# Revolutionizing Test Automation: Harnessing Generative AI to Empower Product Management and Accelerate SDLC

**Artem Golubev**

artem.golubev@testrigor.com

## Abstract

Test automation has existed for a long time, but are we reaping its benefits to the fullest? Do you think your substantial efforts in script creation, maintenance, and upkeep fail to fulfill the real purpose of test automation?

Time is money, and quality is the best business plan. Today, in the era of DevOps, Agile, AI, and, more specifically, generative AI, you have intelligent and better options available for test automation. In this paper, we will discuss the capabilities of generative AI, which lets you streamline and simplify test creation, execution, and maintenance. The power of generative AI lies in its simplicity and the inclusion of everyone on the team, regardless of their technical competency. This helps you significantly reduce manual effort and time-to-market and enhance efficiency drastically. We will discuss examples and case studies that demonstrate these benefits in visible numbers.

Artem will analyze traditional and advanced technologies, methodologies, and future trends in test automation, highlighting where we are and where we could be. He will also discuss the role of product management in this new era, mentioning how AI tools can empower the leadership to make more informed decisions, optimize resource allocation, and introduce innovation.

This paper explains the transformative potential of generative AI in test automation and quality. We will review what generative AI can accomplish, focusing on its impact on product management and the Software Development Life Cycle (SDLC) acceleration.

## Biography

*Artem Golubev is a co-founder of testRigor, a YC company. He is excited about helping companies to become an order of magnitude more effective in QA and deliver software faster with the help of Generative AI.*

*Artem started his career 25 years ago by building software for logistics companies. Since then, he worked at companies like Microsoft and Salesforce, where he learned about best practices and top technologies in QA.*

*It was a frustrating experience to struggle with building automation and, especially, test maintenance at almost any company he worked for. This is how testRigor was born. testRigor AI now empowers companies like Netflix, Cisco, Burger King, and many more to build test automation faster and spend significantly less time on test maintenance by employing Generative AI.*

# 1 Introduction

Manual testing is suitable for ad-hoc testing, but for the repetitive, complex, and fast test execution, manual testing is not the correct answer. With fast DevOps delivery cycles, manual testing has become a bottleneck, not an enabler. Similarly, the traditional approach to test automation involves significant manual effort in script creation, maintenance, and upkeep. This process can be time-consuming and prone to errors, reducing test automation's overall efficiency and effectiveness. As the software industry has evolved, more intelligent and efficient testing methods are urgently required today. Here, we will see the issues with the current software testing paradigm and how to improve this situation with advanced AI features and capabilities for better quality, efficiency, cost, and timelines.

# 2 Current QA Situation

Currently, you might be using manual testing, traditional test automation tools, or falsely marketed “codeless test automation tools” to meet your testing requirements. The question is, are they delivering how you want them to? Are they helping you achieve the required quality with less manual effort and time? Or are you stuck in expensive production issues, missed deadlines, and never-ending script maintenance? Let us discuss the drawbacks of traditional testing methods.

## 2.1 Drawbacks of Manual Testing

- Bugs slip into the production
- Not enough test coverage
- Time-consuming and error-prone
- Longer release cycles hinder DevOps principles

## 2.2 Drawbacks of Traditional Test Automation (Selenium/Appium)

- Too much maintenance effort, cost, and time
- 5% of tests always break and are unstable
- Slow test creation due to programming
- ROI is low due to the extensive effort involved

Here is a simple login test in Selenium. It requires all the imports, drivers, properties, and dependencies in place before you run the actual test case. Also, as we can clearly see, it is dependent on fragile XPath locators.

```
package tests;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.Test;

public class TestSample {
    @Test
    public void login() {
        System.setProperty("webdriver.chrome.driver", "path of driver");
        WebDriver driver = new ChromeDriver();
        driver.manage().window().maximize();
        driver.get("https://www.browserstack.com/users/sign_in");
        WebElement username = driver.findElement(By.id("user_email_Login"));
        WebElement password = driver.findElement(By.id("user_password"));
        WebElement login = driver.findElement(By.name("commit"));
        username.sendKeys( charSequences: "abc@gmail.com");
        password.sendKeys( charSequences: "your_password");
        login.click();

        WebElement welcomeXpath = driver.findElement(
                By.xpath("//*[@id=\"wpbody-content\"]/div[2]/header[2]/div/nav/ul/li[5]/a/span"));
        String welcomeXpathText = welcomeXpath.getText();
        Assert.assertEquals(welcomeXpathText,  s1: "Welcome Peter");
    }
}
```

Any change in the element's attributes will cause the test script to become invalid. Quite evidently, you require programming expertise to write this test case. Here is the same test case written through generative AI:

```
login
check that page contains "Welcome Peter"
```

As you can see, the simplicity of generative AI lets you use just a few straightforward English words instead of the whole chunk of programming code. Generative AI enables you to test complex test scenarios with simplicity and ease. You can save the AUT's login credentials in the Gen AI tool and then use the command 'login' every time you want your test script to simulate user login. On average, you can create 5 test cases in a day using generative AI compared to 0.5 using traditional test automation tools such as Selenium.

Additionally, most codeless test automation tools in the market still work on Selenium or programming scripts, and hence, they inherit the drawbacks of traditional test automation tools.

# 3 Role of Generative AI in Test Automation

According to Gartner, by 2026, more than 80% of organizations will use generative AI applications or its application programming interfaces (APIs). AI has been a prominent topic in technology for the last ten years. When generative AI, especially ChatGPT, came out in 2022, it caused worldwide attention and a massive surge in AI innovation and use.

Let us understand: **What is Generative AI?**

Generative AI refers to a category of artificial intelligence that is designed to generate new content or data that is similar to what it has been trained on. Unlike traditional AI models, which typically focus on recognizing patterns or making decisions based on existing data, generative AI creates new, original outputs. This can include text, images, music, synthetic data, and even code.

## 3.1 Benefits of Using Generative AI in Software Testing

Generative AI offers a modern and intelligent approach to creating software applications and test automation. It uses advanced machine learning algorithms to generate, execute, and maintain test scripts automatically. You can write test cases in plain English or any other natural language to test complex test scenarios easily and quickly. This approach reduces the dependency on manual scripting and allows for continuous (24x7, in parallel) and efficient testing throughout the SDLC.

- **Write test cases in plain English:** If you choose to write test cases yourself, then generative AI with NLP lets you do that in plain English or any other natural language. Here is an example of selecting a t-shirt size and scrolling the page until the "Pay Now" button is visible on the UI:

  ```
  click "t-shirt"

  select 1st option from "Size"

  click "Add to cart"

  scroll down until page contains "Pay Now"
  ```

- **Automated test case creation**: You can quickly create automated tests using generative AI capabilities. Generative AI can take descriptions in plain English of what needs to be tested and then identify the key actions and conditions involved.

Provide the test case title/description or describe the feature for which you need the test cases. The intelligent generative AI engine will automatically generate the test steps.

For example, if a tester describes an application (bestbuy.com) as “an e-commerce website where people can search and shop online products”. Then Gen AI will generate automated test cases such as:

1. Order Confirmation and Payment Processing
2. Product Search, Selection, and Checkout Process
3. User Registration and Account Creation Process

It is intelligent enough to then create the test steps for the test case “Product Search, Selection, and Checkout Process” automatically in plain English as:

```
enter "Laptop" into "What can we help you find today?"

click "submit search"

click "Dell Inspiron 15 Touch Screen Laptop – Intel Core i5"

click "ADD TO CART"
```

These steps are generated automatically by the AI without needing the tester to write any code, even the test data is identified by generative AI automatically. Within minutes, you have an automated test case ready to run without any coding hassle.

- **Record and playback features**: The generative AI-based automation tools also allow for record and playback capabilities if you wish to do so. The test steps will be in plain English, and you will have automated tests ready as you navigate through the user flow. For example, creating tests with our Chrome extension will take 4X less time than manual testing; our client IDT’s manual QA team was able to get from 34% to 91% automation in under nine months.

- **Near zero test maintenance effort:** Instead of relying on fixed locators like XPath or CSS selectors, generative AI uses a more abstract approach, often based on the visual and functional attributes of elements. This means that even if the UI structure changes slightly, the AI can still identify the correct elements based on how they appear or behave, reducing the need for frequent updates to locators. Generative AI has the ability to understand the context in which elements appear.

  For instance, if a button labeled "Login" is changed to "Sign In", the AI can recognize that these labels serve the same function and adjust the test case accordingly. This reduces the likelihood of test failures due to minor text or layout changes. As the application under test evolves, generative AI updates its understanding of the application’s UI, which allows it to adjust the test scripts automatically. This self-learning capability means that the test cases are in sync with the latest version of the software.

  Generative AI uses advanced algorithms to recognize and interact with elements on a user interface, even if their properties change. For example, if the ID or label of a button changes in the application, traditional test scripts might fail because they rely on specific identifiers (XPath/CSS). However, generative AI can adapt by understanding the context and purpose of the element, allowing it to continue interacting with the correct button without needing manual updates to the test script. This lack of XPath and CSS dependency ensures ultra-stable tests that are easy to maintain.

For example, you can support only 200 test cases in Selenium before being stuck 100% at test maintenance. This number is 200,000 test cases for testRigor. Also, our numbers show that it takes 99.5% less time to maintain tests in testRigor than in Selenium.

- **Reusable subroutines:** You can create a subroutine in plain English for a group of test steps, which are often repeated in the test suite. Use the subroutine's name in the test case when you want to run these repetitive steps. Also, if there are changes, you can rewrite the subroutine and need not update each test case separately. Here is a Salesforce test case example of creating a new task (in the Campaigns module):

```
login

create campaign

click "Campaigns"

click stored value "generatedCampaignName"

click "New Task"

generate unique name, then enter into "Subject" and save as
"generatedSubject"

click "Due Date"

click "Today"

click "Save"

check that page contains stored value "generatedSubject" roughly below
"Activity"
```

As you can see, ‘create campaign’ is a reusable subroutine used in all campaign-related end-to-end test cases. Here are the test steps of the ‘create campaign’ reusable subroutine:

```
click "Campaigns"

click "New"

generate unique name, then enter into "Campaign Name" and save as
"generatedCampaignName"

generate unique name, then enter into "Description" and save as
"generatedCampaignDescription"

click "Save"

check that page contains stored value "generatedCampaignName"
```

- **Cost and time reduction**: Automation reduces the need for extensive manual testing, leading to labor and resource savings. Since test cases can be generated quickly in plain English, time that was previously diverted to test script programming is now saved. Additionally, the early detection of defects reduces the cost associated with fixing bugs later in the SDLC.

- **Inclusion of everyone in testing activities through BDD (Behavior-driven Development) and SDD (Specification-driven Development):** Use the power and advantages of shift left testing with generative AI. Create test cases early, even before engineers start working on

application code using SDD. With intelligent test automation, BDD is used in its true sense, which is an attempt to express end-to-end tests from an end user's perspective. With plain English test cases, you can write test cases as the end-user will see them without any implementation (programming) overhead. The application code and automation test cases can be generated using the same product specifications.


- **Consistency and stability of test results:** The test results are stable and consistent because there is no reliance on details of implementation (CSS/XPath locators). The non-reliance on basic implementation details makes the test scripts robust and ultra-stable.

- **Empowers manual testers to create robust test automation:** With the power of generative AI, you need not hire separate automation engineering to build test automation. Your manual testers have the right testing skills to test complex test scenarios, which will otherwise take much coding and error handling.
  Here are some English commands to accomplish complex test scenarios:

  1. **Database**: `run sql query "select top 1 UserID, LastName, FirstName from Users;"`
  2. **File**: `check that file "instruction.pdf" was downloaded`
  3. **Email:** `send email to "user@customer.com" with subject "Test message" and body "Hi, this is a test, this is just a test message."`
  4. **SMS**: `sms to +15344297154 with body "hello"`
  5. **Phone call**: `call "+15344297154" and validate it was picked up`
  6. **QR code**: `scan qr code value from stored value "saved-qr-code" and save as "code"`
  7. **Captcha resolution**: `resolve captcha of type recaptcha`
  8. **Audio**: `check that audio from "my-recording" is "70%" similar to "my-specific-recording"`
  9. **Visual testing**: `compare screen to previous version treating error as "minor"`
  10. **Tables**: `check that table "actions" at row "102" and column "Name" contains "Spock"`

- **Improved test coverage and efficiency:** Generative AI allows you to write tests in plain English, and your whole team can create and execute test cases. This leads to more thorough testing, identifying edge cases and potential defects that might otherwise go undetected. It increases test coverage and efficiency tremendously.

- **Test scalability:** Generative AI can handle large-scale testing efforts, making it ideal for projects with extensive test requirements. It can quickly adapt to the increasing complexity and volume of tests needed for modern software applications. It can help you find the new corner cases that might get missed by the human eye. Also, the complex workflows (banking applications), and scenarios such as file handling, SMS and phone call testing, Captcha resolution, etc., are handled by generative AI-based testing in plain English.

  Test data management, parallel execution, modular test design, parameterization, continuous integration, and regular maintenance are some of the capabilities that generative AI-based test automation provides. This, in turn, helps achieve test scalability.

- **Single tool to test multiple platforms:** The Gen AI-based test automation tools are versatile enough to test web, mobile (native, hybrid), desktop, API, database, etc. This greatly reduces the overhead of using different testing tools for different platforms or devices.

- **Supports continuous testing:** Generative AI being the latest technology, supports continuous testing practices. It seamlessly integrates with continuous integration/continuous delivery (CI/CD) pipelines through CI/CD tools such as Jenkins, GitLab CI, CircleCI, Buddy, etc. This integration ensures that testing is conducted at every stage of the development cycle. It helps with real-time feedback to the development team to fix bugs and facilitate rapid iterations.

# 4 Use Cases of Generative AI Features

## 4.1 Write automation test cases in plain English and copy-paste manual tests from test management systems to automate directly

You can either write your test scripts using the test case editor in English or the record-and-playback tool to record test cases in plain English. There's also the option to let these intelligent tools use their generative AI feature to create a fully functional test case for you with just a description and URL.

Here is a test case example to provide more clarity:

*enter "Kindle" into "Search"*

*enter enter*

*click "Kindle Scribe"*

*click "Add To Cart"*

*check that page contains "Item Added To Cart"*

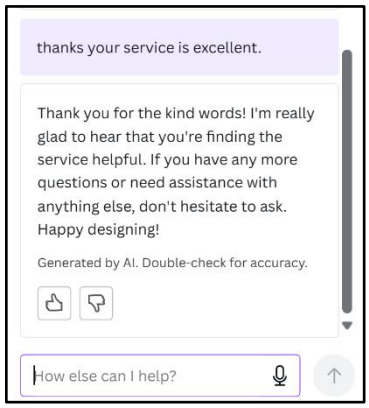
As you can see in the above example, you can just mention the text visible on the UI, and the elements are identified by AI. Even if the labels are changed, such as "Login" changed to "Sign-in" the AI is able to identify it, and the test case will work flawlessly. Also, the actions and assertions on these elements are in plain English (or any natural language). This is possible because generative AI-based test automation tools use artificial intelligence to reduce the maintenance effort typically associated with automated tests. The reason is simple: traditionally, the automated tests have used XPath/CSS identifiers as locators. Now, with Gen AI, this has changed; the AI uses multiple attributes to recognize an element: context, purpose, behavior, etc. Therefore, it can automatically adjust tests

based on application UI changes. This decreases the time spent in maintaining tests after each application update.

You can also import or copy-paste your existing manual test cases from test management tools such as TestRail, PractiTest, Zephyr, etc. Tweak them if necessary, and you can run these test cases directly as automated tests.

### 4.2 Test LLMs and AI features through intelligent AI agents

You can use LLMs (Large Language Models) to analyze and test real-time user sentiments. These findings will help you act immediately based on that information to solve the customer issue and provide greater user satisfaction. Test your LLMs and AI using other LLMs' intelligence. If the real-time user sentiment is not positive (e.g., a user is not able to book a hotel), then you can immediately let customer support take action and aid the user in their user journey. For example, you can verify whether the customer chat has a positive message or not using AI as below:




```
check that "chat" "contains a positive message" using ai
```

Another example is you can utilize Vision AI to validate a natural language statement about the webpage as below:

```
open url "https://bestbuy.com"

check that page "has a search bar on the top of the page" using AI
```

All of the above commands invoke AI to analyze the page/screen and perform complex validations that were previously only possible to do manually. Generative AI makes testing advanced AI features such as LLMs straightforward.

# 5 Case Studies: Test Automation with Generative AI

### 5.1 A Fortune 1000 company (IDT) went from 34% automation to 91% automation in under nine months, with only manual QA personnel.

IDT spent 32 person-years of QA Engineers building automated tests. After that, they were not progressing towards their 90% automation goal and were stuck at around 33-34% automation because all of their QA Engineers were 100% busy maintaining existing tests instead of building new ones.

*"We spent so much time with maintenance when using Selenium, and we spend nearly 0 time with maintenance using testRigor*", says **Keith Powe**, **VP of Engineering at IDT Corp**.

With generative AI, they achieved the following milestones:

- Turned their manual testers into sophisticated automation engineers with a reduced learning curve when compared to test automation tools based on scripting.
- They are saving a minimum of $576K worth of costs per year and received 7X ROI
- Since they were using Selenium for test automation, where flaky tests are a real problem, switching to generative AI-based testing reduced considerable time and effort spent on maintaining these Selenium tests. They saved costs by eliminating precious engineering hours spent on maintaining flaky tests or application updates.
- Converted Selenium tests or manual test scripts to stable, easy-to-maintain, plain English-based autonomous tests

They were automating about four test cases per week per person. Now, since they started with generative AI-based testing, they are automating double.

### 5.2 A MicroMobility company (tembici) went from 100% manual testing to 100% automated with an 80% decrease in bugs.

With manual testing, tembici encountered more bugs in production and postponed deliveries. They needed consistent quality environments and use cases where they could execute all their tests at once rather than scattered among multiple developers. Each developer had one device, and they had no concept of a device farm where they could test multiple devices and configurations.

*“testRigor is an easier platform to achieve the consistency of maintaining and running the tests in a speedy way*.”, says **Carolina V.**, **IT Management at tembici.**

With generative AI, they achieved the following milestones:

- Developers could begin making tests within an hour of starting testRigor
- Able to test multiple devices without having to write new tests for each device
- Went from testing six builds a week to six builds in just 3 hours
- Eliminated all manual testing

Furthermore, the entire team, from product managers to UI/UX engineers, had a consistent framework for creating, communicating, and executing automated tests. This ultimately led to a more productive team utilizing a tool every member could use.

# Conclusion

By simplifying the creation, execution, and maintenance of test cases, generative AI not only addresses the limitations of traditional testing methods but also enhances the overall efficiency and effectiveness of the Software Development Life Cycle (SDLC). This technology empowers teams across all levels, whether they are seasoned automation engineers or manual testers, to contribute to the testing process with minimal training and technical expertise.

The benefits are clear: reduced maintenance efforts, faster time-to-market, improved test coverage, and significant cost savings. Case studies from companies like IDT and tembici depict the transformative impact of generative AI, showcasing dramatic improvements in test automation levels, cost efficiency, and product quality.

As we progress, product management will continue using these AI-driven tools to optimize resources, make informed decisions, and implement innovation. Generative AI is not just a tool for today but a cornerstone for the future of software testing. It successfully enables teams to meet rapid development cycle demands and consistently deliver high-quality products.