The Essential Guide to Low-Code

By Chris Souther

Author’s Forward

I began writing this eBook while working at OutSystems. I left before it was completed and to my knowledge, no one ever picked it up.

Because it was written with OutSystems particular solution in mind, I have left their product name and relevant information in this eBook. That said, all opinions in this eBook are the author’s alone.

A grammatical note: OutSystems preferred to refrain from any use of an apostrophe when dealing with possessiveness since their company name ended in an “s.” Per the style guide, we always just used “OutSystems” regardless of possession.

Introduction

Welcome to the Essential Guide to Low-Code. This guide is intended for people who either may have only started learning about low-code and want to get the right information about low-code up-front or for those who know a little about low-code and want to clarify and deepen their understanding about what low-code is and what it can and cannot do.

OutSystems is a low-code vendor. We would like nothing more than for you to download the OutSystems free Personal Environment, start using it, love it(!), and become a customer for life. But that's not our goal here. We believe our business, and IT communities at large, will benefit when everyone clearly understands the options and benefits afforded by ALL low-code.

With that as our goal, we have refrained from offering specifics about our platform as much as possible in this guide. However, not all low-code platforms include capabilities described in this guide, so in those cases, we will refer to OutSystems specifically so that you are fully versed in low-code's true potential.

What Is Low-Code?

Forget what you've heard and let's begin anew. Low-code is a visual way to design, develop, and deliver software applications fast.

If you are a developer, imagine moving around visual representations of code (squares, hexagons, circles) in your integrated development environment (IDE) and having the software automatically connect them rather than you typing out the code and logic manually.

"But I'm not a developer." That's the great thing about low-code; you don't have to be a developer.

For business users and others with a technical bent, imagine building an application to make something you do every day, simpler. And instead of creating a complicated spreadsheet or workflow by hand and then trying to document a process that may or may not be followed by everyone, you instead build an app that makes it easy for people to follow an established process. If they use your app, they automatically have to follow the process and provide the mandatory fields of data.

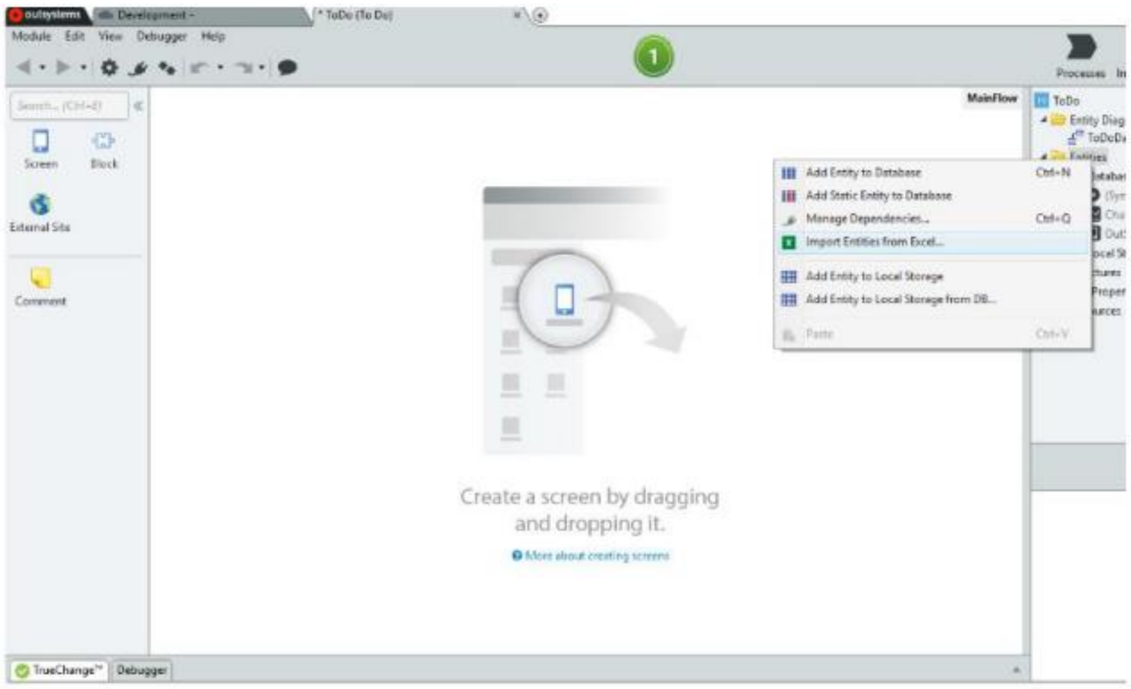

outsystems
Development -
* ToDo (To Do)
Module Edit View Debugger Help
Processes
Search... (Ctrl+E)
Screen
Block
External Site
Comment
MainFlow
ToDo
Entity Diag
Add Entity to Database Ctrl+N
Add Static Entity to Database
Manage Dependencies... Ctrl+Q
Import Entities from Excel...
Add Entity to Local Storage
Add Entity to Local Storage from DB...
Paste Ctrl+V
Create a screen by dragging and dropping it.
More about creating screens
TrueChange™
Debugger

Illustration 1. Low-code development using visual objects

In years past, application development was strictly an IT pursuit, but no more. Low-code platforms have evolved both from a functionality and usability standpoint so that IT can build complex systems to replace aging legacy ones, while line of business (LoB) users can also use it to build applications and process flows to improve their day-to-day. But we will get more into that later.

There is a level of governance (oversight by the IT department) necessary here to ensure that people aren't building and installing apps and tools that are not secure or that may not meet industry or government requirements for things like privacy. Low-code can enforce such governance, allowing "the business" and IT to collaborate and develop software that addresses business problems.

It's a win-win for everyone because, ultimately, low-code is a way to get more done in less time. You spend more time creating and building applications and software, and less time on repetitive work.

Who Uses Low-Code?

Just about every business benefits from some form of a digital footprint, which means that almost every business needs someone, with at least a digital curiosity, to service their digital needs.

According to the OutSystems 2020 State of Application Development Report, which is the most comprehensive annual survey of its kind, 53 percent of organizations are currently using low-code in some way. This matches almost perfectly with a similar survey by Devops.com in 2018 that reported 28 percent of global developers used low-code then and an additional 22 percent planned to in 2019 (for a total of 50 percent).[1]

Independent analyst firm Gartner reports that, "By 2024, low-code application development will be responsible for more than 65 percent of application development activity."[2]

Specifically, low-code is still primarily used by those in IT, although its use by line of business users is growing. We call these business users, "citizen developers."

Let's stop for a moment and talk about this term "citizen developer" because its definition bears heavily on the discussion here and in the future.

What Is a Citizen Developer?

Gartner is credited with coining the term "Citizen Developer" and their definition says:

"A citizen developer is a user who creates new business applications for consumption by others using development and runtime environments sanctioned by corporate IT."

We would like to point out two key points of Gartner's definition of a citizen developer:

1. Citizen Developers builds business applications using tools their in-house IT team knows about and approves of.

2. Citizen Developers builds these business applications for other people in the business.

In the years since Gartner first defined this category of employees who create new tools and applications, a new class of business developer has risen.

According to Gartner's definition, anyone deemed a citizen developer is also carefully monitored and "governed" by their in-house IT. By governed, we mean that the citizen developer doesn't use any tools or build, deploy, and use any software or applications without the knowledge and permission of IT. It's a "perfect world" descriptor in a world that is rarely thus. In reality, there are business developers who:

[1]Low-Code Has Taken Development by Storm. Why Not the Web? 2019, https://devops.com/theimpact-of-low-code-on-the-web/
[2] Magic Quadrant for Enterprise Low-Code Application Platforms, August 8, 2019, Kimihiko Iijima, Mark Driver, Paul Vincent, Jason Wong

- Build business apps without IT's approval, and in many cases, without IT's knowledge.

- Build business apps, not necessarily for other people to use, but for themselves.

Why they do what they do without telling IT is a subject of much debate. But, the industry has, at least, agreed on a term for these business developers: shadow IT. Presumably, these shadow IT workers also install and run their business apps inside the corporate network, also without IT's knowledge.

At issue here is all the gray area in the middle and the "whys" and "whats":

- Why do business users spend their time building apps they want, or need, for their job? Did they ask IT and IT said "No"? Did IT give them something that doesn't fit their needs, so they took it upon themselves to build what they needed? Were they just curious whether or not they could build something on their own? Were they bored?

- What qualifies as a business application? Does a spreadsheet-based tool that helps accounting better track payments to outside vendors qualify as an "application?" It's doubtful IT knows about a random spreadsheet Joe in accounting is using, but is it secure? How is it used? Who uses it? How widely is the information in it shared?

- What if a line-of-business worker heard about low-code from someone who uses it for their IT role, and she takes it upon herself to download a free low-code development environment and then builds an app and starts using it for work? In this case, the "tool" was IT-approved, but the app she built and started using is probably not on IT's radar.

The old definitions of citizen developer and shadow IT are incomplete at best and this has led to rampant misclassification of the activities of people who are just trying to get their job done. They aren't willfully jaywalking across IT's procurement processes; they just need something that does "X."

The bottom line is that low-code democratizes the act of application development, allowing those without a deep background in 5th-generation languages, or specific training in User Design, to build useful applications that increase business agility and efficiency. They aren't bad people, so we shouldn't classify them with some shady "Shadow" descriptor. We should, instead, try and understand what their motivations are, and if in fact they aren't getting what they need from their IT, figure out how to fix that problem.

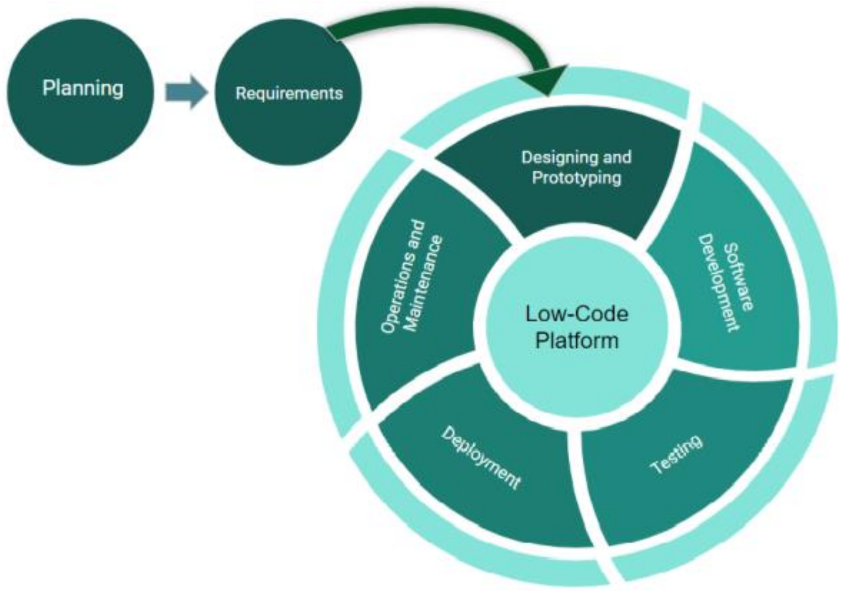
"Are you saying low-code ends reliance on professional development?"

Unequivocally, "No." There will always be a role for professional developers. They are masters of their craft and even if we eventually automate all software development, their experience outside the confines of application development is a necessary and valuable part of the business. In addition, there are novice developers or developers who have specialized in one area or language only (like mobile or Python) who can up their games and become full-stack developers with low-code.

What we are suggesting is that there is a way for non-professional developers to build tools to assist them in their jobs, in a way that both satisfies IT's requirements and that doesn't require IT to build or manage all of the tools they use in their day-to-day jobs.

Low-code and the Software Development Lifecycle (SDLC)

Low-code isn't just an application "development" tool. Low-code automates the software development lifecycle (SDLC) process and combines the separate components of traditional development into one platform.



Software Development Lifecycle (SDLC)

Once the planning and requirements are completed, nearly all other activities related to application development, testing, deployment, and ongoing maintenance, can be done on or through the low-code platform using its native and 3rd-party-integrated capabilities. But let's start at the very beginning of the SDLC and see how low-code improves all parts of the application development process.

Planning and Requirements

The planning phase, also sometimes called the "discovery" phase, should be the first item on any new application development project's "to do" list. In fact, InfoWorld Reports that nearly 70 percent of all development firms require a discovery phase.[3] Discovery phases are necessary because a vast majority of IT projects fail in some way, shape, or form, with 16.5% of them falling apart completely.[4] Since IT and business executives admit that, quite often, these projects are doomed from the very start, it pays (from development firms down to mom-and-pop service companies) to be thorough.

But let’s say we skipped this advice and barreled ahead, confident that we knew what we needed and our customers wanted. Rather than spending weeks interviewing customers and storyboarding out the user experience (UX), we instead spent just a handful of days on it. Now in the development phase of our project, we realize that maybe, just maybe, our requirements weren’t totally spot-on.

Low-code is forgiving. Rather than having to scrap weeks of work and going back to the drawing board, low-code allows for fast prototyping and changing in hours or days what might normally have taken weeks or months. Low-code doesn’t give you permission to cut corners on planning and requirements; it just doesn’t punish you for getting it “not quite right” the first time.

Prototyping and Design

In this phase, the goal is to design and build a working first draft of your final application. There is every expectation that this prototype design will probably not look or work the way the final product will and so you should not strive to make your first draft perfect.

Most people find it very difficult to define what exactly they want, but they have no issues at all looking at something and defining where its shortcomings are and what they would like to see done differently. That’s precisely the point of the stage of the SDLC.

With low-code, in the time it might normally take to develop one prototype to show a customer, you can instead create several, giving the prospective customer, or the UX designer, many different options to pick and choose from on the way to the final design.

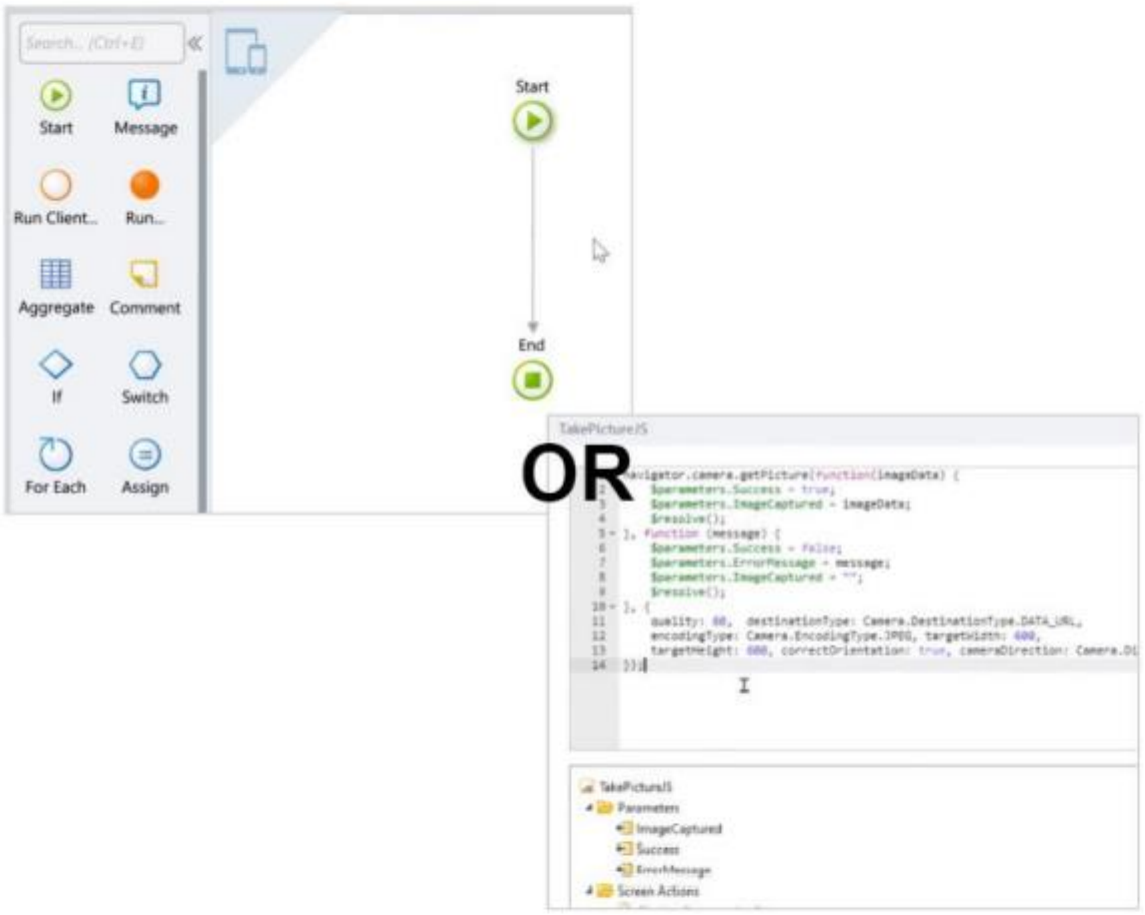
Whether you need to change a few little things or the main “thing” the application is designed to do, you can make those changes using a low-code platform quickly and easily.

[3] Discovery is the most underrated phase of software development, 2019, https://www.infoworld.com/article/3294201/discovery-is-the-most-underrated-phase-of-softwaredevelopment.html
[4] Pulse of the Profession 2018: Success in Disruptive Times, 2018, https://www.pmi.org/-/media/pmi/documents/public/pdf/learning/thought-leadership/pulse/pulse-of-the-profession-2018.pdf

Development (or Coding)

The difference between this stage and the prior one is the level of detail and functionality that goes into it. The development phase is typically the lengthiest phase, but you may find that when using low-code, this phase goes faster than some of the others.

Low-code greatly simplifies the act of application development. Using a drag-and-drop interface, modeling, and pre-built UIs and components, you can assemble and configure applications, all without writing a single line of code—unless you want to. The low-code environment is still a familiar one; it includes an editor where you can see the actual code and make any manual changes you want. The big change, however, is the visual development interface, shown here.



```
avigator.camera.getPicture(function(imageData) {
    $parameters.Success = true;
    $parameters.ImageCaptured = imageData;
    $resolve();
}, function (message) {
    $parameters.Success = false;
    $parameters.ErrorMessage = message;
    $parameters.ImageCaptured = "";
    $resolve();
}, {
    quality: 60,  destinationType: Camera.DestinationType.DATA_URL,
    encodingType: Camera.EncodingType.JPEG, targetWidth: 600,
    targetHeight: 600, correctOrientation: true, cameraDirection: Camera.Di
});
```

Using low-code, you can work primarily in the visual editor or you can split the screen with the visual editor on one side and the code editor on the other, or you can switch back and forth.

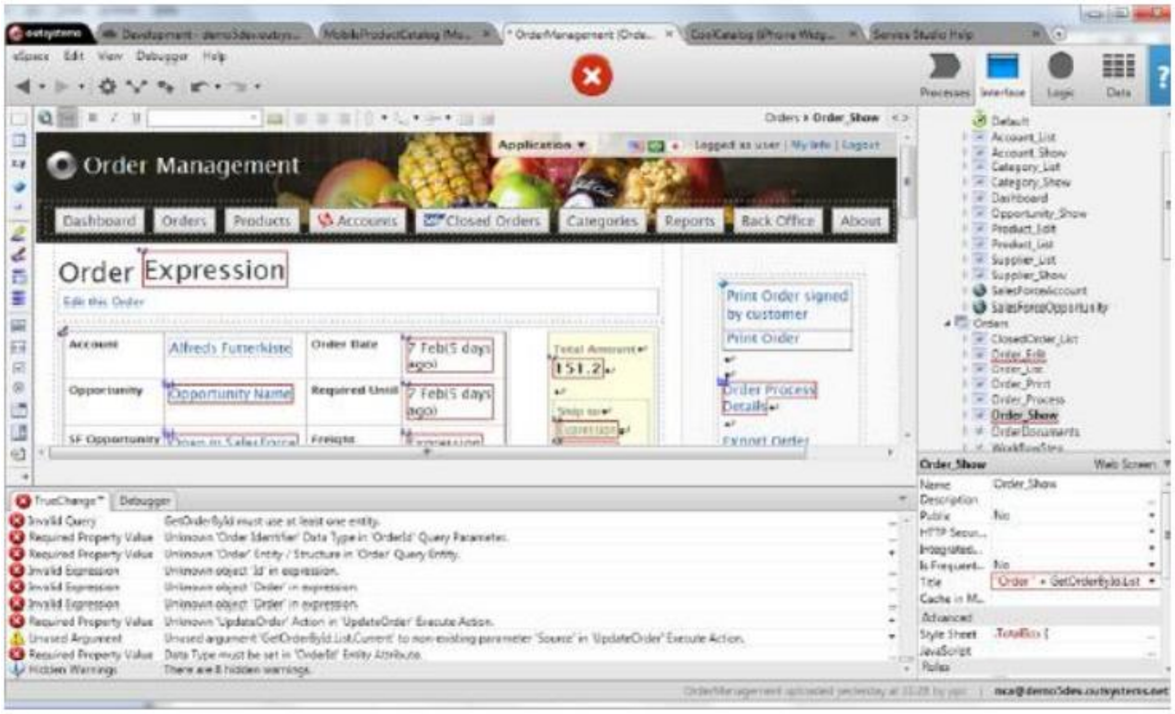
Testing and Deployment

In traditional development, the testing phase and the deployment phase are two distinct phases because they both take time, and they also both typically require specialists. You need a tester who understands how to properly test every part of the software, from its integration components to basic functionality, to ensure it works in all of its deployment scenarios (cloud, on-prem, hybrid). For mobile and web apps, it must be tested for compatibility across form factors, web browsers, or both.

Any problems found require involving the developers again, and that’s before trying to deploy the application, which brings its own set of challenges both before, during, and after deployment.

Here again, low-code streamlines and simplifies these processes. Since the vast majority of the code used (up to 90%) is pre-built, and therefore, pre-tested, you know that most of the code already works.

Before the code goes live, “some” low-code platforms offer automated staging with full impact and dependency analysis, ensuring that what you’re deploying won’t break something already in production.



```
Invalid Query             GetOrderById must use at least one entity.
Required Property Value   Unknown 'Order Identifier' Data Type in 'OrderId' Query Parameter.
Required Property Value   Unknown 'Order' Entity / Structure in 'Order' Query Entity.
Invalid Expression        Unknown object 'Id' in expression.
Invalid Expression        Unknown object 'Order' in expression.
Invalid Expression        Unknown object 'Order' in expression.
Required Property Value   Unknown 'UpdateOrder' Action in 'UpdateOrder' Execute Action.
Unused Argument           Unused argument 'GetOrderById.List.Current' to non-existing parameter 'Source' in 'UpdateOrder' Execute Action.
Required Property Value   Data Type must be set in 'OrderId' Entity Attribute.
Hidden Warnings           There are 8 hidden warnings.
```

(Source: https://www.outsystems.com/evaluation-guide/how-does-outsystems-support-testing-and-quality-assurance/)

In this image, the red "x" marks in the code section at the bottom of the IDE correspond to red outlined fields in the visual order management environment just above it. These red alerts tell the developer what must be addressed before the application is deployed.

But if somehow, something does go wrong when your code is deployed, low-code is your best friend. An enterprise-worthy low-code platform can automatically roll back your changes, preserving uptime and giving you the time you need to identify the problem and fix it before your customers start ringing up the call center.

Operations and Maintenance

We're talking here about what happens post-deployment. How do you keep an eye on things and make sure your applications are working the way they are supposed to—and even get feedback on what users like, or don't?

In addition to general availability, which low-code platforms can certainly monitor, enterprise-worthy low-code platforms include substantial feedback mechanisms, allowing developers, business users, and even end-users the ability to provide feedback from just about anywhere in your application.

And as mentioned previously, low-code's rapid iteration capabilities allow you to constantly update and deploy changes as needed based on feedback or design needs rather than the 1-4 times per month the average application sees an update.

The Layers of Low-Code

In the security industry, the layers of an onion are often used as a metaphor for the different levels of security precautions organizations should employ. Though overused, the onion metaphor is an apt description for low-code platforms as well. On the outside of the platform is a very user-friendly wrapper of functionality and capabilities. But the deeper you move into the platform, the more technical acumen it demands, while becoming far more useful overall. Let's peel the onion.

The No-Code Layer

What sounds like it should be its own category--no-code--is often equated with low-code. Even industry analysts do it. Probably more than any of the other flavors of low-code we will discuss here, no-code bears the most explanation.

No-code simplifies the act of application development to the extreme. It strips away most of the technical know-how and gives users nearly everything they need to create apps completely from scratch. Some use-cases where no-code is a great fit:

- A local school PTA wants to build an app to accept payments for fundraising.

- A line-of-business user wants to build a process-based app for tracking the onboarding of new customers; the type of app that requires registering new customer information across systems and setting up "next steps" that can include anything from "Welcome" email, to dispatching an installation technician.

But, no-code in its purest form is far too limiting for most enterprises to use as a primary development platform. For starters, no-code assumes users are not primarily concerned with building something different from what everyone else has. Most business developers just need something that works and if it works similar to something they have used before, all the better. No-code's design considerations also assume that users will not want to, or have access to, modify the actual codebase.

While no-code as a "feature set" is desirable under the right circumstances, platforms operating as a no-code, one-show-pony, will never enjoy the full blessings of IT.

The Niche Low-Code Layer

Niche low-code platforms are a significant step up from strictly no-code tools, but they are also limiting. Niche low-code platforms are designed to satisfy a specific business need or built for a specific industry. They are built using components, functionality, framework, and architecture built specifically for that industry.

An example of a niche low-code app could be one developed specifically for business process management. It may be very good at building workflows and structuring data handling from the moment a customer calls into a support line to dispatching a technician onsite. But, how do you apply that to logistics or healthcare?

It can be done, but the result will be an extremely customized and inefficient system that gets worse with every update. Far better, would be to start from scratch and build something specifically for your business and needs.

Rapid Application Development

Rapid application development (RAD) is less a technology as it is a methodology. RAD focuses more on iteration and user feedback and less on following a strict plan. RAD and low-code are tightly intertwined and some low-code vendors have included RAD descriptors in their own messaging. However, pure RAD low-code platforms require other platforms for turning the prototypes and front-ends into mission-critical applications. So, as a technology, it’s incomplete.

Application Platform as a Service

Here’s a mouthful: Application Platform as a Service (aPaaS) is a category of cloud computing services that provides a platform for creating, running, and managing applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app.

Put more simply, an aPaaS low-code platform minimizes the on-premises instance of the platform by hosting it in the cloud. Instead of purchasing a behemoth piece of commercial software installing it on your network and being responsible for all of its upkeep and maintenance, all you have to worry about is making sure you have network access when you’re ready to use it.

An aPaaS low-code platform can be a very simple offering that lets you perform fairly limited development, up to a fully-functional enterprise low-code platform, which we will discuss next.

Enterprise Low-Code Application Platforms

The holy grail of low-code, enterprise low-code platforms have few limitations. If the pinnacle of application development is to tell an AI “assistant” what you want and the assistant creates it, enterprise low-code is about as close as we can get to that today without completely turning our jobs over to the bots. Enterprise low-code is the best of the best.

It offers:

1. The flexibility and customization of handwritten code.

2. A suite of pre-built templates, connectors, and APIs that you can customize with your branding and the je ne sais quoi that makes your offering unique.

3. The simplicity and speed of a drag and drop interface can predict your next action and present you with the requisite options for completing it.

As the low-code market matured in the last decade, most analyst firms have realized the differences in the types of low-code tools and platforms offered by vendors. They are only now beginning to re-evaluate their criteria for each and putting vendors in their respective places, causing some “x”-code vendors no small amount of anxiety.

From this point on, assume that any capabilities discussed in this eBook fall under the heading of “enterprise low-code application platform” since that is the most capable platform for most enterprise needs.

What Can You Create with Low-Code?

A more appropriate question might be, “What can’t you create with low-code?” If you ask professional developers what they think about low-code, the ones who aren’t using it will likely tell you, “It’s too limiting.” Unless your job is that of a developer creating first-person shooter games, there’s very little low-code can’t do. Anyone who claims otherwise is expressing what Forrester’s Rob Koplowitz calls, “Developer Machismo.”

Developer machismo typically stems from a one-part misunderstanding of what low-code is and what it can do, very possibly due to past exposure to an inferior x-code tool. Another reason developers may not embrace low-code is over a fear that low-code will mean the end of what has been a fairly profitable career pursuit, which is understandable.

Though understandable, this fear is unnecessary. There will always be a role for developers. Developers build low-code platforms, and who knows better how to help developers do more than themselves? Certainly not artificial intelligence (AI). Artificial intelligence is nowhere near capable of accepting vague inputs and porting out anything usefully creative. So, while tomorrow’s developers may not be doing the exact kinds of things they do now, and in the same proportions, they will be needed.

The best we can offer now in terms of development automation is fully reflected in today’s modern enterprise-class low-code platforms. We can automate logic and workflows and we can tell the software to do “that” if “this” happens. That’s easy. But we still need humans for creativity.

Low-Code for Reusable Components

It’s estimated that anywhere from 60-90% of most applications use the same basic code and functionality. Meaning, we can automate the majority of most application development. But we still need developers to tell the computers what our intent is and to build that unique 10-40% that makes our applications different.

If so much code is the same across much of the app dev world, what does even the minimum amount of code commonality (60%) get us? Sixty percent means we can pre-build and automate things like:

- **Design templates**: Much the way online blogging platforms made it easy for anyone to apply and customize hundreds of beautiful design templates, we can do the same with applications. We can build templates that are specific to industries like banking or healthcare that offer common functionality such as appointment setting and money transfers. These templates can be built to any form factor and according to the most current industry best practices and guidelines.

- **Automated Operations and Logic:** We know from experience, through lengthy customer interviews and years of close relationships, that business applications and core systems each have similar operational patterns. ERP systems, for example, tend to follow a similar process including order entry, customer verification, warehouse/supplier identification, processing, payments, logistics. This is oversimplified of course, but it illustrates a point that we can generate this code and logic so that it’s pre-built, pretested, and ready for use.

- **Modernize Legacy Systems:** It’s not enough to bolt on a new mobile app and hope your brownfield architecture can breathe new life into old, legacy systems. True digital modernization requires a more “from the ground up” approach. Low-code guides developers and provides assistance in designing new core systems.

OutSystems 4-Layer Canvas architecture, for example, simplifies the creation of service-oriented-architecture (SOA). Simply put, it promotes the correct abstraction of reusable (micro)services and the correct isolation of distinct functional modules. This is important because in an average enterprise architecture implementation, it’s common to support 20+ business applications and include more than 200 interdependent modules. Being able to build that, using reusable elements that still have independent lifecycles, ensures your new core systems are scalable and agile for years to come.

Low-Code for Mobile

Sometime near the middle of 2018, coincidentally right around the time the “app” celebrated its 10th birthday, we started spending more time on mobile apps than we did watching television. And while most of us spend the majority of our time on nine mobile apps in a given day, we routinely use 30 each month on average.[5]

It’s not a stretch to say that mobile apps are key to the success of most organizations’ digital strategy. Mobile application design is a distinct skill set in modern IT, and traditional mobile app developers are fairly unique in that they require a combination of technical skills as well as soft (people) skills. In addition to knowing how to actually code a mobile app so that it works across form factors, they also need a keen understanding into how people use their phones. They need to understand UX/UI design, and it certainly doesn’t hurt if they are highly creative. Other than technical trainers, there are few career choices requiring such a combination of right-brain/ and left-brain talent.

If you don’t have such talent on staff or if you’ve tried to find it and for whatever reason that particular rectangle is still blank on your org chart, there’s help yet. Low-code is exceptional at mobile application design and delivery.

In fact, mobile apps are the second most common type of application built with low-code (41% according to the 2019 OutSystems State of Application Development report). Most enterprise-class low-code platforms can generate code that’s compatible for each form factor you desire (e.g., iOS, Android) without having to use a Mac or work in Android Studio specifically.

Other functionality, such as the ability to customize your app for offline use and the ability to integrate user feedback mechanisms into your apps helps ensure that you’re giving your users the best experience possible. And that is good for business.

Low-Code for Web Applications

The death of the desktop has been greatly exaggerated. We mentioned previously that mobile apps were the most commonly developed software on low-code platforms in 2019, but we didn’t mention what got the numbers one and two spots— that would be portals and web apps.

[5] 60+ Fascinating Smartphone Apps Usage Statistics For 2019, Social Media Today, https://www.socialmediatoday.com/news/60-fascinating-smartphone-apps-usage-statistics-for-2019-infographic/550990/

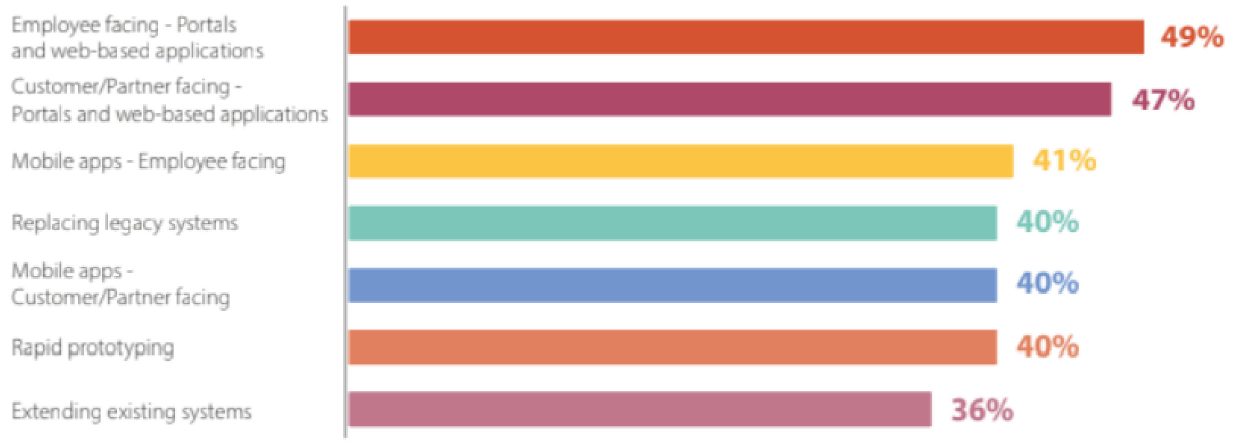


Fig. 37: **Projects Using Low-Code**

Source: State of Application Development 2019/2020. OutSystems ©

But web apps are increasingly becoming more like mobile apps, thanks, in no small part, to technology that helps them look and feel like the mobile apps we use every day. If you're fairly new to this topic, here's a quick primer on these technologies, and this [blog post on Medium] offers some great insight into how OutSystems uses all three.

Responsive web design

Responsive web design has been around for at least a decade and it is considered bare minimum from a UX standpoint. What responsive web design addresses is how a web site/page adapts to fit the user's screen. Traditional desktop-based web browser pages don't render well on smaller form factors without modification to the code. Responsive web design uses flexible layouts, along with CSS, to automatically adjust the display as needed.

Progressive web applications

Progressive web applications are web pages that have properties similar to mobile apps. They incorporate responsive design so that they look like a mobile app. But then there is an entire backend suite of capabilities that make them "feel" like mobile apps as well, such as:

- You can install them rather than only visiting them via URL.
- You can link to them so you can share them or even just find them later.
- They can send push notifications to mobile devices.
- They work almost as equally as well offline as they do online.

Reactive Web Applications

This one is tricky because we typically think of things that are reactive as being a little late to the game. But quite the opposite is true here. Websites built with this technology functionally maximize response times thanks to a combination of server-side logic and client-side rendering. As you interact with the web app, it’s ready for you. It doesn’t need to wait for you to click on a menu item, then send the query to the server-side, then wait for the response and present it to you. Most of the information is already there on the client-side waiting for you to ask for it.

If this all sounds complicated and requires someone who specializes in building sophisticated web applications, you’re right. But if we convince you of nothing else about low-code, we hope we convince you that low-code simplifies much of today’s complex modern application design requirements by including these capabilities natively. As you use the platform, you’re automatically incorporating best-in-class technologies.

Low-Code for Core Systems

It is a well-circulated myth that low-code isn’t suitable for large-enterprise legacy modernization endeavors. And for some low-code platforms, that is certainly understandable. Basic no-code and niche low-code tools lack the advanced capabilities, and the breadth of functionality needed to build and manage core systems.

But, enterprise-class platforms, such as those described by Gartner in its Magic Quadrant^{TM} for Enterprise Low-Code Application Platforms and Forrester’s Wave^{TM}: Low-Code Development Platforms For AD&D Professionals reports, are capable of building these kinds of systems, and more. Consider these critical capabilities for building and managing critical core systems.

CORE SYSTEMS CRITICAL CAPABILITIES

Critical Capability	Benefit
Speed of Delivery	Low-code significantly speeds up application development, even of core systems. OutSystems research shows that 67% of organizations either have used, or are using, low-code for their own legacy transformation projects.
Scalability	Though difficult to quantify, scalability is achieved by either increasing demand capacity on the delivery side or by increasing capacity on the production side. Enterprise-class low-code platforms can do both, on demand, without sacrificing administration transparency or availability.
Performance	Enterprises demand high-performance without sacrificing anything, and low-code can deliver. From code reviews during design, to dependency checking prior to deployment, and ongoing monitoring and threat detection, enterprise low-code can help ensure digital business continuity.
Future-proof architecture	Rebuilding your core systems every ten years or so is not a digital strategy. The only recipe for long-term success is to "build once and update continuously." But it takes the right core, one designed to evolve with technology and whose applications rely on standard applications and frameworks so that, no matter what happens with your business, your code always belongs to you and not some proprietary vendor.
Deployment of your choice	There are some very good reasons to deploy your systems and applications in the cloud. And there are good reasons not to. The decision should be yours alone, and not a requirement of your low-code vendor. Enterprise-class low-code platforms today offer the flexibility of on-prem, single cloud, or multi-cloud deployments, giving you the scalability and flexibility you need; even if your needs change tomorrow.

Today's modern operating systems, the ones that let you simply click on something to perform complex operations, comprise tens of millions of lines of code. Today's modern low-platforms are equally as complex. Both are designed to greatly simplify otherwise difficult tasks. As such, the answer to the question of “What can you build with low-code” is “Just about whatever you need, no matter how big or small.”

The Benefits of Low-Code

Now knowing what low-code is how it can be used and by whom, let’s talk about some of the real benefits of low-code; minus the marketing-speak that usually comes with a discussion from a vendor about a technology. Let’s start with people; because, any discussion about the benefits of technology always comes down to people.

Low-Code Benefits People

Global Knowledge, the largest dedicated IT training company, last year released its top 10 most in-demand skills for IT departments. Application development came in at number three. So, while application development continues to be a strong career pursuit in terms of growth, salary, and longevity, it is but one part of the full IT toolkit from a skills standpoint.

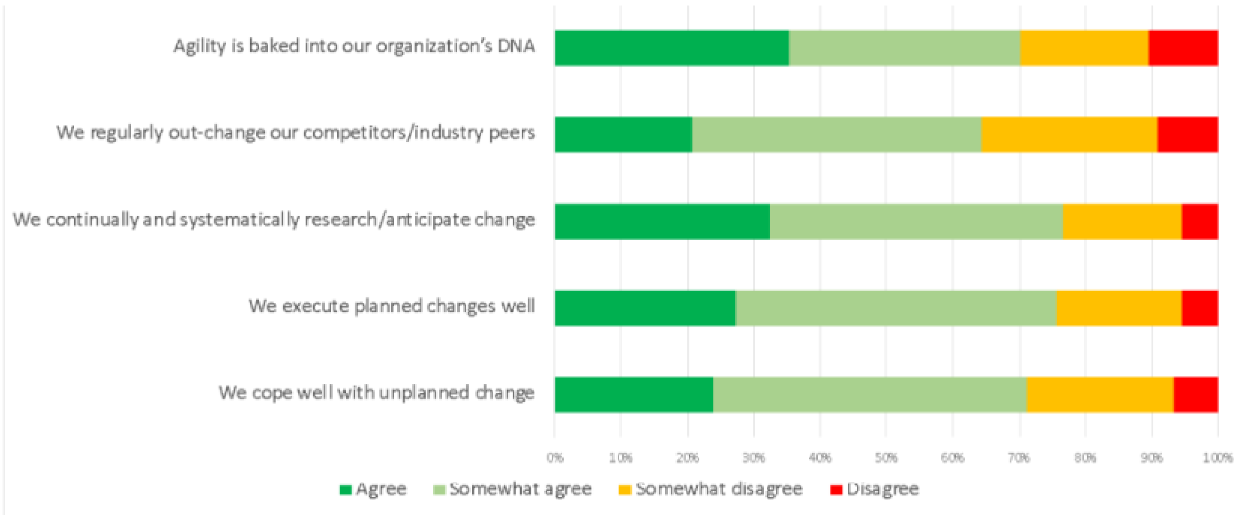
Low-code lets workers who specialize in other things add “app dev” to their portfolio of skills. It allows someone who understands cloud computing (#1 in the list) or AI and machine learning (#8 on the list) to apply their specific expertise to build and create new and amazing things. Before low-code or without low-code, an IT specialist in any of these other areas would have had to rely on someone else to bring these amazing things to life, allowing specialists to add #3 to their list of skills.

Conversely, it helps traditional application developers expand their pool of contributors, bringing in someone with a deep background in project management (#7 on the list), to help workflow a new business app that could save the organization hundreds of thousands of dollars. That looks pretty good on a resume.

The bottom line is that developers should no more fear low-code than they fear a new programming language. Rather than see it as a threat to their livelihood, they should embrace it as part of their growth as a professional. And technically-minded workers may just find that low-code scratches a curiosity itch and helps them bring to life something that they may have felt was unobtainable before.

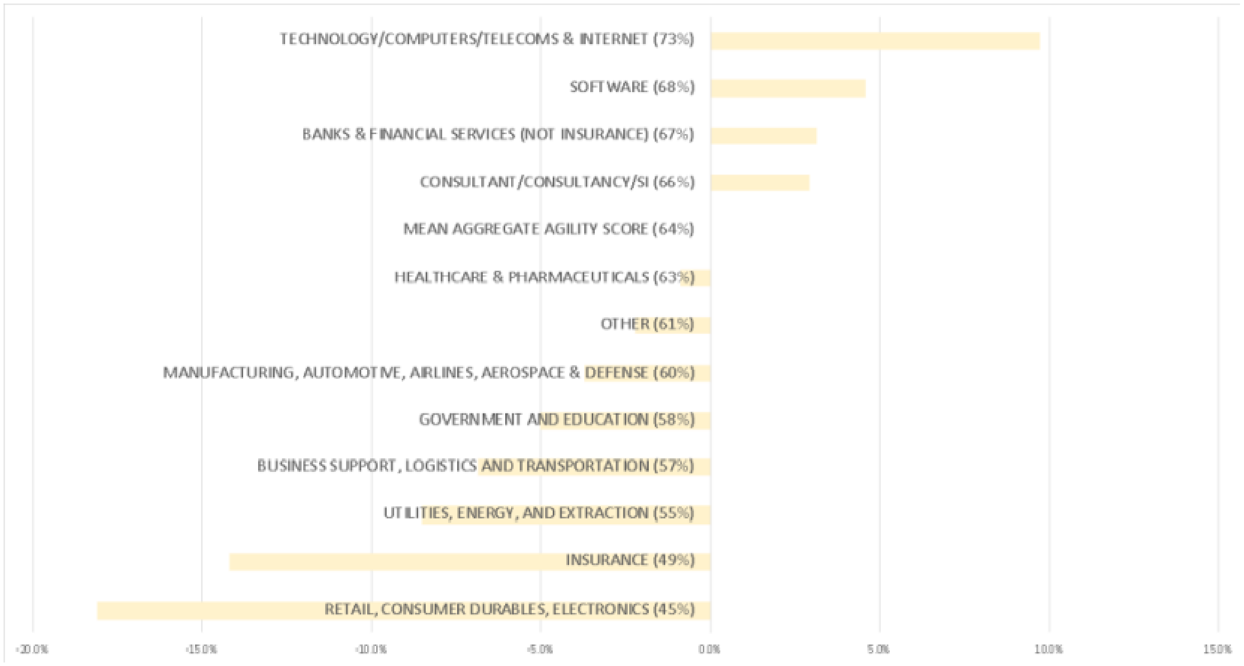
Low-code Helps Make Business Agile

Business is cyclical. What is popular today can easily fall out of favor tomorrow, only to resurrect itself again to begin anew. So it is with agility. What was designed to foster collaboration and efficiency in application development was quickly adopted and promoted as a solution to all things silo'd in the business. And like anything that strays too far from its roots, it lost its focus. But low-code is helping agility make a comeback. Agile (capital "A") development is an approach that focuses on evolutionary development, early delivery, and continuous improvement. In the OutSystems 2020 State of Application Development report, we asked several questions gauged to measure organizations perceived level of agility. We used the highly respected Prosci's "Agility Attributes Assessment" as our basis for questions and scoring.



Results: For the majority of companies, there is a sense of confidence in overall organizational agility, but it appears to be a cautious one.

Next, we measured how various industries self-scored in various Agile measurements.



Result: Those industries with roots back to the beginnings of the Agile movement appear best poised for success. A sidenote concerning these results is that they were measured at the height of the COVID-19 crisis, and in general, these results closely mirror data reflecting the impacts of the crisis on various industries. Those with a higher agility score here were less likely to see significant disruption from COVID-19.

Hoping to get a better understanding of the differences in agility preparedness among organizations, we looked at the agility scores of organizations already using low-code versus those not using low-code.

From the OutSystems State of Application Development Survey - 2020

On multiple measures, the survey results attest to superior speed and agility enjoyed by users of low-code compared to those not using low-code:

- A 19% higher self-assessment score for digital transformation maturity
- A 5% higher score for organizational agility
- An agile maturity advantage of 22%
- A 15% innovation advantage—devoting more app dev time and resources to innovation
- An app delivery speed advantage—12% more deliver web and mobile apps in less than four months
- Shorter release cycles—15% more say they release new software versions biweekly or more often
- Twenty-two percent more say their business is satisfied or somewhat satisfied with the release cadence of new software
- 11% more likely to have reduced their app dev backlog in the past year.

Low-code Future-Proofs Your Apps and Architecture

Admittedly, this section title sounds ridiculous without context. Honestly, if we've learned nothing from the tech industry (dot.com bust, Y2K, Blackberry) nothing is future-proof, right?

Low-code very well could be. Imagine a platform built using the most advanced languages and tools available at the time. But one built knowing it will need to constantly evolve in such a way that not only are the applications and systems built on it capable of being updated as needed, but the platform itself can be.

Three basic criteria for future-proofing your apps and architecture with low-code:

1. **You own your code, not your low-code vendor**. Sounds like a no-brainer, doesn't it? But it's less common than you might think. If your low-code vendor goes out of business or is acquired, and the technology is killed or rolled up into something else proprietary, if

you don’t own your code, you are effectively dead in the water. Or, barring that, you are subject to your vendor’s pricing whims until you can rebuild on your own.

2. **Your architecture is scalable--horizontally and vertically**. What does this mean? With a horizontally scalable architecture, when your business needs more capacity to meet user demand, you can add and manage as many additional (or fewer) servers as needed to meet capacity. With vertical scalability, no matter how complex your IT needs are, your architecture can handle it by adding or reducing processing capacity.

3. **A consistent commitment to continuous improvement.** This is as much on your low-code vendor as it is on you. If your low-code vendor is constantly updating its low-code platform with functionality and underlying technologies, like reactive web apps and support for exciting new technologies like AI and IoT, then you always have access to the latest and greatest for your business.

Death and taxes, right? Maybe there’s another certainty we can add to life, after all, thanks to low-code.

Low-Code Helps Improve Your Security Posture

Of the following two scenarios, which sounds more secure:

Scenario 1: IT team “A” has 32 workers, each with different backgrounds, skills, and varying levels of security awareness. Some use tools provided only by IT, while others prefer to use some tools they have been using throughout their career, and which IT is not aware of.

Scenario 2: IT team “B” has 32 workers, same makeup. The majority of them use a single platform to design, build, test, and deploy applications. This platform employs several security protections, including identity and access management (IAM), database encryption, auditing, and log monitoring, and governance for all users. The platform also allows seasoned developers to integrate many of their beloved tools, for added governance and security. If security patches are needed due to a newly disclosed vulnerability, the platform can simplify the process of patch delivery by managing the process and giving administrators full control over rollouts and rollbacks as needed.

Scenario 2 sounds more secure, right? That’s because it is, obviously. Any time you can reduce the number of disparately maintained internet-facing tools, which includes just about every piece of software in use today, it makes sense to do so.

There is also a governance consideration here. As developer teams grow, and in organizations that offer a citizen developer program, managing permissions and access gets more complex.

But low-code platforms can offer relief with complex usage policies and team models that help segment which teams can access what and what tools they can use.

Wrapping Up

We hope this guide has provided you with enough information to whet your appetite to want to learn more about low-code and its benefits to organizations of all sizes. OutSystems' low-code platform is capable of everything described in this guide, and more. We barely scratched the surface. In addition to the platform itself, OutSystems has a full training program that can walk anyone through the simple use of the platform, starting with building an entire app in just a few minutes, all the way up to full certification on the platform.

In addition, OutSystems has an active community of users and moderators in the “Forge” where you can ask questions, get ideas for your apps and deployments, and even submit your own, personally-developed components and APIs so that everyone in the Forge benefits.

We invite you to sign up for your own free “Personal Environment” and start working with the platform on your own. Of course, we’re here to help when you need it