EMA

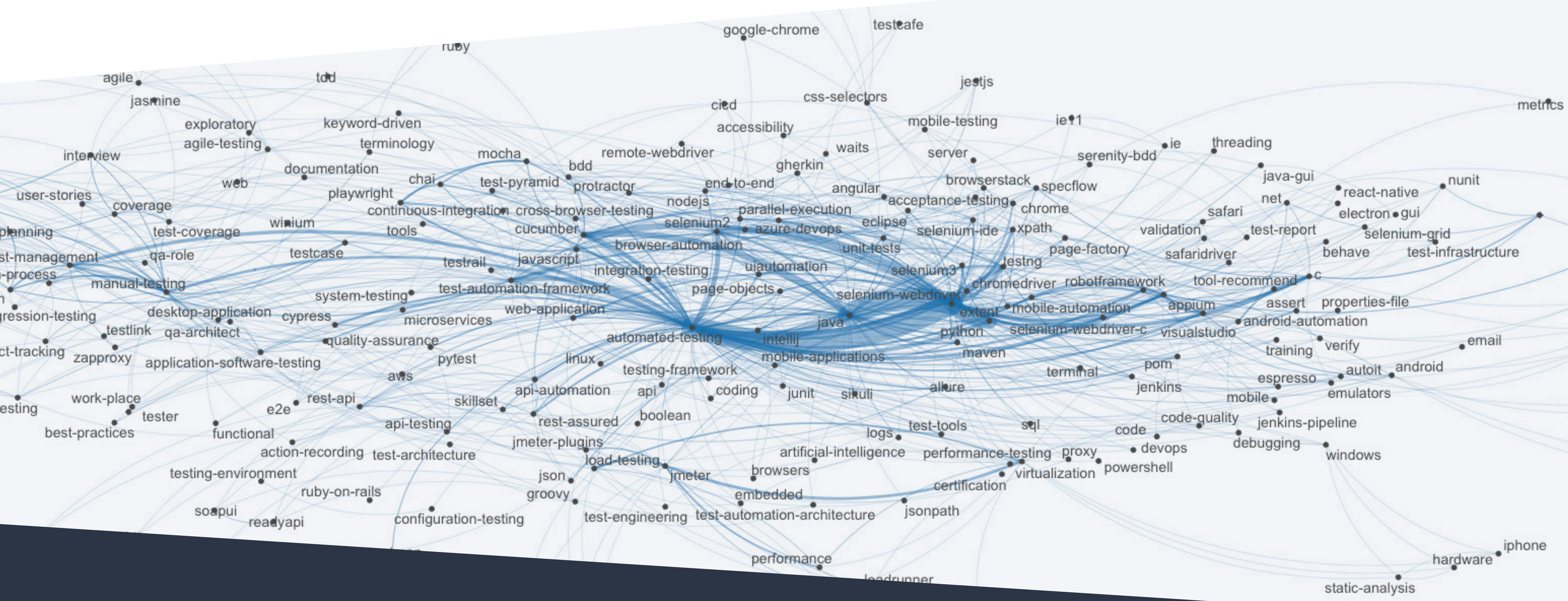
IT & DATA MANAGEMENT RESEARCH, INDUSTRY ANALYSIS & CONSULTING

Disrupting the Economics of Software Testing Through AI

September 2021 EMA Research Report

By Torsten Volk

Managing Research Director for Cloud-Native, DevOps, Machine Learning, and AI



Disrupting the Economics of Software Testing Through AI

As the ability to accelerate the delivery of customer value through innovation, and at lower cost, has become today's critical source for achieving competitive advantages, traditional software testing practices can no longer scale to meet business demands. Test automation frameworks typically rely on a jungle of test scripts written in different languages, using different sets of runtime parameters, and lacking consistent compliance test capabilities. This forces the organization into the unfortunate choice of adding cost or risk to their agile development processes. They can either hire additional staff and increase test infrastructure to cope with the increasing test overhead, or they can accept the added risk that originates from incomplete testing practices.

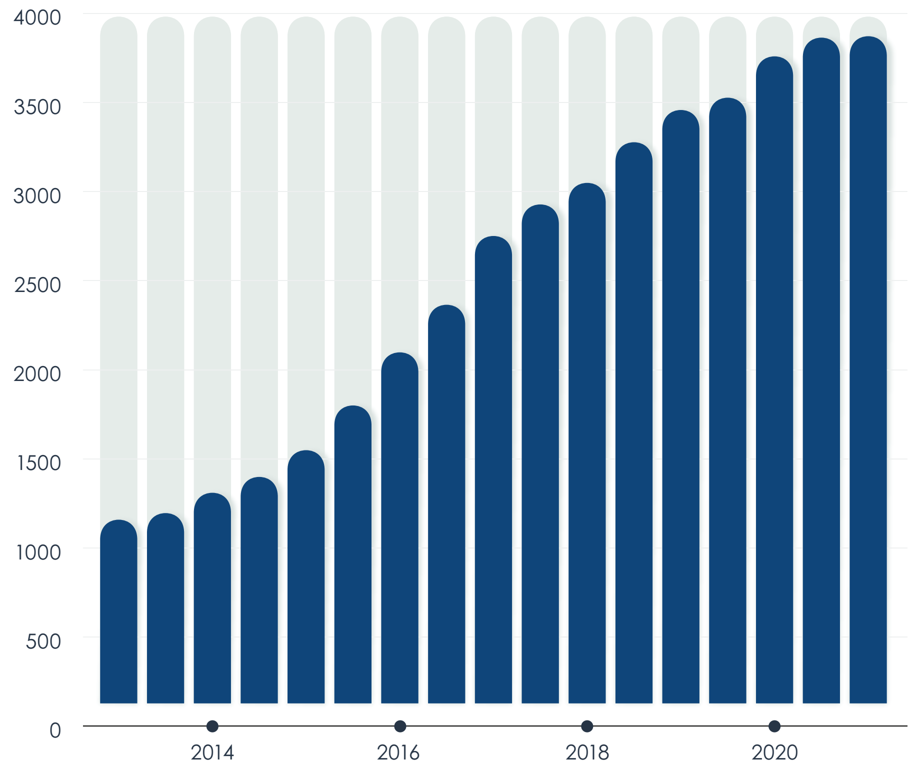
How AI Can Help

An AI-driven approach to software testing disrupts the traditionally linear relationship between the number of software releases and test overhead cost by continuously enhancing the efficiency of the existing quality assurance engineering team through:

- filtering out issues without user impact that do not need human attention
- automating an increasing share of the overall test workflow
- consolidating human tasks for optimal efficiency
- providing human engineers with actionable recommendations and decision context
- learning from human decisions

Ultimately, AI-driven testing can eliminate the vicious rectangle that forces organizations into a tradeoff among releasing faster, reducing cost, improving the customer value of the product (quality), and reserving time for innovation.

Increased Importance of Automated Tests



Increase in the number of test automation questions (per six months) posted to StackOverflow between 2012 and 2021

Why is the Cost of Quality Control Out of Control?

There are key causes for today's escalating cost of quality control.

Mobile Device and Browser Proliferation

Since 2017, the number of different Android smartphones released per year increased by an annual average (CAGR) of 30%, placing an immense amount of pressure on quality assurance engineers to verify functionality, compliance, performance, and user friendliness of their applications on the bulk of smart devices.

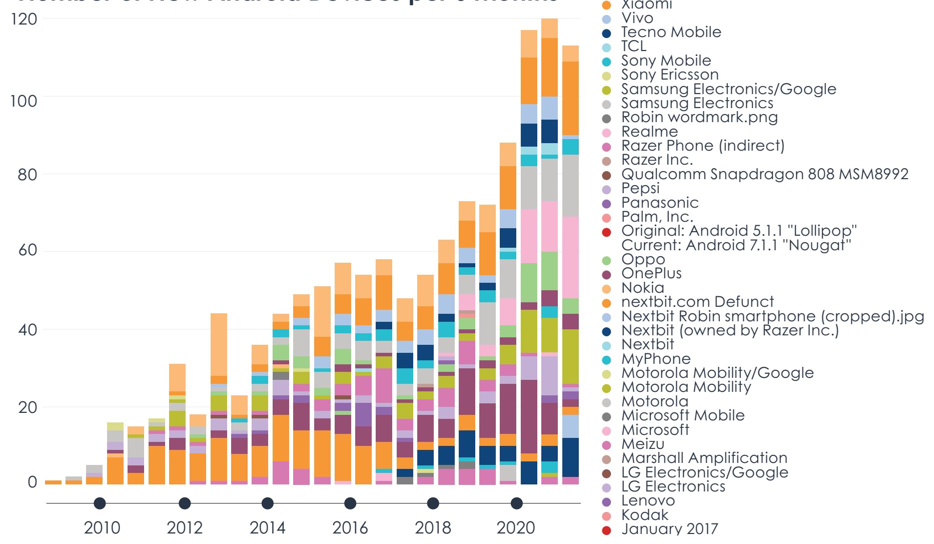
Faster Releases, Shorter Cycles

Business stakeholders ask for continuously faster and shorter releases to deliver increased customer value, without the ability to hire additional quality control staff. EMA research has shown that organizations with a high level of release automation, including test automation, were able to release new software 70% faster and with four times more features compared to their peers with low degrees of automation.

A Declarative Approach is Critical to Controlling Cost

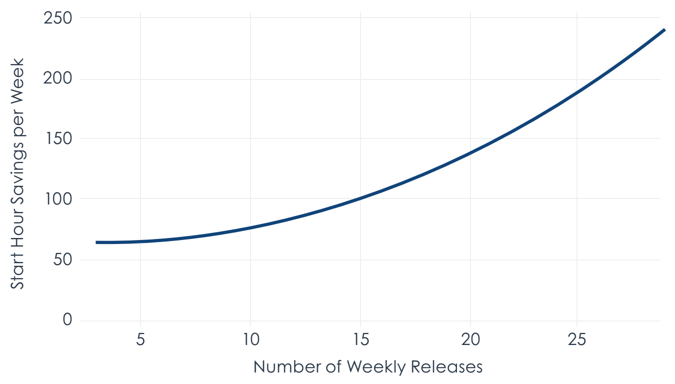
Preventing this escalating complexity from exponentially increasing quality assurance efforts or, alternatively, from decreasing application quality requires a declarative approach to test automation in which the test platform automatically enforces the desired application state wherever possible and escalates cases to human engineers where automatic enforcement is unfeasible.

Number of New Android Devices per 6 Months



List of mobile Android devices by vendor on Wikipedia

Automation Savings Fuel Exponential Increase in Release Frequency



The chart shows the potential weekly staff hour savings for software developers and test engineers because of an increasing number of highly automated releases.

Challenges for Software Engineers and DevOps

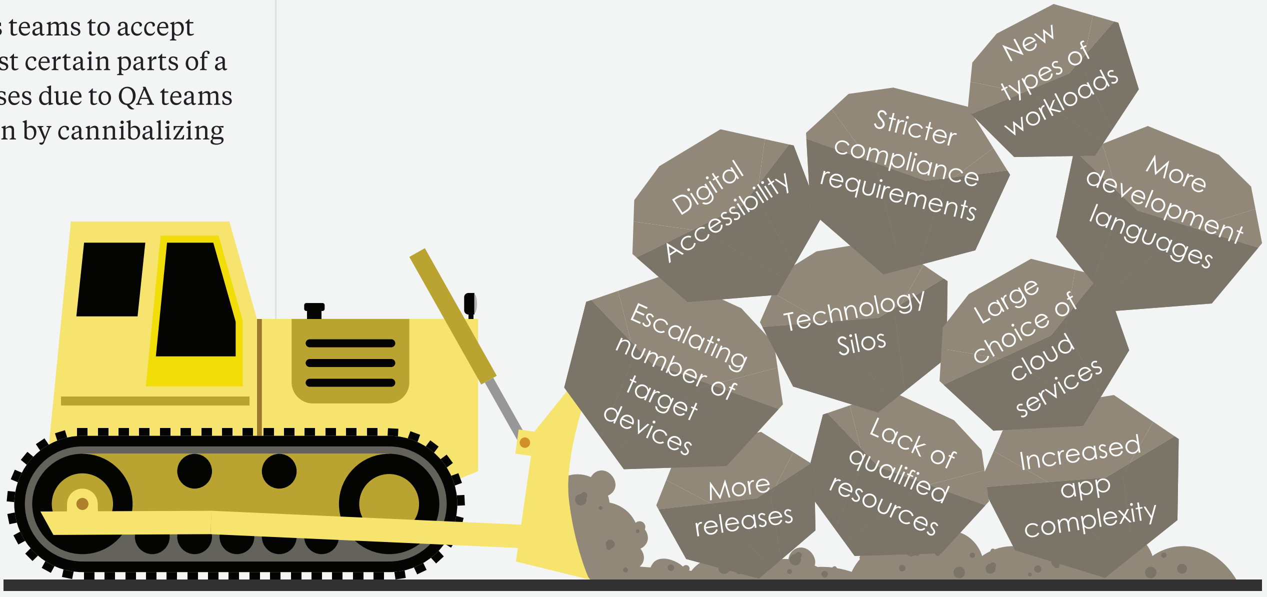
Breaking up traditionally monolithic enterprise software into small chunks of independently released code, referred to as microservices, enables enterprises to rapidly respond to user requirements without having to wait for the next quarterly or even annual release. Instead, each individual product team is responsible for testing their own microservices individually and for continuously monitoring them in production.

Enabling individual development and DevOps teams to deploy, update, and operate their own microservice(s) leads to a higher level of technology choice and compounds the already complex integration and overall quality challenge posed by the escalating number of mobile devices, IoT integrations, cloud service offerings, open-source software platforms, and separately managed DevOps toolchains.

This increase in complexity leads development and DevOps teams to accept increasingly risky choices between "waving through" at least certain parts of a release without comprehensive testing, slowing down releases due to QA teams lagging behind, or recruiting additional test engineers, often by cannibalizing existing software engineers.

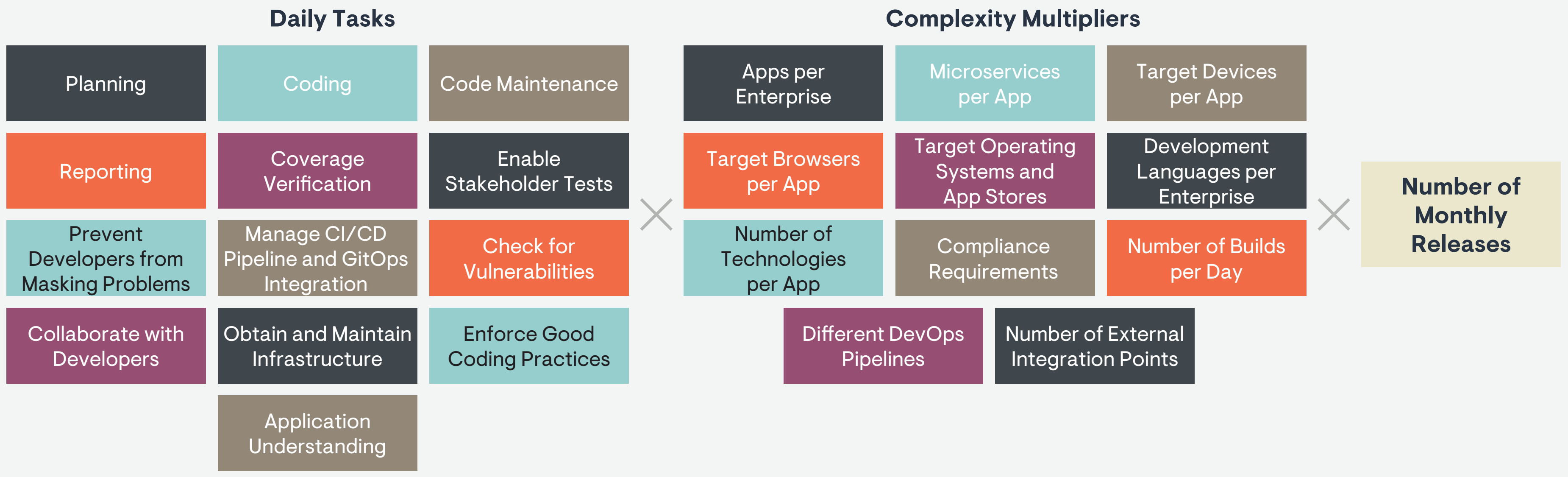
"We see teams struggle to stay on top of executing their test plans, no matter whether they release once a year or every day. The escalating complexity of technologies and applications makes it tricky for everyone to start accepting QA gaps."

QA Team Lead,
Global Telecommunication Provider


New types of workloads
Stricter compliance requirements
More development languages
Digital Accessibility
Technology Silos
Large choice of cloud services
Escalating number of target devices
Lack of qualified resources
Increased app complexity
More releases

Increasing Testing Effort = Daily Tasks x Complexity Multipliers x Number of Monthly Releases

The increase in technology complexity and the growing number of monthly code releases, multiplied by the traditional set of daily tasks to be completed by test engineers, results in an exponential increase in the testing effort required for comprehensive software quality control. Without a constantly increasing number of test engineers who are fluent in automation coding, organizations face decreasing test quality and incomplete coverage.


Daily Tasks
Planning
Coding
Code Maintenance
Reporting
Coverage Verification
Enable Stakeholder Tests
Prevent Developers from Masking Problems
Manage CI/CD Pipeline and GitOps Integration
Check for Vulnerabilities
Collaborate with Developers
Obtain and Maintain Infrastructure
Enforce Good Coding Practices
Application Understanding
×
Complexity Multipliers
Apps per Enterprise
Microservices per App
Target Devices per App
Target Browsers per App
Target Operating Systems and App Stores
Development Languages per Enterprise
Number of Technologies per App
Compliance Requirements
Number of Builds per Day
Different DevOps Pipelines
Number of External Integration Points
×
Number of Monthly Releases

The increase in technology complexity and the growing number of monthly code releases, multiplied by the traditional set of daily tasks to be completed by test engineers, results in an exponential increase in the testing effort required for comprehensive software quality control. Without a constantly increasing number of test engineers who are fluent in automation coding, organizations face decreasing test quality and incomplete coverage.

Automation, Coding, and Tools

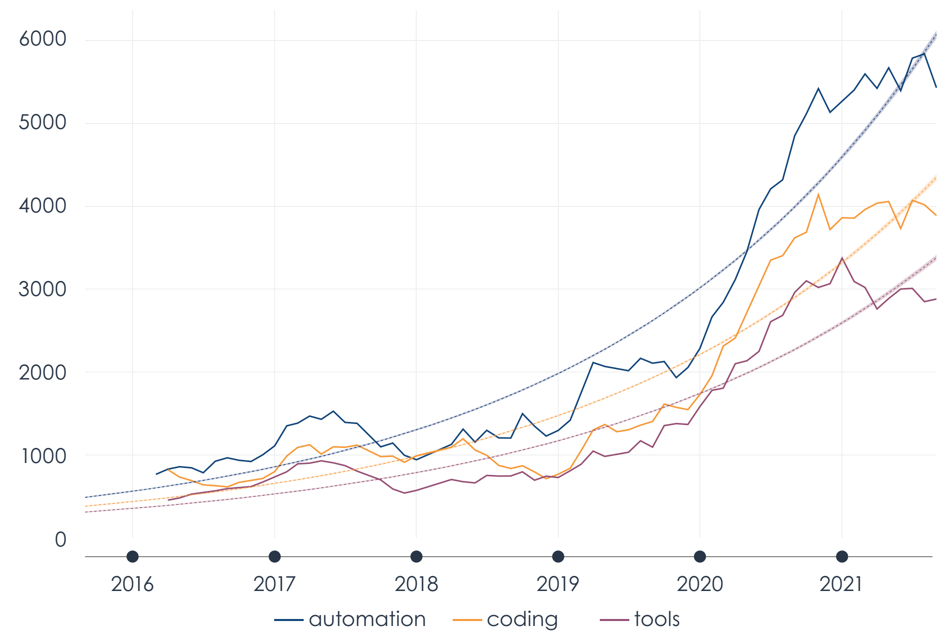
Since the beginning of 2020, test automation platforms and test automation scripts have dominated conversation in the area of software testing. In the ideal case, there is no "code complete" without the corresponding test scripts checked into Git.

In reality, enterprise teams are becoming more and more accepting of incomplete test coverage. The combination of continuously writing and updating code to comprehensively test applications that are composed of numerous distributed microservices while working on a growing number of devices across a wide array of browsers and accessed from network connections seems like a neverending uphill battle.

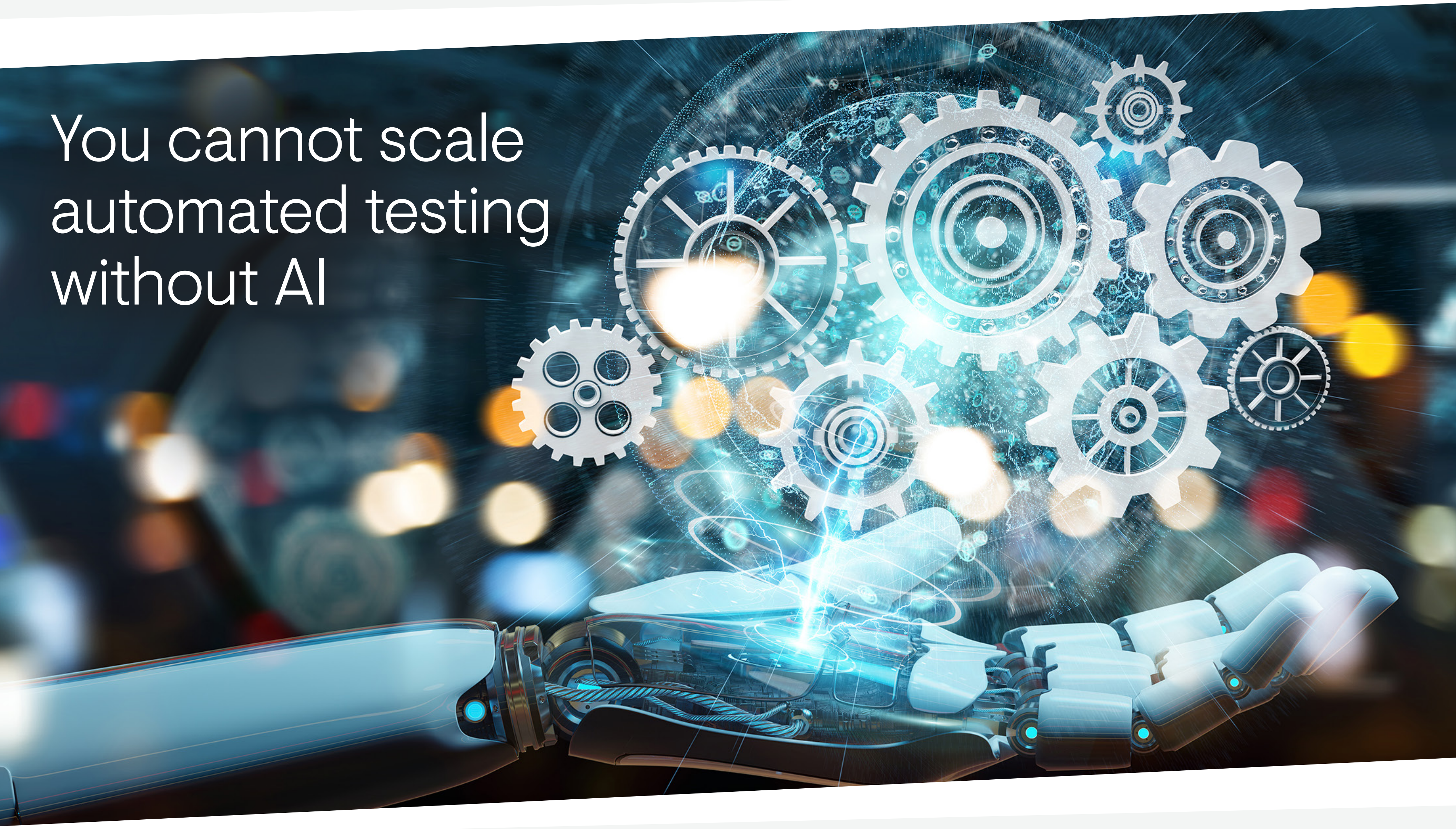
Traditional test automation tools can no longer scale to the complexities of modern software delivery.

"The automated test process would simply take too long for us to test new releases on every imaginable device."

Test Engineer, U.S. Telco



Number of posts per day to the SoftwareTesting Subreddit



You cannot scale automated testing without AI

AI in Testing in 2021

To streamline and automate parts of the test workflow while enhancing human productivity, software test teams can draw from a menu of five key AI capabilities. These can be combined and tailored to the individual organization's requirements.

Category	Description	Advantage	Impact in 2021	Expected Impact in 2024
Test Creation/Smart Crawling (e.g., test.ai, test rigor)	Automatic discovery of new and changed test requirements through the continuous analysis of changes in the application and natural language process (NLP) of documented requirements	Streamlines upfront test creation and decreases risk of coverage gaps	★☆☆	★★☆
Self-Healing (e.g., Parasoft, Testim)	Continuous and automated remediation of broken test workflows	Frees test engineers from repetitive test maintenance tasks	★★☆	★★☆
Visual Inspection (e.g., Applitools)	Training of deep learning models to inspect the application through the eyes of the end user	Provides complete and accurate coverage of the user experience. Learns and adapts to new situations without the need to write and maintain code-based rules	★★★	★★★
Coverage Detection (e.g., SeaLights.io)	Automatic detection of the different paths that end users can take through the application and reporting of gaps in code coverage	Enables end user-focused testing and optimizes testing activities	★☆☆	★★☆
Anomaly Detection (e.g., Cypress.io)	Automatic detection of system behavior that is inconsistent with the predictions of the AI/ML model	Enables auto-alerting and self-healing, and increases test engineer productivity by automatically prioritizing tasks	★★☆	★★★

Conclusion: Smart crawling, self-healing, anomaly detection, and coverage detection each are point solutions that help organizations lower their risk of blind spots while decreasing human workload. Visual inspection goes further compared to these point solutions by aiming to understand application workflows and business requirements.

6 Critical Pain Points of Test Automation

False Positives	Test Maintenance	Inefficient Feedback Process	Application Complexity	Device/Use Case Coverage	Toolchain Complexity
The inability to reliably automate all relevant test cases leads to "false positives" piling up and congesting the test process.	Traditional test automation requires continuous and ongoing updates. Frequent application changes result in the inability to efficiently deliver continuous and complete automated regression testing.	The lack of continuous feedback loops prevents automated test processes from improving. This leaves test engineers to deal with the same problems over and over.	Without the ability to prioritize the use of their limited resources, enterprises often drown in application complexity. This results in random gaps in test coverage and inconsistent testing of the UI/UX.	The continuously growing number of devices and complexity of user scenarios forces organizations to only focus on critical use cases, to prevent slowing down delivery of minor releases and "quick fixes."	A lack of integration of tests into DevOps toolchains leaves enterprises with a continuous overhead of manual integration tasks and increases the risk of issues falling through the cracks.
"We are often flooded with piles of false positives to manually review, as our automated test workflows cannot distinguish between an actual bug and an insignificant change." *Test Engineer, U.S. Automotive Manufacturer*	"Full regression does not exist." *Development Lead, U.S. Financial Services Firm*	"I love when my bug reports all come back with replies instead of fixes." *DevOps Engineer, International Mobile Payments Platform*	"In my opinion, it doesn't make sense to validate all generated messages (tens of thousands per day per Microservice)." *Test Engineer, B2B SaaS Platform*	"It is simply not feasible to test minor releases and 'quick fixes' on every potential target device and API. This would slow us down tremendously." *Test Engineer, International Financial Services Platform*	"Our test stack contains separate tools for testing UI, API, scalability, compliance, integration... Chaining all this together is not a simple task." *Test Engineer, Online Learning Platform*
"We continuously review the same type of non-issues because our system does not seem to be able to learn." *DevOps Lead, Fantasy Sports Platform*	"Regression test? Ain't nobody have time for that." *Developer, B2C E-Commerce Platform*	"I take notes in my mind to learn as much as I can from code reviews." *Test Engineer, UK Food Delivery Service*	"While we keep track of direct dependencies between microservices, we often run into integration issues." *Test Engineer, Gaming Company*	"If the issue occurs for someone crossing the country on a train, I need to know these circumstances." *DevOps Engineer, Large Civil Engineering Firm*	"As a test engineer, I'm expected to own the CI/CD toolchain." *Test Engineer, B2B SaaS platform*
"Since we cannot afford to miss anything, lots of potential problems end up on our test engineers' desks." *Development Lead, Embedded Systems Manufacturer*	"Regression testing can never fully be automated." *Developer, Publishing Company*	"You should not do many tests in production stages." *Test Engineer, Pharma Company*	"We dump all messages generated during our test process into a queue for our test engineers to sort through." *Test Lead, Enterprise Software Vendor*		"We have not had time to evaluate the API test capabilities of our existing platform, but we just bought a new tool just for API testing." *Product Owner, B2C E-Commerce Platform*

Overloading Test Engineers with False Positives

Ideal Scenario

Test teams create all test cases based on exact knowledge of how the different end-user personas will use the software.

In Real Life

Test engineers often do not have sufficient application knowledge or time to write robust test workflows that can reliably differentiate between bugs and insignificant changes. Instead, they tag most or all deviations for human review, slowing down the release process and adding overhead cost.

How AI/ML Can Help

AI-driven visual inspection (visual AI) can continuously map and monitor usage patterns for test engineers to understand normal end-user behavior.

AI can learn to distinguish between desired and undesired application behavior by observing changes in the volume, content, and severity of coinciding support tickets, by identifying changes in application error logs, or by directly correlating the number of completed end-user transactions with changes in application behavior.

AI can learn the impact of normal application use on the individual layers of the overall application stack. If the application starts consuming substantially more resources despite usage patterns staying mostly the same, the AI will alert test engineers of potential issues in the application code.

Continuously Maintaining Regression Testing

Ideal Scenario

Each new release has a complete set of regression tests to completely prevent any negative interaction between old and new code.

In Real Life

Rule-based regression tests are labor-intensive because they require test engineers to continuously adjust their test code based on the potential impact of changes to the underlying application code. All application workflows need to be retested for each target device, browser, screen resolution, and input device, then retested under realistic network latency and potential temporary connection outages.

How AI/ML Can Help

Self-healing enables runtime remediation of broken locator-based navigation workflows.

Automated maintenance of test results based on AI analysis, and replication, of human interactions with test results.

As AI-driven visual inspection (visual AI) provides complete and accurate regression of the UI, development teams spend less time managing test code and are able to complete a full set of regression tests before each new code release. Ultimately, AI-driven visual inspection could learn and evaluate the desired input and output values of the application, based on the analysis of real-life end-user transactions. This would allow the visual AI to automatically find bugs in the underlying application code.

The analysis of real-life transaction data allows the AI to alert humans of problems with actual revenue impact while deprioritizing issues that do not seem to negatively influence end-user behavior.

Inefficient Feedback Process

Ideal Scenario

Test workflows provide the guardrails needed for combining human strengths with the strengths of test automation. Test engineers are able to easily access feedback on all bug reports, code reviews, and all workflow and compliance requirements in order to fully understand all relevant context. This understanding is the basis for optimal test efficiency.

In Real Life

Test engineers often lack structure, clear instructions, and the ability to learn end-user requirements. This leads to unnecessary email chains, a lack of knowledge retention, and overall poor test quality.

How AI/ML Can Help

AI-driven workflows can continuously monitor bug reports, code reviews, and other relevant context factors to provide test engineers with alerts and situational context for enhanced productivity.

AI can identify and consistently implement knowledge, processes, test parameters, and remediation scripts that perform well across teams.

AI-driven visual inspection (visual AI) can alert test engineers of compliance requirements, deviations from agreed upon UI/UX standards, and potential end-user experience issues identified through the analysis of log data and performance metrics from the rendered UI.

Application Complexity

Ideal Scenario

Each new release of any microservice is tested within its entire (multi) application context, because in modern distributed applications any change to the code, state, or data of any one microservice can affect the functioning of any of the other microservices and ultimately the end-user experience of the application they are part of.

In Real Life

Without the ability to prioritize the use of their limited resources, enterprises often drown in application complexity, because when microservices are shared between multiple applications, it becomes even trickier to keep track of actual and potential dependencies. This can result in random gaps in test coverage.

How AI/ML Can Help

AI can identify the impact of changes in the code, application stack, and external dependencies on the end-user experience, simply by correlating application logs and metrics with the results of visual UI inspection.

Device Use Case Coverage

Ideal Scenario

New application releases are tested on all target devices.

In Real Life

By the end of 2021, there will be approximately 250 additional Android smartphones and numerous additional Apple devices in the marketplace. Combining this number with the count of older devices still in use means over 1,000 different devices. Most organizations only test their applications for a few of the most popular devices in order not to slow down their development lifecycle. This often leads to user experience issues due to applications displaying incorrectly for different browsers and browser versions, screen sizes, and device configurations.

How AI/ML Can Help

AI can learn critical use cases and devices to help test engineers prioritize efforts and can detect common problems of specific use cases on certain types of devices to alert test engineers of potential issues. AI can also alert test engineers when user experience degrades below an acceptable level.

By focusing on the standardized functional execution across devices/platforms, AI-driven visual inspection can be leveraged at scale to enable complete cross-browser/cross-device validation.

Toolchain Complexity

Ideal Scenario

The test toolchain integrates with the CI/CD pipeline to continuously provide the services needed to pass the current release gates. Passing these gates includes a combination of completed tests from all areas: UI, API, scalability, compliance, integration, performance, stress, scalability, etc. Testing along the lines of user workflows is more efficient, faster, and more reliable than testing these aspects separately.

In Real Life

A lack of integration of test processes and platforms into DevOps toolchains leaves enterprises with a continuous overhead of manual integration tasks and increases the risk of issues falling through the cracks.

How AI/ML Can Help

AI can coordinate between different test suites to align the testing process with end-to-end business workflows and replace the need for certain specialized tools by testing functionality through the front end.

AI can also overcome differences in development languages by offering a declarative approach toward specifying desired inputs and outputs at the interface layer.

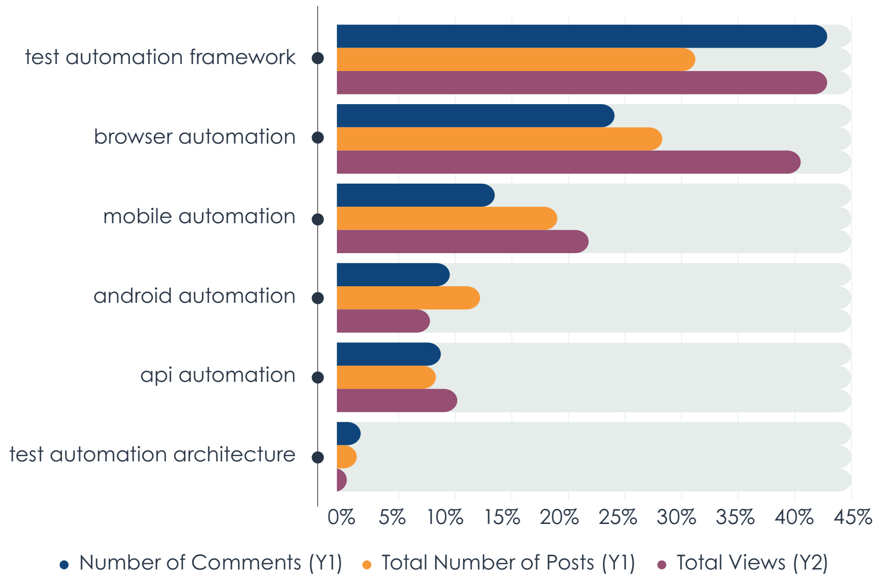
EMA Perspective

AI-based test automation technologies can deliver real ROI today and have the potential to address, and ultimately eliminate, today's critical automation bottlenecks.

Currently, organizations face the challenge of cutting through the complexity of test automation frameworks that only attack the same problem in a slightly different way, not from a different angle. To be successful, organizations need to select automation platforms that can curb release overhead through a combination of automatic issue qualification, prioritization, resolutions, and assignment to human test engineers when necessary. This includes the essential requirement of eliminating today's flood of false positives overloading test engineering teams. This requires a new kind of test automation platform that can understand developer intent to automatically determine the significance of unexpected test outcomes.

At the core of AI-enabled test automation is the idea of decision-making based on human intent in combination with large historical datasets that provide the required decision context. Humans declare their intent through test code that describes the desired state of the tested application in response to human and machine interactions. While this may sound like a technical detail, everyone involved in software testing must understand the importance of this declarative approach because it lays the technical foundation for scalable testing within complex and rapidly scaling environments. For example, instead of creating a set of manual rules that exactly define the language and layout of an order confirmation, users can simply declare the requirement for a standard confirmation screen that at least includes the customer name and an order number. All other development teams can then use the same declarative statement to ensure speed and consistency, without adding cost.

Of the currently available applications of AI to software testing, AI-driven visual inspection has the highest impact. This discipline aims to provide test engineers with an additional "pair of eyes," leaving them to focus on areas that really need human intelligence. It provides humans with the contextual information needed to accelerate their test and remediation efforts, recommending solutions wherever necessary and remembering human decisions, and actions, so they can be automated and shared across the organization.



The seven test automation challenges most discussed on the StackExchange Software Quality Assurance engineering forum in 1H 2021