Low-Code Platforms Deliver Customer-Facing Apps Fast, But Will They Scale Up?

To Answer "Yes," Look For Scalable Architectures, Flexible Pricing, And Appropriate Features

by John R. Rymer and Clay Richardson

August 11, 2015 | Updated: August 13, 2015

Why Read This Report

Low-code platforms are an important strategy to speed delivery of software to win, serve, and retain customers. Because application development and delivery (AD&D) professionals adopt low-code platforms by starting small and then growing, the question of whether these platforms will scale to support large apps and high numbers of apps is crucial. Low-code platforms will scale up as vendors architect them to do so. Look for product architectures designed to support high scale, features to coordinate the efforts of multiple development teams, fully expressive tools, support for governing large portfolios of applications, and flexible pricing models. Read this report for a guide to obtaining both fast app delivery and scalability from low-code platforms.

Key Takeaways

Most Low-Code Platforms Will Support Large Applications

Low-code platforms are best used for apps that scale using well-known web architectures -- systems of customer engagement and adaptive front-ends to systems of record. Vendors report average app sizes of between 200 and 2,000 concurrent users, but we've found one app serving 41,000 users globally and another serving 21,000 call-center agents.

To Assess A Low-Code Platform's Scaling Potential, Evaluate Five Factors

Five factors determine how well a low-code platform will scale: platform architecture (with emphasis on data management), tool expressiveness, common services and components, application life-cycle management, and pricing models.

Make A Strategic Commitment Only To Scalable Low-Code Platforms

AD&D teams use low-code platforms for limited purposes at first. If those low-code platforms can scale up to large apps, large portfolios, and multiple scenarios, the product is worthy of strategic people, technology, and process investments. If not, use the platform for prototyping, departmental apps, and other limited purposes.

Low-Code Platforms Deliver Customer-Facing Apps Fast, But Will They Scale Up?

To Answer "Yes," Look For Scalable Architectures, Flexible Pricing, And Appropriate Features

by John R. Rymer and Clay Richardson
with Christopher Mines, Duncan Jones, Dominique Whittaker, Joseph Miller, and Ian McPherson

August 11, 2015 | Updated: August 13, 2015

Table Of Contents

Notes & Resources

Forrester conducted 33 in-depth interviews during 2014 and 2015 with customers using low-code platforms to deliver systems of engagement. Forrester conducted in-depth interviews on low-code platform scaling with Appian, Bizagi, ClaySys Technologies, CrownPeak, Intuit, K2, MatsSoft, Mendix, MicroPact, OutSystems, and Salesforce.

Related Research Documents

Five Customer-Facing Scenarios Shape "Low-Code" Platform Choices

New Development Platforms Emerge For Customer-Facing Applications

FORRESTER®

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

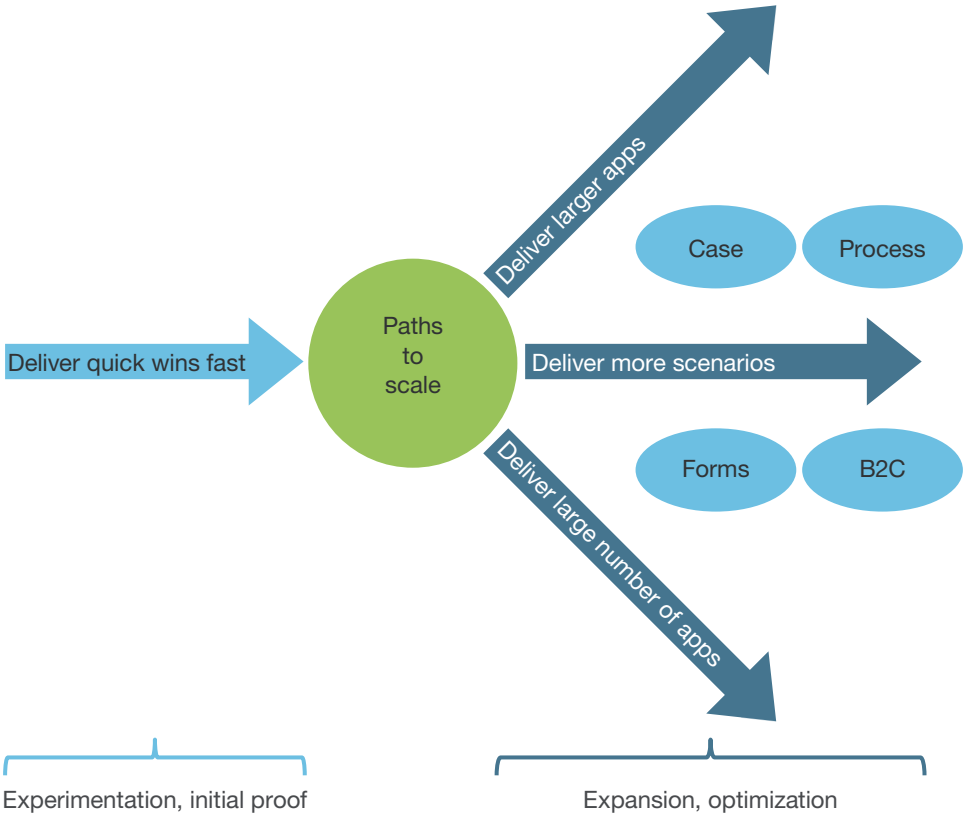
Will Today's Low-Code Success Hit A Brick Wall Tomorrow?

Low-code platforms are catching on with AD&D organizations desperate to rapidly create and update applications to win, serve, and retain customers. With these platforms, customers have delivered apps ten times (sometimes more) faster than conventional methods.[1] Because adoption of these products starts small and then expands project by project, the experience with the first successful project won't reveal much about the platform's ability to scale. Thus, after the first one or two successful projects, low-code customers ask, "How far can I go with this platform?"

Will low-code platforms scale? This question breaks down into three additional paths customers hope to take with low-code platforms after their initial wins (see Figure 1). They use them to deliver:

- **Very large, complex apps.** Customers typically start small with low-code platforms — often with "long-tail" or departmental applications serving small numbers of customers or employees. Before long, they're contemplating mission-critical business applications supporting large numbers of people, managing large numbers of transactions and data/objects, and/or operating with high reliability.
- **A wide range of application scenarios.** Customers often adopt a low-code platform because its integral application framework suits a particular application type very well: forms-based applications, for example, or straightforward workflows. Then they seek to apply the platform's framework(s) to additional scenarios: for example, a sales tool for tablets.
- **Large numbers of applications.** What is the largest number of applications we've seen on a low-code platform? 15,000; 7,000 of which were in active use. That's an extreme case, but it illustrates that initial success with low-code platforms can breed many application projects — and result in the need for features to manage those several applications.

FIGURE 1 The First Low-Code Success Leads To Three Scale-Up Demands



The Stakes Are High

The stakes are high for both customers and vendors. AD&D teams — often embedded in a business unit — use low-code platforms for limited purposes at first. As they attempt to expand from initial successes, two scenarios are possible:

- **A tactical solution can become strategic.** If the low-code platforms scale up, they could become as strategic as, say, .NET or Java. In this case, the delivery teams will derive higher and sustained value from the product. They'll also have to invest in the governance needed to manage a strategic platform.
- **A tactical platform can become tomorrow's legacy.** If the low-code platform either cannot scale up or can only scale up in limited ways, that platform will be just another in a long line of hot-today, legacy-tomorrow products.

Low-Code Platforms Scale Within Their Architectural And Cost Limits

Low-code platform vendors report an average application size of between 200 and 2,000 concurrent users. But most of the vendors can cite larger applications as well. For example, PointBeyond built a Nintex application that serves 41,000 users globally. Appian has a financial-services customer running 1,025 discrete applications on its platform and a travel-services firm running 2,300 process instances on its cloud platform. ClaySys' largest customer runs applications supporting 21,000 call-center agents, and another runs 500 forms in a rebuilt front-end for a customer's ERP system. A large Portuguese insurer built and deployed its entire claims management system on OutSystems' platform.

But not all of the products that embrace a low-code approach scale equally well. Low-code platforms can scale if architected to do so, subject to functional and cost limitations, as well as customer configuration, of the product. Some low-code platforms have the features and functions needed to scale in all three scenarios described above, while some scale best in one or two of the scenarios.

To assess a low-code platform's potential scalability, evaluate these five factors:

1. **Platform architecture should be designed to scale.** The platform's structure, particularly its approach to data management, is crucial to scaling. Some platforms take full responsibility for data management, adding developer convenience, while others simply integrate with established systems of record databases and content stores.
2. **Tool expressiveness should be expansive.** Low-code's promise is to dramatically reduce the amount of coding required to deliver an app. To deliver, a platform's declarative tools should address all aspects of an application's design from user experience to logic to data operations, as well as change control and administration within the platform's target scenario(s). This challenge grows as customers address multiple scenarios with the low-code platform of their choice.
3. **Shared components, services, and catalogs help fuel large portfolios.** The name of the game in low-code platforms is fast initial app delivery and updates. Platforms that provide common app components — for user experience construction, data integration, process types, and the like — meet this requirement best. Tools that help AD&D teams create and manage common components and services are also very valuable.
4. **Multiple delivery teams need built-in life-cycle management.** Common components help speed the work of multiple delivery teams, but not to coordinate their work. For that, teams need project and life-cycle management facilities either built into the low-code platform or easily integrated with it.
5. **Costs should reflect real value to the customer.** AD&D decision-makers select low-code platforms in part because initial costs are low compared to typical enterprise software. The pricing models should support cost expansion that reflects the platform's business value, giving clients the ability to influence how costs rise as platform usage expands.

Architectural Patterns Offer Tradeoff Between Speed And Control

Low-code platforms will support building large, complex apps, but AD&D leaders will need to look under the hood at each vendor's application architecture, and to weigh tradeoffs that the vendor's architecture offers between speedy delivery and future control. When they do, AD&D leaders will find two architectural patterns.

Architectural Pattern No. 1: Comprehensive Application Management

The first pattern speeds app delivery by shielding developers from tasks like configuring database connection pools, platform scaling, and managing user web sessions. These platforms take full responsibility for managing an application's architecture, offering user-interface frameworks, virtual data layers, automatic scaling, and other features.

The disadvantage of this architectural pattern is that it limits AD&D's ability to control the application architecture. AD&D has only limited ability to configure platform components for performance and scale. The risk is that the low-code platform vendor is unable to effectively scale its proprietary frameworks and platform components over time. To get a handle on how this type of comprehensive application management will affect future scale and performance, evaluate how the low-code platform:

- **Implements virtual data layers to hide data and integration complexity.** Virtual data models present data from numerous back-end databases and systems of record as uniform collections of objects, helping to manage data connectivity. These virtual data models can be configured to read, update, and synch data across different data sources, but are a potential scaling bottleneck. They must be kept in sync with the back-end sources they represent, and as the volumes of objects virtual data layers manage grows, the layer itself must be able to scale up.
- **Leverages rendering engines to translate forms for Web and mobile.** Applications built on low-code platforms will need to be accessed by users across web, smartphone, and tablet devices. Some low-code vendors manage this presentation complexity by embedding their own custom-built rendering engine as part of their platform architecture. These rendering engines allow developers to build once, and write everywhere; translating a single form or user interface to fit and operate across different target devices.

  While this can eliminate the need to write multiple custom user interfaces to support different devices, it also restricts developer control over pixel-perfect layout and the ability to embed user experiences into third-party applications.
- **Embeds state engines for processing application and flow logic.** Some low-code platforms manage and execute application and workflow logic using state engines and built-in business rules engines that drive the logic based on a triggering event or condition. Evaluate how the vendor's state engine and rules engine perform under the stress of large transaction volumes and how these engines can be tuned for performance.

Architectural Pattern No. 2: Limited Application Management

Many platforms rely on the well-known web architectures to scale up apps. To scale, add resources to the web, application-server, or database tiers of the architecture. In these cases, the vendor's platform is only one element in the overall application architecture, but not all of it. When evaluating web architectures for low-code, consider each vendor's ability to:

- **Configure core databases and connections using available DBA tools.** These vendors rely on conventional relational DBMS but don't take responsibility for how that database software is scaled. Some vendors leave it to the developer to configure and optimize their own database connections using the built-in management tools provided by popular relational databases. This approach offers maximum scalability and control when it comes to database performance, but it also requires developers with DBA skills and can introduce an additional layer of complexity to manage.
- **Rely on existing web architectures that support scalable deployment.** Instead of providing custom rendering engines, some low-code vendors rely on the native web rendering engines provided by web application containers, such as Apache Tomcat, Jetty, and the .NET Common Language Runtime. These low-code platforms typically don't rely on embedded state engines and business rules engines but translate application logic and rules into programming constructs. The result is that developers can optimize code that is generated by the platform. Also, developers can apply custom stylesheets, forms, and user interfaces to their apps.

  The downsides are that the speed benefits of low-code are available for only a portion of app projects, and change management is made slower by app architectures incorporating multiple platforms.

Tool Expressiveness: Deliver 100% Of Apps Without Coding

Programming languages are effective at addressing many types of applications; the declarative and visual tools at the heart of low-code platforms are most effective in narrow domains. The promise of low-code platforms depends on how well a product's declarative and visual tools allow developers to define all aspects of their apps within the platform's domain. After your team's initial project, scaling up will require that the platform's tooling be expressive enough to address additional applications and scenarios. Your vendor should have a commitment to incorporate new technologies into its tool set over time.

Products with declarative tools that scale to cover all or most of your application's requirements will yield the greatest productivity benefits. Look for:

- **All of the functions your application requires.** Low-code platforms are all either database- or process-centric, and this orientation is usually reflected in their built-in tools. However, all platforms must address the fundamentals: user permissions, either data modeling or data access, various forms of logic, user interface creation, reporting, and platform administration. Some platforms directly address application concepts like case-management logic, social-style user experiences, API creation, and documents and web content as well (see Figure 2).

Each low-code platform vendor addresses the problem of comprehensiveness in one of two ways: Either with a single tool set adaptable to many purposes or with a tool set incorporating multiple designers. A single tool set will be simple to learn but may be an awkward fit to new concepts like activity feeds and machine learning. A multidesigner tool set is more difficult to master and will slow down app delivery until your developers gain mastery.

- **All of the user experience designs and logic.** User experience is the most challenging area, for two reasons. First, customers report struggles in using low-code platforms for complex forms and pixel-perfect mobile applications. Some customers are forced to use external products (sometimes coding) for their forms or mobile apps, which breaks the low-code promise. Second, user experience is a fast-moving area, and thus challenging for the vendors to keep up with new ideas. Mobile offline is an example.[2]
- **The data-handling functions your app requires.** Customers have two choices in evaluating the data-handling tools provided by low-code platforms: Delegate all but data binding and integration to an external database layer, or build a robust data layer into the platform itself. If taking the latter approach, the platforms should provide tools for modeling the data layer and synchronizing it with primary data stores. If the platform delegates data-management to external data stores, then look for mapping, query, and CRUD logic tools.
- **Extensions and APIs to address missing, new, and external functions.** Technology is evolving too quickly for any low-code vendor to keep up with in its tooling. The next best thing to explicit support for a new application concept in a product's declarative and visual tools is easy integration of functional libraries (e.g., specialized math or encryption) and external platforms addressing that concept. Easy integration requires architected extension points and APIs for the low-code platform.

FIGURE 2 Potential Application Concepts In A Low-Code Platform

Data	R/F modeling
	Globals/constants
	Data stores
	Data and record types
	Queries
	Integration
Logic	Business rules
	Transactions
	Process/workflow
	Case-management
	Social
User experience	Forms
	Pages
	Mobile/adaptive design
	Mobile offline
	Navigation
	Search
	Apps
Identity/security	Permissions
	Groups
	Sharing rules
Reporting	Reports
	Dashboards
	Analytics
Platform	Administration
	APIs
Content	Documents
	Web content
	Document folders

Large Portfolio And Team Support Demands Life-Cycle Features

Our interviews confirmed that low-code platforms can support large and diverse portfolios of applications developed by multiple teams across the enterprise. For example, one company used its low-code platform to deliver 40 different applications across eight different development teams. This embedded life-cycle management helps facilitate collaboration and coordination across multiple development teams and development environments.

Look Beyond Table Stakes To Evaluate Support For Large Portfolios

Low-code platform vendors provide basic life-cycle management features, including version management, shared repositories, and application testing. But even with these basic features we encountered some end user companies that complained of deployment and life-cycle management challenges. To understand whether the low-code vendor's life-cycle management features will help or hinder delivering across a large portfolio, ask how they:

- **Speed deployment by providing visibility into application dependencies.** Some low-code vendors provide shared repositories for building, managing, and deploying applications. These shared repositories provide a single place to view apps, components, and services, and provide insight into which deployed apps are actively used and assist in managing application and component versioning.

  Going a step further, some low-code platforms maintain indexes of relationships and dependencies between apps and design components (such as web services and database connectors). OutSystems and Appian, for example, provide dependency diagrams developers and administrators use to quickly evaluate how a change to a shared artifact will impact different apps in the repository.

- **Simplify agile delivery through built-in team collaboration tools.** Coordination, communication, and collaboration are constant challenges when managing development across multiple teams. At the most basic level, low-code platforms address this challenge through built-in version management features that keep developers from stepping on each other's toes while building and updating applications. Vendors also provide built-in version management support for branching and merging app functionality built across different teams.

  Going beyond table stakes version management features, we found some platforms providing built-in Agile development collaboration tools. These built in tools allow team leads to delegate development tasks and features and track progress through burn-down charts.

- **Scale due diligence through automated performance testing.** Each newly deployed app has the potential to degrade performance across the entire portfolio of apps running on the low-code platform. As part of normal due diligence before deploying new apps, development teams will need to test the impact a new app will have on the environment, and make any necessary adjustments.

Some low-code platforms are introducing automated tools that help teams test the performance of their apps for large scale deployments and roll-outs. This means teams can run automated performance tests across large bases of users and sets of data.

Costs Scale Depending On Pricing Models And Scope

Developers rarely consider cost when they first adopt a low-code platform. The initial costs are lower than traditional enterprise software — even free in some cases — and their first project has a limited scope. However, after the first project success, AD&D leaders must model how costs are likely to grow based on the vendor's real prices rather than introductory prices. when looking to expand after the first project successes, customers must be concerned about long-term costs.

The nightmare scenario for low-code platform customers: The product is too expensive to apply to many projects — particularly exploratory, "test-and-learn" projects — and so the initial project becomes first an orphan and ultimately an oddball legacy. Understanding how vendor pricing models work is, thus, essential to a successful strategic adoption of low-code platforms.

Low-code platform vendors seem to use a wide variety of pricing models, but they are all variations on two basic pricing models (see Figure 3):

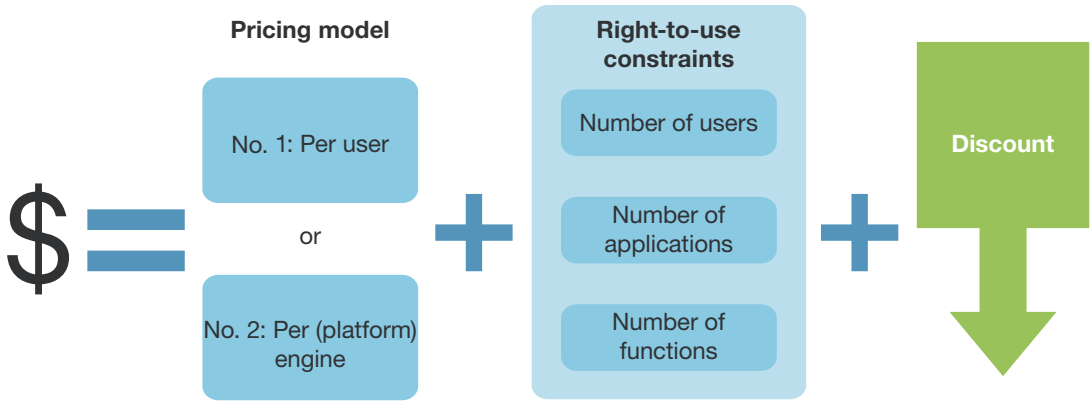
- **Per-user pricing is fine for specialist apps but may prevent broad adoption.** Vendors that use this model usually have a BPM or workflow background. Per user pricing should reflect different personas' depth breadth and frequency of usage.[3] Many low-code providers do this, but some, including Salesforce, have one flat price per user. A single price per use is fine for specialist apps that deliver high business value to a small user base, but they can make the platform too expensive for simple apps that everyone will use.

  Per-user pricing is good at full adoption when everyone is using lots of low-code apps, but you may never reach that point. Negotiate a low or zero price for infrequent users if the vendor does not have one in its standard model, or avoid using that platform for broad use apps. You should also ensure that there is no charge for unauthenticated users (people visiting, say, a marketing website).

- **Infrastructure-based pricing is better initially, but costs may spiral out of control.** This model is similar to Java application-server pricing; in most situations it ignores user counts. Under this model, vendors charge customers for instances of their engines or platform runtimes. They either measure processing capacity, or factors that drive that such as the number of apps and/or functions.

  These models are best for widely used applications, particularly customer-facing ones, but they can make costs hard to understand, and thus control. The model keeps costs low initially, but greater numbers of users, growing data volumes and increased app complexity can drive up costs unexpectedly. One way to retain control is to negotiate an unlimited enterprise-wide license when you are approaching critical mass adoption of the platform.

FIGURE 3 How Low-Code Platform Pricing Works



Recommendations

Move Scaling To The Front Of Your Low-Code Platform Due Diligence

Our research provides a framework to evaluate the scaling potential of low-code platforms — a dimension we've found is missing from most product-adoption decisions. In most adoption decisions, the need for speedy app delivery is the primary concern. These customers adopt low-code platforms based on their productivity potential, trusting that scalability will be good enough. We recommend that scaling potential be a first-order consideration in adopting one of these products.

Structure your due diligence of low-code platforms as follows:

- **Decide on the role low-code platforms will play in your app architectures — and why.** You've got two choices: 1) Use a low-code platform to create and manage front-end applications and processes only; or 2) use a low-code platform to manage end-to-end application architectures, including data management. The first architectural pattern leverages your enterprise's investments and support processes for core data, but slows down change management. The second architectural pattern allows fast changes, but introduces new data infrastructure (the low-code platform's) that must be coordinated with primary data sources and must be built to scale.
- **Design processes to enable rapid app change at scale.** Speedy ongoing app changes are crucial to customer-facing apps, and your selected platform should help address the challenge. As noted in the prior recommendation, though, you'll have to set up processes and practices that allow your chosen platform to meet your enterprise's requirements for speedy app changes.

If you choose a low-code *front-end* platform, design a process to ensure demands for database and apps changes get the highest priority, and design testing regimes to account for dependencies between the layers of your application architecture. If you choose a comprehensive low-code platform, define processes to ensure prompt synchronization of your virtual data layer with primary sources and capacity planning for the platform.

› **Go for completely expressive tooling, but invest in patterns and practices as well.** Your goal: Declarative tools that cover 100% of requirements in your chosen scenarios. Incomplete coverage will require you to incorporate another product and usually to write code, reducing the speed benefits of your low-code platform choice.

At the same time, invest in creating common rules, workflows, integrations, and other components for your developers to reuse, saving time, as well as patterns for application design, life-cycle progression, error handling, version management, and deployment to promote apps that perform well and scale. Inefficient user experience designs can cause slow app performance, especially at scale. Thus, vendor guidance on design can save a lot of time in creating applications that perform well.

› **Prioritize support for large app portfolios first, then for multiple teams.** Only some of the low-code platforms offer good support for managing large portfolios of apps, services, and components. And only some provide more than basic app versioning features to help multiple development teams coordinate their work. Look for dependency analysis and management and application deployment tools. If you can't get these features from the low-code platform vendor, you'll have to build them yourself using external tools.

› **Determine how you'll influence costs with your usage and practices.** The simple rules-of-thumb apply only when applications are uniform. For example, an industrial distributor used a low-code platform strictly to automate its processes for maintaining master data about customer accounts, but for no other applications. The scope and functions required to deliver these applications are narrow, and only a small number of people use the applications. Thus, the per-user model worked well.

With more diverse application sets, you will need to look at the cost controls that the vendor provides to you — and your tolerance for their complexity. In strategic uses of low-code platforms, visibility into cost drivers and controls over those costs are much more important than pricing models. However, models that give AD&D pros control over factors that drive costs are complex. The per-engine vendors provide such controls, but your organization must be set up properly to really benefit from using them.

What It Means

Low-Code Platform Usage Will Expand Into Mission-Critical Apps

Customers adopt low-code platforms first to build portals, dashboards, web apps, workflows, mobile apps, and other systems-of-engagement patterns. These are crucial applications, but usually not classified as "mission-critical" applications. As customers prove the scaling potential of low-code platforms, they will inevitably begin applying those platforms to mission-critical applications — back-office record-keeping and transaction processing applications that support customer, supply-chain, and production operations.

Why? The age of the customer's imperative for quick delivery of new apps and changes applies also to mission-critical systems. Indeed, the typically slow-paced change cycles for back-office systems gave stimulus to creation of independent systems of engagement that developers could build and evolve on faster schedules, and thus, to low-code platforms.

AD&D leaders attack this problem by applying agile methods to their back-office development projects — even COBOL and SAP ABAP projects — and by replacing older packages with SaaS and other packages employing modern architectures. And some move back-office mission-critical apps onto low-code platforms. It is a trend we expect to accelerate as more examples surface of large-scale, critical apps developed and running on low-code platforms.

Engage With An Analyst

Gain greater confidence in your decisions by working with Forrester thought leaders to apply our research to your specific business and technology initiatives.

Analyst Inquiry

Ask a question related to our research; a Forrester analyst will help you put it into practice and take the next step. Schedule a 30-minute phone session with the analyst or opt for a response via email.

Learn more about inquiry, including tips for getting the most out of your discussion.

Analyst Advisory

Put research into practice with in-depth analysis of your specific business and technology challenges. Engagements include custom advisory calls, strategy days, workshops, speeches, and webinars.

Learn about interactive advisory sessions and how we can support your initiatives.

Supplemental Material

Companies Interviewed For This Report

Appian

Bizagi

ClaySys Technologies

CrownPeak

Intuit

K2

MatsSoft

Mendix

MicroPact

OutSystems

Salesforce

Endnotes

[1] Precise comparisons between app delivery using low-code platforms versus using conventional coding and package implementations are difficult to find. Four of the examples we've found illustrate the range of possible speed improvements. A health ministry delivered a patient administration system in five weeks using a low-code platform versus an estimated three years required using a packaged app. A European insurer delivered the minimum viable product for an agent portal in 10 days; this was a project that had been on the IT backlog for at least two years with little prospect of being completed. A large call-center operator reduced the time necessary to implement software supporting new clients to three weeks from four months. And a Spanish insurance carrier implemented a new web portal in 13 weeks, far faster than the 2.7 years originally estimated for the project.

[2] "Unfortunately, our experience shows that offline support is the mobile app feature continually underscoped by developers and over-simplified by stakeholders." For more information, see the "The Offline Mobile Challenge" Forrester report.

[3] Systems of engagement such as business intelligence, collaboration technologies, and mobile applications are high priorities for business and technology decision-makers at enterprises. These applications expand data capture and information access beyond the primary user populations of the underlying applications. Persona-based per-user pricing of these applications matches each user's cost to not just their usage breadth (i.e., modules they access), but also its depth (features within those modules) and frequency (how often they use the software, and for how long). This role profiling should be primarily top down (from analysis of employees by job function) rather than bottom up (from detailed tracking or limitation of access to specific functions). The goal is to match price to value, not to create a usage-tracking nightmare. See the "Four Software Licensing Trends That Support Your Business Technology Agenda" Forrester report.

FORRESTER®
CHALLENGE THINKING. LEAD CHANGE.

We work with business and technology leaders to develop customer-obsessed strategies that drive growth.

PRODUCTS AND SERVICES

- Core research and tools
- Data and analytics
- Peer collaboration
- Analyst engagement
- Consulting
- Events

Forrester's research and insights are tailored to your role and critical business initiatives.

ROLES WE SERVE

Marketing & Strategy Professionals

CMO
B2B Marketing
B2C Marketing
Customer Experience
Customer Insights
eBusiness & Channel Strategy

Technology Management Professionals

CIO
- Application Development & Delivery

Enterprise Architecture
Infrastructure & Operations
Security & Risk
Sourcing & Vendor Management

Technology Industry Professionals

Analyst Relations

CLIENT SUPPORT

For information on hard-copy or electronic reprints, please contact Client Support at +1 866-367-7378, +1 617-613-5730, or clientsupport@forrester.com. We offer quantity discounts and special pricing for academic and nonprofit institutions.