PARASOFT®

WHITEPAPER

Guide for AI in Software Testing

INTRODUCTION

Artificial intelligence is one of the digital marketplace's most overused buzzwords. The term "AI" conjures up images of Alexa or Siri, computer chess opponents, and self-driving cars. If we believe Hollywood, AI will evolve into all-powerful supercomputers bent on human destruction.

Contrary to fiction, AI is transforming human experiences positively. It can help humans in a variety of ways, including reducing errors and automating repetitive tasks.

Software test automation tools are maturing and have incorporated AI and machine learning (ML) technology. Let's evaluate the role of AI in software testing and investigate:

» How software development teams use AI testing.

» How to leverage AI to improve productivity, quality, and security.

THE HYPE OF AI VERSUS REALITY

It’s important to realize that the scope of what is considered AI changes over time. The Turing test was created as a way to help define whether a computer or program had achieved real intelligence. To date, this test hasn’t been passed, so effectively, true AI doesn’t exist yet. However, there's no reason to not use the AI and ML technologies we've invented along the way.

Most people are accustomed to question-and-answer responses from Siri and Alexa these days, but not everyone considers this AI. Or, at the very least, these aren’t considered to be state of the art.

The same will happen with software tools. Innovations in automation today will be expected as new capabilities evolve. However, let’s consider what is meant by using AI technology in software test automation tools today.

Teams at companies like Parasoft use AI to make jobs easier and to assist in efforts to deliver top-quality projects quickly and contribute to the organization's overall success. The objective is not to replace a tester or insert a robot coder. Not at all. The emphasis is on seeking a solution.

The key point that separates the hype of AI from reality is that AI is not magic or the silver bullet promised with every new generation of tools. However, AI and ML do offer impressive enhancements to software testing tools.

SOFTWARE CHALLENGES THAT AI & ML HELP SOLVE

The application of AI in software testing tools focuses on making the software development life cycle (SDLC) easier. Teams can use AI to help automate and reduce the amount of mundane and tedious tasks in development and testing by applying reasoning, problem solving, and, in some cases, ML.

You may wonder, "Don’t test automation tools do this already?" Test automation tools already use AI technology, but they have limitations.

Where AI shines in software testing is when it’s applied to remove those limitations, enabling software test automation tools to provide even more value to developers and testers. The value of AI comes from reducing the direct involvement of the developer or tester in the most mundane tasks. We still have a great need for human intelligence in applying business logic, strategic thinking, creative ideas, and the like.

For example, consider that most, if not all, test automation tools run tests and deliver results. Most don’t know which tests to run, so they run all of them or some predetermined set of them.

What if an AI-enabled bot could review the current state of test statuses, recent code changes, new or changing requirements, code coverage, and other metrics, and then decide which tests to run and run them for you?

Bringing in decision-making that adapts to change, then making decisions without the direct intervention of a human, is an example of applying AI.

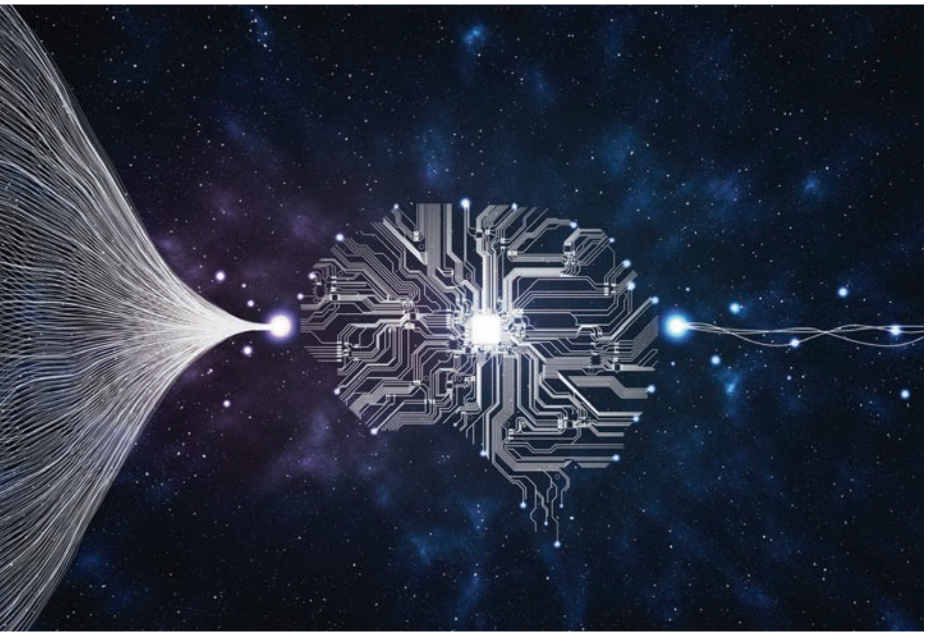
MORE SOFTWARE, MORE RELEASES

As worldwide demand for software continues to surge, the demand for developers increases. More software and more developers mean more software releases are expected to hit the market. A recent report by Statista corroborates this expectation with a projection that suggests that the global developer population is expected to increase from 24.5 million in 2020 to 28.7 million by 2024.

The exponential growth of software and its constantly shifting areas of application equates to increased load on software teams, developers, and testers. Automating testing is a practical way to help deal with this influx of software demand. All this software needs to be tested and it can't all be done manually.

Software testing is the process of subjecting a software infrastructure to a series of functional and nonfunctional testing scenarios. It's a process of evaluating software to ensure that it can do what it's designed to do correctly and efficiently. When teams test software, they can discover and resolve runtime defects, scalability issues, security vulnerabilities, and more.

The software testing process is usually rigorous, hence the need for automation. However, for software automation to be super-efficient and seamless, there's a need to incorporate AI.



HOW ML ENHANCES AI

ML can augment AI by applying algorithms that allow the tool to improve automatically by collecting copious amounts of data produced by testing.

ML research is a subset of overall AI research with a focus on decision-making management based on previously observed data. This is an important aspect of AI overall, as intelligence requires modifying decision-making as learning improves. In software testing tools, though, ML isn't always necessary. Sometimes an AI-enabled tool is best manually fine-tuned to suit the organization using the tool, and then the same logic and reasoning can be applied every time, regardless of the outcome.

In other cases, data collection is key to the decision-making process, and ML can be extremely valuable, requiring some data initially and then improving or adapting as more data is collected. For example, over time, code coverage, static analysis results, test results, or other software metrics can inform the AI about the state of the software project.

THE OPPORTUNITY FOR AI & ML IN SOFTWARE TESTING

The opportunity for using AI and ML technology is where it makes the most impact in software testing. Specifically, focusing this technology on key areas.

- Easier and more efficient test creation, expansion, and completeness.
- Reducing test maintenance.
- Adapting to change with self-healing tests.
- Decreasing execution time with intelligent test selection.
- Reducing noise and increasing relevance.
- Reducing business risk.

APPLYING AI & ML ACROSS THE SDLC

The ideal test pyramid defines where it's best to invest time and effort in a project. The test pyramid outlines the relative investment at each phase of testing. Investing early effort into a foundation of good code quality and a comprehensive suite of unit tests is fundamental and forms the foundation of the pyramid.

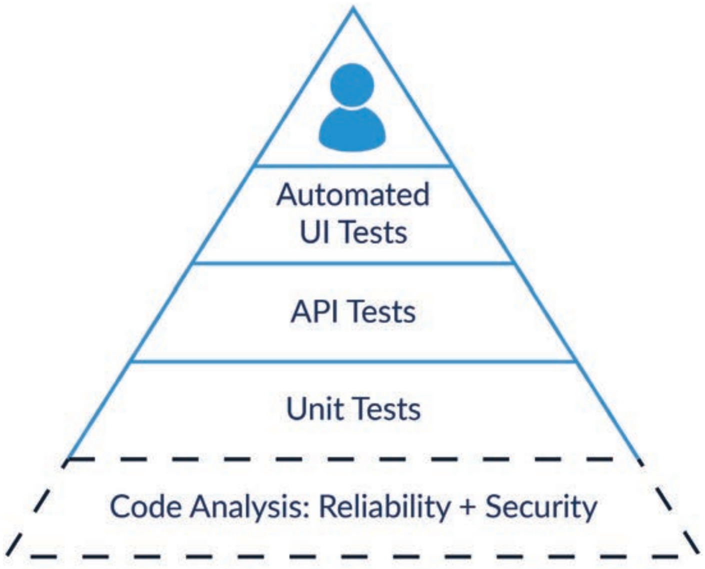


Figure 1: The ideal test pyramid is built on a foundation of high quality code and unit testing.

Further along in the SDLC, API and service testing for functional testing will shift QA away from the traditional emphasis on manual UI testing. Beyond that, the focus is on automating as many of the remaining UI tests as possible. A smaller number of system and GUI-based tests focused on essential tasks such as usability and customer satisfaction testing are located at the top of the pyramid.

Since testing and development resources are finite, there's a need to make testing more efficient while increasing coverage to do more with the same. Focusing testing on exactly what needs to be validated after each code change is critical to accelerating testing, enabling continuous testing, and meeting delivery goals.

AI and ML play a key role in providing the data needed by test automation tools to focus testing while removing many of the tedious, error-prone, and mundane tasks.

- **Improve the adoption of static analysis.** Successful adoption of static analysis tools is often difficult or abandoned due to a large number of warnings and the perception that these warnings are false positives. AI and ML group and prioritize the findings reported by static analysis to improve the user experience and adoption of such tools.

- **Improve unit test creation.** Unit testing is seen as necessary but tedious and adds to the developer's workload with more coding and more maintenance. AI reduces tedium and errors with automated unit test creation from existing code and provides assistance to improve and reuse existing tests.
- **Reduce test maintenance.** Unit tests are more code that needs fixing and updating as the application changes. For developers, unit testing often seems like more work with poor results. AI reduces test maintenance and proactively helps identify tests that need updating.
- **Reduce test execution.** Testing takes time—more time than is provided for in most development schedules. A lot of testing time is spent waiting for results from large test suites. It's often difficult to know what to test when changes are made, and regression testing tends to be "all or nothing" to make sure nothing is missed. AI can help identify which code needs to be retested, including dependencies, in the application and only execute those associated tests.
- **Increase automation of API testing.** Functional testing is more efficient and portable at the API layer, but teams struggle to create and run enough API tests. In most cases, it's insufficient to relieve the testing burden from manual and automated UI testing. AI helps generate meaningful and reusable API tests from monitoring of API traffic during existing UI testing and during API usage in general.
- **Improve UI test automation.** Just as software organizations struggle with all levels of testing, automating UI testing also remains a key challenge. Many developers use Selenium tests for UI automation, but this solution and other UI test automation tools require lots of maintenance and attention. AI can help improve automated UI testing by making tests more adaptable to change and self-healing broken tests due to UI changes.

REAL EXAMPLES OF AI AND ML IN SOFTWARE TESTING

The following are examples of what happens when you apply AI and ML technology to software testing problems.

- Improve static analysis adoption.
- Increase unit testing coverage.
- Improve API testing.
- Automate UI testing efficiently.
- Remove redundant work with smart test execution.

IMPROVE STATIC ANALYSIS ADOPTION

Static analysis prevents and detects bugs and security vulnerabilities in code as soon as it is written, without the need for execution. These tools prevent software errors early on during the development of an application and are essential to ensuring quality and security. In some industries, coding standards, like ISO 26262, SEI CERT, and more, are required to enhance the safety and security of the source code. Static analysis tools are critical in automating the enforcement of these standards.

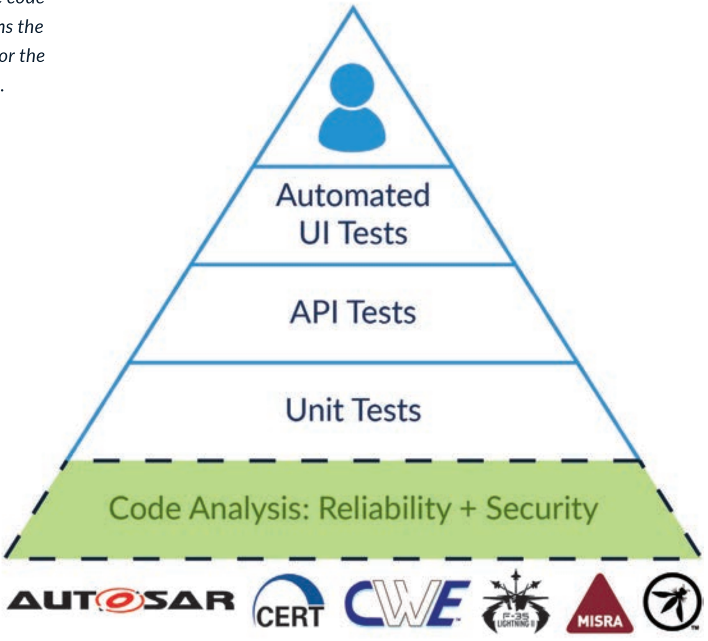


Figure 2: Static source code analysis forms the foundation for the test pyramid.

Software teams often struggle with the initial results they get from static analysis, especially when using the tools for the first time. The number of standards, rules/checkers, recommendations, and metrics that are possible with modern static analysis tools can be overwhelming. It often turns teams off enough that they won't put forth further effort.

Software development teams have unique quality requirements. Each team has its own definition of a false positive, often meaning "don't care" rather than "this is technically incorrect." One solution is to apply AI and ML to prioritize the findings reported by static analysis to improve the user experience and adoption of such tools.

Teams can use AI to quickly classify the findings in the output of a static analysis tool as either something they want to see or something they want to suppress. The tool does this by reviewing a subset of findings and constructing a classifier based on the metadata associated with those findings.

This classifier is based on results from previous classifications of static analysis findings in the context of both historical suppression of irrelevant warnings and prioritization of meaningful findings to fix inside the codebase.

The end results are classified in two ways.

1. Of interest for the team to investigate.
2. Items that can be suppressed.

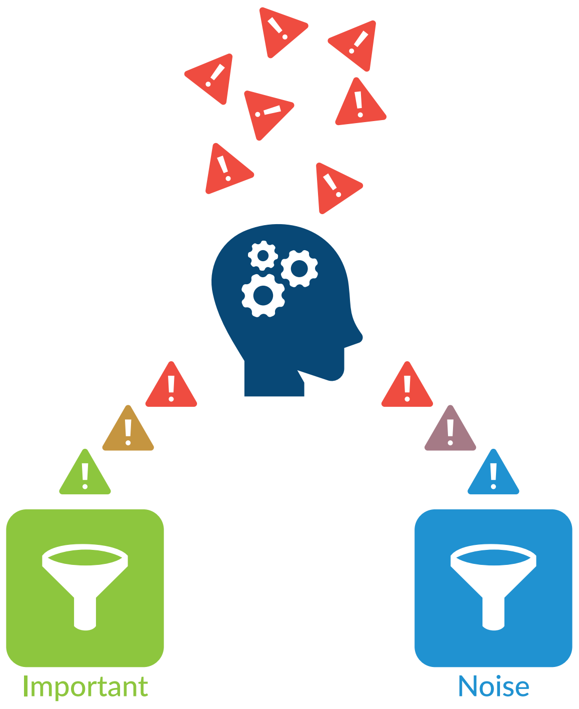


Figure 3: AI can help filter less important static analysis warnings. These warnings are suppressed rather than deleted for analysis later.

Developers can focus on important code issues before they propagate throughout the code base. However, for safety-critical development, violations that are considered noise will eventually need to be reviewed.

AI can be used to perform hotspot detection. A hot spot is a group of violations that have a single common "root of the problem," which causes all those violations to be detected in different places of the code. Each execution path leading to violations must go through that shared "root cause" line of the code. The value of assigning a hotspot cluster to a developer is that they can fix several issues with a single piece of code.

Leveraging the development team's individual strengths, AI can be used to assign violations from the unordered queue according to each developer's skills. Developers get recommendations that match their personal skills and preferences and become more productive by working on the problems they feel comfortable with.

While fixing violations, developer preferences are determined by the previous violation selection and assignments. Future violations are grouped into separate queues "recommended" for individual developers. Combining preferences and hot spot detection, static analysis results can be better organized for an optimal response from the development team.

An AI model can be used to vectorize code snippets around all violations. These code vectors are easy to compare, so when a specific violation is fixed, a search is done for other violations that are reported for "similar" code based on that comparison. The developer is advised to look for warnings inside recommended code to avoid context switching between completely different sources. Since developers are already familiar with the code and the fix, they spend less time analyzing and understanding the source code.

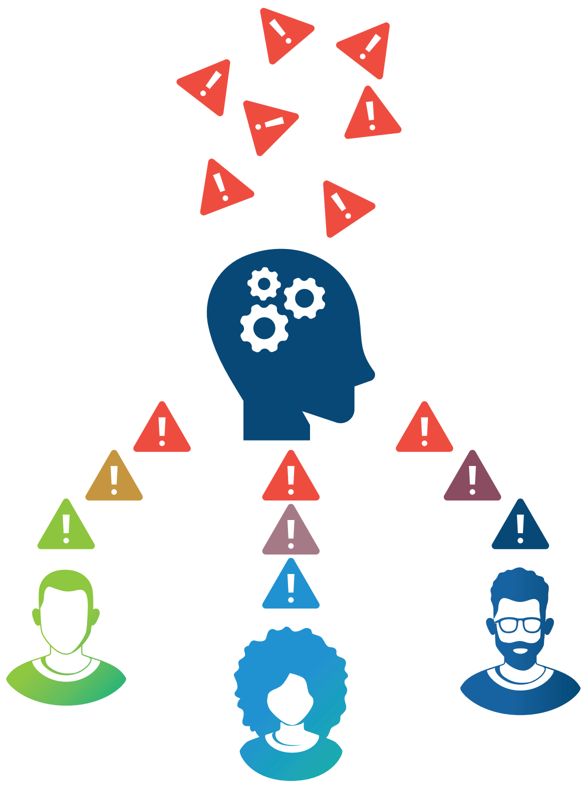


*Figure 4:
AI can assign static analysis warnings based on developer skills.*

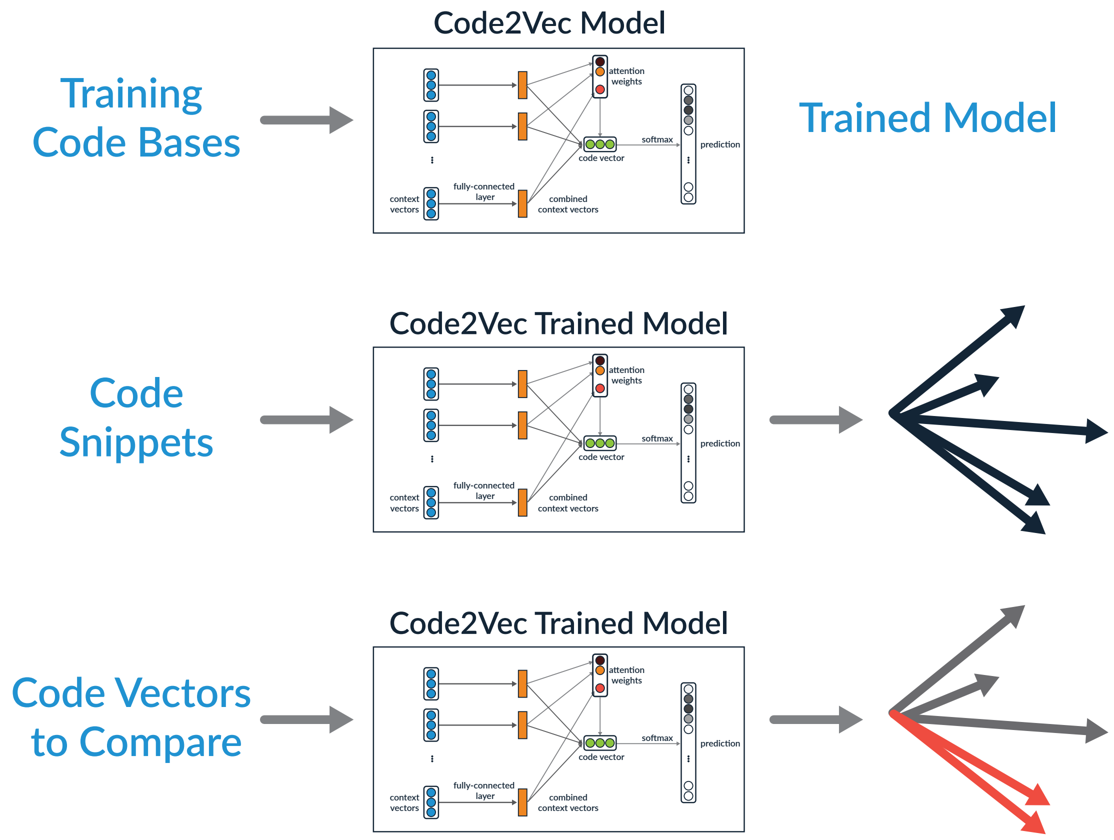


*Figure 5:
AI helps to cluster static analysis warnings based on similarity of the code.*

The application of AI benefits the entire static analysis workflow. The overall workload is reduced by filtering out ignored violations and detecting hotspots if they exist. The remaining to do list of violations is distributed to individual developers based on their skills.

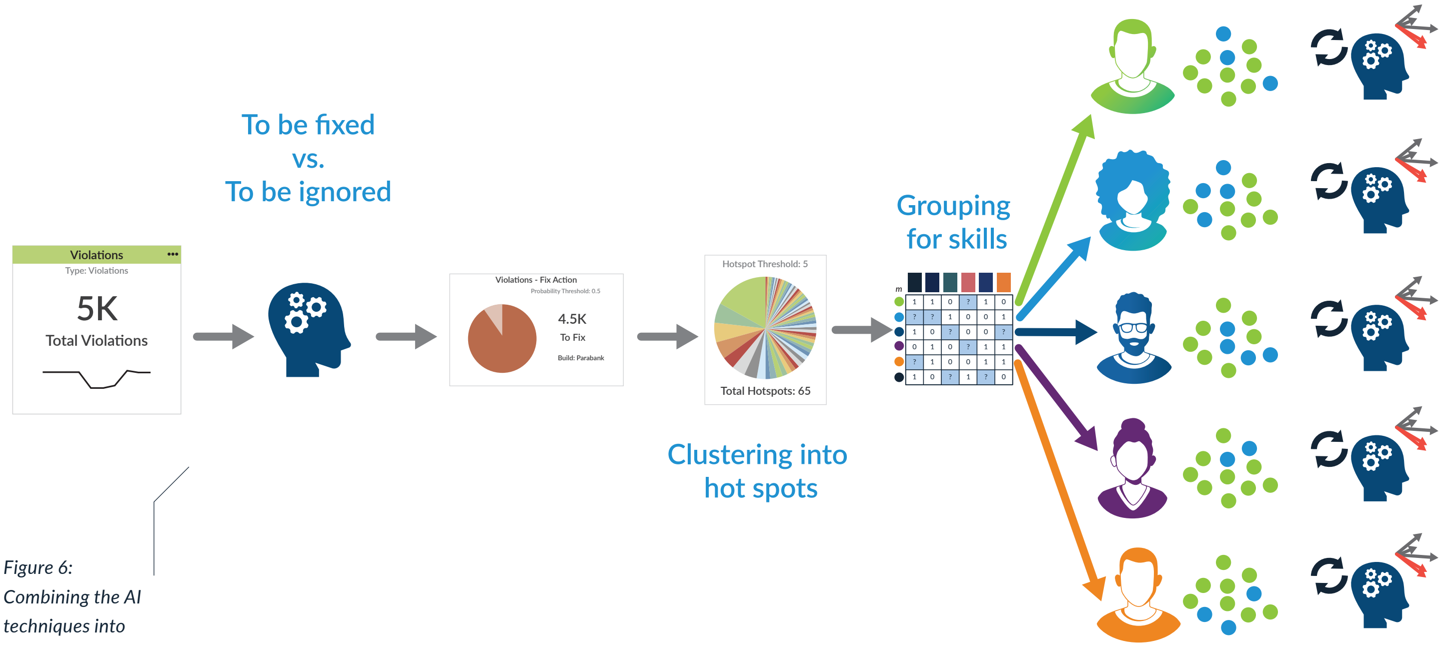


*Figure 6:
Combining the AI techniques into the static analysis remediation workflow results in an optimum workload for the team.*

INCREASE UNIT TESTING COVERAGE

The next layer of the test pyramid recommends a solid foundation of unit tests. The implication is that software teams place substantial effort, schedule, and budget into testing at this early stage rather than leaving it until integration or UI testing.

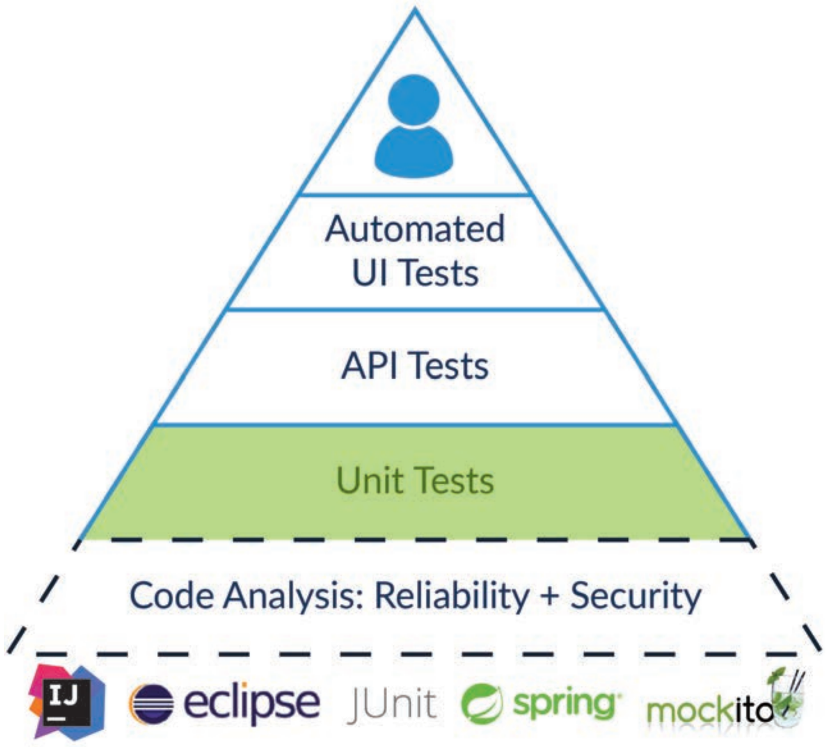


Figure 7:
The test pyramid recommends investing in a foundation of unit tests to shift left testing as code is being developed.

However, there are barriers to unit testing that many teams struggle to overcome, which can easily take up 40% of development time.

» Making meaningful, isolated unit tests with appropriate stubs and mocks, as well as with the necessary assertions in the code.

» Increasing code coverage following the company's policy to ensure the application is sufficiently tested.

» Maintaining unit tests with continuous changes of codebase.

Unit testing is viewed as time-consuming, tedious, and overwhelming by the majority of developers. It doesn't have to be this way.

Unit test creation in particular is a difficult task since it can be time-consuming to create unique tests that fully test a unit. One way to alleviate this is by making it easier to create stubs and mocks with assisted test creation for better isolation of the code under test. AI can be used to analyze the unit under test to determine its dependencies on other classes and then suggest mocking them during assisted test creation to the user in order to create more isolated tests.

Another key problem area for unit testing is covering enough of the code so it can be considered fully tested. This usually means executing every path through a unit and exercising the decision logic within.

AI can help address the problem by detecting code that isn't covered by existing test suites by traversing the control path of the source code to figure out which parameters need to be passed into a method under test and how stubs or mocks need to be initialized to reach uncovered code. This makes it possible to automatically generate new unit tests, applying modified parameters to increase the overall code coverage of the entire project.

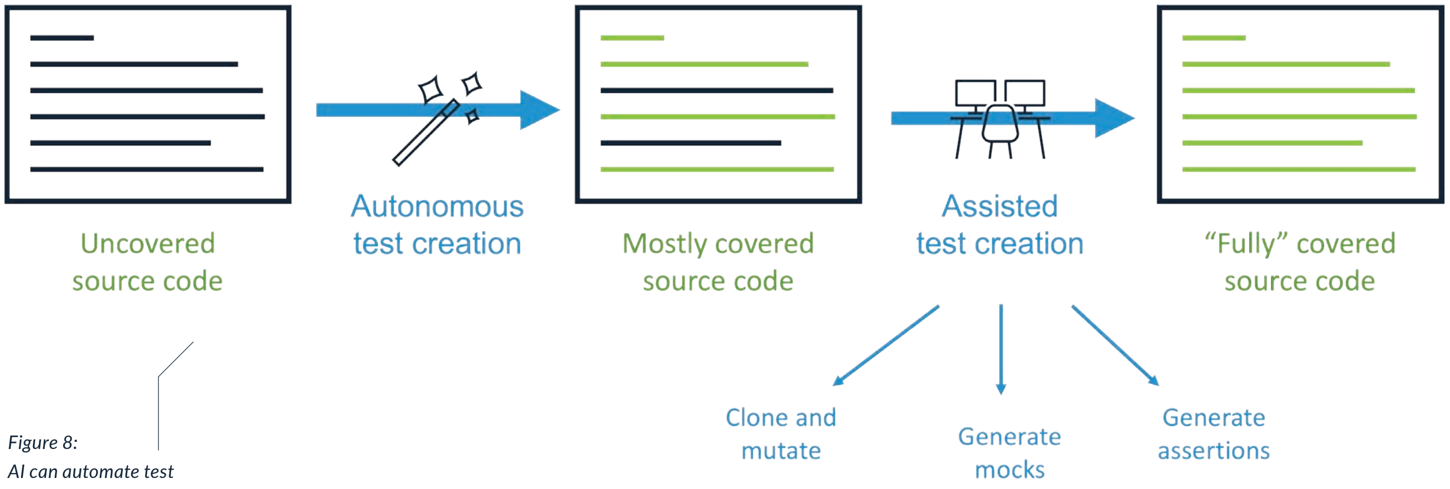


Figure 8:
AI can automate test creation to a fairly high level of coverage. However, test creation assistance brings coverage to a new, higher level.

The capabilities of AI in producing tests from code are impressive. However, it's up to the developers to continuously invest in and build their own tests. Again, using AI test creation assistance, developers can extend code coverage through clones and mutations, create the mocks, and auto-generate assertions. Combining these approaches makes it possible to achieve a high level of code coverage with much less manual effort.

These capabilities are part of our Parasoft Jtest product, which is a comprehensive testing solution for Java developers that includes static analysis, unit testing, coverage, traceability, and so on. Parasoft Jtest's IDE plugins add useful automation to the unit testing practice with easy one-click actions for creating, scaling, and maintaining unit tests. With Jtest, users can achieve higher code coverage with the help of AI while significantly cutting the time and effort required to build a comprehensive and meaningful suite of JUnit test cases.

IMPROVE API TESTING

The struggle to improve API testing traditionally relied on the expertise and motivation of the development team since APIs are often outside the realm of QA. Moreover, APIs are sometimes poorly documented and creating tests for them is difficult and time consuming.

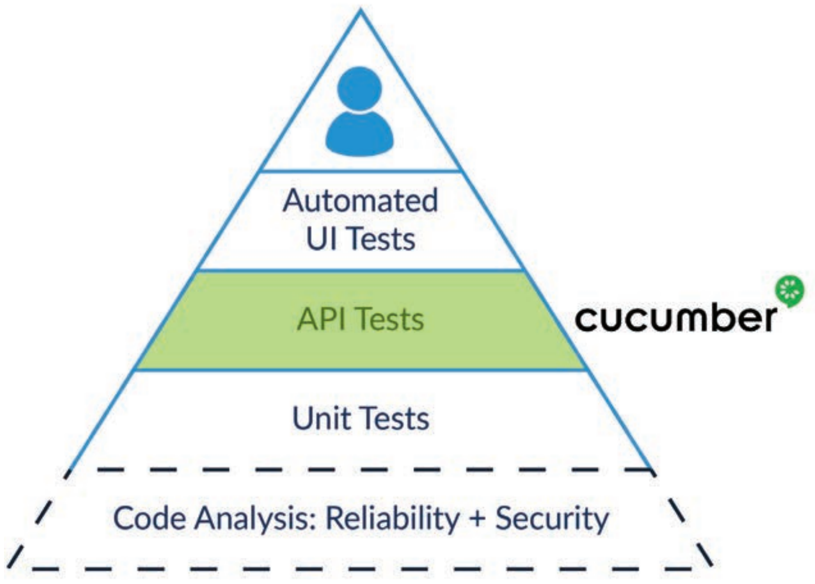


Figure 9:
API testing is an important part of a test strategy. The aim is to move the functional application testing away from the reliance on manual and UI testing to automated API tests.

When it comes to API testing, AI and ML aim to accomplish the following.

» Increase functional coverage with API and service layer testing.

» Make it easier to automate and quicker to execute.

» Reuse the results for load and performance testing.

The solution is an API smart test generation technology that analyzes the API traffic observed and recorded during execution of manual UI tests. However, it goes beyond record-and-playback by leveraging AI and ML to convert UI tests into complete, automated API test scenarios.

A smart API test generator uses reasoning to understand the patterns and relationships in the different API calls made while exercising the UI. From that analysis, a series of API calls is constructed that represent the underlying interface calls made during the UI flow.

Then, ML is applied by observing interactions within the different API resources and storing them as templates in a proprietary data structure. This internal structure is updated by examining other test cases in the user's library to learn different types of behavior when exercising the tests, for example, an assertion or adding a particular header at the right spot.

The goal of AI here is to create more advanced parameterized tests, not just repeat what the user was doing, as you get with simple record-and-playback testing. Here's how the smart API test generator works.

1. Recognizes patterns inside the traffic.
2. Creates a comprehensive data model of observed parameters.
3. Generates and enables automated API tests by applying learned patterns to other API tests to enhance them and help users create more advanced automated test scenarios.

The resulting automated API tests are more complete, reusable, scalable, and resilient to change.

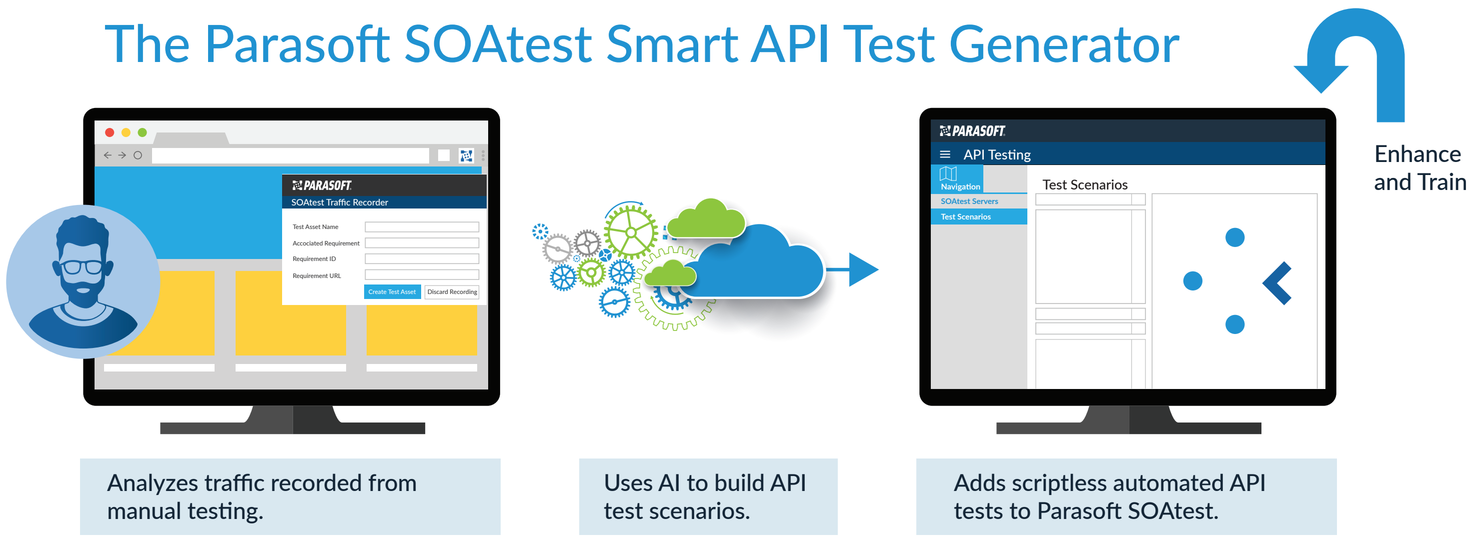


Figure 10: The smart API test generator analyzes traffic recorded during UI testing to build test scenarios that can be expanded and reused.

AI-driven API testing can be expanded to other areas of testing, such as security. It is possible to incorporate penetration testing as part of the shift left of API testing by extending the API test automation tools with industry standard tools such as OWASP ZAP or Burp Suite.

During API testing, it's easy to enable penetration testing in parallel. Security-specific results are captured for the security team to analyze and triage. The security team can feed the analysis back to the developers to fix. The process flow is shown below.

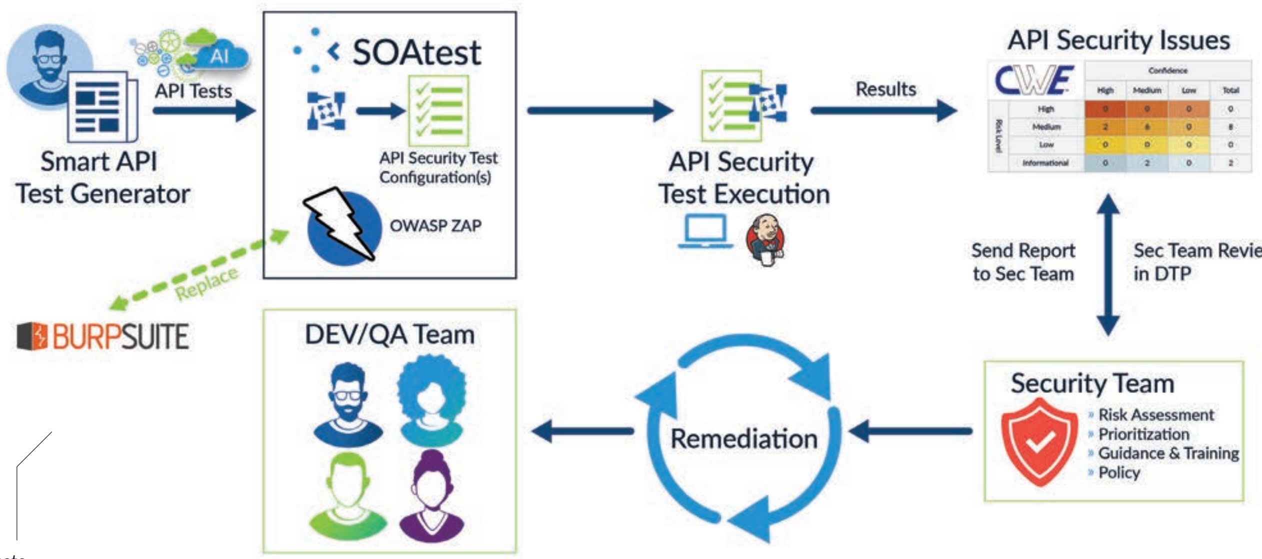


Figure 11: Generated API tests can be extended with OWASP ZAP or Burp Suite to create pen tests that are run in parallel.

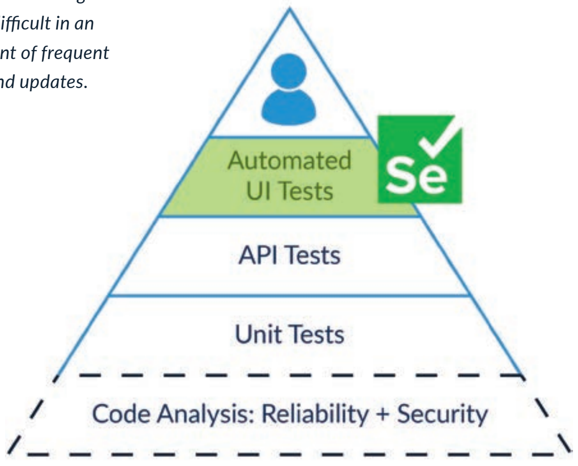
AUTOMATE UI TESTING EFFICIENTLY

Validating the application's functionality with UI testing is another critical component of your testing strategy to ensure that the product is fully verified before going into production. The Selenium UI test automation framework is widely adopted for UI testing, but users still struggle with the common Selenium testing challenges of maintainability and stability. This is where AI technologies and, particularly, ML can help by:

» Creating tests from recording the user interaction within the browser during manual UI testing.

» Providing self-healing capabilities during runtime execution to address the common maintainability problems associated with UI testing.

Figure 12: Automated UI testing reduces the reliance on manual UI verification. However, automating UI testing is difficult in an environment of frequent changes and updates.



AI can learn about internal data structures during the regular execution of Selenium tests. Parasoft Selenic does this by monitoring each test run and capturing detailed information about the web UI content of the application under test. It extracts DOM elements, their attributes, locators, and the like, and correlates them with actions performed by UI-driven tests. Selenic employs a proprietary data modeling approach, storing that information inside its AI engine. The model is updated continuously, analyzing the historical execution of all tests to continue building the knowledgebase.

Self-healing of tests is a critical time-saver in cases when UI elements of web pages are moved or modified, causing tests to fail. The AI heuristics used by the engine can match those changed elements with historical data represented by the model and automatically generate "smart locators" that are resistant to changes to recover execution of Selenium tests at runtime. Information about these changes is automatically propagated through the model, and the future generation of new locators is adjusted based on those changes.

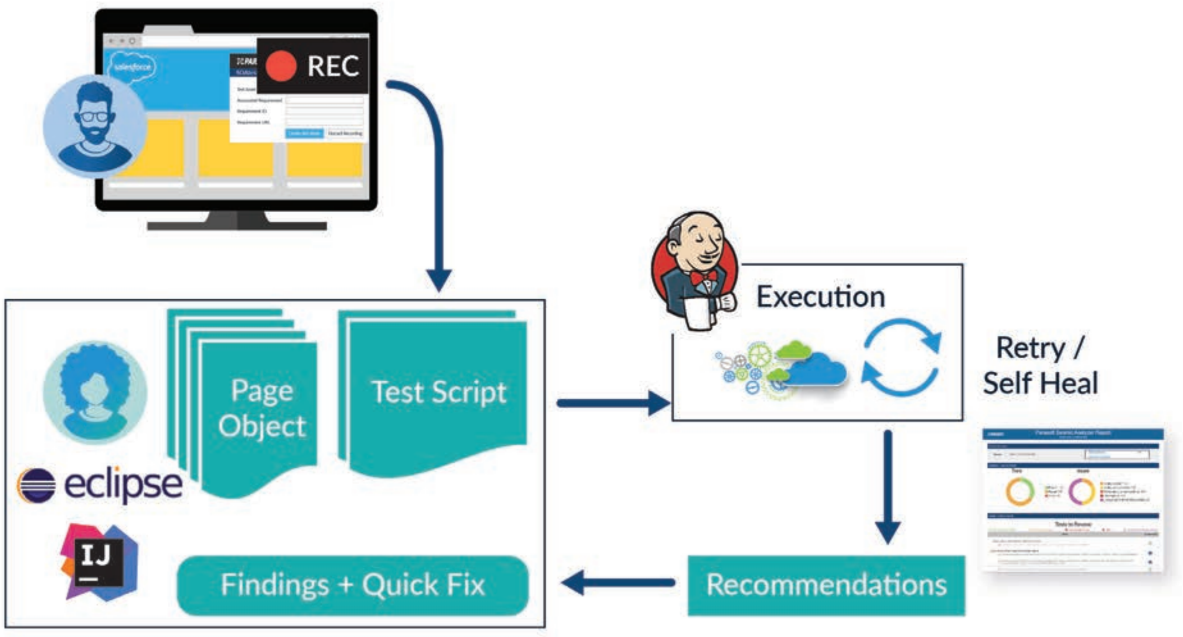


Figure 13:
AI increases efficiency of Selenium-based UI test automation with self-healing applied during execution.

In addition to this, Selenic can self-heal different types of waiting conditions, addressing instabilities associated with the performance characteristics of systems under test. It also measures the time associated with running each test case on web pages and compares it to the historical average captured from the previous runs. In cases where deviation exceeds a certain threshold, an alert is flagged inside the report to notify the user of significant changes.

REMOVE REDUNDANT WORK WITH SMART TEST EXECUTION

Test impact analysis (TIA) assesses the impact of changes made to production code. It helps to reveal test cases affected by code changes. The primary benefit of TIA is that it eliminates the need to run tests on your entire code base after modifications have been made. This saves time and costs while keeping your development process running efficiently.

There's also benefit from TIA technology during the execution of manual tests. Or you can leverage the integration of TIA-based tools with CI/CD pipelines. This can optimize the run of your automated tests and provide faster feedback to developers about the impact of changes on the quality of their projects. The analysis and tests selection are available to optimize the execution of unit tests, API tests, and Selenium web UI tests.

Plan Testing Based on Impacted Requirements

To prioritize test activities, a correlation from tests to business requirements is required. Keeping track of user stories and requirements provides an important view, in real-time, into the quality of the value stream. User stories and requirements should be reviewed for priority.

Test tools provide the traceability capabilities needed to plan execution for tests that validate items being worked on in-sprint. However, more is required since it's unclear how recent changes have impacted the code.

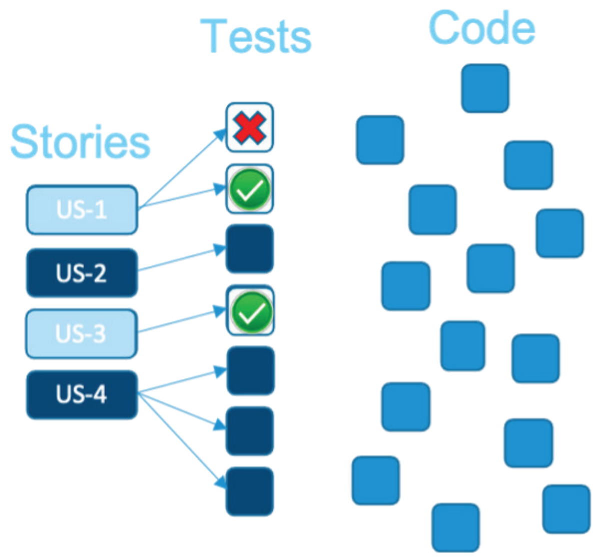


Figure 14:
Prioritizing tests based on impacted user stories is the first step to optimizing test execution.

Use Test Impact Analysis to Validate Only What Has Changed

To fully optimize test execution, it's necessary to understand the code that each test covers and then determine the code that has changed. Test tools provide this capability, which is the central repository for test results and analysis. Test impact analysis allows testers to focus only on the tests that validate the changes.

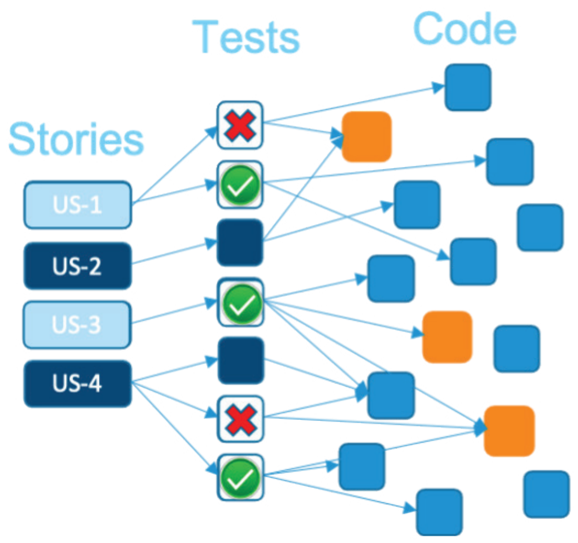


Figure 15:
Test impact analysis determines which code has actually changed to focus testing only on what should be tested.

Regardless of the test practice used as part of CI/CD pipelines, all the results are pushed into a centralized reporting and analytical system. This not only enables test impact analysis but also data display in terms of sophisticated, customizable dashboards. Integrations are supported with external tools such as test management and requirements management systems, including codebeamer, Polarion, and Jira.

In addition, data such as requirements coming from other external systems is used to generate traceability reports, creating associations between requirements, tests executed, modified files, and test results. This is illustrated in the flow diagram below.

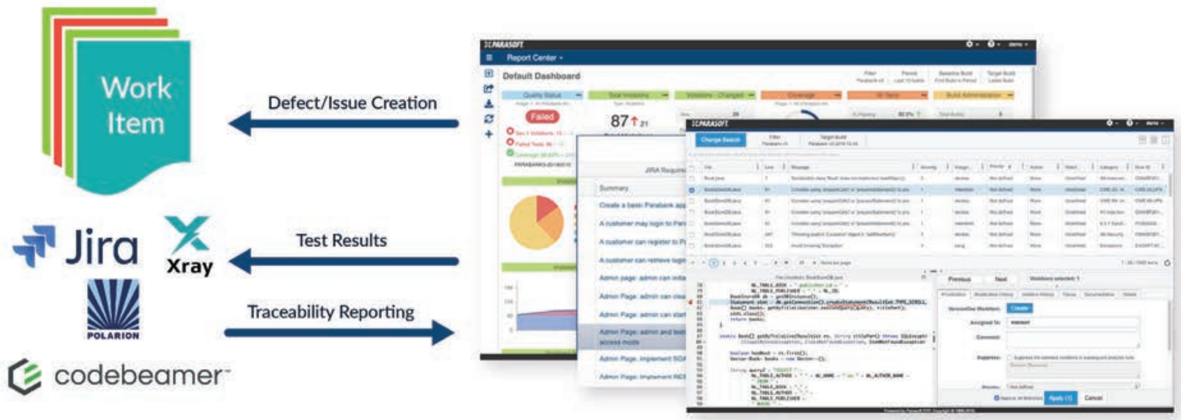


Figure 16:
A reporting and analytical system can integrate with various requirements management and test management systems.

An AI and ML enhanced reporting and analytical system provides more than test impact analysis and smart test execution. There are numerous capabilities that provide a holistic view of quality of the projects such as:

» Modified code coverage

» Risky code coverage

» Test stability reporting

» KPI calculations

» Custom analytics and derived metrics

BENEFITS OF AI/ML IN SOFTWARE TESTING

AI and ML provide benefits throughout the SDLC and among the various tools that assist at each of these levels. Most importantly these technologies amplify the effectiveness of tools with:

- More efficiency and productivity
- Lower costs and test time
- High quality deliverables

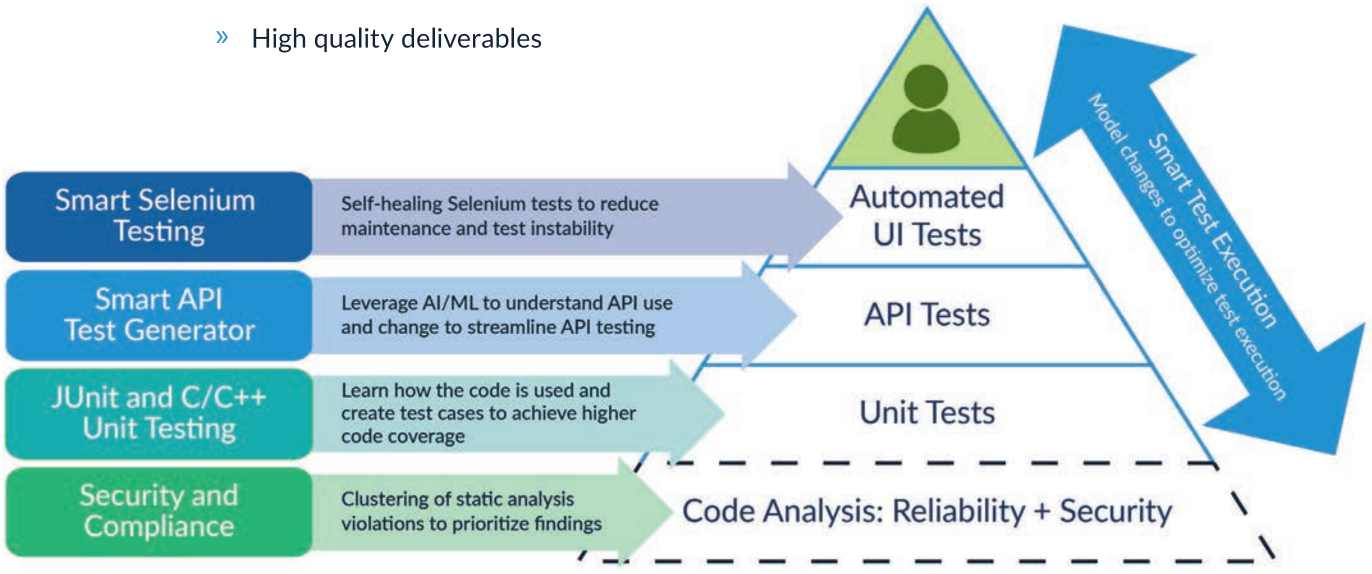

Smart Selenium Testing
Self-healing Selenium tests to reduce maintenance and test instability
Automated UI Tests
Smart API Test Generator
Leverage AI/ML to understand API use and change to streamline API testing
API Tests
JUnit and C/C++ Unit Testing
Learn how the code is used and create test cases to achieve higher code coverage
Unit Tests
Security and Compliance
Clustering of static analysis violations to prioritize findings
Code Analysis: Reliability + Security
Smart Test Execution
Model changes to optimize test execution

*Figure 17:
AI and ML have applications from the top to bottom of the test pyramid. At each level, AI is providing smarter, more productive software testing capabilities.*

AI-powered test automation solutions provide benefits across the development team.

- **Development managers** can achieve production schedules with no late-cycle defects crippling release timetables. They get more out of test automation with an AI-enhanced testing tool suite that reduces risk and increases the stability of existing projects.
- **Developers** know where the real issues are. The results from unstable tests complicate diagnosis. Automated test creation, assisted test modification, and self-healing streamline application testing so they can focus on fixing only true test failures directly within their development environment.
- **Testers and QA** get quick feedback on test execution so they can be more strategic about where to prioritize testing resources. They improve the stability of functional testing for increased productivity and have better confidence in software quality and security.

The explosive growth witnessed in the software market suggests that more software will continue to be released to solve problems in our daily lives. However, for software to function efficiently and reach the market as quickly as possible, there's a need for the powerful combination of automation and AI in software testing.

Parasoft leads the industry with incorporating AI and ML in leading-edge solutions to help teams achieve milestones and companies to reach their goals.

TAKE THE NEXT STEP

Talk to an expert to learn more about how your development team can maximize the benefits of AI/ML-powered test automation solutions.

ABOUT PARASOFT

Parasoft helps organizations continuously deliver quality software with its market-proven, integrated suite of automated software testing tools. Supporting the embedded, enterprise, and IoT markets, Parasoft's technologies reduce the time, effort, and cost of delivering secure, reliable, and compliant software by integrating everything from deep code analysis and unit testing to web UI and API testing, plus service virtualization and complete code coverage, into the delivery pipeline. Bringing all this together, Parasoft's award-winning reporting and analytics dashboard delivers a centralized view of quality enabling organizations to deliver with confidence and succeed in today's most strategic ecosystems and development initiatives—security, safety-critical, Agile, DevOps, and continuous testing.