Bloor

White Paper

**White Paper** by Bloor

Author **Daniel Howard**

**March 2025**

# Testing Trends in 2025

Your guide to the most significant trends, strategies, and considerations for testing in 2025.

# Introduction

Software testing is challenging: software continues to evolve at a rapid pace, as does the need for high quality and reliability in software from both a functionality and a governance perspective. This means that software testing must be fast enough to keep up with development, while being thorough enough to satisfy stringent quality requirements. These demands are very much in tension, meaning that your testing will need to thread the proverbial needle if you want to take advantage of it to build high quality software at speed. While this is certainly possible, it is also far easier said than done. This is where this paper comes in: as a guide to the most significant trends, strategies, and considerations for testing in 2025, aimed at helping you to build an effective and productive test solution that maximises the value of your testing practice.

## Automated testing

For starters, we should make one thing very clear: if you are serious about your software testing, it must be automated on at least some level. Manual testing is simply too slow, too poorly covering, and too error-prone for serious consideration as the backbone of a modern testing solution (and this is to say nothing of the financial expense, which is also very considerable). We say this confidently, despite manual testing's continued, and unfortunate, pervasiveness in the market.

The real question is not whether you should automate, but how much you should automate, and your answer might be anything from just the execution of test scripts to very nearly everything in the testing process, including high-level processes such as test design. Generally speaking, the more you automate, the faster, more efficiently, more thoroughly, and more consistently you can test; hence, we would generally recommend that you automate as much of your testing as you are reasonably able to.

> "...your testing will need to thread the proverbial needle if you want to take advantage of it to build high quality software at speed. While this is certainly possible, it is also far easier said than done."

# Value-based testing

Like any software solution, it behoves you to monitor the impact of your testing on your business objectives (such as your KPIs – Key Performance Indicators), and thereby measure its value. This is, in essence, value-based testing, and it is vital for ensuring that your testing efforts are having a positive effect. Similarly, it is important to understand your testing requirements and to have a meaningful way of evaluating whether or not you are meeting them. Having a way to associate your tests with your requirements – or, even better, a way to derive your tests from your requirements directly – goes a long way here.

Note that an effective testing solution should both find defects and improve overall application quality, often in less discrete ways such as performance. The latter will often fall into nonfunctional testing, which despite its name is only a hair less important than the more defect-driven functional testing, and you ignore it at your peril. When compared to manual testing, your testing solution should also improve your testing speed, and thereby the productivity of your development lifecycle as a whole. In any case, you will want to monitor your testing to ensure that it is not unduly delaying development.

You will also want to have a robust way to understand test coverage (which is to say, which parts of your system have been tested) and risk (how likely each of those parts are to fail a test). In particular, the former can help you identify tests which provide no additional coverage, and are most likely redundant, while the latter can inform how often you run each of your tests, perhaps even in an automated manner. For example, areas of your system with high risk can be tested more frequently than areas of low risk. This all helps to minimise overtesting – running tests which provide no, or very minimal, benefit – and thence free up computing resources that could be better spent elsewhere (such as on other tests).

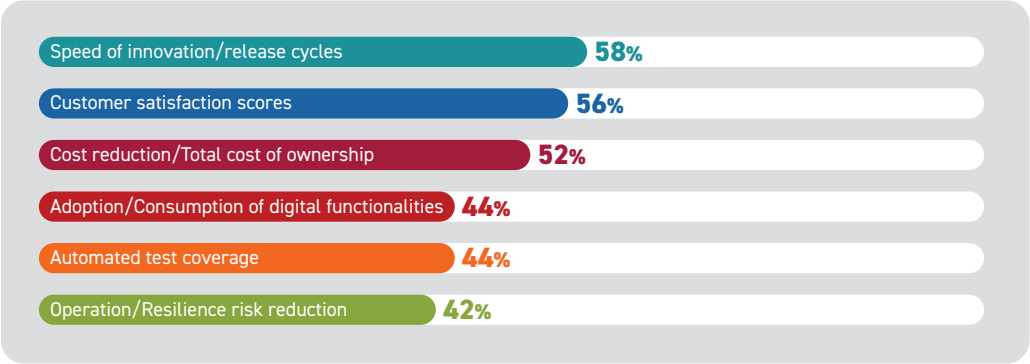


*Figure 1 – "The Most Common KPIs to Measure the Success of QA and Testing Programs", IDC's Quality Assurance and Testing Survey, Q3 2023, N = 400 respondents*

> "Having a way to associate your tests with your requirements – or, even better, a way to derive your tests from your requirements directly – goes a long way here."

# Automated test execution

The most basic approach to test automation is to automate test execution, most often in the form of executable test scripts (see ***Figure 2***). This means having a way to create and run said scripts, such as an open-source framework like Selenium. The result is that you have a suite of test scripts that can be run consistently and repeatedly (perhaps even continuously, with the appropriate tooling), reducing the risk of errors or bias in your test execution and improving coverage due to the increased quantity and frequency of tests that you can run. While automating test execution in this way is only the bare minimum in terms of automated testing, it should still not be understated how much of an improvement it is from relying on manually executed tests.

That said, this method remains the bare minimum. While it forms the basis of more sophisticated approaches, it has significant pain points when used by itself. For instance, building and maintaining a suite of executable test scripts is difficult and time-consuming, especially for large, enterprise-wide systems. Creating these scripts is more akin to programming than writing manual tests, and may even be beyond the expertise of some test engineers. There is an upside to this: test engineers that are experienced with writing test scripts benefit from significant flexibility and customisation, due to their low-level, programmatic nature. Even then, however, we have to imagine that most test scripts do not significantly benefit from this kind of attention, which would usually be better spent elsewhere.

Moreover, since you are still relying on manual processes to create your test scripts, it is all too easy for some scenarios – the ones the test writers did not think of – to go untested. Needless to say, this can have disastrous consequences.

In short, while automated test execution is a good starting point, and while retaining the ability to write test scripts manually can be useful, we posit that most enterprises would be better served by a greater degree of automation in their testing. For example, model-driven testing, which we discuss in the following section.

> “...while automated test execution is a good starting point, and while retaining the ability to write test scripts manually can be useful, we posit that most enterprises would be better served by a greater degree of automation in their testing.”

**Scenario: Successful log in**

- **Given** the user is on the login screen
- **When** the user enters valid credentials
- **Then** the user is redirected to their home screen

**Scenario: Failed log in**

- **Given** the user is on the login screen
- **When** the user enters invalid credentials
- **Then** the user is asked if they forgot their password

*Figure 2 – Example test script for a login process*

# Model-driven testing

In contrast to automated test execution, model-driven testing tries to automate both test execution and test design. This is achieved by creating a (usually) visual model of the system under test (as seen in ***Figure 3***). This model can be thought of as a "digital twin" for your system (though technically digital twins are a distinct concept), which is furnished with various testing artefacts – often but not always including test data, requirements, executable actions, and so on – and used by the testing solution to create a suite of tests (and often, test scripts). These tests can then be executed automatically, in a similar (sometimes identical) manner to the automated test execution methodology described above.

Starting with a model and deriving tests from it has a number of advantages. For starters, the test suite created can be truly exhaustive, systematically covering every meaningful possibility outlined in your model. Indeed, when new tests are generated, it is often with the goal of maximising test coverage and/or minimising risk, and sophisticated coverage and/or risk analysis is frequently provided as part of model-driven testing. Likewise, maintenance can be automated using change management: when part of your model is altered, it can be detected automatically and (the relevant part of) your test suite can be regenerated. You may have to update your model manually, but this is often a fairly light task, especially compared to manually maintaining a test suite. Similarly, while it is of course possible for your model to be incomplete, it is far less likely than a test scenario being overlooked during a manual test design process. Moreover, the visual nature of the model often makes it obvious when something is missing.

Speaking of, having a visual model of the system under test is a significant boon in and of itself: a good model will make it significantly easier to understand both the system under test and the testing layer on top of it. This is especially useful for onboarding new testers and for collaborating with non-technical users, particularly the actual users of the system under test, who may possess important tribal knowledge in regard to its operation. In addition, fostering this sort of collaboration makes it easier to ensure your testing efforts are aligned with your business objectives. Particularly intuitive test solutions can also help enable testing outside of your dedicated testing teams, with users across the software development lifecycle acting as "citizen testers".

All of this said, model-driven testing is not perfect: creating a model for a large system by hand can be a lengthy process, and this has historically given model-driven testing a relatively long time to value. Fortunately, many of the more advanced model-driven testing platforms have addressed this by providing the means to generate your model automatically. There is a caveat here, however, which is that this is often accomplished by reverse engineering your existing test suite, and will therefore be as complete (or not) as it is. Even so, we would argue that even in the worst-case scenario for model-driven testing, it is not appreciably more effort to create a model than it is to write a comprehensive suite of test scripts – and it has far greater potential payoff. Moreover, there is significant potential in using AI to generate your model from whole cloth, which we discuss further below.

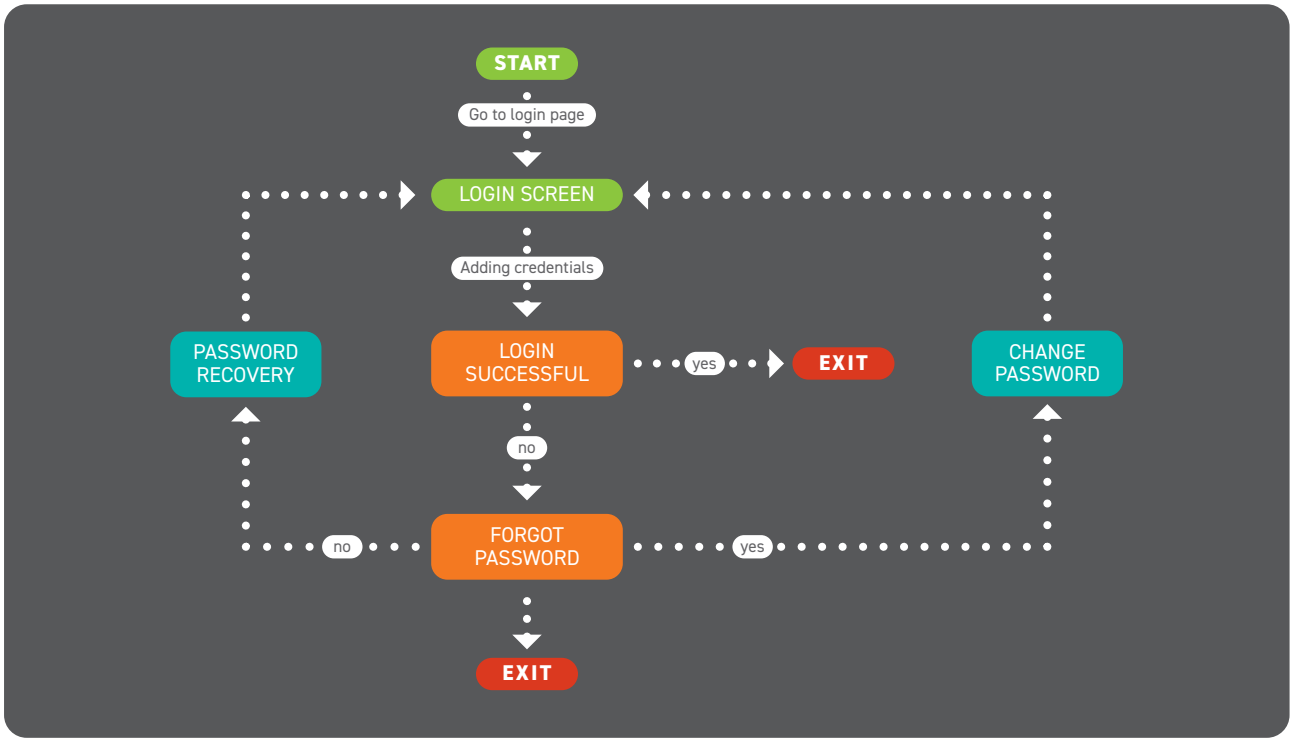


*Figure 3 – Example test model for a login process*

# End-to-end testing

There are two particularly important aspects to enabling truly end-to-end testing: first, your testing solution should be as far-reaching as possible in what kinds of system it can test, in order to maximise test coverage (and therefore value); second, it should slot as seamlessly as possible into your greater software architecture, so that it can be used either automatically or on demand wherever in your system it is required.

To elaborate on the former, there will be a wide variety of different ways that you can test any given system. Traditional testing is done at the user interface layer: a tester (or a test execution framework) will open a browser or desktop window and make their way through the application click by click. This is all well and good, but it is not the be all and end all. For example, you will likely have APIs which will need to be tested separately; there will be physical components to your system that you may want to test; and so on. Hence, it is important to have the capability to test these different elements using various connectivity options. Moreover, while it is possible to test all of these disparate elements individually, it is generally more effective to unify their testing within a single solution. Similarly, you will need your testing solution to be able to interact with a diverse array of data types, stemming from multiple kinds of data sources, in order to create realistic and comprehensive test scenarios.

The latter aspect, on the other hand, is about enabling continuous testing. This is the ability to incorporate testing as part of your automated software delivery pipeline, thereby helping to ensure that your system functions as intended after every release. Ideally, this means running automated tests at every stage of your system's lifecycle, usually testing both earlier and later than you might otherwise. Testing earlier ("shifting left") allows you to find problems sooner in development, when the cost to fix them is lower, while testing later ("shifting right") enables you to monitor your system after deployment and proactively detect and correct any lingering problems. In both cases, you learn about and act on any issues as soon as possible. In addition, continuous testing is essential for creating reliable CI/CD pipelines.

"Testing earlier ("shifting left") allows you to find problems sooner in development, when the cost to fix them is lower."

# Generative AI

Generative AI is likely the most prolific current trend in software, so it should be no surprise that it is being applied to testing as well. Indeed, test automation – particular high-level test automation, such as model-driven testing – is ripe for it: after all, its raison d'être is to automate the creation of all kinds of testing assets, and it makes perfect sense to augment that creation with AI.

For example, above we discuss the difficulty of building a model to drive your model-driven tests. Reverse engineering existing tests is one thing, but what if you could just use AI to create your model from a natural language description, or from analysis of the solution itself? No reverse engineering – and therefore no existing test assets – would be required. Moreover, generative AI can be used to create copilots and other natural language interfaces that promise to help users more easily and intuitively interact with your tests. At the organisational level, in addition to the model creation described above, it can be employed to help create various test assets, including test cases, code snippets, and so on. Other potential uses include more sophisticated defect prediction and test prioritisation, both in deciding which tests would be most productive to create next (in terms of coverage and/or risk) and how often each test should be run in order to maximise efficiency and minimise overtesting. Generating textual insights into the testing process, particularly the root cause of failing tests, is yet another potential use for generative AI in a testing context. It can also be leveraged to help make tests more resilient, by allowing them to interact with an application intelligently and adapt to its user interface when it changes. The core benefit that generative AI provides in all of these use cases is an even higher degree of automation, in turn leading to greater testing speed and efficiency. It may even be necessary for your testers to keep pace if your developers start using it to accelerate development in a similar way.

But it is worth being reminded that AI is not a magic wand: given how in vogue it is, it is all too easy to get lost in heady promises of what AI can do in theory and lose track of what it can actually do in practice. In far too many cases, the degree to which a given solution makes use of AI is grossly exaggerated (this is increasingly known as "AI washing") and comes off as less a technological advancement and more a marketing gimmick. This has created an atmosphere where appetite for AI tends to veer wildly between heady, uncritical acceptance and total distrust, with little in between. Naturally, we advise you to take the middle road of maintaining a healthy, but not overwhelming, level of scepticism. AI can genuinely be of great benefit to your testing solution, but only when it is deployed thoughtfully and meaningfully.

In particular, there is an increasingly urgent need for AI assurance and governance within generative AI solutions. This is being spurred on by rapidly developing legal regulation, such as the EU's new AI act (the "first-ever comprehensive legal framework on AI"), that place significant security and privacy requirements on AI models and the data they use and produce. We expect the ramifications of this act to be similar to GDPR, in that a) it will herald a wave of international AI-oriented compliance regulation, and b) sooner or later the vast majority of AI-using organisations will be forced to take the issue seriously, or else face significant financial (and reputational) repercussions for refusing to do so.

Finally, it is worth discussing agentic AI. This is positioned as the successor to generative AI, and promises to deliver AI that is essentially autonomous – or, at least, that is significantly more autonomous that it has been in the past. For instance, it could be able to make proactive, intelligent decisions, such as providing relevant (perhaps even urgent) insights and analysis without needing to be asked. For testing, utilising agentic AI could involve leveraging tests that deeply and dynamically understand and interact with your environment rather than relying on relatively static test scripts, resulting in tests that are more automated, resilient, and informative. Although it is still early days for this technology, we are waiting with bated breath to see what will come of it, both for testing and in general.

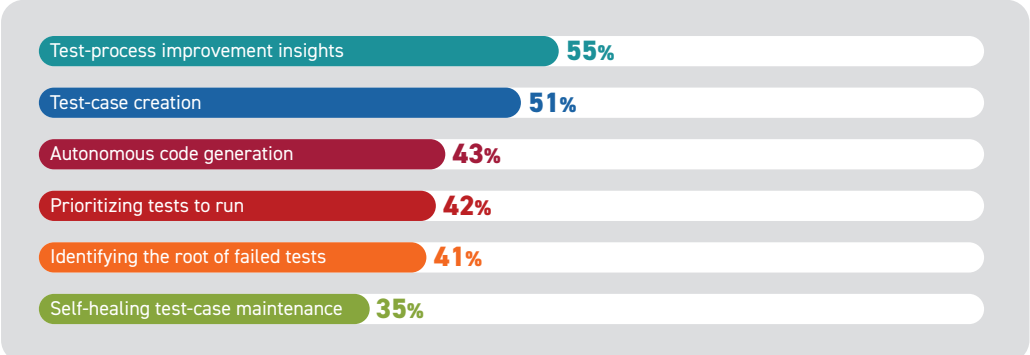


*Figure 4 – "Key Test Automation Areas Likely to Benefit from AI/ML", IDC's Quality Assurance and Testing Survey, Q3 2023, N = 400 respondents*

# Conclusion

We began this paper by saying that software testing is difficult, and it is, but that does not mean it has to be laborious: it can, and should, be thoroughly automated, from the design of your tests on down. We have talked through several strategies for delivering that level of automation and for maximising the value of your testing solution, from architectural issues like connectivity to advanced technologies like generative (and, indeed, agentic) AI. In short, we have highlighted the most significant issues and opportunities that you should be aware of when building a testing solution in 2025.

# About the author

**DANIEL HOWARD**
Senior Analyst

Daniel is an experienced member of the IT industry. In 2014, following the completion of his Master of Mathematics at the University of Bath, he started his career as a software engineer, developer and tester at what was then known as IPL. His work there included all manner of software development and testing, and both Daniel personally and IPL generally were known for the high standard of quality they delivered. In the summer of 2016, Daniel left IPL to work as an analyst for Bloor Research, and the rest is history.

Daniel works primarily in the data space, his interest inherited from his father and colleague, Philip Howard. Even so, his prior role as a software engineer remains with him, and has carried forward into a particular appreciation for the development, DevOps, and testing spaces. This allows him to leverage the technical expertise, insight and 'on-the-ground' perspective garnered through his old life as a developer to good effect.

Outside of work, Daniel enjoys latin and ballroom dancing, board games, skiing, cooking, and playing the guitar.

## Bloor overview

Technology is enabling rapid business evolution. The opportunities are immense but if you do not adapt then you will not survive. So in the age of Mutable business Evolution is Essential to your success.

***We'll show you the future and help you deliver it.***

Bloor brings fresh technological thinking to help you navigate complex business situations, converting challenges into new opportunities for real growth, profitability and impact.

We provide actionable strategic insight through our innovative independent technology research, advisory and consulting services. We assist companies throughout their transformation journeys to stay relevant, bringing fresh thinking to complex business situations and turning challenges into new opportunities for real growth and profitability.

For over 25 years, Bloor has assisted companies to intelligently evolve: by embracing technology to adjust their strategies and achieve the best possible outcomes. At Bloor, we will help you challenge assumptions to consistently improve and succeed.

## Copyright and disclaimer



**Bloor Research International Ltd**

20–22 Wenlock Road, London N1 7GU, United Kingdom

+44 (0)1494 291 992

info@Bloorresearch.com

www.Bloorresearch.com

This information is subject to change without notice. © Keysight Technologies, 2025,
Published in USA, April 16, 2025, 7124-1026.EN