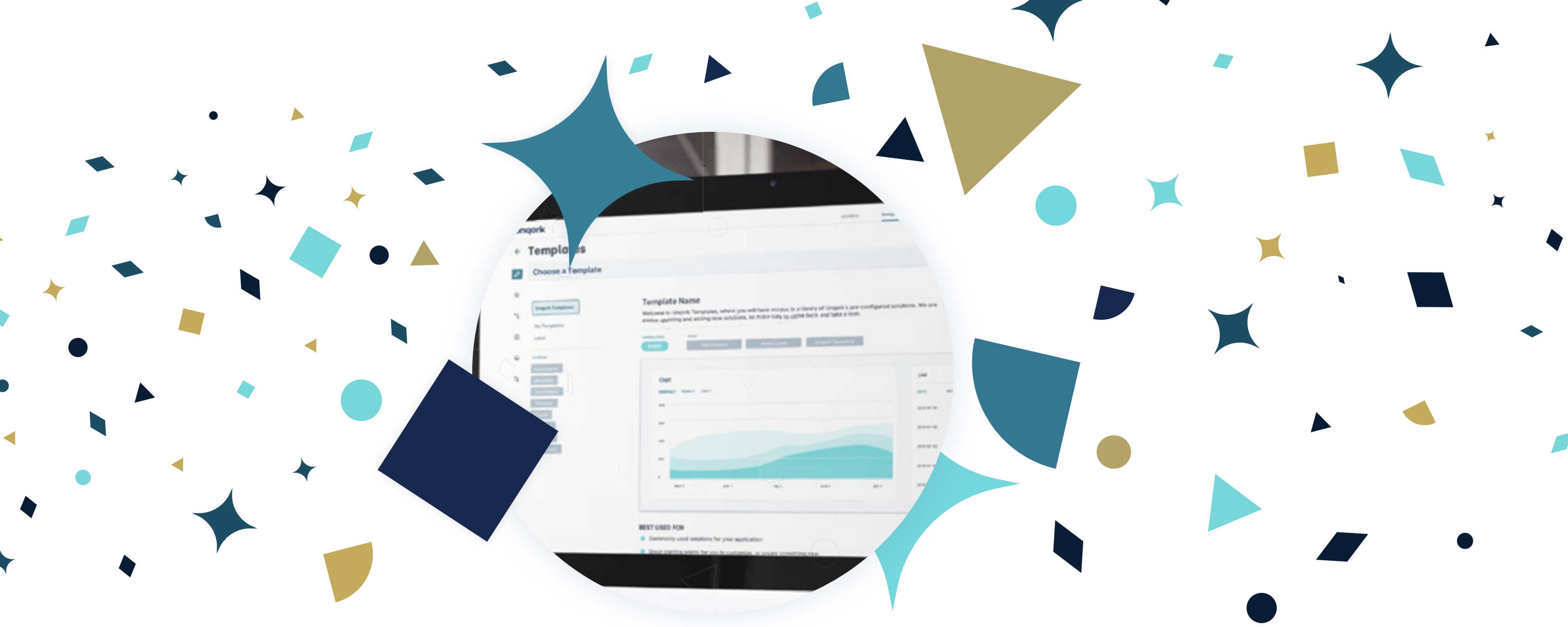
unqork

The Field Guide to No-Code Application Platforms

How your enterprise can significantly speed up time to market, improve quality, and lower the costs of your custom software projects.

unqork

Contents



This year, enterprises are expected to spend **$550 billion** building custom software.

Custom solutions can address the specific needs of a business and—ideally—secure an advantage over any competition who deign to rely on non-configurable, packaged software. Developing custom enterprise software, however, is not for the faint of heart.

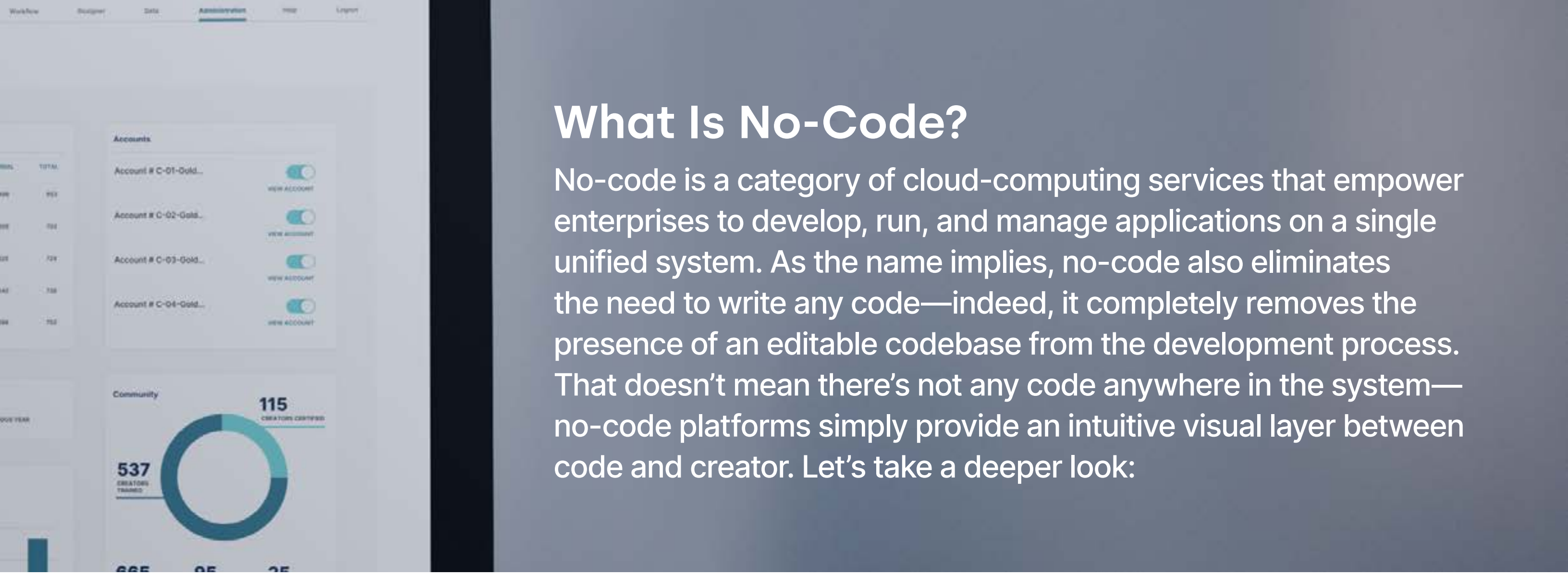
Each new solution has to play nice with legacy systems and processes, balance the (occasionally competing) interests of multiple internal teams, and be continually maintained over its lifespan. To complicate matters further, companies are entirely reliant on a small pool of experienced engineers with the coding expertise to build robust enterprise-scale applications.

At least, that's the way it used to be.

No-code platforms dramatically improve how enterprises build, deploy, and manage custom software. By simplifying and streamlining development, no-code shortens the time to market, improves quality, opens the process up, and lowers the costs of both initial builds and ongoing maintenance.

Many solutions on the market make similar claims, of course. Indeed, the application platform market (also called Application Platform as a Service, or "aPaaS") can admittedly be pretty confusing. The sheer number of vendors, claims, and lingo can be overwhelming. But there are very important differences between enterprise-scale no-code and those other solutions.

We hope this guide will be a helpful resource for any enterprise looking to inject new efficiencies into their development function.



What Is No-Code?

No-code is a category of cloud-computing services that empower enterprises to develop, run, and manage applications on a single unified system. As the name implies, no-code also eliminates the need to write any code—indeed, it completely removes the presence of an editable codebase from the development process. That doesn't mean there's not any code anywhere in the system—no-code platforms simply provide an intuitive visual layer between code and creator. Let's take a deeper look:

HOW DOES IT WORK?

When you are building an application with code, what you're doing is reproducing a set of commands over and over again. The commands happen in different ways in different parts of your program, but they are the same commands. What a no-code platform does is repackage these commands in a graphical form, allowing you to configure and manipulate them visually. The platform then executes those commands as if they were written in code.

By stringing together such commands, you can build your program without having to see any of the code or write any of it yourself.

The application is configured visually from start to finish, and it runs entirely from the platform after it's deployed. Changes are made by simply logging in and reconfiguring the visual interface.

WHY IS IT HELPFUL?

Removing the need to write, edit, and debug lines of code speeds up the time to market, improves quality, and lowers the costs of initial builds and ongoing software maintenance.

WHO SHOULD USE IT?

Let's look at this one in a little more detail, and break it down by:

- **Company size:** No-code platforms can be used by companies of all sizes, but the companies that will benefit the most are the ones spending an outsized portion of their IT budgets on complex custom applications. The larger the company, the more it stands to gain from streamlining its application development process.
- **Industry:** Companies across all industries are using no-code platforms. Early adopters were primarily in industries like Financial Services and Insurance. But increasingly, organizations across all categories that spend a significant portion of their budgets on custom software projects are adopting the no-code approach.
- **Roles:** No-code platforms can be used by anyone that understands basic logic and conditional statements. This means everyone from classically trained engineers to business analysts, and anyone in between.

A Brief History of Software Development Productivity

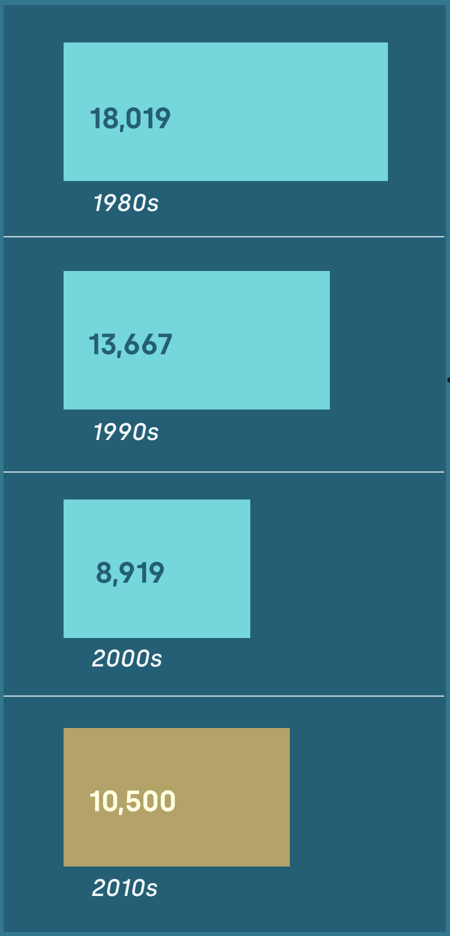
For decades, the standard in software development has been based on the idea of building faster.

This isn't a new challenge. In fact, many years ago, the problem was even worse, as many of the stopgap solutions we rely on today didn't even exist yet! Let's take a quick look at how building applications has evolved over the last several decades:

- **The 1980s** were a difficult period for software development. COBOL was the dominant language, but it was really difficult to use. Many projects failed to get off the ground or, worse, were released but malfunctioned. That's why many refer to this period as "The Software Crisis."
- **Then the 1990s** came along and things got a bit better. COBOL continued to dominate, but methodology innovations like Rapid Application Development (RAD) and higher-level (and more user-friendly) programming languages like Java started to gain traction in the enterprise and building software got more efficient and easier. Applications became more useful and the time spent creating them decreased.
- **Next came the 2000s**, which saw Java surpass COBOL as the dominant language. Innovations like frameworks (e.g., Spring) and Integrated Development Environments (IDEs) along with low-code platforms like Appian, Mendix, and Outsystems all helped developers become more productive.
- **Then the 2010s** came along, higher-level languages like Python started to gain adoption, and low-code platforms and frameworks became more advanced. Methodologies like Agile started to permeate enterprise development projects. And despite these advancements, the average time to complete a typical software project was **10,500 hours**, a 20% loss in productivity.

So, why this recent downtick in productivity? There's no doubt software got more complex in the 2010s, but it got more complex in previous periods as well. What's different is that the most recent increases in complexity have not—yet—been matched with a new set of development technologies that are sufficiently able to address these challenges, and as a result, budgets are exploding, backlogs are growing, and projects failing to meet requirements and timelines.

HOURS REQUIRED FOR TYPICAL ENTERPRISE APPLICATION



Source: QSM Software Development Database, 2019

What Can You Do With a No-Code Platform?

No-code platforms can power modern application development practices that incorporate a wide variety of systems. Even beyond the efficiency benefits that come with having a single consolidated system, no-code platforms' visual UI gives more people the ability to build powerful applications. The result is less time to market for initial builds, better collaboration across organizations, and lower cost overall.

BUILD GREENFIELD SYSTEMS

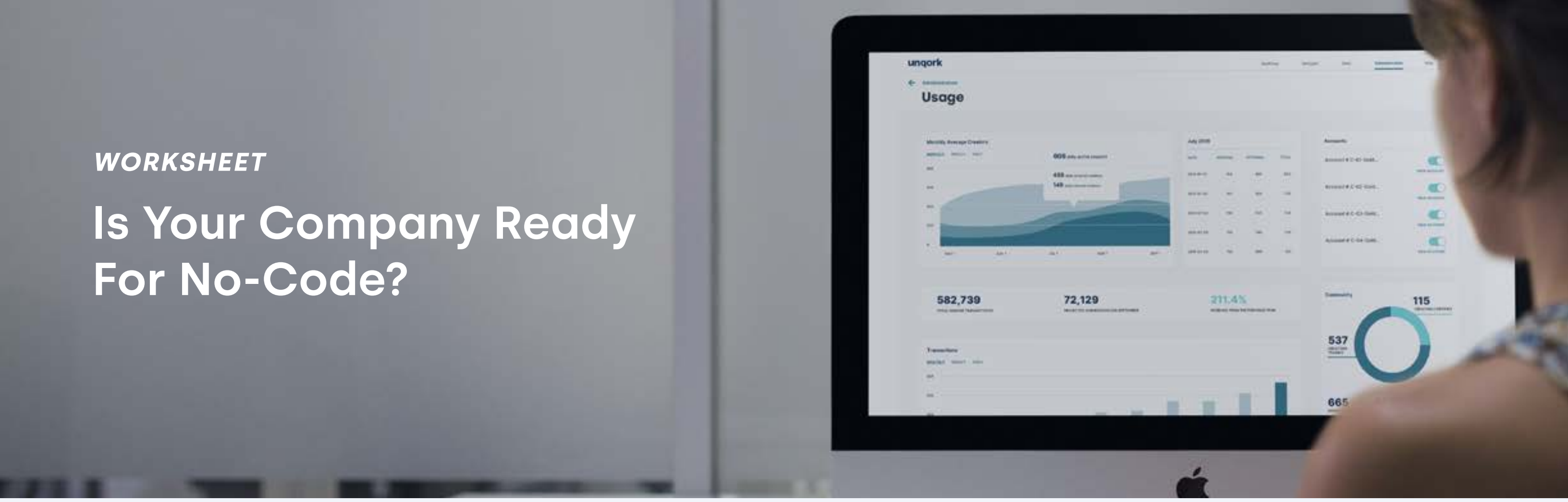
Today, a typical enterprise application takes months, sometimes even years, to build. No-code platforms dramatically accelerate this process and allow organizations to build applications in just weeks—or, in some cases, days.

INTEGRATE SYSTEMS

Pulling or pushing data to external systems is a natural prerequisite for any modern enterprise application. Best-in-class no-code platforms streamline this process by not only supporting modern APIs, but also by supporting legacy systems and paper-based processes.

MANAGE AND MONITOR

Building an application and integrating it with external systems is only the first step in the long lifecycle of an enterprise application. The full SDLC life cycle, as well as the process of updating, re-deploying, managing, and monitoring applications, can also be supported within a no-code platform.



WORKSHEET

Is Your Company Ready For No-Code?

Use this worksheet to determine if your enterprise is ready for a no-code platform. Answer each question on a scale from 1 (strongly disagree) to 5 (strongly agree).

	1	2	3	4	5
1. **Growth objectives:** Our company's top priorities include producing differentiated customer experiences and getting products to market faster.	▰	◆	◗	●	✦
2. **Margin improvement:** Our company is striving to reduce our technology costs and footprint, improve operational efficiencies, and reduce our dependence on paper-based processes.	▰	◆	◗	●	✦
3. **Existing project backlog:** Our company has a long queue of IT projects waiting to be started. Many of these backlogged projects are very important to our business.	▰	◆	◗	●	✦
4. **Project delays:** Many of our recent IT projects missed their deadlines, failed to meet the necessary requirements, or had to be significantly rescoped.	▰	◆	◗	●	✦
5. **Requirements:** When our company does launch applications, we're forced to significantly scale back business requirements to deliver them on time.	▰	◆	◗	●	✦
6. **Time to market:** Our company is working on getting new ideas to market faster, including rounds of testing and iterating after launch.	▰	◆	◗	●	✦
7. **IT recruitment and turnover:** Engineers at my company seem to be spending a lot of their time on fairly mundane tasks.	▰	◆	◗	●	✦

SCORE 25 POINTS OR MORE?

it's probably worth looking into whether no-code can help your software development process.

Why NOW for No-Code?

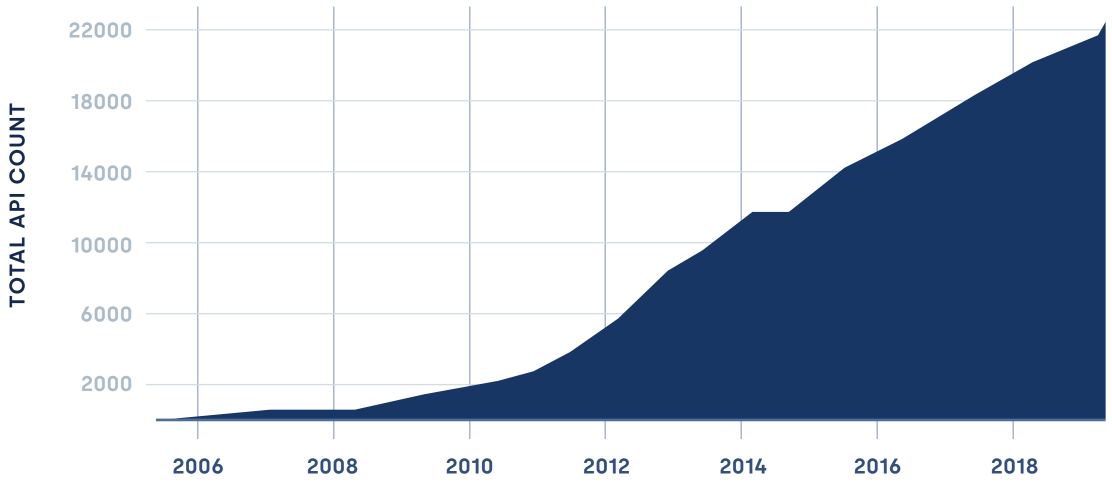
We'll be the first to admit that no-code as a concept isn't new. In some ways, it's been the holy grail of software development for a long time, but previous attempts have fallen short of the true promise of enterprise-grade visual development. However, in the last decade, a few factors have converged to not only make no-code a possibility, but a competitive necessity. Let's take a look of some of these converging trends:

TREND ONE
The explosion of microservices

A microservice is an accessible, self-contained piece of business functionality with clear interfaces. Stated another way: Microservices give developers the ability to easily extend the capabilities of their applications.

Over the past 15 years, an increasing number of enterprise developers have made their services easier to use by creating APIs and endpoints for their internal services. In doing so, developers have created a layer that made other developers' lives easier, but most of these microservices still require code to attach to existing services, which can add time and complications. No-code allows organizations to quickly and efficiently stitch numerous microservices together and build intelligent workflows to orchestrate inputs and outputs—all without code as a bottleneck.

Growth in Web APIs since 2005



Source: ProgrammableWeb

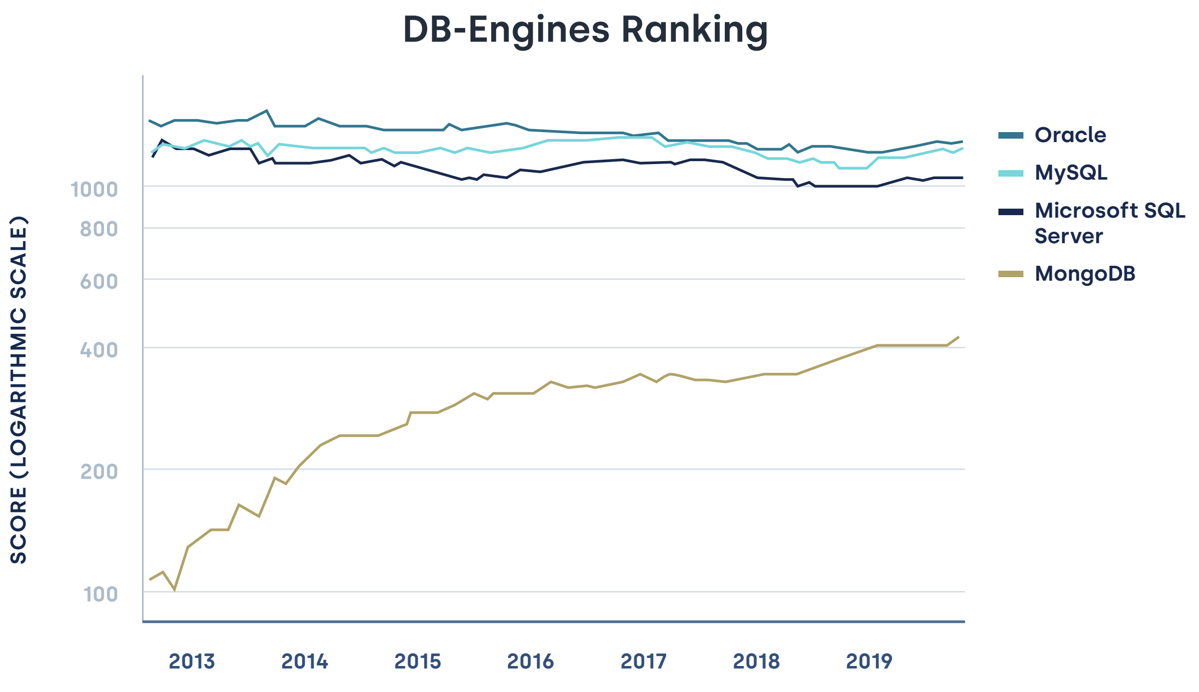
TREND TWO

Document-based databases

The second factor has been the advent of document-based databases, sometimes also called "NoSQL" databases. Instead of storing data in rows and columns, document-based databases store data in documents. These documents typically use a structure similar to JSON (JavaScript Object Notation), a format popular among developers.

These databases have enabled unparalleled flexibility—from accepting the flat files prevalent in legacy systems to modern JSON—while also maintaining critical features popular among tabular and relational databases.

No-code platforms make it significantly easier for developers to integrate and move data between these databases. This flexibility can be particularly helpful for modeling unstructured and/or constantly changing data. It also makes changing an application during its lifecycle easier—for example, if an enterprise needs to add new fields to an existing product.



Source: DB-Engines.com, 2019

TREND THREE

Enterprise cloud adoption

The advent of cloud services has been a game-changer for the enterprise. Platforms hosted in the cloud are secure, scalable, upgradeable, and future-proof, so it should come as no surprise that almost 75% of businesses are at least partially deployed in the cloud according to IDG. That same study found that almost 40% of technical decision-makers feel pressure to move entirely to the cloud.

The cloud has also unlocked key capabilities for no-code application platforms. Because the codebases of these platforms are completely abstracted from the user, software upgrades for these cloud-based platforms can happen completely behind the scenes. As a result, the components of a no-code application are always up-to-date with the latest versions and bug-fixes.

Worldwide Public Cloud Service Revenue Forecast (Billions of US Dollars)

	2018	2019	2020	2021	2022
Cloud Business Process Services (BPaaS)	41.7	43.7	46.9	50.2	53.8
Cloud Application Infrastructure Services (PaaS)	26.4	32.2	39.7	48.3	58.0
Cloud Application Services (SaaS)	85.7	99.5	116.0	133.0	151.0
Cloud Management and Security Services	10.5	12.0	13.8	15.7	17.6
Cloud System Infrastructure Services (IaaS)	32.4	40.3	50.0	61.3	74.1
Total Market	**196.7**	**227.8**	**266.4**	**308.5**	**354.6**

BPaaS: business process as a service; IaaS: infrastructure as a service; PaaS: platofrm as a service; SaaS: software as a service. Totals may not add up due to rounding.

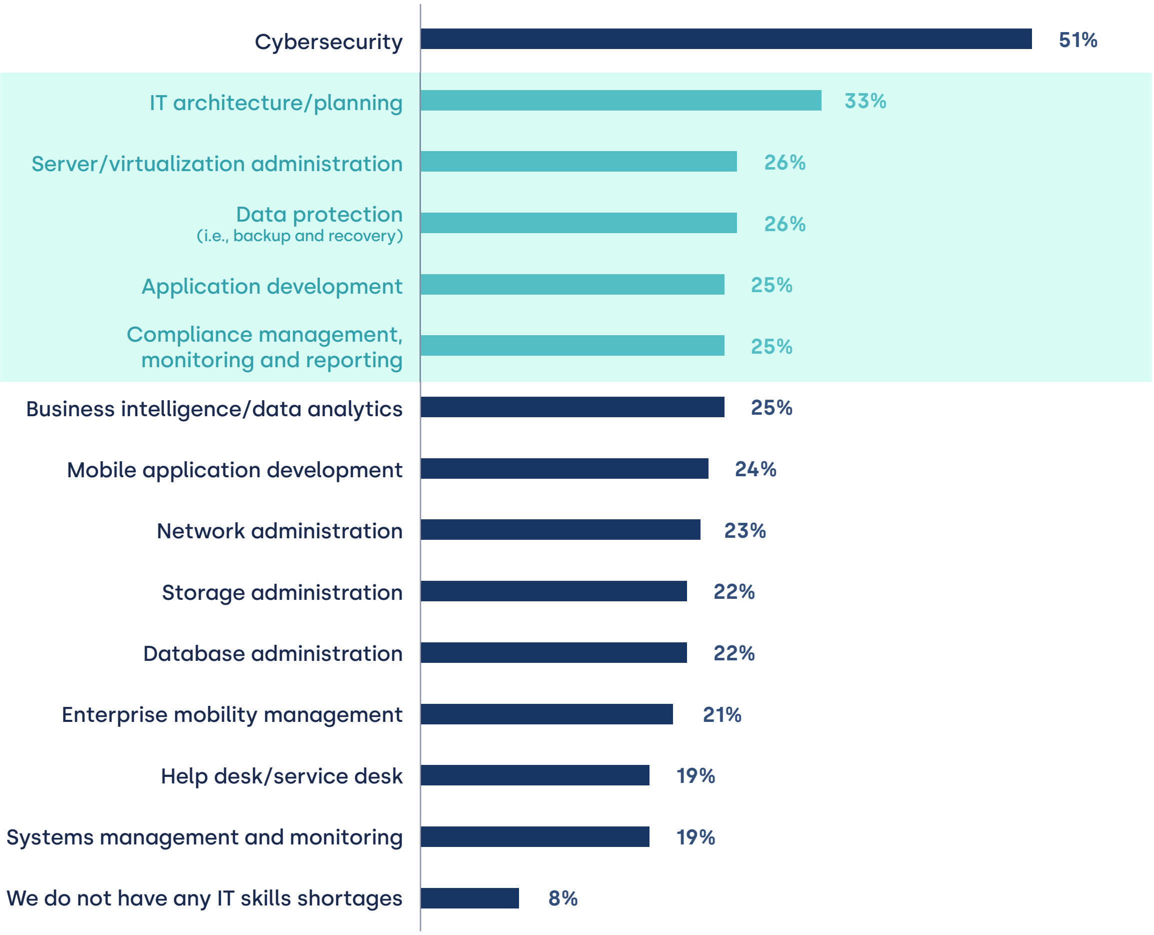
TREND FOUR

Skill shortages in key IT roles

The final trend worth highlighting is the increased difficulty of hiring engineers across almost every business. Today, experienced engineers who are capable of solving complex problems are in higher demand than ever, and the pool is limited. While this has always been the case, a recent LinkedIn study found that 70% of businesses felt the talent gap is getting wider. Ask any enterprise IT recruiter, even in the face of a global economic disruption brought out by COVID-19, there is still intense competition for experienced engineers.

In which of the following areas do you believe your IT organization currently has a problematic shortage of existing skills?

Percent of respondents, N=620, multiple responses accepted



How Is No-Code Different Than ________?

There are many ways to build enterprise software. The purpose of this section is to explore in detail why no-code is different (and better) than traditional code, as well as some of the other development approaches on the market.

NO-CODE VS. TRADITIONAL CODE

Traditional code refers to any type of development that directly uses a programming language to build software, e.g., Java, Javascript, C++, C#, Python, or PHP. Most modern developers use “frameworks” that complement these languages to accelerate development. For example, many applications built in Java also use a framework called Spring to more easily build certain functionality.

Traditional development requires engineers to be well-versed in the chosen development language and any relevant frameworks. After defining the initial requirements, the engineer will configure a development environment and begin translating requirements into application logic. This generally involves the engineer (or, most likely, group of engineers) making decisions about how to solve the application’s logic challenges using their knowledge of the language.

Traditional code comes with four major challenges:

1. **A reliance on specialized skill sets:** The first key difference between no-code and traditional development is that the latter requires developers with highly specialized skills. This makes finding the right talent very challenging and expensive. This specialization has resulted in skill shortages in almost every major IT role.
2. **Long initial development times:** The second key difference is the length of initial development times. While some framework advancements have helped to accelerate the process, it’s not uncommon to see a large enterprise project take multiple years to ship with traditional approaches. This is because, on some level, it’s necessary to reinvent the wheel with each new traditional development project.
3. **Legacy code:** Making changes in legacy code can be challenging—and expensive. The original engineers might not even be available to explain the system they’ve built. This requires new engineers to research and relearn the application’s logic before rewriting in the way that they think is best. These rewrites turn what used to be new code into legacy code, and the cycle continues…

4. **Upkeep:** Finally, even without any functional changes, most software needs to be upgraded and maintained over time. This is because fundamental aspects of both programming languages and frameworks are always evolving. If you've built software that uses these elements, you will likely need to rebuild parts of your application to keep up.

So as you can see, traditional development is fraught with downsides. Yet even today, this approach comprises the vast majority of software projects. That said, these downsides prompted the next evolution in software development: Low-code.

NO-CODE VS. LOW-CODE

Another common point of comparison for no-code platforms is low-code. Even though the term "low-code" was coined by Forrester in 2014, most of these tools have been around since the early 2000s and grew out of ideas like Integrated Development Environments (IDEs) like Microsoft Visual Studio and Eclipse or the CASE tools of the late 1990s.

While every low-code solution is slightly different, these tools represent common application elements in graphical form. Some may even offer some drag-and-drop functionality. The engineer configures the basics of their application, and the tool generates the foundational codebase. Once this foundational code is created, developers are still needed to append, modify, and finalize the codebase.

Even though low-code platforms require less hand-coding overall, IT skill sets are still always needed to create a functional enterprise application—particularly when any complex functionality is involved. This means low-code platforms share many of the disadvantages of traditional code.

1. **Faster, but not *fastest*:** All things being equal, low-code is faster than traditional coding. This is because low-code automates the production of foundational coding tasks, and brings the average time down from 9-12 months for a typical application to 3-6 months. (No-code, on the other hand, removes the need for any hand-coding and can typically complete an application of equal complexity in just 2-3 months.)
2. **Reliance on IT skills (still):** Low-code systems can help experienced coders do certain tasks more efficiently, but these systems are still designed specifically with IT workers in mind. (No-code is completely visual, meaning that with little training, new IT workers as well as non-IT workers can easily access the development process.)
3. **More errors:** Syntax can be very tricky and fragile. Every time a human types code (as you still must do in a low-code setting), there's a potential for mistakes. Any small mistake can cause a problem that needs to be debugged, which adds time and frustration. (No-code handles all the syntax—and does so with machine efficiency—so creators only need to focus on business logic and UX.)
4. **Legacy debt:** In low code, business logic is completely separated from technology upgrades. For example, if the low-code vendor upgrades its platform and no longer supports certain features you rely on, your enterprise would be exposed to significant upgrade costs. With no-code, users are only concerned with business logic, which will keep functioning even as the technology "underneath the hood" evolves.

In summary, while low-code platforms may enable faster builds than traditional code, these platforms still require expensive engineers, expensive changes, and significant legacy costs year after year, project after project.

	Low-Code Platforms	No-Code Platforms
Time to first build	**Faster** - typically a low-code application of equal complexity can be completed in 3-6 months relative to 9-12 for a typical enterprise application	**Much Faster** - typically a no-code application of equal complexity can be completed in 2-3 months
Ease of making material changes	**Difficult** - because code is involved an engineer must decipher and debug often idiosyncratic lines of code	**Easy** - all configuration takes place within the confines of business logic, only changes to business logic are required to change the application
Ease of hiring and training	**Difficult** - requires either consultants trained in specific language or seasoned developers that already understand code	**Easy** - anyone versed in business logic and decisioning can configure on a no-code platform
Total cost of ownership	**Slightly less** - basic elements of code maintenance and support still required	**None** - no legacy, no editable codebase to maintain or upgrade

WHAT ABOUT WEBSITE BUILDERS?

There have been many software products released in the last five-to-ten years that help non-engineers build websites without having to write any code. Products like Wix, Squarespace, and Webflow all make it possible, for example, to build seriously great websites without any previous knowledge of engineering, code, or software development. The main audiences for these platforms are small businesses and individuals—not the enterprise.

These tools are designed to help users build “brochure” type websites, which display static data, images, and basic forms. They’re great for getting a message across simply, but they aren’t able to incorporate advanced functionality and business logic for enterprise-ready applications. These may *technically* be no-code platforms, but in this ebook, we’re referring to advanced platforms designed for complex, scalable, enterprise-ready applications.

The Impact of No-Code

No-code platforms open up a world of benefits for the enterprise. In this section, we'll explore the benefits of no-code to large enterprises, including some that can begin making an impact right away.

FASTER TIME TO MARKET

Today, building or customizing enterprise software with traditional methods takes a tremendous effort. Most projects require integrating several different tools, and once development actually begins, the custom coding required for integrations and legacy data migrations alone often dominate the effort.

No-code fundamentally changes this equation. With no-code, you can build rich functionality much faster than you can with conventional methods. How? There are two major drivers.

The first is the fact that no-code platforms are fully built environments where the best technologies have already been selected, which means little time needs to be invested in analyzing a project before you even get started. Each tool on the platform works together during the build, deployment, and management processes, removing complexity and effort and allowing you to move fast.

The second is the complete removal of the need to work in code. A no-code platform allows you to focus entirely on the business logic of your application without having to worry about syntax or processes associated with traditional programming languages. This reduces the friction between the definition and implementation of requirements, allowing you to move at a speed of business.

IMPROVED COLLABORATION

In most large enterprises, there's a significant gap between IT and the business on understanding the objectives and functionality of a technology build. This often results in functionality being lost in translation because of poorly defined (or poorly understood) requirements, leading to disappointing ROI.

No-code bridges the gap via an intuitive visual development process that empowers business teams to directly participate in the building process. This improved collaboration results in applications that all teams can be assured will address the fundamental needs of the business.

EASIER MAINTENANCE

It's not uncommon for the average large enterprise to spend 50-75% of its total IT budget just maintaining existing systems.

Anytime you want to make a modification, integrate a new system, or upgrade an underlying technology, you're typically faced with a sizable undertaking. There could be several reasons for this in a given organization. First, code is fragile and nuanced. Five different engineers may solve the same problem in five different ways. And the sixth engineer brought into the project months later may have a sixth solution to the problem that's incredibly brilliant and efficient, but requires a complete overhaul.

Another reason is the cascading effect of tools used to build software. If, for example, a change is made to the front-end of an application (e.g., adding a field to a form), a change must also be made to the database. Similarly, that change then has to be propagated to any other form that may use the same field. With a no-code system, cascading tasks are automated, which makes modifying software much easier than with code. To put it simply: no-code gives organizations the freedom to make changes based on business needs, not budget cycles.

BETTER UTILIZATION OF IT RESOURCES

Today, almost every technical project requires multiple skilled developers. As a result, this limited group of people needs to be involved with every one of your enterprise applications. This creates bottlenecks within your organization. By moving applications to a no-code platform, you can free up highly-skilled engineers to work only on projects that require specialized engineering solutions to drive business value; not to keep dozens of patched-together systems up and running.

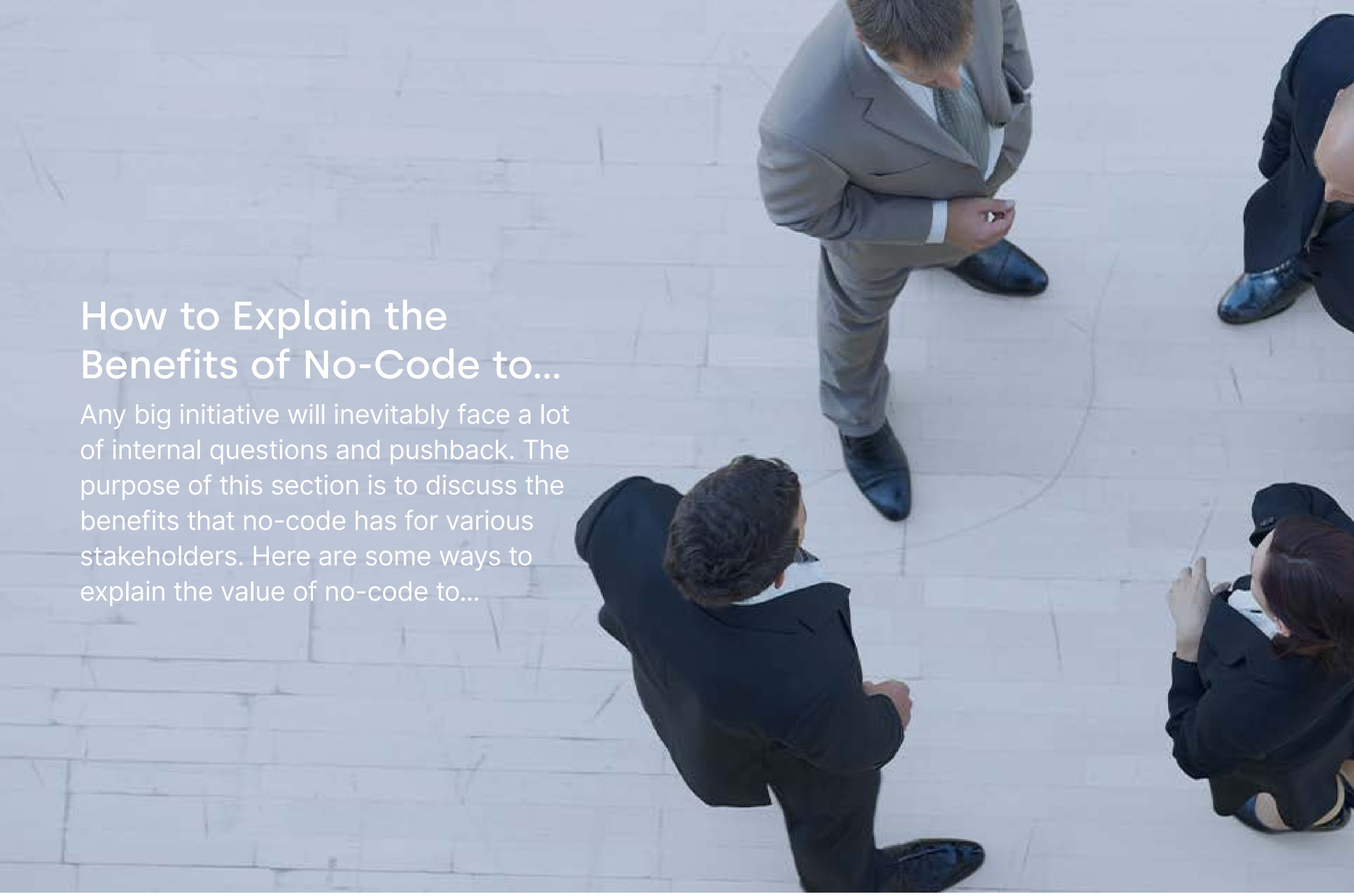
Highly-skilled engineers are not only expensive, but they're also in short supply. This means projects risk getting backed-up whenever an engineer leaves. Because most no-code platforms can be learned in a matter of weeks, they widen the potential talent pool to include a greater variety of backgrounds including less-experienced engineers and non-technical business analysts. As detailed above, this leads to better collaboration, but it also lowers the overall cost of delivering applications.

MORE INNOVATION

No-code application development enables large enterprises to focus more investments and resources into innovation. As mentioned above, almost 60% of IT budgets today are spent simply maintaining the status quo—and anecdotally, we believe that number is actually much, much higher.

Moving a large portion of development tasks to a no-code platform frees up engineers to focus on engineering solutions to complex or unique business problems. The speed and flexibility of no-code makes it easier to experiment and bring brand new ideas to market. Because coded projects have high fixed costs to begin with, limited budgets mean that only the projects that have an obvious value end up moving forward. While this makes sense within the constraints of traditional application development, it often means the boldest ideas (which are potentially also the most lucrative ones) never see the light of day.

Experimentation can be an expensive undertaking, but this is where competitive advantages are secured. Large technology companies like Google and Facebook, for example, are able to constantly test new ideas because they have the resources to invest exclusively on innovation. This allows them to quickly bring new ideas to market, test them, and iterate. Your company probably doesn't have the resources to do this, but lowering the cost of development can open the door to being more experimental.



How to Explain the Benefits of No-Code to...

Any big initiative will inevitably face a lot of internal questions and pushback. The purpose of this section is to discuss the benefits that no-code has for various stakeholders. Here are some ways to explain the value of no-code to...

...the CEO

- **Lowered costs**: Depending on the industry, IT spend today represents a significant percentage of total revenue. Some of this cost is due to the specialized resources required to build applications, but a majority of it goes toward maintaining the status quo with small changes and maintenance. No-code significantly reduces these costs.
- **More high-value projects**: No-code frees-up highly skilled engineers so they can focus on the most important projects that drive your business forward. This is a more effective use of resources than paying the most experienced engineers to build the simplest applications.
- **Minimized risk**: Explain how no-code minimizes the risk of a project going “off the rails” by reducing the number of moving parts. This helps projects reach their full potential and mitigates the risk of significant delays or lost business requirements.
- **Accelerated time to market**: No-code can accomplish initial builds faster, which means projects start generating revenue much more quickly. This can have a huge impact on financials, and also impacts the overall throughput capabilities of the enterprise technology team.

…the CIO

- **Decreased security risks**: The CIO has likely been forced into the "better is harder" dilemma for their entire career. No-code can eliminate this tradeoff. Highlight no-code's ability to dramatically reduce the overall complexity and risk of large projects by having security built-in as a fundamental part of the system.
- **Improved IT hiring**: With no-code, the CIO can widen their pool of candidates qualified to build applications. While it takes years to learn how to build enterprise-grade software with code, business users can typically be trained to use no-code platforms effectively in mere weeks.
- **Increased collaboration**: No-code leads to productive conversations between IT and the business. What happens in the platform is exactly what gets executed in the application, so business users are more easily able to contribute to requirement conversations.
- **Reduced occurrences of shadow IT**: Today, large application development queues contribute to the presence of "shadow IT" in enterprises. Rogue IT projects are often created and funded without the knowledge of IT. No-code can put a stop to this by accelerating the application development process and freeing-up IT resources to work on more projects.

…the CSO

- **Enhanced controls**: Stress no-code's enhanced control over the development process. Even though no-code might seem like the Wild West of application development, it actually gives the CSO more control. Since there is no editable codebase, the people creating applications are naturally constrained by the platform's functionality.
- **Security at scale**: Advanced no-code platforms such as Unqork were built with security as a fundamental building block. That means security remains even as the application scales outward.

…the CFO

- **Amplified productivity**: At its core, no-code stretches every dollar of IT investment. Highly skilled engineers are free to work on the most impactful projects, and less experienced team members are made more productive.
- **Improved employee retention:** Replacing an IT employee is incredibly expensive. No-code is a tool that gives employees a way to avoid tedious development tasks and spend more time taking on high-value—and ultimately more rewarding—tasks. The result is more productive, happier employees doing the best work of their careers.
- **Accelerated time to value:** Every IT project has either a revenue impact or cost reduction as part of its business case. When projects are delayed or miss requirements, these benefits go unrealized. No-code can help ensure that projects are delivered on time and on budget.

Common Concerns About No-Code Platforms

Of course, convincing teams to use no-code platforms has the added challenge of going up against the status quo. Change is difficult and innovation always comes with an element of risk. Here are some common reasons we've seen enterprises give for *not* investing in a no-code platform, along with our response:

"No-code is the same as low-code, it's just marketing hype."

Professional analysts tend to lump low-code and no-code systems together, sometimes calling the category "LC/NC" (low-code/no-code). At Unqork, we believe the industry is doing itself a huge disservice by combining these product categories.

But it's not all the fault of the analysts. The truth is, most platforms that have called themselves "no-code" aren't really "no-code" at all, and as soon as you try to build something even remotely complex, the platform fails and you have to call in the coders or begin again from scratch. Advanced platforms like Unqork are built with enterprise-ready complexity in mind.

"No-code can't build everything we need."

This is certainly true of the first generation of no-code platforms, but today it's highly dependent on your project and chosen no-code vendor. While no-code may not be the best choice for literally *everything* your company needs (for example, a space shuttle operating system), today's enterprise no-code platforms can help you efficiently build both complex customer-facing and internal workflows.

In addition, no-code doesn't have to be the only way your company builds software. Most organizations use a variety of different tools depending on the needs of the project. The key is to start small with a single project before deciding that no-code is your platform of choice.

“No-code is only for business teams.”

The early iterations of no-code tools were positioned solely for the business user, or “citizen developers.” These tools weren’t designed to be capable of complex tasks and were therefore relegated to very simple projects. Today’s no-code platforms are different. By taking core engineering concepts (e.g., objects, variables, and rules) and placing them in a completely visual context, they can be used by skilled engineers to dramatically amplify and accelerate their output.

Citizen development refers to the idea that any business user can simply conceive and construct their own application with little to no help from an engineer. While no-code is related to this idea, we believe that citizen development represents a myopic view of the powerful implications of no-code technologies.

If you think that no-code will simply provide business teams with the tools to build small, inconsequential applications, you aren’t seeing the full picture—and you’ll miss a huge opportunity over the next 10 years.

“No-code can’t support enterprise security.”

Again, while robust enterprise security wasn’t a common feature in many first-gen no-code products, today’s enterprise no-code platforms provide baked-in security functionality that adheres to the highest standards. If you choose the right vendor, applications built with no-code platforms are perfectly capable of having enterprise-grade security, compliance, permissions, and infrastructure capabilities.

“No-code drives vendor lock-in.”

To some extent, this is a true objection against a no-code system. However, we would argue this objection is true of any application platform, as you’re counting on one system to become the foundation for your application development lifecycle. That said, we’d argue the alternative—an entirely open system with a custom codebase—is even less desirable, as it creates a highly complex environment that your organization is locked into.

Choosing a No-Code Platform

We believe Unqork is the best no-code platform on the market, but we're biased, of course. That said, here's an unbiased process you can follow to choose the no-code platform that's right for your company.

STEP ONE

Pick a project

Moving all of your applications to a no-code platform (or anywhere for that matter) in one fell swoop simply isn't practical. Instead, the first step to implementing no-code is to pick a single project or application you've been wrestling with. Maybe it's one that's been backlogged due to resource constraints, or maybe it's an application your team has been thinking about building that isn't fully scoped yet.

At Unqork, we take prospective customers through a simple experiment: We ask them to identify a business process that currently involves a physical paper form as well as all the manual or email-based processes associated with that from. Then we work with customers to turn it into a fully digital process on our first phone call.

STEP TWO

Determine your high-level requirements

This step is about outlining the high-level requirements for your system. This doesn't need to be extremely detailed, but you need to identify the broad strokes of the functionality you're trying to build.

Here's a good initial checklist of things to identify:

- A simple diagram of the business logic
- Any paper-based processes
- Any external systems (legacy or modern APIs) you need to pull data from
- Any calculation logic or decision criteria
- Who in your organization performs each step in the process
- Typical reports you'd want to see from the system

STEP THREE

Establish a baseline

Now that you've defined a use case, you should establish a baseline of what the project would take to build internally with existing IT resources. Even if they won't actually be building the project now, this will give you a strong basis for comparison. You can then use this estimate to measure the benefit of moving to a no-code approach.

At a high-level, we recommend scoping the following areas:

- **Costs of the initial build:** This includes the number of required engineers, project managers, and general management.
- **Time of initial build:** Project timelines and release dates.
- **Ongoing maintenance:** Required maintenance costs for support, changes, and annual upgrades throughout the life of the software.

STEP FOUR

Build a list of vendors

Now that you have an idea of the investment required for an internal build, you should explore no-code vendors that might meet your needs. We won't go into what your list should be here, but a quick scan of the application platform market should give you a good idea of the initial list.

STEP FIVE

Initial demo

In our experience, a strong no-code platform should be capable of building software as soon as you begin your vendor evaluations. After all, if a platform is truly no-code, it should allow you to immediately begin visual software assembly without technical intervention.

STEP SIX

Ask the tough questions

You most likely won't build your project during the purchase process—but seeing the platform in action should enable you to ask questions that really get to the heart of the no-code value proposition. Some key questions include:

- Does the platform operate in a truly no-code environment, or does it simply generate code using visual tools (low-code)?
- Can the platform tackle complex functionality, such as rater systems or complex algorithms and formulas?
- Is it cloud-based, and if so, is your data mingled with other customers' data (single vs. multi-tenant)?
- What happens if you encounter a function that doesn't have an existing component?
- Can the platform serve as the "system of record" for your most sensitive, regulated documents and transactions?
- What roles and security permissioning does it support?
- What product usage analytics are available?
- What software development lifecycle (SDLC) controls and auditing capabilities does the platform provide?

STEP SEVEN

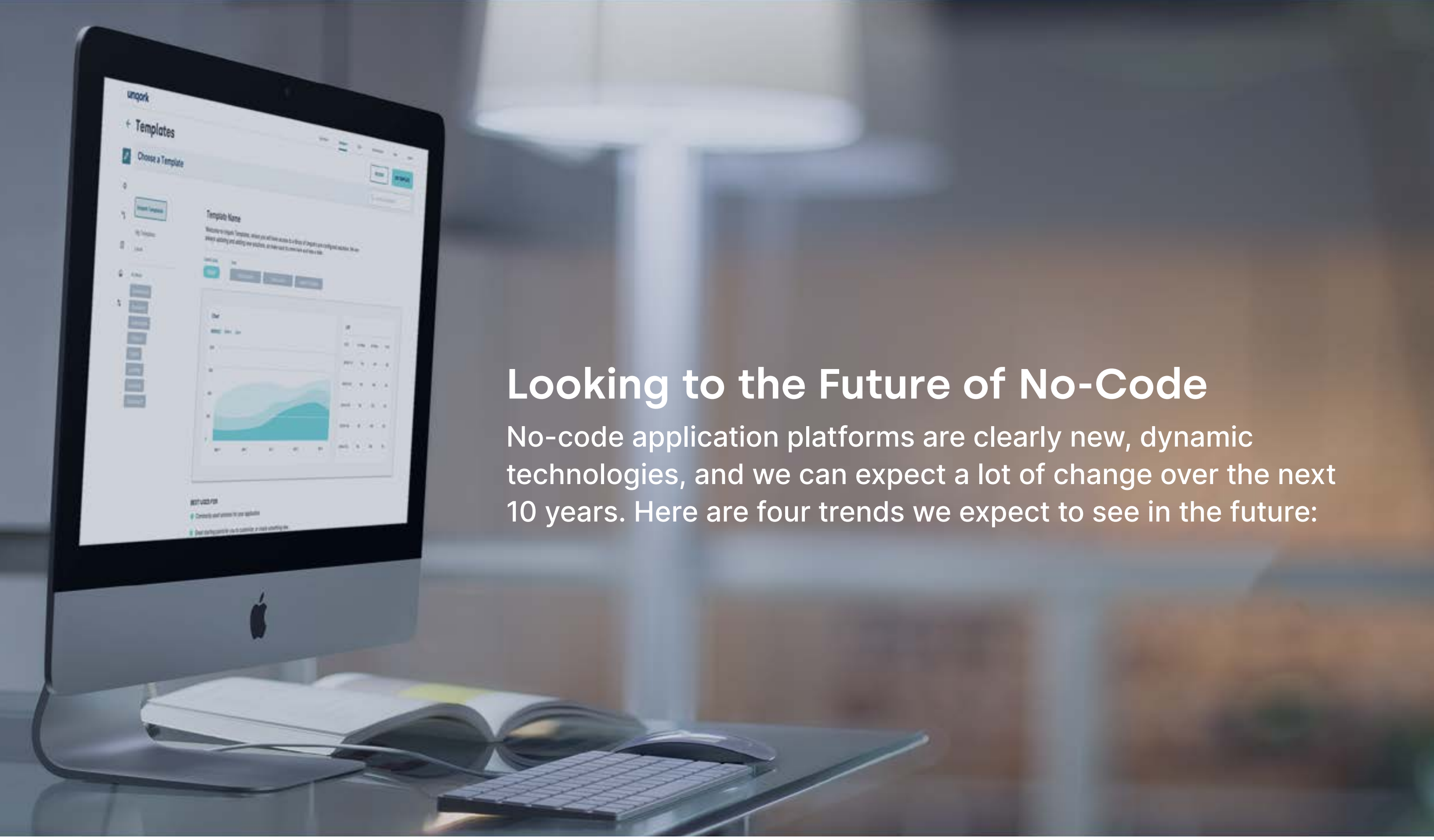
Build a shortlist and talk to references

Ask your vendors for references. You can also do this with your personal network. Try to find references that are similar to your organization. Beyond that, make sure their situation before adopting the platform was similar to yours. Drill-in beyond simple questions like "would you recommend" and try to understand why.

STEP EIGHT

Make a decision

Time to make a decision! You've done the research, seen the demos, and weighed the pros and cons. As you weigh a decision, it's important to remember that even though details like contract terms and subscription fees (within reason) will seem very important during the purchase process, they pale in comparison to delivering your applications on time, spec, and budget.



Looking to the Future of No-Code

No-code application platforms are clearly new, dynamic technologies, and we can expect a lot of change over the next 10 years. Here are four trends we expect to see in the future:

TREND ONE
No-code capabilities will expand

A key differentiator among today's no-code vendors is the complexity and functionality of their components. Some components are built for more basic functionality—for example, capturing an attribute in a form—while others are for more complex functionality, like executing an underwriting algorithm for a life insurance policy.

As more enterprises adopt no-code and deploy it against complex use cases, the overall capabilities of the system will continue to improve and allow for even more advanced functionality. As a result, these platforms will be able to handle increasingly more complex use cases over time. At Unqork we call this "Partner in Platform"—our platform will go as far as the enterprises we work with need it to.

TREND TWO
Empowering a new type of engineer

Today, being an engineer is synonymous with being able to write code. This will change dramatically over the next 10 years. To date, no-code has been fairly synonymous with the idea of citizen development—non-engineers producing applications without the need for code. That said, it still requires a strong working knowledge of how software works to build a truly great no-code application.

We envision a new type of engineer emerging over the next decade. This engineer isn't necessarily versed in the nuances and specifics of any given programming language's syntax, but is well-versed in general engineering concepts like objects, variables, and application logic. This engineer will be able to design effective systems and processes, and then quickly translate them into working applications using a visual no-code system.

There are significantly more people who are qualified to learn these skills than there are classically trained engineers. This opens up huge opportunities during a critical IT skills shortage.

TREND THREE

A return to true innovation

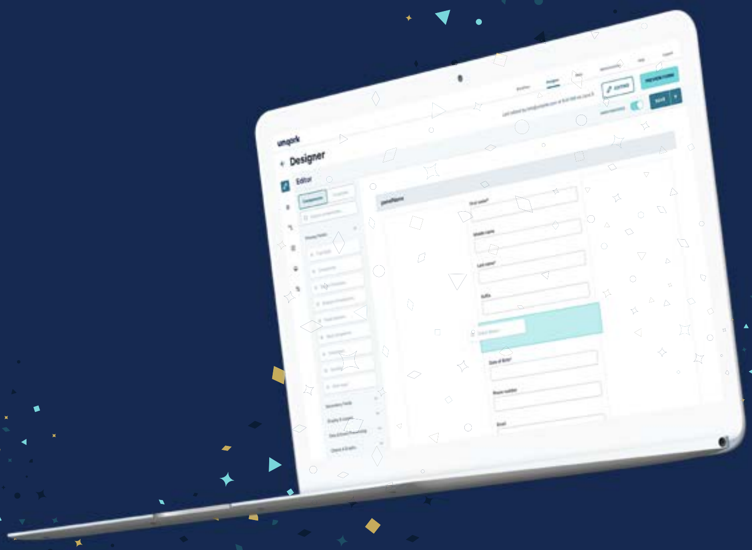
We believe no-code application development will spark a return to innovation for large enterprises. Almost 60% of IT budgets today are spent simply maintaining the status quo. Moving these simpler development tasks to a no-code platform will free-up sophisticated engineers to focus on high-value tasks that truly move the business forward. It will also bridge the gap between the engineering and business sides of companies, allowing teams to better collaborate as no-code blurs the line dividing the two sides.

TREND FOUR

A massive reduction in legacy maintenance costs

A no-code project will typically require much less legacy maintenance than a traditional code-based project. As companies begin to build more applications using no-code platforms, a side effect will be a massive reduction in legacy maintenance costs over time. In the long run, this will result in an "unlocking" of IT resources to not only reduce costs, but also enable a renewed focus on projects that move the business forward.

Unqork's No-Code Application Platform



Enterprise technology leaders have always been forced into a tradeoff. Simple applications can be built with drag-and-drop tools, but more functionality requires a variety of specialized tools, expensive developers, and most importantly, lots of custom code.

Unqork was designed to eliminate this tradeoff. Combining the intuitive nature of no-code with the power of a truly enterprise-grade application platform, Unqork has made it possible to build complex software without having to write a single line of code. The result is faster time to market, improved quality, and reduced cost that's just not possible with conventional tools. Let's dive into how it works.

BUILD

With Unqork, enterprises can build feature-rich, complex applications and APIs within an entirely visual interface.

It's all centered on an intuitive workflow designer that keeps the focus on structuring your application logic. With a rich component library featuring a deep set of industry-specific templates, you can visually build business rules across your application, send messages, and assign work to specific roles or automated tasks.

On the back-end, Unqork enables you to configure your application with the flexibility of a schema-free, document-based database that adapts to your application's logic. That enables you to move structured or unstructured data out of your system for analysis.

INTEGRATE

Unqork makes it much easier to configure full-stack applications, but we recognize that no software exists in a vacuum—especially in the enterprise. That's why Unqork's platform allows users to integrate with everything from modern APIs to legacy systems, and everything in between.

Unqork can also serve as a data hub, centralizing how APIs are accessed across your entire organization and surface them to anyone building an application.

Dealing with unwieldy legacy systems? No problem. Unqork can hook into any legacy file format and extract data, enabling you to use it in an intuitive visual interface.

MANAGE

Enterprise software development is truly a team effort. Unqork lets you manage and monitor performance within a best-practice development process. This will empower your team to maintain version control and reversion capabilities for every configuration, no matter how long ago it was created.

The Unqork platform operates across the entire software development life cycle, from staging to testing to production environments. You can use Unqork to monitor application performance, find bottlenecks and, improve efficiency using deep integrations with your favorite DevOps tools.

SECURE

For any enterprise actively shipping its own software, security can no longer be treated as an afterthought. With our background in enterprise technology, Unqork was built from the ground-up to meet even the most stringent enterprise requirements.

That means your team can encrypt data to and from the applications they're building, with best-practice security controls and compliance standards. Unqork teams operate with confidence in a single-tenant, cloud-agnostic enterprise infrastructure and maintain a comprehensive audit trail with immutable versions of their data.

In Conclusion

Now you’re armed with everything there is to know about no-code application platforms to make these decisions with your company.

Of course, this space changes all the time. We’ll create new versions of this guide from time-to-time and keep you up to date on our blog.

Advanced no-code platforms like Unqork can help your organization achieve adeptly engineer around challenges of any scale. Get in touch to see what we can do for you.

unqork

Enterprise application development, reimagined

Unqork is a no-code application platform that helps large enterprises build complex custom software faster, with higher quality, and lower costs than conventional approaches.

Request a Demo Learn More