



Artificial Intelligence in Test Automation: The Next Frontier

GlobalLogic Research | 2020-21
by Prageet Pathak, Parvendra Singh

Contents

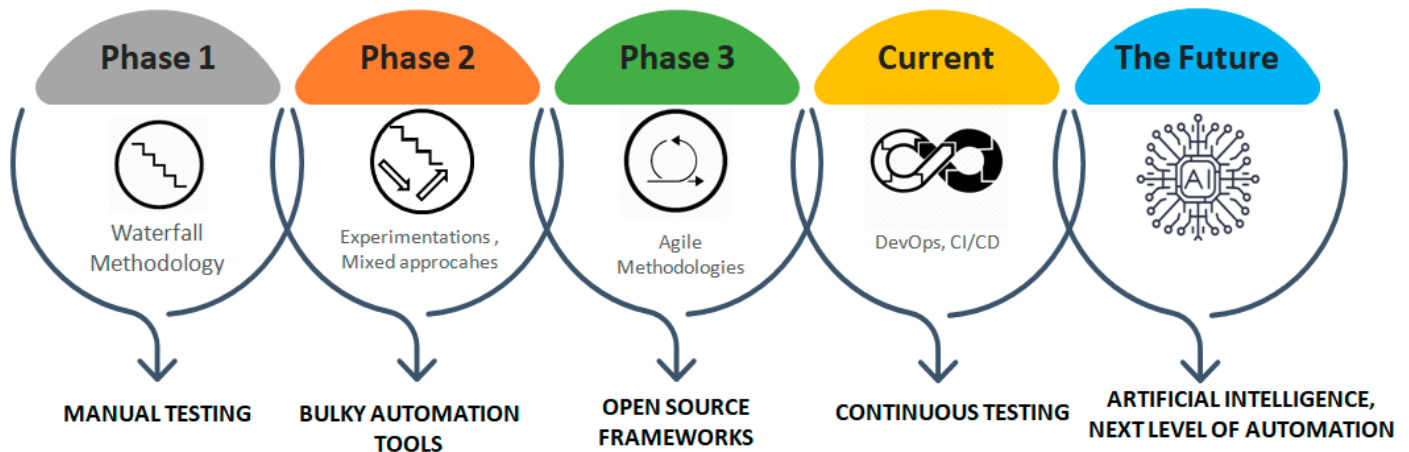
Introduction	1
Challenges with Traditional Automation	2
Challenges in Test Planning Phase	
Challenges in Test Authoring Phase	
Challenges in Test Maintenance Phase	
An Overview of AI and Machine Learning	5
Leveraging AI and ML in Test Automation	6
Changing the way we identify our web elements	
Self-Healing Test Scripts	
Replace Repetitive tasks	
Smart Regression	
Visual/Snapshot Testing	
Other Uses of AI in Test Automation	12
Tools Leveraging AI in Test Automation	13
Conclusion	14

Introduction

“Success in creating AI would be the biggest event in human history. Unfortunately, it might also be the last, unless we learn how to avoid the risks.”

— Stephen Hawking, famous theoretical physicist, cosmologist, and author

We have come a long way from the time Automation of Test Cases was introduced in the QA industry with some bulky tools. Today we have more robust open-source frameworks and integration with devOps to provide continuous testing. As we have moved from the traditional Waterfall Model of software development to agile models with quick deliveries, testing strategies evolved to support the lifecycle and provide quick feedback at every stage of development.



Now that continuous integration and continuous deployment has become the bare minimum for every software development project (with releases going out every week), our testing teams are always on a time crunch to complete testing cycles.

The first solution for this problem is automation, but automation takes a lot of time. The main problems with automation are identifying the scenarios and coding, and maintaining the automation code against the new changes. These challenges can be solved by using AI in automation. AI can be used for writing test cases, generating data sets, analyzing the test results, etc. In other words, AI can make our automation smarter.

The purpose of this paper is to help solve the challenges of automation testing using artificial intelligence and machine learning. Let's first discuss the challenges of automation testing.

Challenges with Traditional Automation

Current Test Automation tools and practices have evolved to help with testing needs, but they are still bound by certain limitations.

In the list below, we have highlighted some of those challenges and categorized them into different phases of test automation life cycle.

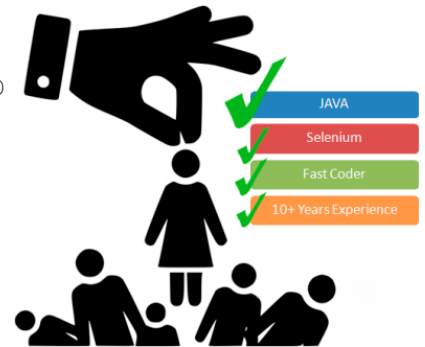
Challenges in Test Planning Phase

During the planning phase of Test Automation, it can feel difficult to start a new project quickly. We are forced to go through certain steps or processes that can very likely be avoided with the next-gen automation practices we will discuss in this document.

- **Finding team members with required skill set**

Current automation tools require team members to have a certain skill set to start creating test scripts. This very often becomes an obstacle to setting up a new team quickly.

Various next-generation Test Automation tools have incorporated artificial intelligence to provide testers a semi or fully codeless scripting platform to overcome this challenge. Using these tools, team members of any skill set can start contributing to automation with minimal effort.



- **Finding the right automation tool for the project**

There are numerous automation tools and models popular in the market. Finding the best one for your needs is an important task for every team.

An AI algorithm or tool can come to our rescue here as well. Such a system can be trained with previous project data and outcomes to help us predict a suitable automation model for the project we are about to start.



Challenges in Test Authoring Phase

Once we have gone through all the planning steps and moved on to writing test scripts, the next set of challenges begins.

- **Starting from scratch (doing repetitive tasks in every new project)**

Starting our day with the same routine chores might not trouble us. When we are forced to do that in our work, it will be frustrating even for the most patient amongst us. Automation and devOps practices in use nowadays have saved us from a large number of routine testing tasks. However, when it comes to starting a new test automation project or writing a new test suite, no matter how reusable the components are, we still end up writing a lot of similar code again.

For example, in every web automation project, we write page classes or create data models for our web forms. Imagine a world where these are already available for our application. It would be easy and quick to start automating our test cases! This is another area where we can utilize artificial intelligence to automatically generate a skeleton automation to start test script authoring quickly.

Challenges in Test Maintenance Phase

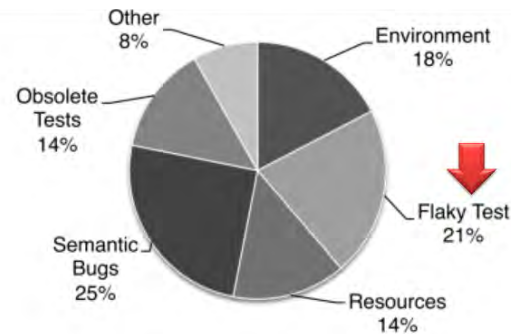
The Test Automation phase, where we feel traditional automation techniques are underused the most, is when it comes to maintaining existing test scripts. A lot of improvement is expected with artificial intelligence incorporated into next-generation automation tools.

Below is a list of a few of the challenges every test automation engineer faces while maintaining test scripts during daily regression runs or application updates.

- **Flaky tests**

Very often we have to spend hours to determine if a failing test is an issue in the application or just a random failure of a poorly written test case. These deterministic and inconsistent test failures are referred to as flaky tests. These are capable of holding back a release for no good reason and are a real challenge for every QA team.

Another aim of next generation automation tools is to help us write more robust test cases, and enable us to find patterns in random test failures so we can take the necessary actions.



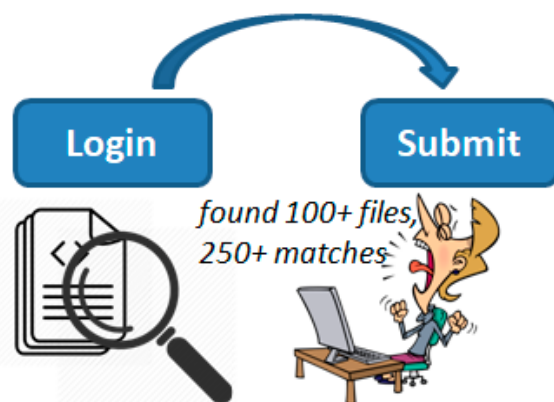
(a) Distribution based on bug categories.

Ref: A. Vahabzadeh, A. M. Fard and A. Mesbah, "An empirical study of bugs in test code," 2015 IEEE International Conference on Software Maintenance and Evolution (ICSM), Bremen, 2015, pp. 101-110, doi:10.1109/ICSM.2015.7332456.

- **Script updates with every minor UI change**

In the process of improving customer experience, web applications often undergo minor UI changes. Although the changes are hardly noticeable to a human performing a transaction on the application, it would normally fail our test scripts while performing some actions on the page. Even a one-line change in frontend code can cost multiple hours of test automation code updates.

A test automation framework equipped with algorithms powered by artificial intelligence and machine learning can eliminate human intervention in script updates for such minor changes. Models can be trained in such a way that they are able to distinguish between a minor UI change and a bug in the application, and then take further action accordingly.



- **Determining test size to ensure quality of a change**

Changes are an inevitable part of making any software application better. While every change brings more value to the end-user, it comes with an overhead of regression testing. In a fast-paced development environment, it is not always possible to run an entire regression suite of test cases for every change. Rather, we have to be selective.

Determining the right size and set of test cases to be run for a particular change has always been a challenge to QA teams. Artificial intelligence can help us to build algorithms that can predict the regression test suite for the given change based on different parameters.



- **Maintaining different test suites/scripts**

A test automation repository often consists of many different test suites, defined to collect a set of test cases to be run on different occasions. These include a change in a specific functional area, sanity test, smoke test, regression test, sprint closure, etc., to name just a few.

As the size of an application grows, maintaining a large number of test scripts and different test suites becomes a challenge unless a strict process has been followed throughout.

We can defer this job to a next-generation automation framework to maintain and extract the required test scripts as per the testing requirements.



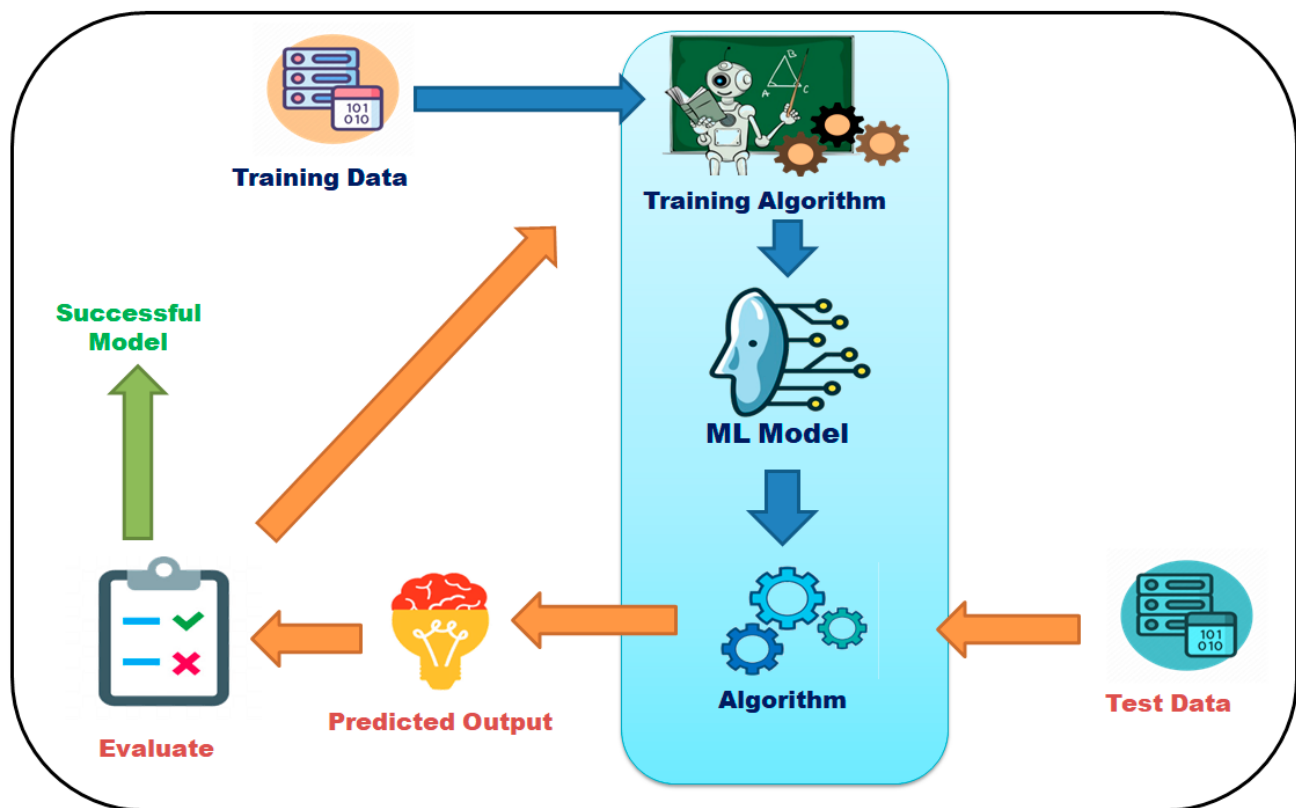
An Overview of AI and Machine Learning

Artificial intelligence (AI) refers to the capability of machines to perform functions that mimic a human mind, such as learning and problem-solving. AI should not always be associated with futuristic robots or machines that look like humans. Any device or software that can perceive its surroundings and make the correct decisions to achieve its goal can be said to have some level of artificial intelligence.

Machine learning (ML) is a sub-field of AI and is the science of getting computers to act without being explicitly programmed (Stanford 2018). It provides systems with the ability to learn from data or experiences and continuously improve themselves.

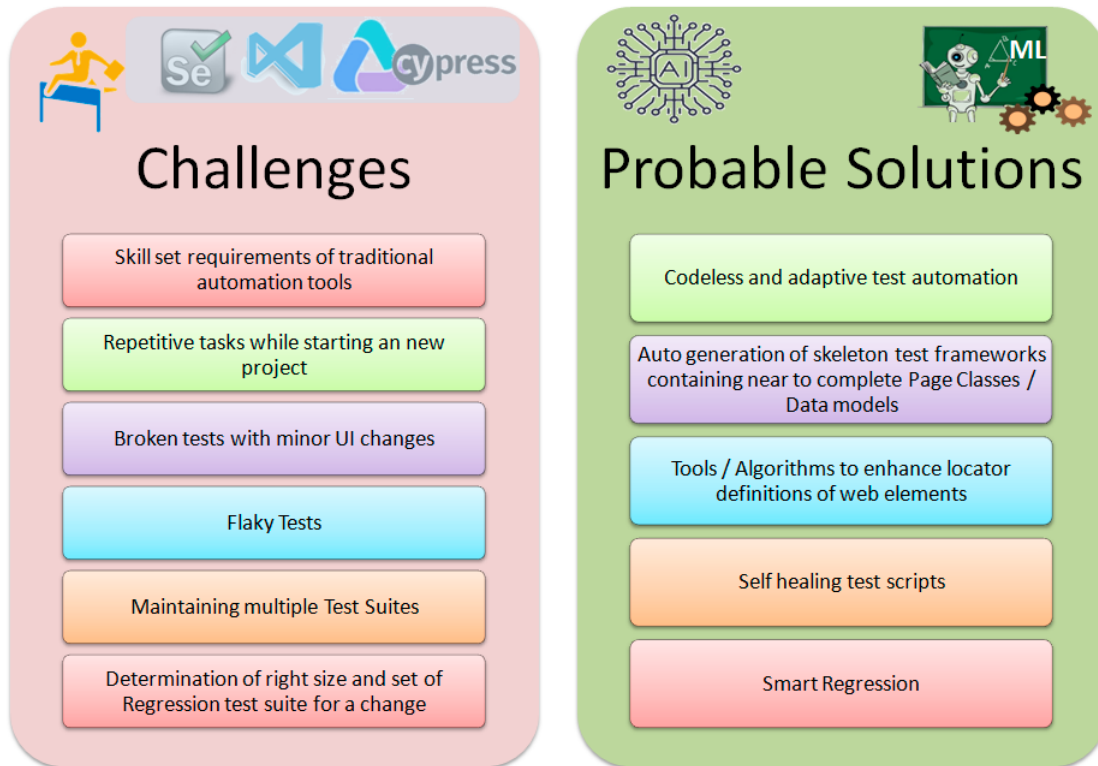
Two major applications of ML are referred to as unsupervised learning and supervised learning.

- **In unsupervised learning**, unlabeled data is fed into a training algorithm with the goal of discovering patterns and relationships. One example is clustering algorithms that attempt to organize data points into groups.
- **Supervised learning**, in contrast, involves human-labeled training data. A supervised learning workflow typically involves multiple iterations of constructing labeled-training data, extracting features from the data, choosing an algorithm, training a model, evaluating the model using test or input data, and improving the model. The diagram below portrays this workflow.



Leveraging AI and ML in Test Automation

We have addressed various challenges with currently used Test Automation techniques. Now let's take a look at how we can leverage different AI or machine learning algorithms to overcome those limitations and further enhance our Automated Testing.



Changing the way we identify our web elements

Defining an efficient and robust locator for web elements is a key contributor towards stable Test Script. In any automated regression suite we observe a large number of failed tests due to failure to locate desired elements.

This has become more frequent with the increased use of rapid web development frameworks. Nowadays, most web applications are built on top of these frameworks, which provide a quick and easy way for developers to generate content-rich and dynamic web pages. On the one hand, this is a boon to developers. However, it's also a nightmare to test automation engineers because the more dynamic the content on a web page is, the more dynamic Ids and other properties will be in the HTML DOM.

Writing test scripts to automate actions on these web elements involves more vigilant locator definition and extra time if done manually. There are multiple element-locating tools available to make it easier for us. Unfortunately, the robustness of these auto-generated locators is hard to rely on for many. Most traditional tools rely on the single algorithm they were built on and tend to generate frail element locators when used for such a dynamic web page.

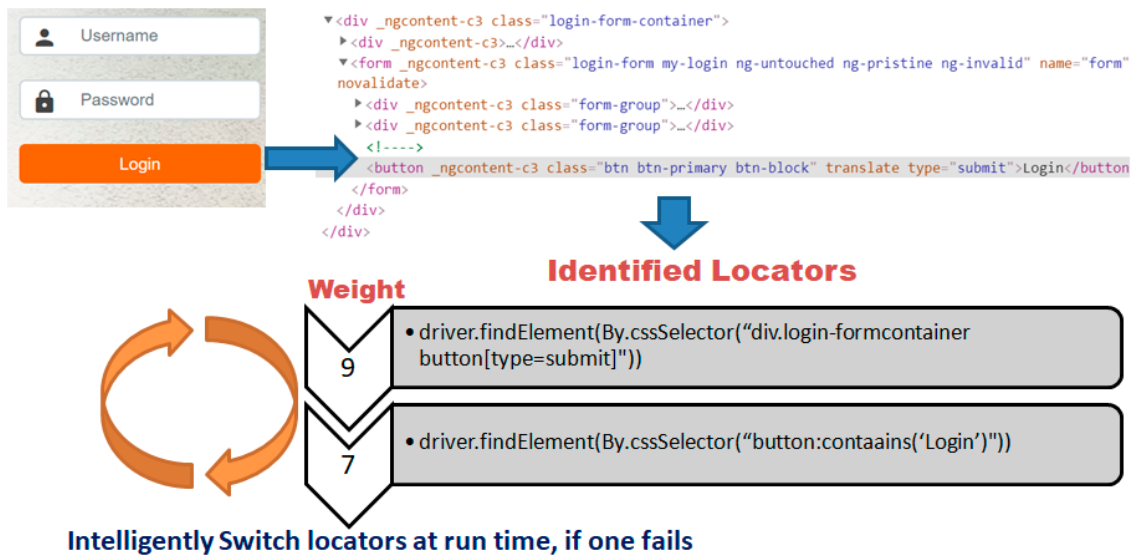
The use of machine learning and other artificial intelligence algorithms in combination can improve this to a much greater extent than we had previously seen. A simple example is a model trained with a large data set from different websites, which uses a weighted set of locator algorithms rather than relying on just one.

Below are a few of the various approaches that are leveraging artificial intelligence to take Test Automation of Web Applications to the next level.

- **Using a weighted set of locators**

As briefed above, relying on a single locator to identify web elements in interactive and dynamic web applications may lead to flaky test scripts. This often takes up a lot of our time just to maintain regression test scripts for random test failures and minor UI changes.

The use of artificial intelligence can take us a step closer to finding a perfect solution to this problem. An element-locating tool can be developed that is trained with not just a huge data set comprising different websites and locators of their web elements, but also with a set of different algorithms to generate locators to be used in our test scripts. The outcome of using these tools for web elements could be a list of locators with an associated weight. Test automation tools can then be introduced to an artificial intelligence that is able to switch from one locator to another from the list at runtime to provide a more reliable and robust test framework.



- **Visually locating an element**

AI can locate elements without consulting DOM properties. Instead, it uses a visual representation, similar to what any user would do (e.g. icons - My Account, play, pause, mute, cart, and settings). AI models can be trained with a large number of known icons and then be ready to automatically identify if any similar image, button or icon is present on the web page. In the test scripts, users can simply mention which type of element is targeted instead of providing locators based on HTML DOM properties. This can be as simple as saying, "Click on My Account." An AI-powered automation tool will automatically find where the My Account icon is on the web page and perform user action on it.

Self-Healing Test Scripts

In this context of continuous improvement and always-changing customer expectations, our web applications have become highly dynamic. This has made the work of test automation engineers more difficult. With even a minor change in the UI, we often need to revisit multiple test scripts.



What if our test scripts were given the ability to do this themselves! That is exactly what artificial intelligence can do for us.

We can write algorithms so that whenever a minor UI change is detected, rather than failing the Test Case, our framework can update the test script accordingly. The algorithms need to be capable of differentiating between an acceptable change and a bug and should take action accordingly. Additionally, it should highlight any change it makes in the test scripts, on the Test Result reports or on some dedicated alerts for a human to double-check the decision.

Below are various approaches an intelligent framework can follow to self-heal a failing script.

- **Correct a failing web element locator**

Most test failures we see in our daily test reports are due to the script not being able to identify a particular element on the page due to its changed HTML properties. An artificially intelligent test automation framework can identify such minor changes and take corrective action. Actions could be determining an alternative locator on the fly and making script updates or flagging an alternative locator in the Test Report for a human to take the corrective measures.

- **Intelligent retry of a failing step**

Another intelligence our test automation frameworks can have is to be able to determine if a failing step is a random failure and the script should retry the action. This can also eliminate all the failures due to synchronization issues.

- **Handle any unexpected error as per the error content**

It is quite easy for a human to read an error message in the application and take further actions accordingly. Introducing the same capability in our test scripts has always been a challenge. We often end up writing multiple if-else blocks of every possible error to explicitly define what the script should do in case one of them appears. This traditional approach is very expensive for test authoring and still lacks the capability to handle any unexpected error. No matter how much effort we spend writing exception handling code, we end up with failed tests due to some unexpected error or warning appearing on UI.

Leveraging artificial intelligence can help us build algorithms to automatically handle any unexpected error or warning and take corrective steps. A model trained to understand the text of any error/warning popup and interpret corrective steps would be a great inclusion in our test automation frameworks.

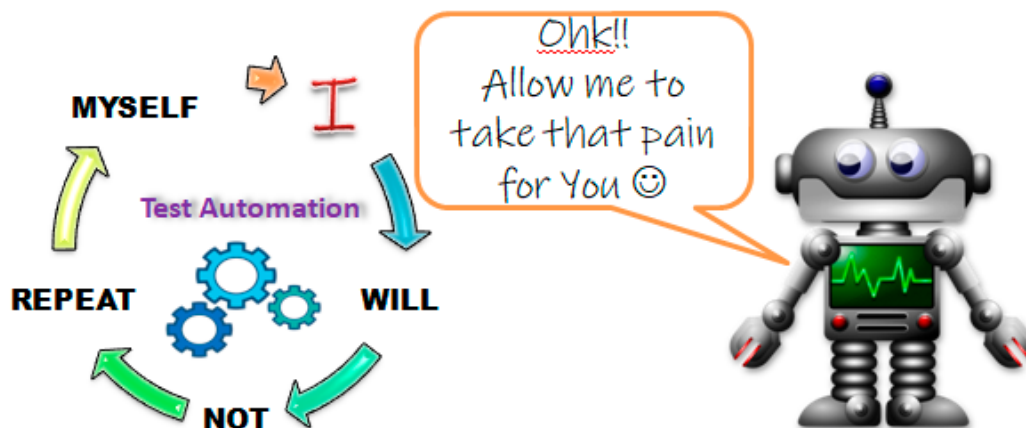
- **Update obsolete test data resulting in error**

The next set of failures we see in our test reports are due to obsolete test data. Test data that was once valid but no longer acceptable in the application can lead to test failures and a held build. Just like a human would simply update the test data and corresponding expected result, our test scripts can also be trained to do so with the help of artificial intelligence and machine learning.

A simple example of obsolete test data is a date range that no longer exists in our date picker control.

Replace Repetitive tasks

No matter how efficiently we write our Test Frameworks to be reused in different projects, there still are numerous tasks we need to perform that are repetitive in nature. So far, we have found very limited ways to avoid them.



Below is a list of few of them and how leveraging AI algorithms can help us automate them.

- **Writing Page Classes**

Page Object model is a widely used design pattern for test automation of web applications to create readable and maintainable test scripts. A page class abstracts internals of a page (web elements and actions to be performed on those) into high level functional flows that are then called in test scripts. Idea is to separate operations on the web pages from verification. While this is a very effective approach, writing these page classes for every single page in our application requires a substantial coding effort, and actual test case automation is often delayed due to this.

The previously discussed approach to automatically identify robust web element locators can further be extended to create some tools, which can perform reverse engineering on the web application and generate page classes for us. If not complete, at least we can get a page class which would be ready to use with very minimal updates. This can be a very useful kick start for any web test automation project.

- **Creating data models (classes representing test data for different web forms)**

In addition to page classes, most test automation frameworks also consist of Data Model classes, which often represent test data for different web forms in our web applications.

Deriving all such Data Models by performing analysis or reverse engineering on the application under test can also be a great use of artificial intelligence and machine learning. This will give any test automation team a skeleton framework that is ready to start test authoring.

Smart Regression

Determining the minimum number of test cases to ensure a particular change in an application is still a million-dollar question for every team. The best we can do is to pick regression test cases that were already categorized according to functional areas, or pick tests based on experience with the application under test. This often leads to some bug leakage or delay in CI/CD pipelines.

Tools leveraging artificial intelligence and machine learning can help us solve this problem to a large extent. Such models and algorithms can take a greater number of parameters into consideration than are humanly possible and determine the size of the regression test suite with great precision. These tools would provide recommendations of test cases to be run for any change, based on different information collected over time.

Below are a few of the many parameters that can be considered for this purpose.

- **Pick up tests based on impacted functionality**

A machine learning-driven tool can better identify the test cases that are touching the functional area that could have been impacted by the change under test, and improve our regression suite selection.

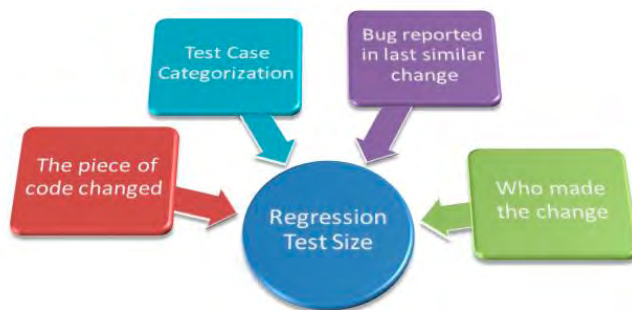
- **Identify tests that frequently fail with a similar change**

Test cases more prone to fail can also be analyzed from the past test reports very efficiently by an automated tool.

- **Which developer wrote the piece of code**

From previous data on application predicting, tools can determine if a particular developer is more likely to write buggy code.

These are just a few parameters for illustration. There could be many data points and parameters (both general and project specific) that can be used by our next generation tools to help QA teams to run only the tests that matter and not the entire regression test suites every time a change in application happens. This will optimize testing efforts and enable faster project releases.



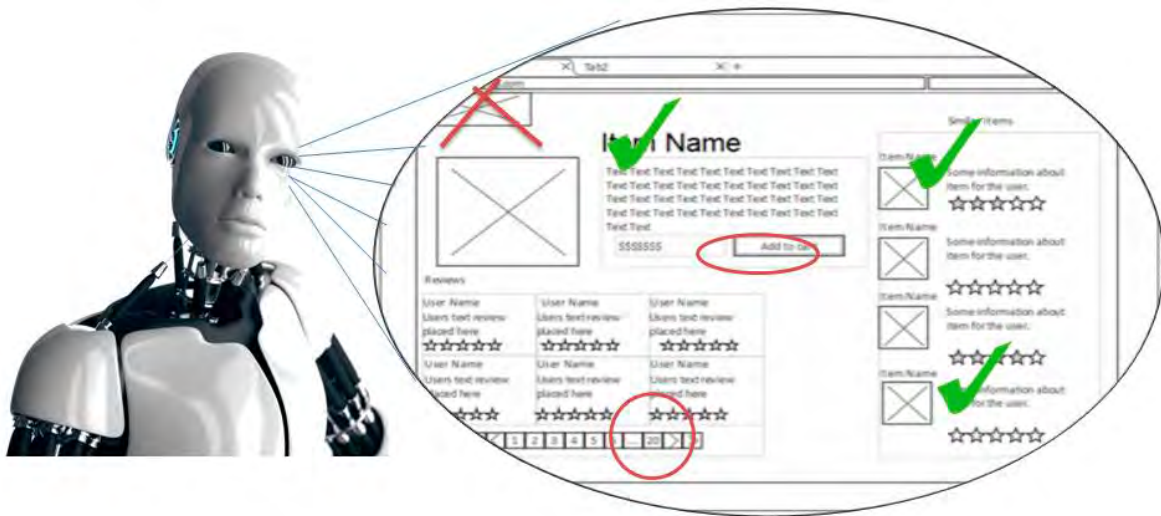
Visual/Snapshot Testing

Visual testing is the verification of visual appearance on a web page with its baseline expected image. The traditional approach to automate visual test cases is to compare a web page pixel by pixel with the baseline image. This wasn't an effective approach and tended to give unreliable results. Modern tools are using artificial intelligence and evolved approaches to view the web page as a human eye would, and are capable of categorizing a mismatch as a defect or an acceptable difference. Automated visual testing is very helpful to uncover UI defects in different browser configurations.

Snapshot testing is a term used to describe an initial approach of visual testing where an algorithm compares each pixel of a snapshot of the web page to check if the hex code is the same as the baseline image. This approach wasn't effective with modern web applications and reports a high number of false positives.

Visual testing techniques can be taken to a whole new level with the incorporation of artificial intelligence. The test scripts can eliminate the need to write assertions on individual web elements and can rely on the visual appearance of the entire web page if the same test data is being used in every test iteration. The idea is to record the snapshot (either an image, HTML DOM or a combination of both) of the outcome web page after each test step and perform an intelligent comparison with the recorded baseline snapshots instead of verifying assertions on element properties.

Apart from speeding up test authoring, this will also help to uncover any UI defects that are not always reported by functional test automation.



Other Uses of AI in Test Automation

In addition to the different approaches discussed above, there are many other ways we can implement artificial intelligence or machine learning in Test Automation. Below are a few of the other probable implementations that are still in their initial stages and will be an integral part of Test Automation once they are mature enough.

- **Spidering**

Spidering or web crawling is an approach where a bot systematically browses through a web application via different links and web forms on the page.

This technique is typically used to browse the Web for indexing. It can be extended further to perform reverse engineering on an application under test and identify Test Cases automatically.



- **Test Bots doing exploratory testing on an application**

As fascinating as it sounds, test bots performing testing on any application with very minimal human intervention would revolutionize the software development life cycle.

While our traditional testing approaches run in parallel, these test bots can be trained to perform exploratory testing on any web application.

By nature, exploratory testing cannot be automated with traditional approaches, but some futuristic tools that incorporate artificial intelligence can learn to do so while silently observing when a QA engineer explores an application manually and notices defects. Another approach is crawling through an application and finding any unusual patterns during future test runs.







- **Early automation - deriving test scripts from mockups or architecture documents**

Another probable use case of artificial intelligence in Test Automation is deriving test automation scripts from the design documentation.

If followed, this approach can help us to start test automation as early in the development of an application. For example, the UI mockups created for any web application are a rich source of information we need to define our Page Object and test scripts. Similarly, other design and architecture documents can be analyzed by an AI-powered automation tool to perceive major functional flows and define test scripts accordingly.

Tools Leveraging AI in Test Automation

While they are in the very initial stage, there are several test automation tools that are already leveraging artificial intelligence and providing rich experiences to users. Below is a list of a few such tools in no specific order.

	<p>Testim</p> <p>Testim is an AI-based UI testing tool. This tool leverages AI and machine learning to provide fast test authoring and reliable test scripts.</p>
	<p>Applitools Eyes</p> <p>Intelligent Automated Testing Platform Applitools Eyes is powered by Visual AI, which replicates human eyes and the brain to spot functional and visual regressions quickly.</p>
	<p>Testcraft</p> <p>Testcraft is a codeless selenium test automation platform for regression and continuous testing, as well as monitoring of web applications. Their revolutionary artificial intelligence technology eliminates maintenance time and cost as it automatically overcomes changes in the app.</p>
	<p>Test.ai</p> <p>Test.ai is an AI-powered mobile test automation platform, which scales and tests thousands of apps in parallel without the need to code or maintain tests.</p>

Conclusion

While still in its infancy, the inclusion of artificial intelligence and machine learning in software testing is a reality and a requirement for coming industry demands. These next gen technologies can consume a large amount of information and make predictions, which can be used by test automation tools to build efficient and robust test frameworks. With this paper, we have tried to uncover a few of the many implementations of AI and ML technologies in test automation.

A question might arise: As a QA engineer, should I learn about artificial intelligence right now?

The answer is 'Probably, yes!' Very soon all the test automation tools will be powered by artificial intelligence to some extent, and being able to understand or be competent enough to train a machine learning model will be a valuable addition to every test automation engineer's portfolio.

About the Authors



Prageet Pathak is a Senior Quality Assurance Lead at GlobalLogic with extensive experience in Functional and Automation Testing. Prageet contributes to various streams through his automation skills in mobile and web application.



Parvendra Singh is a Quality Assurance Consultant at GlobalLogic with extensive experience in Web and API Automation. Parvendra works to ensure quality throughout the product engineering life cycle, which includes Functional Testing, Test Automation and Process improvements.

References

Vahabzadeh, A. M. Fard and A. Mesbah, "An empirical study of bugs in test code," 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), Bremen, 2015, pp. 101-110, doi: 10.1109/ICSME.2015.7332456

5 great ways to use AI in your test automation - Joe Colantonio, Founder, TestGuild

GlobalLogic®

GlobalLogic is a leader in digital product engineering. We help our clients design and build innovative products, platforms, and digital experiences for the modern world. By integrating strategic design, complex engineering, and vertical industry expertise,— we help our clients imagine what's possible and accelerate their transition into tomorrow's digital businesses. Headquartered in Silicon Valley, GlobalLogic operates design studios and engineering centers around the world, extending our deep expertise to customers in the communications, automotive, healthcare, technology, media and entertainment, manufacturing, and semiconductor industries.



www.globallogic.com