

Welcome Letter From Sauce Labs

JOHN KELLY, CHIEF STRATEGY OFFICER

Time to market is everything. When the cloud started to proliferate in the 2000s, it created a revolution in software development, allowing developers to minimize and sometimes skip entire procurement processes, environmental configuration, and network security for the running of the software. You could “rent” the equipment, and have a prototype environment ready within minutes. All from a web browser.

A similar revolution is taking place today *in the actual development of the software*, through low-code development platforms.

Traditional software development processes tend to create below-the-line work, requiring developers to cobble features together from disparate libraries using the programming equivalent of duct tape and bailing wire. It’s not much different from the procurement process to get new equipment.

Some teams still spend all their time on tech debt and library maintenance, and may never even touch customer-facing features. Low-code/no-code tools give developers a head-start, allowing them to jump-start the software process, and relying on the tool vendors to keep up with tech debt and library maintenance. This allows them to focus on the business outcomes, not the duct tape: they can prioritize the customers’ needs and delights, not the tech at the bottom of the stack.


However, what hasn’t gone away—and likely never will—is the fact that your business solves a unique problem, in a unique way. This requires rigorous design, development, and testing to make sure it’s delivering value to your customers without the risk of crippling software defects.

As you’ll read in this Forrester Report, software developed with low-code solutions requires a different approach to testing. Low-code obviates some of the testing altogether (which is great!), but some of the work may require more energy and attention to detail than before

In short, you need a holistic test strategy. In a low-code world, the onus is on how your business processes and workflows are assembled. Bugs are likely harder to find overall, they might be more subtle, and they might manifest themselves differently depending on different data sets or changing workflow conditions.

To add yet more complexity, the world now revolves around SaaS applications - managing sales workflows, HR processes, customer support ticketing, and a myriad of other functions. These platforms have mostly supplanted the spreadsheet-based nightmare that companies have been living since the 1970s.

More and more, companies are writing software to interact with these platforms, and the integration of low-code solutions with these tools makes it easier than ever, right? Well, yes—but with the added risk that inattention to detail and thorough testing might miss some key edge cases. Low-code testing solutions that combine purpose-built SaaS platform testing with custom software capabilities is a promising new field, aiming to cover all bases using intent-based, declarative test scripts.

Testing is too often isolated from real business outcomes and the key journeys that real customers take. Management sometimes misses the mark, hoping the testers will just rubber stamp the latest features so they can put it into production. Some teams get it right, relying on the tester as a customer advocate and advisor. With low-code software development, much of the testing process can be centered on the user experience, data consistency, and business outcomes. In short, low code platforms help both developers and testers focus more of their time and energy on the real results that the business needs. 

We Must Address Testing In Low-Code Development

Start With Key Areas: Functional, Unit, And Component Testing

October 2, 2020

By John Bratincevic, Diego Lo Giudice, John Rymer, Julia Caldwell with Chris Gardner, Andrew Dobak, Kara Hartig

FORRESTER®

Summary

The role of low-code platforms is growing in application development. Application development and delivery (AD&D) professionals are starting to use these platforms to speed delivery of mission-critical high-scale projects and must now address application quality by testing their low-code creations. Testing low-code applications is different from testing traditionally developed apps. Read this report to understand these differences and why they matter.

THE RISKS OF NOT TESTING LOW-CODE APPLICATIONS ARE CLEAR

More and more enterprises are delivering applications using low-code development platforms, and not just departmental teams. Businesses are taking advantage of low code for 1) speed to market; 2) cheaper bespoke development; 3) more easily tailored core systems; and 4) extendable architecture. Moreover, AD&D leaders are starting to use low-code platforms to deliver large-scale, complex, highly reliable, and/or secure/compliant applications. (see endnote 1) As low-code applications assume more business risk in mission-critical contexts, AD&D leaders and teams ignore testing and quality assurance at their own peril.

Those using low-code platforms who ignore testing face two pitfalls:

- **As low-code developers take on mission-critical work, the stakes are higher.**

More low-code development introduces the potential for more bugs hitting production, more failed integrations, more performance slowdowns, and more business functionality gone awry. An inconvenience in a limited rollout might have widespread consequences for employee and client experiences as use of low code expands. Objectivity, a services firm with practices in three low-code platforms, built testing practices for each to validate the quality of its mission-critical projects.

- **What you build might not match what you wanted.** Just because low code abstracts code does not mean users can have blind faith their applications will behave as envisioned. An administrator at a US utility asks reviewers to validate and feedback on simple apps. But for apps with complex logic, including rules driving the data field engineers see, she relies on the low-code platform's validation tool. As logic grows in complexity, the chances that the logic doesn't match the outcome increases and thus the need for testing intensifies. "This is important when writing logic addressing multiple fields," the administrator said.

LOW-CODE VENDORS AND CUSTOMERS SHARE THE TESTING BURDEN

In low-code application delivery, testing is different than it is in development for traditional coding. It also can be less of a burden on developers. Why? Because it is a shared responsibility:

-

Not Licensed For Distribution.

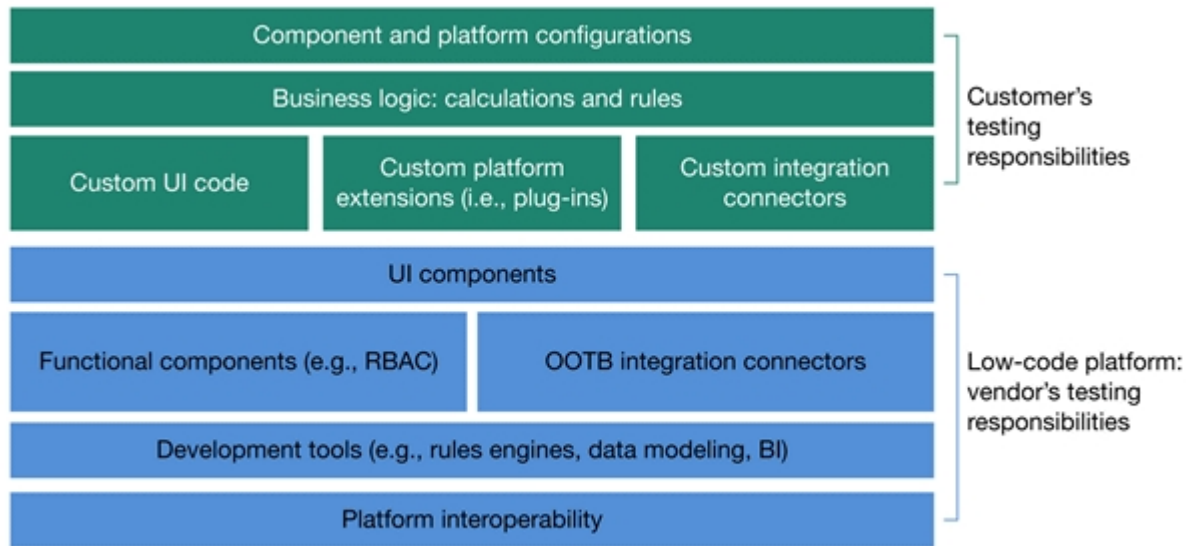
© 2021 Forrester Research, Inc. All trademarks are property of their respective owners.

For more information, see the [Citation Policy](#), contact citations@forrester.com, or call +1 866-367-7378.

Developers use vendor-tested components to build apps. Low-code platforms include building blocks developers reuse when developing applications. The low-code platform vendor already tests and validates these building blocks, so they do not require unit testing by developers. Your vendor should guarantee these building blocks and their interoperability with one another. Thus, customers do not need to test vendor-provided content, which reduces the volume of necessary application testing (see Figure 1).

- **Many platforms provide automatic validation and warnings.** Incomplete process logic, malformed syntax, gaps in application models, etc., commonly trigger alerts or are outright prohibited for developers in low-code development environments.
- **Coding and libraries to supplement low-code platforms must be customer tested.** The customer still owns testing beyond the platform building blocks when using open source components, writing customizations, developing plug-ins, etc.
- **Low-code platform vendors may not provide complete testing tools.** A further challenge exists for AD&D pros: Vendor tooling for supporting customer testing practices is uneven. Those vendors focused on the citizen developer market typically have minimal tooling for structured testing or test automation (beyond simple error trapping of syntax). So testing of any type must happen manually. Vendors focused on professional developers are typically more mature, with some testing tools built into the platform or provided through integrations with other vendors, but these approaches also vary widely.

Figure 1: Low-Code Testing Is A Shared Responsibility



162135

Source: Forrester Research, Inc. Unauthorized reproduction, citation, or distribution prohibited.

SEPARATE TWO METHODS OF DEVELOPMENT IN LOW-CODE TESTING

The key to new testing practices for low-code development: Distinguish declarative development from coding. Fundamentally, the purpose of low-code platforms is to reduce the coding and technical expertise needed to develop applications and test them. However, two development approaches coexist in most low-code platforms, representing opposite ends of a spectrum:

- **Purely declarative (no-code) development using native platform tooling.** These techniques include click-and-drag UI design, formula-style business logic, visual process modelers, rules engines, prebuilt components, and the like. This form of development can still be complex (especially when writing lengthy business logic) but does not employ true lower-level coding.
- **Custom code development to go beyond native platform tooling.** These customizations are typically written in languages such as JavaScript, CSS, and SQL. Examples include plug-ins, custom UIs, database scripts, complex algorithms, and custom integrations.

The Methods Of Development Drive The Modes Of Testing

Not Licensed For Distribution.

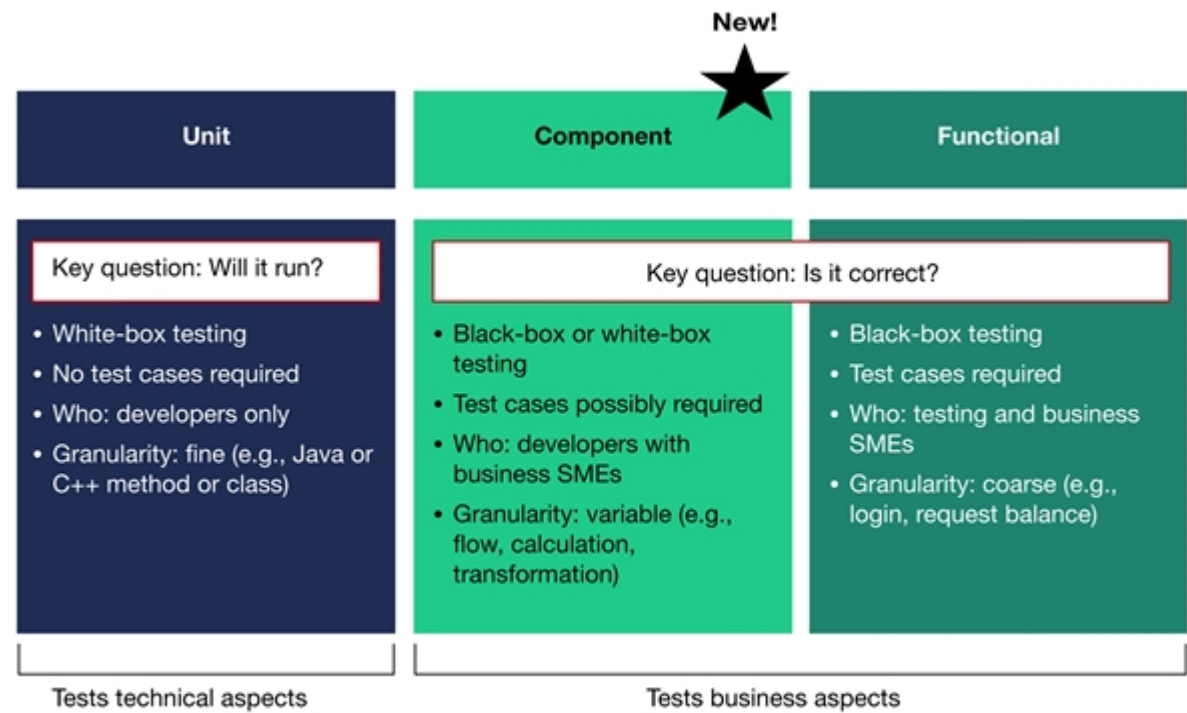
© 2021 Forrester Research, Inc. All trademarks are property of their respective owners.

For more information, see the [Citation Policy](#), contact citations@forrester.com, or call +1 866-367-7378.

Understanding these methods of development informs the way customers can start to think about testing practices in low code (see Figure 2). The level of platform constraints, complexity of the application, and scale of deployment must all be considered to determine the granularity of testing for a given scenario. These essential observations about testing in low code can serve as a starting point:

- **Unit testing applies to few low-code situations.** Unit, or white-box, testing, although ill defined, is the lowest level of testing. (see endnote 2) A unit test will check the correctness of the structure, design, and implementation of the code being tested. In unit testing, the tester chooses inputs to exercise paths through the code to make sure execution and outcomes are correct. Use of purely declarative tooling to develop applications does not generally require unit testing. You should not need to test the validity of the bundles of code from the vendor any more than you would need to test the code of a packaged application. However, once you start extending these out-of-the-box capabilities with custom code, unit testing again becomes valuable.
- **Component testing — a new form of testing — applies to most low-code projects.** Business logic written with high-level formula syntax or visual modelers can be underestimated. As one Mendix executive said: "If you look at the type of decisions made in a nested flow, it's pretty complex. Every piece needs to function right." Developers writing sophisticated or high-risk apps will often test isolated pieces of business logic to validate that these pieces function as intended. This middle ground between unit testing and functional testing is component testing. Component testing can be a gray area but applies when the business logic becomes too complex or risky to trust without testing. For example, one developer in real estate lending broke down and tested the individual inputs into a multiline formula for a complex ROI calculation to ensure each piece worked as intended.
- **Functional testing applies to all low-code projects.** Functional testing is not only relevant across low-code scenarios. Regardless of the platform or application type, end-to-end testing of application functionality must be the core of any organization's testing strategy when using low code. Demand on functional testing is going up.

Figure 2: Figure 2 Foundations Of Low-Code Testing



162135 Source: Forrester Research, Inc. Unauthorized reproduction, citation, or distribution prohibited.

RECOMMENDATIONS

Minimize Custom Code To Reduce The Testing Burden

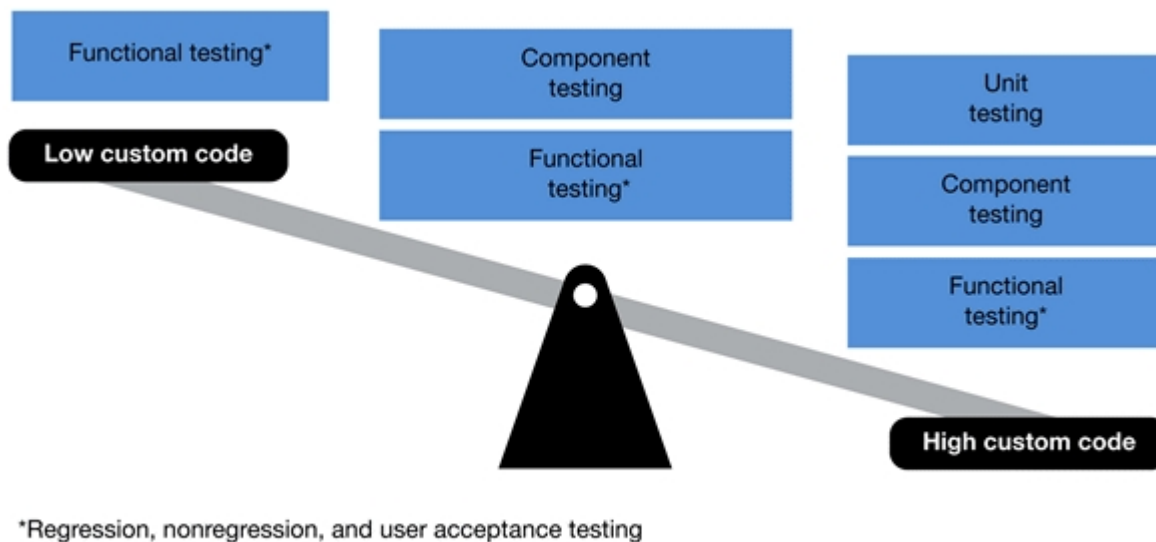
Utilizing the native, declarative tooling of the platform and avoiding custom code provides an opportunity to reduce the application testing burden, speed application delivery, and reduce technical debt, all without sacrificing application quality (see Figure 3). While the tooling and formal practices for low-code testing are still emerging, we recommend testing patterns that vary by scenario:

- **Scenario number one: no custom code and simple declarative logic.** Use functional testing only. In this scenario, a developer constructs an application using only highly abstracted language and development tooling from the platform. For example, a warehouse supervisor with little technical acumen might use drag-and-drop features to build out a digital process for efficiently loading trucks. The supervisor only tests to ensure the application is functioning according to his vision as a domain expert. And he may seek testing experts to provide guidance.

Regardless, applying a functional test scenario to the end-to-end process with varying types of input data suffices.

- **Scenario number two: no custom code and complex declarative logic.** Use functional and component testing. For example, a more advanced developer might construct a complex financial calculation based on a long series of data inputs or business rules. Unit testing is not required (because true coding language is not used), but if the application is high-risk or the logic is too complex to trust without testing, both functional and component testing apply to ensure there are no unseen gaps in the logic. Systematically breaking down the calculation or rules into smaller pieces (components) and testing them individually is advised; some platforms have tooling for this under various labels. For example, Mendix has tooling for testing microflows.
- **Scenario number three: custom code.** Use both unit and functional testing. If possible, leverage existing open source or commercial testing tools. For de facto standard languages such as C++, C#, Java, and JavaScript, use open source unit testing frameworks like xUnit, NUnit, and JUnit. And for functional testing, leverage open source frameworks such as Selenium or Appium and/or market-leading commercial testing tools like Tricentis Tosca, Parasoft, and AccelQ, among others. (see endnote 3) Also use component testing if required. Avoid customizations and only utilize the native platform tooling whenever possible.

Figure 3: More Code Means More Testing



SUPPLEMENTAL MATERIAL

Companies Interviewed For This Report

We would like to thank the individuals from the following companies who generously gave their time during the research for this report.

- Acto Technologies
- Appian
- AppSheet (Google)
- Infinite Blue Platform
- Kentucky Power
- Mendix
- Microsoft
- Netcall
- Novulo
- Objectivity
- OutSystems
- Pegasystems
- Salesforce
- ServiceNow
- Team Resilience

Endnotes

1. For more information, see the Forrester report "[When And How To Modernize Core Applications Using Low-Code Platforms.](#)"

Not Licensed For Distribution.

© 2021 Forrester Research, Inc. All trademarks are property of their respective owners.

For more information, see the [Citation Policy](#), contact citations@forrester.com, or call +1 866-367-7378.

2. For more information on unit testing, refer to the 2014 article by Martin Fowler.
Source: Martin Fowler, "UnitTest," MartinFowler.com, May 5, 2014
(<https://martinfowler.com/bliki/UnitTest.html>).
3. For more information on continuous functional test automation, see the Forrester report "**The Forrester Wave™: Continuous Functional Test Automation Suites, Q2 2020.**"



We help business and technology leaders use customer obsession to accelerate growth.

FORRESTER.COM

Obsessed With Customer Obsession

At Forrester, customer obsession is at the core of everything we do. We're on your side and by your side to help you become more customer obsessed.

Research

Accelerate your impact on the market with a proven path to growth.

- Customer and market dynamics
- Curated tools and frameworks
- Objective advice
- Hands-on guidance

[Learn more.](#)

Consulting

Implement modern strategies that align and empower teams.

- In-depth strategic projects
- Webinars, speeches, and workshops
- Custom content

[Learn more.](#)

Events

Develop fresh perspectives, draw inspiration from leaders, and network with peers.

- Thought leadership, frameworks, and models
- One-on-ones with peers and analysts
- In-person and virtual experiences

[Learn more.](#)

FOLLOW FORRESTER



Contact Us

Contact Forrester at www.forrester.com/contactus. For information on hard-copy or electronic reprints, please contact your Account Team or reprints@forrester.com. We offer quantity discounts and special pricing for academic and nonprofit institutions.

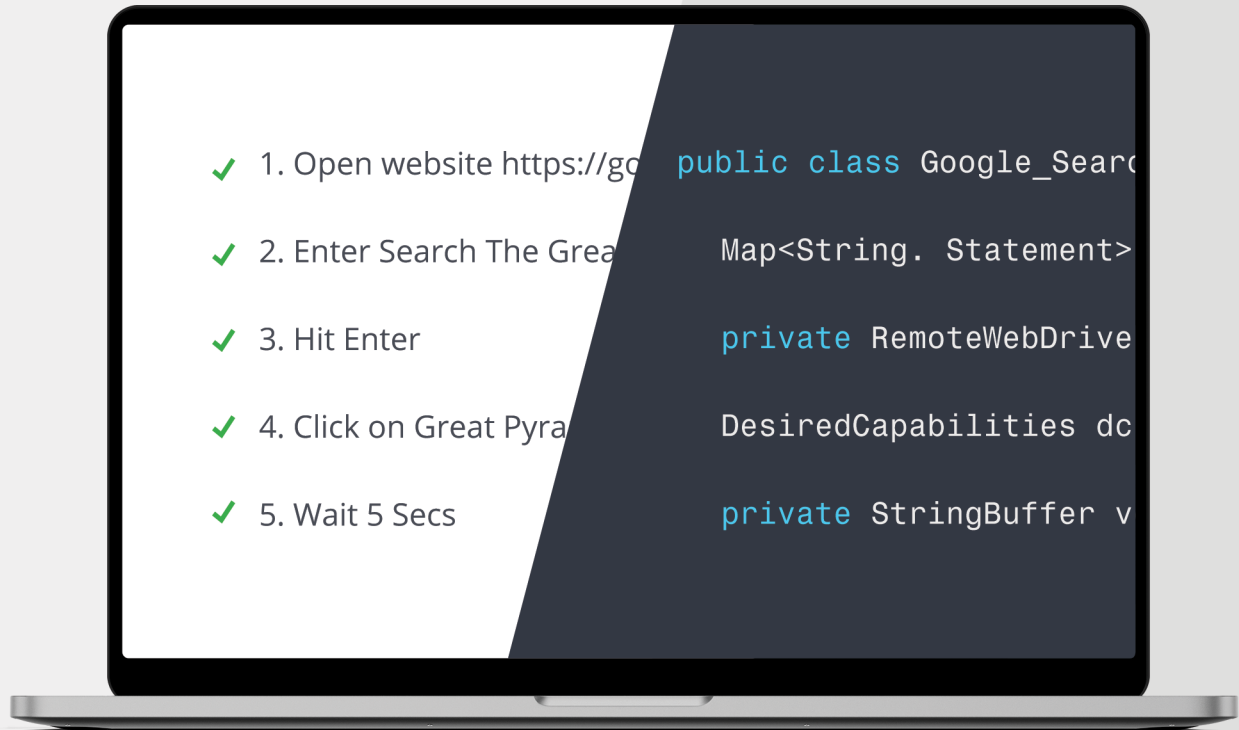
Forrester Research, Inc., 60 Acorn Park Drive, Cambridge, MA 02140 USA
Tel: +1 617-613-6000 | Fax: +1 617-613-5000 | forrester.com

Not Licensed For Distribution.

© 2021 Forrester Research, Inc. All trademarks are property of their respective owners.
For more information, see the [Citation Policy](#), contact citations@forrester.com, or call +1 866-367-7378.

Let Machines Code Your Tests. You Do The Interesting Stuff.

You create the test case. We generate the scripts. AI-driven end-to-end test automation includes powerful natural language processing capabilities.



Intelligent Test Management - Manage testing through constant changes, frequent upgrades and new functionality across end-to-end business processes.



Self-Configuring Reusable Test Assets - Create reusable test components customized for your any web application and add them to new test scenarios with a single click.



API Testing - Increase the breadth, scope, and velocity of your API testing and decrease the time between finding defects and resolution.



Deep-Learning and Machine Learning Capabilities - Detect changes and enable self-healing for test assets giving you critical real-time feedback to ensure long-term durability and scalability.

[Learn more](#)

