AUTOMATION BEFORE AUTOMATION (ABA): AN INTERMEDIATE PHASE IN MODERN SOFTWARE TESTING

BY LIUDAS JANKAUSKAS

[First version: June 2026]
[qaontime.com]
pdf-version: [PDF]
www-version: [WWW]

ABSTRACT

Modern software teams routinely perform rapid testing of newly implemented functionality before investing in traditional automated test suites. This phase commonly blends manual exploration with automated techniques such as fuzzers, mutation tools, schema-driven generators, and AI-assisted test generation to uncover defects, robustness issues, and unexpected behavior.

Despite its prevalence, this activity is frequently misclassified as either exploratory testing [1] [2] [3] [4] [5] or conventional test automation, even though it aligns only partially with either category. This paper introduces Automation Before Automation (ABA) as a distinct intermediate testing phase.

Automation Before Automation (ABA) is defined as the practice of automatically generating and executing exploratory, validation, robustness, and protocol-level tests before creating and maintaining traditional automated test suites — in short, automated exploration before automated verification.

The paper provides a conceptual framework and shared vocabulary for this increasingly important activity, describing its characteristics, relationship to traditional automation, associated tooling categories, limitations, and practical application through a real-world case study.

INTRODUCTION

Modern software testing [8] discussions often distinguish between exploratory testing and automated regression testing. Exploratory testing focuses on discovering unknown problems through investigation, experimentation, and learning. Automated regression testing focuses on repeatedly verifying known behavior and preventing previously solved problems from reappearing. Both activities are widely studied, well understood, and supported by extensive literature, methodologies, and tooling.

In practice, however, software teams routinely perform an additional activity that exists between these two phases. After new functionality is implemented, developers and testers rarely begin by creating large maintainable automated test suites. Instead, they first attempt to understand how the system behaves. They modify inputs, explore boundary conditions, generate invalid data, vary protocols, examine error handling, and intentionally attempt to trigger unexpected behavior. Increasingly, these activities are assisted by tools capable of automatically generating and executing hundreds or thousands of test variations within minutes.

The purpose of these activities is not long-term regression protection. The generated tests are often temporary, disposable, and exploratory in nature. Their primary goal is rapid defect discovery and improved understanding of the system under test.

Despite becoming increasingly common, particularly with the growth of AI-assisted testing, fuzzing [9] [10] tools, property-based testing [12], automated test generators, and protocol analysis tools, this activity has no widely accepted name or formal definition. As a result, it is frequently grouped under either exploratory testing or test automation, even though its objectives, lifecycle, and outcomes differ significantly from both.

This paper introduces the term Automation Before Automation (ABA) and proposes the following definition: Automation Before Automation (ABA) is the practice of automatically generating and executing exploratory, validation, robustness, and protocol-level tests before creating and maintaining traditional automated test suites. ABA can be summarized as automated exploration before automated verification.

The objective of this paper is not to introduce a new testing technique. Rather, it seeks to formally define, characterize, and provide a conceptual framework for a testing activity that is already widely practiced but remains largely unnamed within contemporary software testing literature.

1. THE MISSING LAYER IN MODERN TESTING

Software testing literature typically describes a progression from development to automated testing and continuous integration [6]. In practice, however, the actual workflow followed by most teams contains an additional intermediate stage.

A simplified representation of the commonly assumed process is shown below:

Development → Exploration → Test Automation → CI/CD → Release

This model correctly recognizes the existence of exploratory activities before formal test automation begins. However, the exploration phase is often treated as a single activity. In practice, modern software teams frequently combine manual exploration with automated exploration techniques. Developers and testers not only execute requests manually, but also use tools capable of automatically generating and executing large numbers of test variations, boundary conditions, mutations, protocol checks, robustness scenarios, and AI-generated test cases.

These activities are exploratory in purpose but automated in execution. Despite becoming increasingly common, they are typically grouped under either exploratory testing or test automation, even though they do not fully fit either category.

As a result, the actual workflow more closely resembles the following:

Development → Manual Exploration → ABA → Test Automation → CI/CD → Release

The ABA phase serves a fundamentally different purpose from traditional automated testing. Traditional automated testing focuses on long-term regression prevention. Tests are intentionally designed, maintained, version-controlled, and repeatedly executed throughout the software lifecycle.

ABA focuses on rapid exploration and defect discovery. The generated tests are often temporary and disposable. Their value lies not in long-term maintenance, but in quickly revealing unknown problems, validation weaknesses, robustness issues, and unexpected system behavior before formal automation efforts begin.

Without recognizing ABA as a distinct activity, discussions about testing often incorrectly classify these practices as either exploratory testing or test automation. This obscures important differences in objectives, cost, maintenance requirements, and expected outcomes.

The existence of this intermediate layer becomes increasingly visible as AI-assisted test generation, fuzzing technologies, protocol analyzers, and automated exploration tools become more common throughout the software industry.

2. DEFINING (ABA) AUTOMATION BEFORE AUTOMATION

This paper proposes the following definition: **Automation Before Automation (ABA)** is the practice of automatically generating and executing exploratory, validation, robustness, and protocol-level tests before creating and maintaining traditional automated test suites. ABA can be summarized as: Automated exploration before automated verification.

The defining objective of ABA is not regression prevention. Instead, ABA aims to rapidly discover unknown defects, validation weaknesses, unexpected system behavior, and implementation risks immediately after functionality is implemented or modified.

ABA occupies a distinct position between manual exploration and traditional test automation. It is performed before long-term automated test suites are designed, implemented, maintained, and integrated into continuous delivery [7] pipelines.

For the purposes of this paper, an activity belongs to ABA if its primary purpose is exploratory discovery through automatically generated or automatically executed testing intended to improve understanding of the system under test before formal automation efforts begin.

ABA may include activities such as:

- Automated exploratory testing
- Automated robustness testing
- Automated validation testing
- Automated protocol-level testing
- Automated mutation-based testing
- Automated fuzzing and input variation
- AI-assisted exploratory test generation

ABA does not include:

- Manual exploratory testing without automated test generation or execution

- Traditional regression automation
- Long-term maintainable automated test suites
- Continuous integration test execution
- Acceptance test automation
- Automated verification of previously defined requirements
- SAST (Static Application Security Testing)

The distinction is important because the primary goal of ABA is discovery, while the primary goal of traditional automation is verification.

3. CHARACTERISTICS OF ABA

Although ABA may be implemented using different techniques and tools, several common characteristics frequently distinguish it from traditional automated testing.

Generated Rather Than Manually Authored

ABA commonly relies on automated generation of test cases rather than manually designed test suites. Test variations may be derived from schemas, mutations, heuristics, protocol rules, fuzzing strategies, AI-generated scenarios, or other automated mechanisms.

Exploratory by Nature

The primary purpose of ABA is exploration and discovery. Rather than confirming known behavior, ABA seeks to reveal previously unknown defects, unexpected system responses, validation weaknesses, and robustness issues.

High-Volume Execution

ABA often executes large numbers of test variations in a short period of time. Hundreds or thousands of inputs may be evaluated before a single maintainable regression test is written.

Low Maintenance Cost

Unlike traditional automated test suites, ABA activities typically require little or no long-term maintenance. Their value is concentrated in the information they produce rather than in their continued execution over time.

Frequently Disposable

Many ABA-generated tests are temporary artifacts. Once findings have been analyzed and valuable scenarios identified, the generated tests are often discarded rather than maintained throughout the software lifecycle.

Discovery-Oriented Rather Than Verification-Oriented

Traditional automation primarily verifies known expectations. ABA primarily seeks to discover unknown information about the behavior of the system under test.

Fast Feedback

ABA is designed to provide rapid feedback immediately after implementation or modification of functionality, allowing teams to identify issues before investing in long-term automation efforts.

These characteristics distinguish ABA from traditional automated testing, which typically emphasizes manually designed, maintainable, regression-focused test suites intended for repeated execution throughout the software lifecycle.

4. ABA VS TRADITIONAL TEST AUTOMATION

Automation Before Automation (ABA) and traditional test automation both rely on automated execution. However, their objectives, lifecycle, economics, and expected outcomes differ significantly.

Traditional automation is primarily concerned with regression prevention. Tests are intentionally designed, maintained over time, integrated into delivery pipelines, and repeatedly executed to ensure previously verified functionality continues to operate correctly.

ABA serves a different purpose. Its objective is rapid exploration and defect discovery immediately after functionality is implemented or modified. Rather than protecting against known failures, ABA attempts to expose unknown failures before formal automation efforts begin.

The distinction can be summarized as follows:

Automation Before Automation (ABA)	Traditional Test Automation
Discovers unknown issues	Prevents known regressions
Exploration-oriented	Verification-oriented
Generated automatically	Usually designed manually
Often disposable	Long-term maintained
High-volume execution	Targeted execution
Rapid feedback	Long-term confidence
Low maintenance cost	Continuous maintenance cost
Identifies automation candidates	Implements automation candidates
Executed before formal automation	Executed after automation is created
Optimized for learning	Optimized for stability

Neither approach replaces the other. ABA is not a substitute for automated regression testing, just as exploratory testing is not a substitute for regression testing. Instead, ABA complements traditional automation by helping teams discover defects, identify risks, and determine where automation investments provide the greatest value.

In this sense, ABA can be viewed as a preparatory automation layer that precedes conventional automation efforts. Its purpose is to improve understanding of the system under test before organizations commit resources to building and maintaining long-term automated test suites.

5. TOOL CATEGORIES ASSOCIATED WITH ABA

ABA is defined by its objectives and characteristics rather than by any specific implementation technology. Consequently, ABA should not be viewed as a particular tool category, framework, or product. Instead, multiple existing and emerging tool categories may exhibit ABA characteristics when used for automated exploration prior to long-term automation efforts. The following categories frequently align with the ABA model.

Fuzzing Tools

Fuzzing tools automatically generate large numbers of unexpected, malformed, random, or mutated inputs in an attempt to reveal crashes, validation weaknesses, security vulnerabilities, and robustness issues. Their primary objective is discovery rather than regression prevention, making them a natural fit within the ABA framework.

Mutation and Input Variation Tools

Mutation-based tools systematically alter valid inputs to create alternative test scenarios. Common examples include boundary value mutations, invalid data combinations, protocol violations, field omission, truncation, expansion, and format manipulation. These approaches prioritize exploration and rapid defect discovery over long-term test maintenance.

Schema-Driven Test Generators [11]

Tools capable of generating tests from interface specifications, contracts, schemas, or metadata frequently demonstrate ABA characteristics. Rather than manually authoring test suites, these tools automatically create large numbers of validation and robustness scenarios designed to reveal implementation weaknesses.

AI-Assisted Test Generation Tools

Recent advances in artificial intelligence have enabled automated generation of exploratory test scenarios, input variations, user interactions, and validation strategies. When such tools are used primarily to discover unknown defects before formal automation begins, they align closely with ABA principles.

Protocol and Interface Exploration Tools

Certain tools focus on automatically exercising interfaces, endpoints, commands, or communication protocols using generated variations and behavioral exploration strategies. Their purpose is often to rapidly reveal unexpected system behavior and identify areas requiring deeper investigation.

Automated Robustness and Stress Exploration Tools

Some tools intentionally push systems beyond expected operating conditions through automatically generated workloads, unexpected inputs, edge cases, or environmental variations. When used for rapid exploration and discovery rather than formal performance verification, these activities can also be classified as ABA.

It is important to note that ABA classification depends primarily on how a tool is used rather than on the tool itself. The same technology may participate in ABA activities in one context and traditional test automation in another.

For example, a generated exploratory test executed to discover unknown defects may be considered ABA, while the same test, once selected, maintained, and continuously executed as part of a regression suite, becomes traditional automated testing.

This distinction reinforces the central argument of this paper: ABA is not a product category but a testing activity characterized by automated exploration before automated verification.

6. CASE STUDY: AUTOMATED EXPLORATION OF A PRODUCTION AI API

Context

To demonstrate the practical application of ABA, an automated exploratory session was conducted on a high-profile production AI API using a single valid request captured from normal usage.

Method

Using mutation-based automated test generation, more than 200 test variations were executed within minutes. The variations covered input validation, boundary conditions, malformed payloads, protocol deviations, authorization edge cases, and oversized data scenarios. The objective was exploratory discovery rather than regression verification, and no maintainable test suite was created.

Key Findings

The session revealed several implementation weaknesses, including:

- Inconsistent security header configuration
- Overly permissive cross-origin behavior
- Weak input validation
- Improper handling of oversized payloads
- Protocol-level inconsistencies

Several findings were publicly disclosed and were subsequently addressed by the service provider.

Alignment with ABA Characteristics

This activity exhibited the core characteristics of ABA:

- Generated rather than manually authored tests
- Exploratory rather than verification-oriented objectives
- High-volume execution
- Rapid feedback
- Low maintenance overhead
- Disposable test artifacts
- Discovery of previously unknown defects

Discussion

The case illustrates how ABA can rapidly surface meaningful defects in complex real-world systems, including those developed by highly mature engineering organizations, without requiring deep prior knowledge of the implementation. The primary outcome was not regression protection but accelerated defect discovery and improved understanding of system behavior prior to formal automation efforts.

7. LIMITATIONS

Although ABA provides significant value for rapid defect discovery and exploratory analysis, it should not be viewed as a replacement for traditional automated testing.

ABA excels at identifying unknown problems, exposing unexpected behavior, revealing validation weaknesses, and helping teams understand newly implemented functionality. However, discovering a defect and preventing its reoccurrence are fundamentally different activities.

Traditional automated testing remains essential because software systems continuously evolve. Once a defect has been identified and corrected, organizations require maintainable automated tests capable of repeatedly verifying that the issue does not reappear in future releases.

ABA does not eliminate the need for:

- Regression testing
- Integration testing
- End-to-end testing
- Acceptance testing
- Continuous Integration (CI/CD) quality gates
- Long-term maintainable automated test suites

Similarly, ABA does not replace human exploratory testing. Human testers remain uniquely capable of applying domain knowledge, intuition, creativity, and contextual reasoning that automated generation techniques cannot fully replicate.

The primary output of ABA is information. The primary output of traditional automation is confidence. ABA helps teams discover what they did not know. Traditional automation helps teams continuously verify what they already know.

For this reason, ABA should be viewed as a complementary testing layer rather than a competing methodology. Its purpose is to improve the quality and effectiveness of subsequent testing activities by exposing weaknesses before organizations invest in long-term automation efforts.

In practical terms, ABA answers the question: "What should we automate?"

Traditional automated testing answers the question: "How do we continuously verify it?"

Both activities are necessary, and neither replaces the other.

CONCLUSION

Software teams have long performed activities that exist between manual exploration and traditional automated testing. As automated test generation, fuzzing technologies, AI-assisted testing, and other exploratory automation approaches continue to evolve, this intermediate phase is becoming increasingly visible and important.

This paper introduced Automation Before Automation (ABA) as a formal term for the practice of automatically generating and executing exploratory, validation, robustness, and protocol-level tests before creating and maintaining traditional automated test suites.

The paper argues that ABA represents a distinct testing activity with objectives, characteristics, and outcomes that differ from both manual exploratory testing and conventional regression automation. While traditional automation focuses on continuously verifying known behavior, ABA focuses on rapidly discovering unknown behavior and identifying what is worth automating.

ABA does not replace exploratory testing, nor does it replace traditional automated testing. Instead, it complements both by providing a dedicated layer of automated exploration before long-term verification begins.

By defining and characterizing ABA, this paper provides a common vocabulary and conceptual framework for discussing, evaluating, and improving a testing activity that is already widely practiced but remains largely unnamed within contemporary software testing literature.

[Liudas Jankauskas homepage: qaontime.com]
[More research: qaontime.com/research]

REFERENCES

[1] C. Kaner, J. Falk, and H. Q. Nguyen, Testing Computer Software, 2nd ed. New York, NY, USA: Wiley, 1999. ISBN 978-0471358466.

[2] J. Bach, Exploratory Testing Explained, Satisfice Inc., 2000. Available: https://satisfice.us/articles/et-article.pdf

[3] C. Kaner and J. Bach, Lessons Learned in Software Testing. New York, NY, USA: Wiley, 2002. ISBN 978-0471081128.

[4] M. Bolton and J. Bach, Rapid Software Testing. Context-Driven Testing Community. Available: https://rapid-software-testing.com

[5] J. Itkonen and K. Rautiainen, Exploratory Testing: A Multiple Case Study, Proceedings of the International Symposium on Empirical Software Engineering, 2005. Available: https://www.researchgate.net/publication/4194152_Exploratory_testing_A_multiple_case_study

[6] M. Fowler, Continuous Integration, 2006. Available: https://martinfowler.com/articles/continuousIntegration.html

[7] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Boston, MA, USA: Addison-Wesley, 2010. ISBN 978-0321601919.

[8] ISO/IEC/IEEE 29119 Software Testing Standard. Geneva, Switzerland: International Organization for Standardization. Available: https://www.iso.org/standard/81291.html

[9] M. Sutton, A. Greene, and P. Amini, Fuzzing: Brute Force Vulnerability Discovery. Boston, MA, USA: Addison-Wesley, 2007. ISBN 978-0321446114.

[10] B. P. Miller, L. Fredriksen, and B. So, An Empirical Study of the Reliability of UNIX Utilities, Communications of the ACM, 1990. Available: https://doi.org/10.1145/96267.96279

[11] Z. Hatfield-Dodds and D. Dygalo, Deriving Semantics-Aware Fuzzers from Web API Schemas, 2021. Available: https://arxiv.org/pdf/2112.10328

[12] K. Claessen and J. Hughes, QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs, Proc. Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP), 2000. Available: https://doi.org/10.1145/351240.351266