SAUCELABS

WHITE PAPER

Citizen Code Breakthroughs: Creating Order of Magnitude Leverage for the Development Team

TABLE OF CONTENTS

INTRODUCTION

Frequent software delivery drives innovation. According to the State of DevOps Report, High-performing organizations deploy 200x more frequently and have 3x lower change failure rates than lower-performing organizations. During a presentation at QCon London in 2014, a former Software Engineer at Etsy, Daniel Schauenberg, described how the e-commerce giant is capable of running a fully automated DevOps pipeline that allows their organization to deploy 50 or more new software iterations. Every. Single. Day.

This type of velocity and scale is only possible for Etsy's 150-person engineering team and advanced DevOps infrastructure. At least back then, it allowed them to run 140,000 test suites per day and customize a rapid end-to-end testing framework that gave even junior developers the confidence to deploy code within their first 24 hours on the job.

For emerging organizations trying to level up to this scale, it can be difficult to deliver value hamstrung by having enough engineers and testers trained, available, and be-spoke infrastructure capable of bringing software to market and keeping it updated. In an era where the unemployment rate for IT professionals in the U.S. is sub 2%, hiring and retaining some of the highest in-demand talents is a challenging task. There's a talent war and forward-thinking organizations are suiting up to overcome it.

Custom enterprise SaaS applications like Salesforce, SAP, ServiceNow, and Oracle have vastly propelled the implementation of software delivery for traditional companies looking to modernize or digital native companies looking to accelerate growth. There is also a cost for not adopting these tools: speed to market rapidly outpaced customer demand and keeping up with the ~~Joneses~~ *Etsy's of the world*. Companies must find ways to adapt and deploy new technology in all areas of their business without compromising the digital confidence delivered to customer quality. For enterprises that are struggling to win the talent war and are unable to meet the complex demands of rapid application development (RAD), what alternatives are out there?

Regardless of the compounded challenges the COVID-19 pandemic introduced, digital transformation is accelerating and the citizen-code/no-code/low-code (further referred to in this white paper as low-code) movement promises a new era of rapid and pervasive application development. Scaling the capabilities of the engineering team augmented by the democratization of

coding should lead to breakthroughs across all functions in the organization. It is only a matter of time until we start to witness this.

What is missing is an ability to accelerate software development, across the entire lifecycle, avoid manual delays at critical junctures, while also leveraging the headcount we have today.

Asked simply, can the business produce additional digital transformation value with the people they already have in place? Let's find out.

THE ORIGINS OF NO-CODE

Contrary to popular belief, the era of low—or lower—code application development began in the 1950s with IBM, but in 2014, Gartner helped refine the term:

> "Low-code development both describes platforms that abstract away from code and offer an integrated set of tools to accelerate app delivery."

If you work in an enterprise organization today or have built a software product in the last decade, there is a high probability that you have interacted with a cross-spectrum of low-code development platforms. Whether your organization has embraced this is another thing. The silent insurgence of these tools has revolutionized the way companies, and (citizen) developers work.

The progressive adoption of low-code was inevitable. IBM instituted this trend with their breakthrough innovation FORTRAN (FORmula TRANslator) in the early 1950s. Early assembly languages were needlessly complicated and could not be scaled to other computer devices. FORTRAN helped change this. Computer programming evolved from something only done by computer natives to one that mathematicians and scientists—with no previous exposure to the world of computing— could understand and write functional algebraic statements for. This served as an important benchmark to the no-code development movement.

Fast-forward to the 1970s, C, C++, and C# emerged. These languages were written with natural English syntax and provided major improvements and

usability for creating web applications. The biggest pivot after the C's was the advancement of the programming language Java, Python, PHP, and others. The explosive growth of the internet brought new developers, who were not native computer scientists or mathematicians, and new users who demanded applications be developed at a faster pace.

CHALLENGES ARISE: SHIFTING TO NO-CODE IS NO WALK IN THE PARK

According to a Research and Markets Report, the demand for low-code development is rapidly increasing. In 2019, the generated revenue from low-code application platforms (LCAP) was $10.3B. This number is expected to exceed $187B by 2030 which equates to a compound annual growth rate (CAGR) of more than 30%. With this increased demand is the need for ancillary services like the maintenance, security, and monitoring of low-code platforms. Low-code is not a silver bullet and it can be a detriment to organizational effectiveness if it is not implemented correctly. Given this, where can things go wrong?

PAIN POINT #1: LACK OF OVERSIGHT CREATES SHADOW IT

One of the prevalent concerns with low-code application development is that almost anyone in the organization can spin up a new environment to start building. This is referred to as shadow IT: low-code application platforms make it easier for employees to go around IT governance and operate without the knowledge of management. According to research produced by McAfee, in the era of modern SaaS tools, the average company uses 1083 applications, but the IT department knows only about 108 of them. With governance visibility of less than 10%, this creates new attack vectors for hackers to explore potential data leaks and security gaps that need to be addressed by the organization, especially around critical business processes like lead to cash, revenue to report, and issues to resolution. Gartner has estimated that one-third of successful attacks experienced by enterprises will be on data located in shadow IT resources.

Various LCAPs pose a higher risk to the organization than others and managers need to gain complete visibility into all of the cloud tools adopted, understand the risk assessment, and deploy the right test automation frameworks to ensure critical business processes are not being compromised.

PAIN POINT #2: AUDITS OF VENDOR LOOK DIFFERENT SOLUTION TO SOLUTION

Would a company audit the security protocols of Salesforce Marketing Cloud the same way they audit Microsoft Teams or Discord? Better yet, would they audit vendor tools the same they audit their custom code? The answer is likely no. To find out how secure various vendors are, companies are reliant on compliance certifications, third-party security audits, and cybersecurity insurance. However, this is not enough when customer data and privacy is on the line, and auditing the influx of low-code tools can become a full-time job.

A data breach due to a poor audit can lead to technical staff with added work as they try to find resolutions, loss of productivity and deferred resources, and the adverse impact on a publicly-traded company's share price.

PAIN POINT #3: TRAIN AT YOUR OWN RISK

Low-code platforms promise increased efficiency and agility in the business, but this is often riddled with excessive training requirements and a new user interface that has to be adopted. Sometimes the learning curve of a new LCAP may outweigh the benefits of adding it to the tool stack and not all employees may not be on board with this. Organizations looking to embrace low-code application development must decipher between the value prospective LCAPs can bring versus the full cost of adoption: training, opportunity costs, maintenance, slow or no adoption, etc.).

PAIN POINT #4: MANUAL QA AND TESTING CYCLES

When software development slows down, at either the ideation, coding, testing, or deployment stage, revenue and engagement opportunities are missed.

Once a business has its user requirements translated into code, is there more technical debt that is being inherited by the already backlogged QA team with the introduction of low-code RAD?

On the other side of the equation, when certain features and functionality are nonexistent, users and consumers look elsewhere. If the application doesn't load, doesn't work, or is buggy, there is little tolerance for imperfection. Rushed, undertested, buggy software damages an enterprise's brand.

The last thing a user wants to do is stop mid-task and search for the "help" tab, engage the chatbot, or worst of all, pick up the phone to speak

with the help desk. Those are all fatal delays. They mean the user is not able to use the application, and they're going to be back in the app store looking for the next option.

Shoddy software costs the U.S. an estimated $2.08tr in 2020, according to the Consortium for Information & Software Quality (CISQ). In a well-publicized story, two software bugs that prevented Boeing's Starliner from docking with the International Space Station in December 2019, buggy software is an invisible problem, until it isn't.

In a manual process or an environment of unprecedented demand for testing, the testing team is waiting for headcount approvals in a shrinking pool of available testers. And not only that, once there is an increase in headcount approved, HR still needs to go out and recruit the talent, onboard and acclimate them, which is no guarantee that the quality is even there.

PAIN POINT #5: LOW CODE IS NOT A SILVER BULLET

You can't just inject "low code" across your business, in the same way that a single AI solution cannot possibly address all your business automation or manual operational needs.

Low code platforms come in a variety of forms, ranging from the empowerment of non-technical staff (citizen developers) to productivity tools for professional engineers.

Soon, low code will not replace traditional coding, and it does not address complexity. The next killer app in your industry will not come from low code. Rather, it will be developed on a robust and intuitive code base while being layered by modular low-code applications. A simple link between the CRM and your e-commerce site, for example, or e-commerce to your Accounts Receivable system, is an example of this applicability.

Low code will be an iteration in business processes, not a revolution. The game changers and the heavy lift for the platform will still come from the professional engineers, but with low code, those game changers can happen more quickly with the support of citizen developers.

And don't forget governance. Low code is not an exercise in rogue behavior, outside the oversight of your IT group. The citizen coder may not work for IT, but IT rules and governance still apply.

Otherwise, you're going to end up with shadow IT and application sprawl. Low code potentially creates greater risk than benefit. Gartner has estimated that one-third of successful attacks experienced by enterprises will be on data located in shadow IT resources.

IT, no matter how well funded, provisioned, or staffed, cannot satisfy the application needs of any business on its own. As we learned from the Cloud evolution a decade ago, IT can be the hero of the story by allowing secure access to the stack, with agreed-upon rules and a permission hierarchy. As long as IT clings to a gatekeeper role, success is not possible.

With the advent of low code tools, IT can outsource the lower-risk, lower impact, more mundane applications. Moving from a Protector to Enabler role, IT can provide the templates and building blocks for high-throughput solution development at a business process level, while also turbocharging those user-facing applications that drive engagement and revenue.

In summary, low code is not a silver bullet to comprehensively solve the acceleration of applications, nor is it a replacement for traditional coding. An informed application development strategy recognizes which tools enable and empower citizen coders and which ones enhance engineer productivity, and deploys them in a targeted fashion.

PAIN POINT #6: THE SILOED TESTING TEAM

These four roles can play a significant role in the testing lifecycle if utilized correctly:

- **Business Analyst: defines business requirements and functional flows for integrated applications**
- The BA is a triple threat. The BA knows the business, knows the customer, and has a reasonable idea of the solution, but is unable to deliver on that knowledge with his or her current expertise.

The BA needs to quickly test changes and modifications to business flows on applications, with a testing experience similar to no-code low-code app development tools. And it's not just testing one application. There's also the exponential challenge of coordinating one or more test cases that span multiple applications. How can this be accomplished? How can you leverage an individual BA, and the entire BA team, by giving access to simple IT tools to execute on business logic and application ideas?

- **Functional Tester: functional implementation and validation of the application(s) at test**
- Testers are similarly underutilized. Many testers are adept in just one mode skill set and can't cross into other modes easily. Even if enterprises have enough testers available, they are often committed to other projects.

How can you help your functional testers with limited or no test automation code writing experience? How can you increase agility (speed to test) among the varied skill levels of your testing team, while addressing ongoing testing needs in the development phase, ongoing maintenance, and upgrade cycles?

- **Quality Assurance (QA): ensures the final product adheres to the company's standards and delivers digital confidence**
- QA gets lost in the shuffle, struggling to keep pace with Agile workflows and shorter product release cycles. As product engineering migrates from waterfall to DevOps and Agile on cloud platforms, testing needs to support this transformation.

Learning curves in software development can be extraordinarily steep. It might take days for QA to learn an area of a product well enough to switch gears rapidly. For example, several stories carry over into "Ready for Test" status but are unable to be tested by the end of the sprint, which leaves engineers sitting idle waiting for QA to catch up.

QA is traditionally seen as an add-on step at the end, and not necessarily integrated into the development process. *Is the product complete and thoroughly tested and debugged? Okay, let's toss it over the fence QA.*

Shorter product release cycles require continuous agile maintenance of test assets, across a broad range of device and browser formats. How do we help QA keep up?

- **Software Developer Engineer in Test (SDET): responsible for writing and maintaining the automation code**
- The Software Development Engineer in Test (SDET) role originated at Microsoft and it's a buzzword-worthy job title for those organizations seeking a more technical resource to participate in application development, in place of a manual tester or quality analyst, with no software development interest or skillset.

- The SDET is capable of developing high-performance code useful in the automation of test cases, or in designing the testing framework. SDETs can also assist in reviewing the design and processes of a software product.

As engineers, even the SDETs face challenges in testing, unable to complete testing in short sprint cycles (in-sprint testing). Generating and maintaining test automation with SDETs is not scalable. They need a test automation tool that integrates with a CI/CD pipeline, and one that supports modern application architectures and open source frameworks.

How can we help SDET's sprint and then scale their efforts?

This is an oversimplified explanation of the four roles but aims to provide a picture around the specialized skills and competencies of each. As organizations evolve and as we also start to see the cross-pollination of skills the line that distinguishes these roles continues to blur. While these four roles do not represent sunk capital costs, they do reflect fixed overhead expenses, where leverage is even more valuable and critical in times of talent shortages, given acquisition, onboarding, training, and retention costs.

When collaboration isn't enabled, the BA, Functional Tester, QA, and SDET operate in silos and the outcome is a five-way drag on software development and business velocity:

1. Slow-release cycles
2. Increased defects,
3. Increased QA effort and costs
4. Inefficient business processes
5. Dependency on a finite number of specialized testing knowledge

SOLUTION

A low-code test automation tool is a solution that, when matched with low-code development, can help mitigate risk, accelerate innovation, and increase efficiency across the entire software development life cycle. To add, breakthroughs for organizations will emerge when there is a collaboration between various cross-functional teams.

Low-code testing represents a unique opportunity to enable this collaboration and drive out a major structural cost center—technical debt

– and replace it with an "always-on" infrastructure, that is not only more efficient, but more powerful in terms of its scalability, coverage, awareness of change, and responsiveness to downstream factors. High-performing product teams receive the compound benefits of more frequent deployments, shorter lead times, and lower change fail rates.

As the market for elite knowledge workers tightens month-by-month, business leaders are looking for a more manageable growth strategy. The U.S. Bureau of Labour Statistics calculated in 2020, that the unemployment rate of software professionals was 1.9%.

Competitive advantage in this territory will go to the organizations who drive their variable cost structures to zero utilizing cloud technology, while at the same time incrementally scaling their coverage and reducing their exposure to change disruption.

Autonomous low-code/no-code testing allows an organization to grow technical infrastructure, sustainability with non-technical teams while addressing many of the technical challenges inherent when innovating and delivering new products.

SOLUTION BREAKTHROUGHS

The Test Ecosystem represents just one of the ways to recognize the business value of the low-code, no-code, Citizen code trend. Solutions like AutonomIQ can create an order of magnitude in breakthroughs for the BA, Tester, QA, and SDET roles. It's not enough to double or triple the productivity of these key positions.

A typical resource allocation on a software project might be: Create Build Requirements (35%), Production (40%), and Testing and QA (25%). With low-code tools and automation, you still allocate 25% of your effort to testing. Automation may or may not reduce testing, but it does enable greater coverage and generates more test outcomes (more bugs detected, more code fixed). You get to test more thoroughly and you leverage the expertise of your product owners and business analysts. Nothing slips through because the technical headaches are removed and all specialized workers can collaborate effectively on the same product. A key tenant of Agile development is collective ownership in the delivery of every product line deployed and maintained. Low code application platforms offer this at scale.

Low-code is not futuristic wishful thinking and AutonomIQ helps deliver this winning formula to cross-functional teams: remove bottlenecks to innovation and delivery by encouraging simplicity in an environment riddled with siloed activity and overwhelming complexity. Furthermore, it expedites the delivery of an elevated user experience and digital confidence that surpasses customer expectations, surprises them exclusively on the positive side, and drives community expansion, resilience, and growth. Lastly, this allows teams to overcome the challenges with automated testing and seeding the positive outcomes of test automation from day one and into the future. It's only a matter of time until we start to witness the citizen code breakthroughs that enterprises can leverage to exceed their digital transformation goals.

ABOUT SAUCE LABS AND AUTONOMIQ

As the engine of low/no-code testing in the Sauce Labs DevOps Testing Toolchain, AutonomIQ empowers citizen testers, and IT professionals to automate their functional testing and thoroughly test both the UI and APIs in custom and SaaS applications. With our AI-driven and codeless studio, our customers and partners can significantly shorten test sprints, improve code quality, achieve running full regression cycles, and keep the regression suite up-to-date after each release.

AutonomIQ delivers transformation projects faster without compromising quality. You can deploy AutonomIQ locally on-premises or in the cloud or hybrid cloud.

SAUCELABS

ABOUT SAUCE LABS

Sauce Labs is the leading provider of continuous testing solutions that deliver digital confidence. The Sauce Labs Continuous Testing Cloud delivers a 360-degree view of a customer's application experience, ensuring that web and mobile applications look, function, and perform exactly as they should on every browser, OS, and device, every single time. Sauce Labs is a privately held company funded by Toba Capital, Salesforce Ventures, Centerview Capital Technology, IVP, Adams Street Partners and Riverwood Capital. For more information, please visit saucelabs.com.

saucelabs.com/signup/trial

FREE TRIAL

SAUCE LABS INC. - HQ 116 New Montgomery Street, 3rd Fl San Francisco, CA 94105 USA