AI-Powered Unit Test Synergy to Reduce E2E Dependency

Simone Colosimo
Sonepar Digital – simone.colosimo@gmail.com

Abstract

Unit tests are critical pieces of testing frameworks such as the Testing Pyramid, widely used in the software industry.
However, many organizations treat unit testing as a separate, reserved developers domain. A lack of synergy with QA is the default habit, resulting in overreliance on end-to-end (E2E) tests, costly test maintenance, and inefficient feedback loops.
I want to share how my team integrated unit tests into a shared testing strategy involving developers, Product Owners, and QAs.
By improving collaboration and aligning test coverage early in the development process, we can identify which scenarios are covered by unit tests and which need more focus from E2E automation, reducing unnecessary E2E tests, their scope, and the efforts for their maintenance.
Additionally, we leveraged AI-powered tools to identify and review unit tests more efficiently, further streamlining the testing process and helping focus on areas not covered by unit tests, improving overall test efficiency.
Our approach contributed to measurable improvements in key DORA metrics, such as a higher Deployment Frequency (DF) and a reduction in Change Failure Rate (CFR).

Biography

Simone Colosimo is a Quality Engineering leader with nearly two decades of experience in software quality and a strong interest in collaborative team dynamics and cross-functional collaboration.

He has held QA leadership roles at companies in various industries, including Samsung Electronics and Dashlane, and he currently serves as QA Director at Sonepar Digital.

Simone combines QA leadership with program delivery ownership to transform QA practices into scalable, engineering-aligned strategies. He has led initiatives that bridged the gap between developers, testers, and product stakeholders.

He has spoken at international conferences such as Agile Testing Days and JFTL on topics ranging from zero-bug policies to mental health.

Simone holds a Master's degree in Network and Telecommunications Engineering and is fluent in Italian, French, and English. Born and raised in Rome, he's currently based in Paris.
He enjoys traveling, photography, and has recently embarked on a journey to learn guitar.

1. Problem Statement

In modern software development, the need for rapid delivery and high-quality releases has magnified the role of automated testing strategies.
Widely adopted models like the Testing Pyramid address this need by promoting a balanced and maintainable test strategy that delivers fast feedback and catches issues early.
The model recommends using a large base of Unit Tests, a moderate number of Integration Tests, and a smaller layer of End-To-End (E2E) tests at the top.

Despite this guidance, many organizations still over-rely on E2E tests, replacing manual bottlenecks with automated ones. When unit tests are neglected, E2E tests tend to become too numerous, covering broad scenarios and becoming fragile, i.e. prone to breaking frequently with even small changes. This results in slow execution, high maintenance overhead, and delayed feedback, ultimately reducing the effectiveness and value of test automation.

A major driver of this issue is the organizational mindset that Unit Tests (and Integration Tests) are siloed, reserved domains of developers, while Quality Assurance (QA) ensures solely the E2E tests activities, such as design, creation, execution, and maintenance.
The direct consequence is that QA professionals and Product Owners are typically excluded from the design and evaluation of unit test coverage.
As a result, testing strategies often lack cohesion across different layers of testing.

This fragmentation not only burdens CI/CD pipelines with lengthy test executions but also creates gaps in test coverage, increasing the risk of defects escaping to production. In such environments, the feedback loop becomes inefficient, deployment confidence drops, and engineering teams struggle to achieve desirable DevOps metrics such as high Deployment Frequency (DF) and low Change Failure Rate (CFR), as defined by the DORA framework.

Additionally, even when unit tests are present, they are rarely reviewed or discussed outside code review contexts. Non-developers have limited visibility into what unit tests cover, blurring the “testing image”. The absence of shared ownership and alignment around unit testing leads to suboptimal resource allocation in test automation and diminished trust in the testing effectiveness.

How can we foster a culture in which unit tests are considered a shared asset and a strategic component of the testing process, not just an undermined implementation detail?

Can we leverage emerging AI-powered solutions to facilitate and accelerate a synergy across roles and test layers?

Addressing this challenge requires a shift in mindset, tooling, and practices. It necessitates a collaborative approach to testing strategy that actively includes developers, QAs, and Product Owners in shaping and optimizing test coverage.
Through this shift, organizations can reduce their dependence on costly E2E tests, accelerate feedback loops, and achieve better delivery outcomes as reflected in critical DevOps metrics.

This paper presents a real-world case study of a team that embraced this collaborative strategy. The team achieved measurable improvements in software quality, delivery speed, and operational efficiency by integrating unit tests into a shared conversation and employing AI tools for test discovery and optimization. We will detail the approach, tools, measured outcomes, and the trade-offs encountered.

2. Solution Implementation

To address the disconnect between unit testing and the other test layers, we implemented a structured, simple, collaborative process that integrated unit tests into testing strategy conversations along the development lifecycle.

We established a process in which developers, QAs, and Product Owners jointly participated in test planning sessions. These sessions, held during backlog refinement or early sprint planning, involved discussions on user stories to align on expected behaviors and identify what could and should be covered by unit tests.

2.1 Proof of Concept (POC) and scaling`

The first step was to gather concrete data and demonstrate the value of the new approach. We involved key, experienced QE members to review existing E2E test coverage in their respective teams and compare it feature by feature with unit tests already present in the codebase.
This analysis quickly revealed significant overlap: many E2E steps and scenarios were already covered at the unit level.

Following this discovery phase, we asked them to update their E2E suites by removing redundant tests and steps.
In one extreme example, 27 E2E tests were reduced to just one, with the rest handled by unit tests. Overall, most teams saw a 33% to 70% reduction in E2E test scope, either in terms of the number of tests or individual test steps.

With successful outcomes from the pilot teams, we transitioned from experimentation to formalizing the approach as a simple, structured process. We documented clear guidelines to help QEs expand the analysis done in the POC to all the teams.
Then, we proposed to embed them into team ceremonies like sprint planning and story grooming. The goal was to ensure the process was scalable across teams without adding unnecessary overhead or disrupting existing workflows.

2.2 Keep the process simple and powerful

At the heart of the process, there are just two guiding questions:

- What do we want to test?
- How do we want to test it?

These two questions became the foundation of our quality conversations. They encouraged teams to clarify their intent and consider the most appropriate test layer for each scenario, unit, integration, or E2E test. Despite their simplicity, these questions are powerful because they:

- Align roles and expectations: Developers, QEs, and Product Owners can finally have a shared conversation about test strategy.
- Spot implicit assumptions early: Discussing what to test surfaces hidden risks, edge cases, or business rules that may be overlooked.
- Enhance clever test design: By considering *how* to test something, teams can select the most efficient and effective layer to validate behavior, often reducing their reliance on E2E tests.
- Promote shared ownership: These conversations shift testing from a reactive to a proactive and collaborative approach.

By maintaining this simplicity, the process remained lightweight enough to be adopted across teams yet meaningful enough to drive real improvements in test strategy, coverage clarity, and delivery performance.

2.3 From Manual Collaboration to AI-Driven Synergy

As described above, the synergy between roles was achieved by fostering direct conversations with developers and POs during backlog refinement or story analysis.
We manually reviewed the associated unit tests for each relevant feature or story, to understand what was already covered and what required additional testing. This hands-on collaboration proved its value because it improved mutual understanding and reduced test duplication.

However, maintaining this level of deep analysis across all relevant stories demanded significant time and commitment from both QAs and developers. The process, while effective, wasn't easily scalable in a fast-paced delivery environment.

This is where AI-powered tooling became a true game-changer. By integrating an AI tool (GitHub Copilot in our case) we were able to automatically analyze the codebase, detect existing unit tests, and highlight gaps in test coverage. This allowed us to scale and sustain the collaborative synergy between developers and QEs without requiring the same level of manual effort.

Previously, reviewing pull requests (PRs) and validating unit test coverage was a time-consuming task shared between developers and QEs. With the help of AI, much of this process became automated. The tool accelerated analysis, reduced review time, and enabled quicker, more informed decisions about where additional testing was needed.
We also used AI to suggest or auto-generate missing unit and E2E test scenarios. It helped identify cross-team dependencies and surfaced untested edge cases that might otherwise have gone unnoticed, especially in large and complex codebases.

Another challenge was inconsistent test structures between teams or developers. AI tooling helped standardize test creation practices, promoting more uniform and maintainable test suites across the organization.

Overall, the integration of AI significantly boosted our efficiency, enabled proactive test strategy adjustments, and empowered teams to focus on higher-value testing activities.

3. Challenges

3.1 Mindset shift

One of the most significant obstacles was shifting the perception that unit testing is exclusively a developer concern. QEs and Product Owners were not used to participating in discussions about low-level test coverage, and developers were not used to leveraging unit tests to cover the acceptance criteria.
Once again, valuable contributions from key, experienced QE members helped move the initiative forward.
Through POC results sharing, repeated communication, training, and reinforcement, they helped all roles understand that early alignment on unit test strategy reduces rework and redundant testing later.

3.2 Trade-offs

3.2.1 Pipeline Integration Gaps

By implementing this approach, it's important to ensure that unit tests are fully integrated into regular test pipeline runs. If they're missing or not consistently executed as part of the CI process, some key scenarios may go untested, especially if we assume that they're already covered at the unit level.
This can create a false sense of confidence and increase the risk of bugs slipping into production.
Additionally, not all unit tests are equal in value or structure.
Tests that rely heavily on mocks may fail to provide sufficient confidence in the product's release.
In such cases, it may be more effective to shift the validation to the E2E level, where actual behavior across integrated components can be verified more reliably.
Recognizing these exceptions is important, otherwise the risk is to under-test critical paths or place too much trust in unit test results.

3.2.2. Choosing the right test layer

Finding the right balance requires teams to stay critical and pragmatic, and this is valid not only in terms of where and how to test, but also in the process itself.
It's essential not to place blind faith in a single testing strategy, tool, or layer.
Unit tests must be both present and meaningful. At the same time, teams need to recognize when certain behaviors are better validated through integration or E2E scenarios.
A thoughtful, case-by-case evaluation ensures that the testing strategy remains effective, adaptable, and grounded in real product needs, rather than being loyal to a rigid process.
For instance, in our case, we decided to write E2E scenarios for the four basic data operations: Create, Read, Update, and Delete (CRUD), even though they are already covered by unit tests.

3.2.3 Limits of AI-Driven Testing

AI-generated tests always require human review to ensure they truly align with the project's requirements. This is especially important for edge cases, where the context or complexity may lead to inaccuracies in AI.

Integrating AI tools into existing CI/CD systems, particularly for tasks such as automated test generation or coverage analysis, often requires a custom setup. This can be more complex in projects with non-standard tech stacks or legacy infrastructure.

Additionally, every project comes with its own testing practices, architectural patterns, and conventions. To be effective, AI tools typically need to be fine-tuned or configured to reflect those specific needs, rather than used as a one-size-fits-all solution.

3.3 Unit Test tracking

One of the recurring challenges in quality engineering is the limited visibility and traceability of unit tests within the broader testing strategy. While E2E tests are often tracked via test management tools and linked to feature tickets, hence linked to product mapping, unit tests typically remain isolated within the codebase, tied to pull requests, and thus linked to code mapping.
However, they are not easily traceable over time or across features.
This siloed nature makes it difficult to answer questions like: “Which unit tests validate this feature?” or “Is this area of the code properly tested at the unit level?”

Another problem to face is the fragmentation of testing tools and reporting. Unit tests and E2E tests often exist in separate pipelines, making it harder to consolidate test coverage metrics or pass/fail trends into a single view. This lack of integration weakens confidence in the test suite’s completeness.

Additionally, tracking historical trends and test quality metrics over time is difficult without automation. Teams struggle to measure how coverage evolves, whether regression coverage is maintained, or where gaps might emerge after several releases.

Finally, enforcing a consistent coverage standard for unit and E2E tests is often a manual and error-prone process. Developers may unknowingly merge changes with insufficient coverage, increasing the risk of bugs or regression.

4. Results

Despite the challenges, we were able to observe measurable outcomes in software quality and delivery.

4.1 CI improvements

The synergy between unit and E2E tests brought tangible improvements to our CI/CD pipeline.

By carefully reviewing the E2E suite through the lens of unit test coverage, we identified and removed redundant E2E tests and too broad scenarios that added little value but consumed time and resources.
This clean-up effort resulted in a relevant reduction in test execution time across CI stages.
Builds became faster, and test reliability increased, in particular in pre-release stages where E2E tests were considered “a waste of time”.
By focusing on the added value provided by optimized E2E tests, we had fewer blocked pipelines due to fragile E2E tests, and QA teams could focus more on exploratory testing instead of constantly analyzing test failures.
Our test pass rate for scheduled pipelines went from 50-70% in Q1 2024 to 85-90% in 2025.
The combination of faster test cycles and fewer flaky tests unlocked more frequent and smoother releases, enabling quicker delivery cycles without sacrificing quality.

4.2 DORA metrics

Although it’s not always possible to directly link cause and effect, after introducing the unit test synergy, we started noticing improvements in our DORA metrics.
The deployment frequency (DF) increased from one release per week in Q1 2024 to up to three times per week in Q3 2024, and then remained steady at twice per week in 2025. Teams began pushing smaller changes more regularly, supported by faster feedback loops and improved confidence in test results.
At the same time, we saw the Change Failure Rate (CFR) decrease from 30% in Q1 2024 to 3% in Q4 2024. Since the introduction of the synergy, the CFR average is around 6%.
This appears to be tied to a more stable and reliable testing setup. By removing duplicate or overly broad E2E tests and optimizing coverage where necessary, we reduced the number of bugs that caused reverts or hotfixes in production.
From a QA standpoint, the biggest driver of this improvement was reducing the number and scope of E2E tests. A more targeted E2E suite and earlier alignment on unit test coverage resulted in less time spent on maintenance and debugging.

4.3 Test awareness

Perhaps the most meaningful achievement was the shift in test awareness across stakeholders**.**
In several teams, particularly those where Quality Engineers successfully advocated for the change, we saw a shift in how testing was perceived and discussed.
By involving developers in test planning, teams clarified what to cover with unit tests versus E2E tests. The teams that fostered this environment gained better visibility into test coverage and gaps, leading to more informed risk discussions and a stronger sense of shared ownership during sprint planning, achieving a collaborative test strategy that sustains product delivery confidence.

5. Further expansion and iterations

The achievements of our unit test synergy strategy paved the way for further evolution.
We've identified several key areas where we plan to continue expanding and refining our approach to make testing more scalable, and strategically aligned across all parts of the codebase.

5.1 Backend tests

The next step in our roadmap is to extend this strategy to backend development. While the initial focus was primarily on frontend and cross-functional integration, backend systems often contain a high density of unit and integration tests, especially around APIs and business logic. In many teams, what would typically be considered "E2E" testing is handled directly by backend developers through service-level tests or integration harnesses.

This shift in context presents new challenges. The complexity of backend logic and the absence of a UI layer make coverage analysis and test mapping less straightforward.
We aim to adapt our strategy and AI tooling, considering the various types of validations performed and their context of use.

Specifically, we want to ensure that AI tools can understand and analyze unit and integration tests in backend projects, surface gaps effectively, and provide actionable.
We're exploring ways to bring the same alignment and benefits to backend teams as we did on the frontend.

5.2 Expand AI usage

Our current use of AI for analyzing and surfacing unit test coverage is tailored to our main technology stack, which includes JavaScript and TypeScript.
The tools and workflows we've implemented are effective for these environments, but as our systems grow, so does the variety of tech stacks used across different teams and services.

To ensure broader impact, we plan to extend AI test analysis to repositories built in other languages, such as C#, .NET, and Python. These systems often follow different conventions and testing frameworks, which means the AI tools will need to be fine-tuned for code and test structures specific to each language. Our goal is to enable consistent insights into unit test coverage and quality across all parts of our platform, regardless of language or team structure.

5.3 Enhance Unit Test visibility

Although we've made significant progress in aligning unit test efforts with higher-level test strategy, unit test visibility remains an ongoing challenge.
Unlike E2E tests, which are often tracked in test management platforms and tied to product requirements, unit tests typically live within the codebase and are evaluated mostly during code review.
This creates a disconnect between the tests that validate critical functionality and the broader product development workflow. Stakeholders have limited visibility into what unit tests exist, what they cover, and how they relate to product behavior.
We aim to enhance this by exploring solutions that extract unit test results and context from the code, presenting them in shared, accessible views.
This could involve dashboards or AI-assisted documentation that maps unit tests to features or Jira tickets.
One area of exploration is using AI to summarize or categorize test functions in human-readable terms, helping teams understand what each test is doing without needing to read the raw code.

We believe that better visibility will enable better decisions, reduce duplication, and foster tighter alignment between engineering and product goals.
Making unit tests more transparent and integrating them into day-to-day decision-making is a key part of the next iteration of our testing strategy.

References

Books:

Beck, Kent. 2002. *Test Driven Development: By Example*. Boston: Addison-Wesley.

Crispin, Lisa, and Janet Gregory. 2009. *Agile Testing: A Practical Guide for Testers and Agile Teams*. Boston: Addison-Wesley.

Myers, Glenford J., Corey Sandler, and Tom Badgett. 2011. *The Art of Software Testing*. 3rd ed. Hoboken, NJ: Wiley.

Morris, Paul, and Sander Rossel. 2020. *Continuous Integration, Delivery, and Deployment*. Birmingham, UK: Packt.

Humble, Jez, and David Farley. 2010. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Boston: Addison-Wesley.

Poppendieck, Mary, and Tom Poppendieck. 2003. *Lean Software Development: An Agile Toolkit*. Boston: Addison-Wesley.

Websites:

Fowler, Martin. 2012. “The Practical Test Pyramid.” Accessed September 1, 2025. https://martinfowler.com/bliki/TestPyramid.html

Forsgren, Nicole. 2021. *Accelerate State of DevOps Report*. Google Cloud / DORA. Accessed September 1, 2025. https://dora.dev/

GitHub. 2023. “GitHub Copilot: Your AI Pair Programmer.” Accessed September 1, 2025. https://github.com/features/copilot