Case Study

IoT + AI-Augmented Testing

Fully managed
AI-augmented QA

Up to 69% less time on test creation

~2.4× faster automation on average

Project Overview

Industry:

IoT Security & Automation

Platforms:

iOS, Android, Web

Service type:

Fully managed AI-augmented testing

The client is an international company that designs and manufactures IoT solutions for security and automation. Their ecosystem spans hardware (sensors, controllers, security devices) and software for monitoring and control across mobile and web platforms.

Products tested:

Real-time monitoring and control of security sensors, alerts, and connected device management (iOS & Android)

Door sensor status monitoring with real-time event notifications (iOS & Android)

Admin platform for device ecosystem management, user accounts, configuration, and client support (Web)

The Problem

Test automation on the project initially only covered web. When both mobile apps entered the automation team's scope, the volume of work grew sharply, causing a large backlog of automation tasks.

Automation backlog kept growing

The team needed to build automated tests for three products across three platforms (Android, iOS, Web). Page Object classes, locators, and helper utilities had to be created for dozens of new screens. At the existing pace, the backlog grew faster than the team could close it.

Defect analysis took too long

After each test run, the engineer had to read through execution logs and stack traces to determine whether a failure was a product bug, an environment issue, or an unstable automated test. With multiple products running, this triage consumed hours per cycle.

Codebase maintenance grew

As the test framework expanded from one web product to cover mobile apps as well, repeating patterns accumulated across the codebase. Duplicate logic, oversized methods, and inconsistent patterns made refactoring time-consuming and error-prone.

Routine development was time-consuming

Creating mobile element locators, generating Page Object templates, and writing boilerplate utility code took a significant share of each sprint. The time spent on this routine work left less room for expanding test coverage and improving the framework architecture.

The client decided to bring in QATestLab's AI-augmented automation expertise to tackle the backlog and keep velocity across the entire ecosystem.

Solution: AI-Augmented Testing in Action

QATestLab's QA Automation Engineer integrated Cursor IDE and MCP Agents (Appium, Selenium) into the daily workflow. AI served as a tool to speed up routine engineering tasks, with every output reviewed and validated by the engineer before it entered the codebase

1 Accelerated automation development

AI analyzed the provided test steps and generated full automated tests for both mobile apps and the web platform. It produced Page Object class templates, helper utilities, and mobile element locators in a fraction of the time manual coding would take. The engineer also used AI to optimize existing code and identify opportunities to improve the test framework architecture.

Impact: Broader test coverage built faster, without writing every class, locator, and utility manually.

2 Integration and reporting automation

AI assisted in setting up and maintaining technical integrations: configuring automated test integration with Azure DevOps, generating configurations for automatic Test Run creation, producing test reports, and attaching execution artifacts (screenshots, logs) to test results.

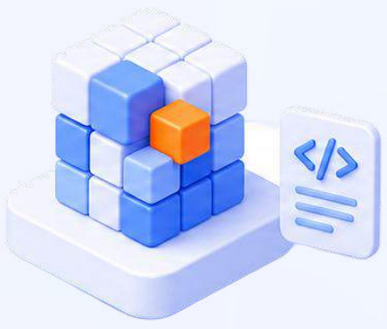
Impact: CI/CD reporting set up faster, with artifacts attached to every run automatically.

3 Defect analysis

AI parsed test execution logs and stack traces to identify the probable root cause of each failure. It categorized problems as product defects, environment issues, or automation instability, cutting the time the engineer spent on triage after each run.

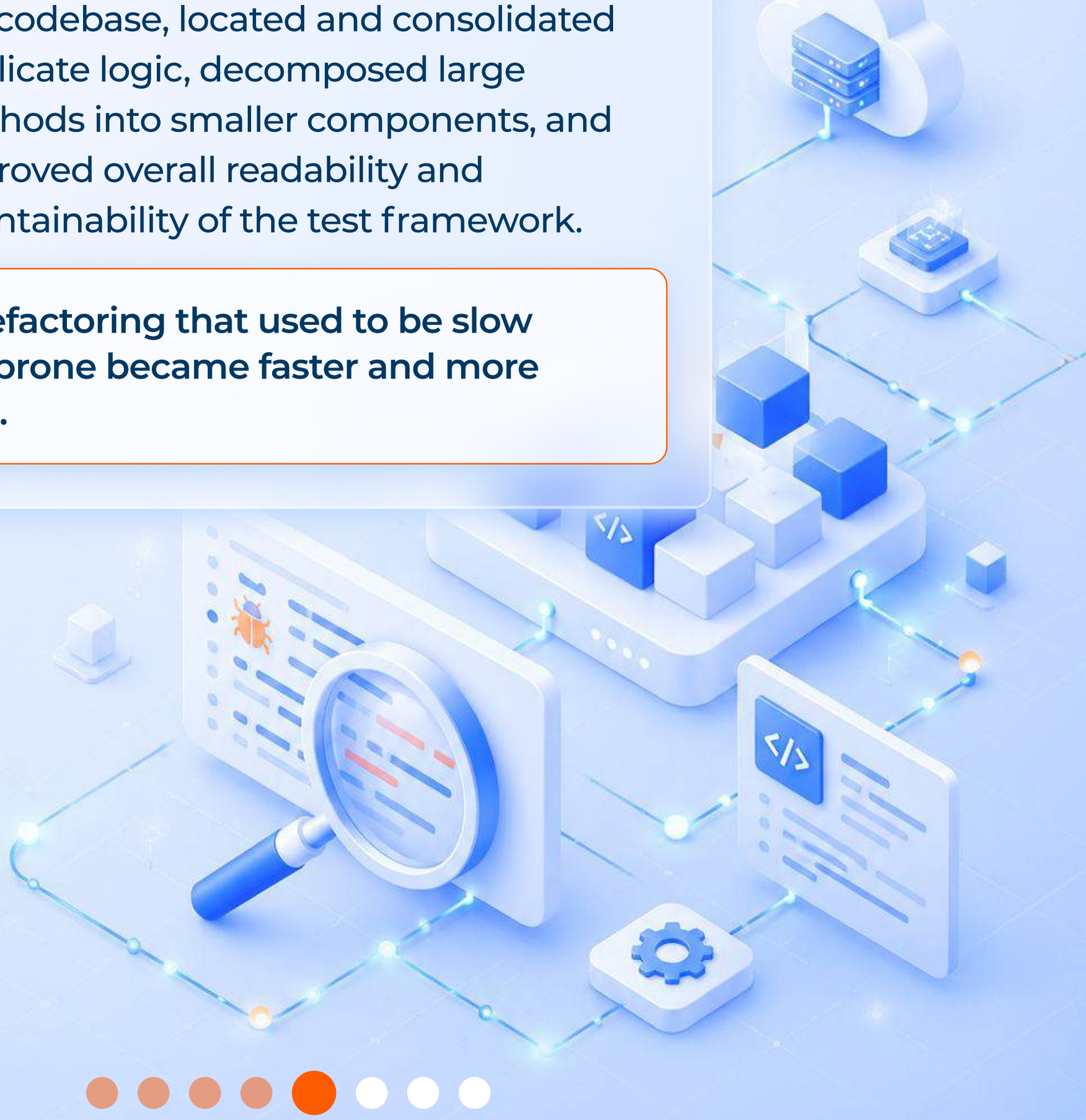
Impact: **Less time reading logs, faster path from failure to root cause.**

4 Codebase refactoring and maintenance



AI handled bulk pattern updates across the codebase, located and consolidated duplicate logic, decomposed large methods into smaller components, and improved overall readability and maintainability of the test framework.

Impact: **Refactoring that used to be slow and error-prone became faster and more consistent.**



What AI Handled vs. What the Engineer Owned

AI operated as a tool under human oversight throughout the project. Every generated test, locator, and code change was reviewed and validated by the engineer before it entered the framework.

What the AI handled	What our engineer owned
Generated automated tests from the provided test steps	Test logic, coverage strategy, and the decision of what to automate next
Produced Page Object classes, locators, and helper utilities	Validation of every generated locator and code review before merge
Analyzed logs and stack traces to identify probable root causes	Final categorization of defects and communication with the dev team
Performed bulk pattern updates and refactoring across the codebase	Architectural decisions and framework design
Generated Azure DevOps configurations and reporting integrations	CI/CD pipeline management, test run execution, and results monitoring

🎯 **This model means no generated code is delivered without human review. The engineer checks, validates, and signs off on every AI output before it reaches the pipeline.**

Results

The team measured performance across 30 automated tests:

Metric	Android	iOS
Speed increase	1.9–2.3×	2.6–3.2×
Time saved	47–57%	61–69%

Without AI

98–123 hours

estimated effort

With AI

46.75 hours

actual effort

Overall:

~2.4× faster

Up to

69% less time on test creation

What these numbers mean

The automation backlog that grew with every sprint started shrinking. Test coverage for mobile apps that would have taken weeks to build was ready within days. The engineer who used to spend hours on boilerplate code and locator inspection now spends that time on test logic, coverage strategy, and framework improvements.

Tech Stack

Languages & Frameworks	C#, NUnit
Mobile Automation	Appium (Android & iOS)
Web Automation	Selenium
CI/CD & Reporting	Azure DevOps, RestSharp
AI Tooling	Cursor IDE, MCP Agents (Appium, Selenium)
Platforms	Android, iOS, Web

When both mobile apps entered our scope, the automation backlog grew faster than we could close it. Cursor and MCP agents changed the pace. AI takes on the routine part – generating Page Object classes, locators, boilerplate – and I focus on logic and coverage. Defect analysis got faster too: sorting through stack traces to categorize failures used to take hours, now it's much shorter. Generated code still needs review, locators need validation, test logic needs human judgment. But as a tool for speeding up routine work, it delivers. The ~2.1–2.6× acceleration is what we measured on real tasks.

– QA Engineer assigned to the project

Get measurable results from AI-augmented testing

Request a Free AI QA Estimation →

CONTACT US