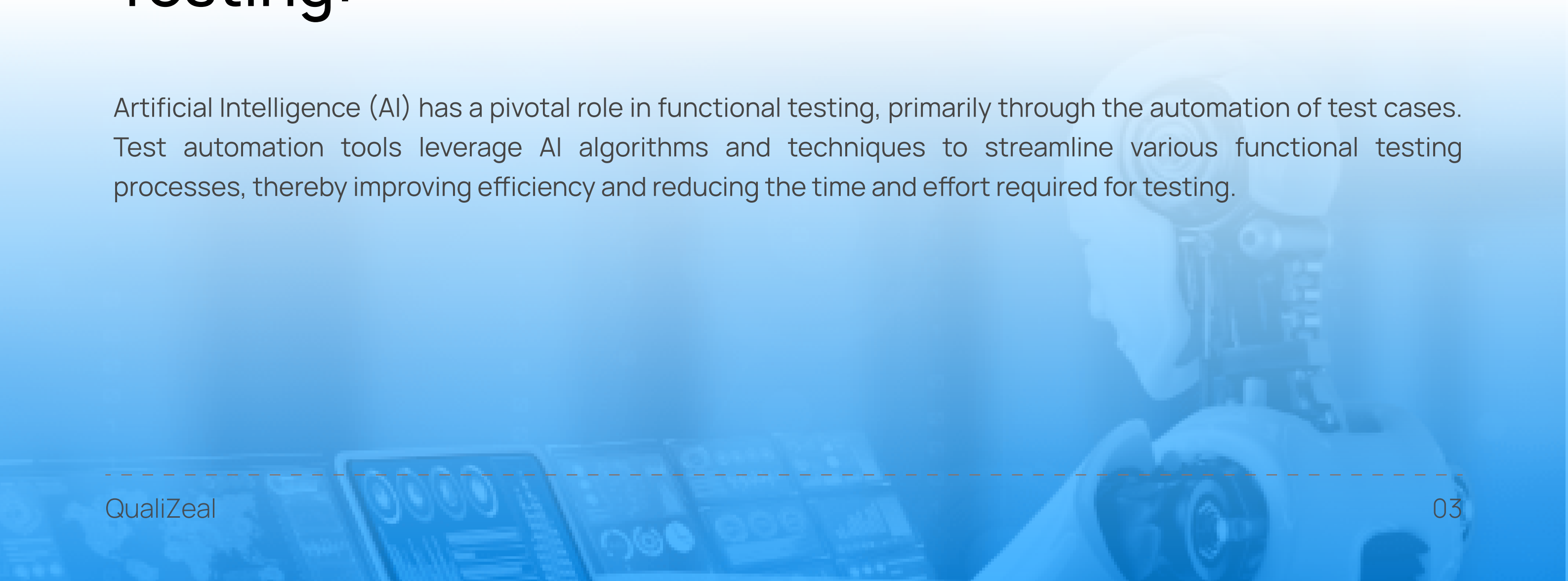
Ultimately, the choice between manual and automated testing depends on various factors, such as the complexity of the software system, the size of the development team, and the available budget and resources. Artificial intelligence (AI) is highly compatible with functional testing as it automates test execution, improves test data management, and enhances the overall quality of testing.

Functional vs Non-Functional Testing:

Parameter	Functional Testing	Non-functional Testing
Objective	Verify functional requirements and specifications	Verify non-functional attributes such as performance, security, reliability, usability, and compatibility
Focus	Functionality	Performance, Security, Usability, Compatibility, Reliability
Methodology	Black box testing	White box testing, Grey box testing
Scope	Unit testing, Integration testing, System testing	Load testing, Stress testing, Usability testing, Compatibility testing
Test Cases	Test cases are based on functional requirements and use cases	Test cases are based on non-functional requirements and use cases
Output	Output is compared against the expected functional requirements and specifications	Output is compared against non-functional requirements and specifications
Tools and Techniques	Automation and manual testing tools, Regression testing, User Acceptance Testing	Performance testing tools, Security testing tools, Usability testing tools, Load testing tools

Role of Artificial Intelligence in Functional Testing:

Artificial Intelligence (AI) has a pivotal role in functional testing, primarily through the automation of test cases. Test automation tools leverage AI algorithms and techniques to streamline various functional testing processes, thereby improving efficiency and reducing the time and effort required for testing.



AI-based test automation tools use complex algorithms such as machine learning, natural language processing, and image recognition to automate the creation and execution of test cases. These tools can analyze large datasets, identify patterns, and generate test cases with minimal human intervention.

Test automation can be categorized into two types: script-based and scriptless. Script-based test automation involves the creation of scripts that automate test execution, while scriptless automation uses AI to generate tests automatically.

Generate & Update Unit Test:

In the field of DevOps, unit testing plays an integral role in continuous testing, and continuous integration/ continuous delivery (CI/CD). However, it is commonly acknowledged that authoring and maintaining unit tests can be a tedious and laborious process, often requiring substantial time and effort from developers. To address this challenge, AI-based products that leverage machine learning for automated unit test creation have emerged, providing a promising solution for organizations seeking to introduce unit tests late in the product life cycle.

Advantages -

- AI-based automated unit tests are a significant advancement over template-based automated unit test generation, utilizing both static and dynamic analysis. The resulting tests are genuine code rather than mere stubs.
- Automated unit tests generated through machine learning can be rapidly created, making them well-suited for large and complex codebases.
- Developers can modify these automated tests as needed and set up the unit regression suite with relative ease.

Limitations -

- As AI-generated unit tests are constructed based on the code itself, they cannot intuit the intended functionality of the code. As such, if the code behaves unexpectedly, the generated unit test will reflect that behavior, which is counterproductive for ensuring and verifying an implicit or explicit contract.
- Automated unit tests generated through machine learning can potentially break existing unit tests that are functioning correctly. Therefore, developers need to take measures to ensure that these tests are properly integrated and avoid any unwanted negative impacts.
- Developers still need to write tests for complex business logic themselves, as these automated tests may not cover every possible scenario.

User Interface Testing:

The amalgamation of artificial intelligence with User Interface (UI) testing has emerged as a promising field. AI-based UI testing leverages test automation tools to perform accurate analysis of the Document Object Model (DOM) and relevant code to establish object properties. Moreover, it employs sophisticated image recognition techniques to navigate through the application and validate UI objects and elements visually, which is then used to generate UI tests.

AI recognizes patterns, and one popular pattern is image-based testing using automated visual validation tools. By using AI-based visual validation tools, testers can find differences that human testers would likely miss. This testing activity makes manual testing a perfect fit for AI testing.

Furthermore, AI-based test systems utilize exploratory testing to uncover inconsistencies or deviations in the application UI and create screenshots for future verification by a Quality Assurance (QA) engineer. Additionally, the visual facets of the System Under Test (SUT), such as layout, size, and color, can be authenticated.

Benefits -

- AI-powered UI testing can facilitate enhanced code coverage.
- AI models are designed to handle minor deviations in UI that do not result in test suite failure.

Limitations -

- The number of platforms, app versions, and browser versions available for modern applications is substantial, making it uncertain how effective AI-based UI automation will be in such complex environments. However, the capability of cloud testing tools to perform tests in parallel makes this area worth monitoring.

Automated API Testing:

Automating API testing is a complex task as it requires a deep understanding of the API and meticulous planning to ensure comprehensive test coverage. Traditional API test automation tools rely on recording API activities and traffic to create tests, which require manual modification and updating for changes in REST calls and parameters.

AI-based API automation testing tools leverage machine learning techniques to mitigate this issue by examining traffic and identifying patterns and connections between API calls, effectively grouping them by scenario. These tools also use existing tests to learn about relationships between APIs, use these to understand changes in APIs, and update existing tests or create new scenario-based tests.

Benefits -

- AI-based automation testing tools can significantly reduce setup time and complexity, especially for novice testers or those without programming experience.
- These tools can also facilitate change management by automating API test updates, freeing up time and effort for manual testing of new scenarios.

Limitations -

- API testing is inherently challenging and requires a deep understanding of the API, making it difficult for AI-based automation tools to accurately identify scenarios and associated API calls.
- Currently, there are limited options available for machine learning-based API testing, and the available tools have rudimentary capabilities.

Automation in Test Maintenance:

AI-based tools are becoming increasingly popular in automating test maintenance. They can evaluate changes made to the code and modify existing tests that no longer align with those changes. These tools are especially effective when the code changes are not too complex. Updates to UI elements or field names no longer need to break the test suite.

AI-Powered Smart Assistant for Precision Testing:

Companies that use continuous integration and continuous testing generate a wealth of data from their test runs. However, it is time-consuming to sift through the data to look for common patterns over time. With ML, AI tools can answer a testing question: What is the minimum number of tests to run to determine whether a change is good or bad? These tools can analyze test coverage and flag areas at risk. For example, a team implemented an ML algorithm to correlate system and debug logs to establish a "fingerprint" of test case failures. This allowed the team to focus their efforts on new test failures.

AI-Powered Spidering for Automated Testing:

ML enables automatic test writing for applications by spidering. New AI/ML tools can automatically crawl the application, collect data on features, take screenshots, download HTML, and measure load times. Over time, the ML models learn the expected patterns of the application. If the tool detects any deviation, visual difference, or running slower than usual, it flags it as a potential issue. Although still in its infancy, this approach may eventually lead to the automatic authoring of tests, helping testers understand which parts of the application should be tested. Ultimately, a human tester needs to validate the flagged issues with domain knowledge of the application.

Test Data Management:

Another promising application of AI in software testing is test data generation. Machine learning can be used to generate data sets that are similar to production data, making them ideal for use in software testing. This is accomplished by using existing production data sets to train machine learning models, which then generate data that is similar to the training data.

The machine learning model that is used for generating data is called a Generative Adversarial Network (GAN). By using GANs, software testers can generate large data sets for testing, saving significant amounts of time and resources.

AI-Powered Tools Used to Automate Functional Testing:

Cypress

Cypress is a JavaScript-based testing framework that facilitates automated test script creation. Its framework-agnostic nature eliminates the need for rewriting new tests when the application is shifted to a different framework. This is possible due to its ability to communicate with different testing libraries and frameworks.

Selenium

Selenium, an open-source automated testing suite, is one of the most preferred choices for web application testing. Selenium is composed of several components, each playing a specific role in aiding test automation. The framework supports multiple test system environments such as Windows, Mac, Linux and several browsers such as Chrome, Firefox, and Internet Explorer.

Puppeteer

Puppeteer, an open-source node js library, is used to automate and streamline front-end development and testing. It contains APIs to interact with and manage Chrome browser in headless mode, but it can also be utilized for non-headless execution on browsers such as Chrome and Firefox.

WebdriverIO

WebdriverIO is an automation framework built to automate modern web and mobile applications. It simplifies the interaction with the app and provides a set of plugins that help create a scalable, robust, and stable test suite. WebdriverIO leverages the power of the WebDriver protocol that is developed and supported by all browsers, ensuring an authentic cross-browser testing experience.

NightwatchJS

NightwatchJS is an integrated, easy-to-use end-to-end testing solution for web applications written in NodeJS. It offers a built-in command-line test runner that runs the tests sequentially or in parallel with retries and implicit waits. NightwatchJS also works seamlessly with BrowserStack out of the box.

From Accuracy to Scalability: Maximize Testing ROI through Intelligence:

The ever-evolving domains of Artificial Intelligence (AI) and Machine Learning (ML) have emerged as a panacea for narrowing the testing gaps that plague software systems and have proven to be most effective when they are synergistically augmented by human expertise in real-time data handling.

As per the "Benefits of Automation & AI in Functional Testing" report, authored by Isaac Sacolick, a renowned technologist and founder of StarCIO, AI and ML-powered testing tools can significantly enhance the quality assurance (QA) process, especially when dealing with complex issues that are difficult to detect through manual or automated testing.

AI's Business Benefits:

A recent survey of 200 IT executives found that almost all of them are experimenting with AI capabilities in testing. The vast majority see the commercial benefits of using AI in Quality Assurance (QA), with areas such as identifying anomalies, computer vision, and natural language processing seen as having the most promise.

As engineers don't have the time or expertise to test thoroughly by hand, more companies are turning to AI for testing, recognizing that the tools have improved significantly over the last five years.

Platform Coverage as a Barrier:

Testing on multiple platforms is a common pain point, with a lack of platform coverage a significant obstacle to test automation. Only 22% of respondents claimed they could develop a single test case that could run on each platform without modification, while 77% said they needed to write separate tests for each platform. There was a significant gap between the CXO level and the teams that truly understands what's going on, with CXOs often not fully aware of what developers are doing.

AI Lowers the Barrier to Automation:

AI can also lower the barrier to entry for people who wish to automate. It is easier to implement and does not require as much subject knowledge as traditional automation, meaning extra testers can be enlisted in the automation process. Additionally, developers can participate in automation, as they know what they programmed and what the product is, but they don't need to know how to automate.

A shift-left testing approach, involving automating more testing throughout the development process, and expanding the scope of testing, can lead to improved customer happiness and app quality without fear of damaging anything. Using machine learning and AI in testing allows teams to uncover more complicated quality concerns.

Make the Transition to Automated Functional Testing:

Identify the Testing Goals:

The initial step of a functional testing plan is to determine the testing goals. These goals are based on the software's expected features in accordance with project requirements. Functional testing goals include validating the software's proper functioning and verifying that it handles errors and unexpected scenarios gracefully.

Create Test Scenarios:

The next step is to develop a comprehensive list of all the test scenarios for a particular feature. These test scenarios depict the various ways the feature will be used. For example, for a payment module, the test scenarios may include multiple currencies, handling invalid or expired card numbers, and generating a notification upon successful transaction completion.

Create Test Data:

After the test scenarios are determined, the next step is to create test data that simulates normal use conditions based on the identified test scenarios. The test data can be entered manually, such as through an MS Excel spreadsheet or printout, or automatically via a script or test tool that inputs the data from a database, flat file, XML, or spreadsheet. Each set of input data should have associated data that describes the expected result that the input data should generate.

Design Test Cases:

The next step is to design test cases based on the different desired outcomes for the test inputs. For example, if an invalid credit card number is entered, the application should display a meaningful error message. The test cases should be designed to verify the expected output based on the input data.

Execute the Test Cases:

The designed test cases should then be executed through the application, and actual outcomes should be compared against the expected results. If the actual and expected outputs differ, the feature has failed the test, and a defect should be recorded.

Deliberate On, Track, and Resolve Defects:

In case a defect is identified, it should be recorded on a formal tracking system accessible to the entire project team. The application should then be modified accordingly, and the test case should be executed again to confirm resolution before a defect is marked as closed.

Challenges Associated with AI Implementation in Functional Testing:

Identify the Testing Goals:

The initial step of a functional testing plan is to determine the testing goals. These goals are based on the software's expected features in accordance with project requirements. Functional testing goals include validating the software's proper functioning and verifying that it handles errors and unexpected scenarios gracefully.

Create Test Scenarios:

The next step is to develop a comprehensive list of all the test scenarios for a particular feature. These test scenarios depict the various ways the feature will be used. For example, for a payment module, the test scenarios may include multiple currencies, handling invalid or expired card numbers, and generating a notification upon successful transaction completion.

Create Test Data:

After the test scenarios are determined, the next step is to create test data that simulates normal use conditions based on the identified test scenarios. The test data can be entered manually, such as through an MS Excel spreadsheet or printout, or automatically via a script or test tool that inputs the data from a database, flat file, XML, or spreadsheet. Each set of input data should have associated data that describes the expected result that the input data should generate.

Design Test Cases:

The next step is to design test cases based on the different desired outcomes for the test inputs. For example, if an invalid credit card number is entered, the application should display a meaningful error message. The test cases should be designed to verify the expected output based on the input data.

Execute the Test Cases:

The designed test cases should then be executed through the application, and actual outcomes should be compared against the expected results. If the actual and expected outputs differ, the feature has failed the test, and a defect should be recorded.

Deliberate On, Track, and Resolve Defects:

In case a defect is identified, it should be recorded on a formal tracking system accessible to the entire project team. The application should then be modified accordingly, and the test case should be executed again to confirm resolution before a defect is marked as closed.

No Goal is Attainable Without a Compelling Strategy:

Define the scope and goals:

Clearly define the scope of AI in functional testing and the goals that you want to achieve. Identify the areas of testing that are most suitable for automation and the types of tests that can benefit the most from AI.

Develop a strategy:

Develop a strategy for implementing AI in functional testing. This should include a roadmap that outlines the steps to be taken, the timelines, and the resources required. The strategy should also consider the impact on existing processes and tools.

Identify the right tools and technologies:

Identify the right tools and technologies that can support AI in functional testing. This includes tools for test automation, test data management, and AI algorithms. Evaluate these tools based on their features, ease of use, and scalability.

Build a team with the right skills and expertise:

Build a team with the right skills and expertise to implement AI in functional testing. This includes data scientists, machine learning experts, and software developers. Ensure that the team has a deep understanding of the testing process and the software being tested.

Implement and continuously improve:

Implement AI in functional testing and continuously improve the process. Monitor the results of AI testing and identify areas for improvement. Use feedback from testers and other stakeholders to refine the process and optimize the use of AI.

Role Of AI

As AI and Machine Learning continue to evolve and improve, they are poised to revolutionize the entire software testing process.

One major advantage of using AI is its ability to automate complex testing scenarios that were previously impossible in traditional approaches. Moreover, AI-powered testing provides deeper insights into the system performance, detecting issues that may have been missed by manual or traditional testing methods.

QualiZeal's expertise in AI and Machine Learning is well-positioned to meet such demand, enabling organizations to achieve their strategic objectives and stay ahead of the curve.

We ensure that the AI-powered testing approach is ethical, transparent, and meets the highest quality standards. The algorithms we use are also unbiased, accurate, and reliable that comply with all relevant regulations and guidelines.

- QualiZeal collects and prepare high-quality data for AI and Machine Learning algorithms. Our team of experts can assist you in identifying and extracting the most relevant data sets and cleaning and pre-processing them for training and validation purposes.

- QualiZeal develops an effective testing strategy for AI aligned with your business goals and objectives. Our quality engineers are trained to identify the most critical testing scenarios and design AI-enabled test cases to ensure comprehensive and accurate testing.

- QualiZeal set up the required infrastructure for AI-powered testing, including the selection and integration of the necessary tools and technologies. We even provide guidance on the optimal configuration and deployment of these tools to maximize their effectiveness.

- QualiZeal offers training and skill augmentation services to empower your in-house capabilities in AI-powered testing. Our training comprises the latest AI and Machine Learning modules that help in developing a skilled workforce capable of handling advanced testing components.