



No-Code vs. Low-Code


Low-code may mitigate the impact of enterprise development challenges, but only no-code can overcome them.

unqork



Contents

Introduction	3
The Enterprise Customization Conundrum	5
Code < Low-Code < No-Code	7
The Advantages of No-Code Over Low-Code	9
No-Code in Action	12
Conclusion	15



Today's enterprise applications have been designed to scale complex business tasks across multiple departments, devices, systems, and timezones. These systems provide unprecedented productivity gains for organizations, new opportunities for workers, and enhanced services for users. However, as the sophistication of these applications increases, so do the difficulties of building and maintaining them.

Following years of steady improvements, development times and costs have begun moving [upwards in the past decade](#). Why? **The complexity of modern enterprise systems is colliding with the natural limits of building with code.**

Enterprises are free to continue swimming against the strengthening tide by repeatedly [increasing their software development spend](#). Or, they can overcome this engineering challenge by tapping solutions that engineer *around it*.

Enter "**low-code**." These simple development tools were introduced in the early 2000s as a way to automate repeatable coding tasks. To be sure, low-code may have helped make some common processes more efficient, but it hasn't been able to overcome today's development challenges because *it doesn't overcome code*.

"**No-code**" platforms are all-in-one development platforms that completely eliminate coding from the process. This means organizations can invest all their time and resources into building value and perfecting the user experience instead of toiling with syntax, bugs, and legacy code.

It's understandable why the marketplace may conflate these two similar-sounding terms, but they are anything but interchangeable. Here at Unqork, we *strongly* disagree with the decision of some analysts to casually bundle both technologies together into a single "low-code/no-code" category. In this eBook, we will demonstrate why **only no-code can meet the needs of the modern digital enterprise** through the exploration of several key differences:



Low-code
is a tool



No-code
is a platform



Low-code
is for coders



No-code is for coders,
business teams, SMEs,
executives, citizen developers,
and everyone in between



Low-code is
somewhat faster
than hand-coding



No-code is the
fastest of them all



Low-code still
requires building
with code



No-code allows organizations
to focus all their attention on
building value



Low-code
was great for
tech from the
early 2000s



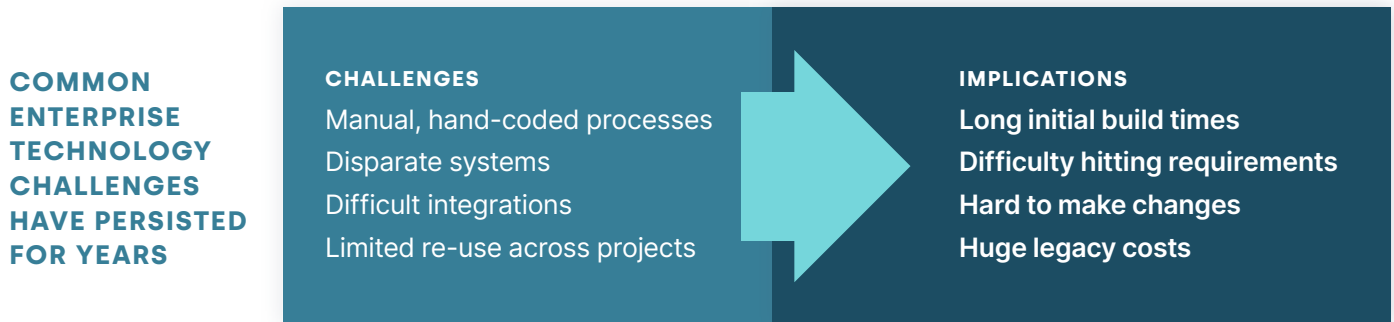
No-code is designed with
mobile, IoT, and AI in mind—
and is built to incorporate the
technologies of the future

The Enterprise Customization Conundrum

Custom software is a crucial part of the modern enterprise. Forrester predicts that enterprises will spend \$550b on creating custom software in 2020, or about half of the total global software spend.

Over the years, packaged enterprise software has grown in both sophistication and popularity, but it can't provide the configurability of a custom-built solution. Only custom solutions allow companies to differentiate their products and processes, and—ideally—secure an advantage over the competition.

However, building custom enterprise software isn't for the faint of heart—particularly when relying on traditional code-based development methodologies. And it's only becoming more challenging.



Custom Development Is Hard (and Getting Harder)

A study by QSM found that between 1980 and 2010, the average time needed to develop custom enterprise software **steadily declined**. These initial improvements were due to factors such as new strategic development strategies, better reuse techniques, and the fact that enterprise IT has become more efficient overall. But then, in the past decade, a strange thing began to happen: Average development times began to tick upwards. In fact, they're **accelerating** (in the wrong direction).

This reversal of fortunes is due to complexity. Today's digital architectures are disparate, expansive, and built on decades of deeply-ingrained legacy systems (some of which have been built with **now-obsolete programming languages**). Beyond the initial build, these challenges impact ongoing system maintenance, as well. In fact, these issues are arguably *even more of a resource-suck* in the latter phases of the software development life cycle (SDLC). Studies have shown that large enterprises can spend up to 75% of their total IT budget just maintaining existing systems.

As these troubling trends grow, so do the expectations placed upon enterprise software. Today's businesses expect development to move at the speed of the economy. Indeed, the ability to build robust software rapidly is a fundamental aspect of **digital resilience**, an important competitive differentiator defined by an organization's ability to mount a robust digital response to marketplace disruptions of any scale.

It has become painfully clear that traditional code-based methodologies are simply not able to keep up with the demands of the modern enterprise. Low-code tools may mitigate the impact of these challenges, but only no-code platforms can overcome them.

A Brief History of Software Development Productivity

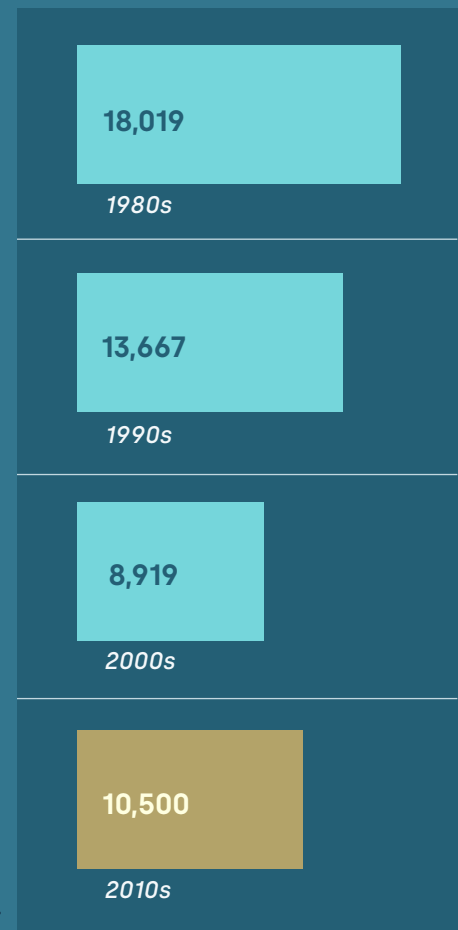
For decades, the standard in software development has been based on the idea of building faster.

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

Code < Low-Code < No-Code

Low-code and no-code both rose to prominence with the promise of replacing hand-coding with visual elements, so developers can spend less time on coding and syntax and focus more attention on building business logic and perfecting the user experience.

To be sure, low-code has been able to inject some efficiencies to the process, but only no-code can meet the efficiency demands of the modern enterprise (see page 13 to read about how, in just a matter of days, NYC was able to use no-code to develop a series of digital hubs that could remotely connect citizens to vital services).

It might be useful to frame these two technologies this way: Low-code is a somewhat-useful development tool, while no-code is a complete development platform that comes with all the tools needed to build and maintain enterprise-ready software. Let's dive in...

WHAT IS "LOW-CODE?"

Low-code (LC) was developed in the early 2000s to amplify developer productivity. These solutions work by inserting repeatable scripts into specific parts of the platform. Some of these tools may have drag-and-drop functionality, but to achieve complex—or unique—custom functionality, coding is still necessary.

LC tools work with various common programming languages, while some use a proprietary language that users have to learn. These tools were designed with coders in mind, not for business or other non-IT users. Furthermore, most LC platforms focus only on internal, back-end workflows and few have any consumer-focused capabilities (e.g., styling, anonymous users).

WHAT IS "NO-CODE?"

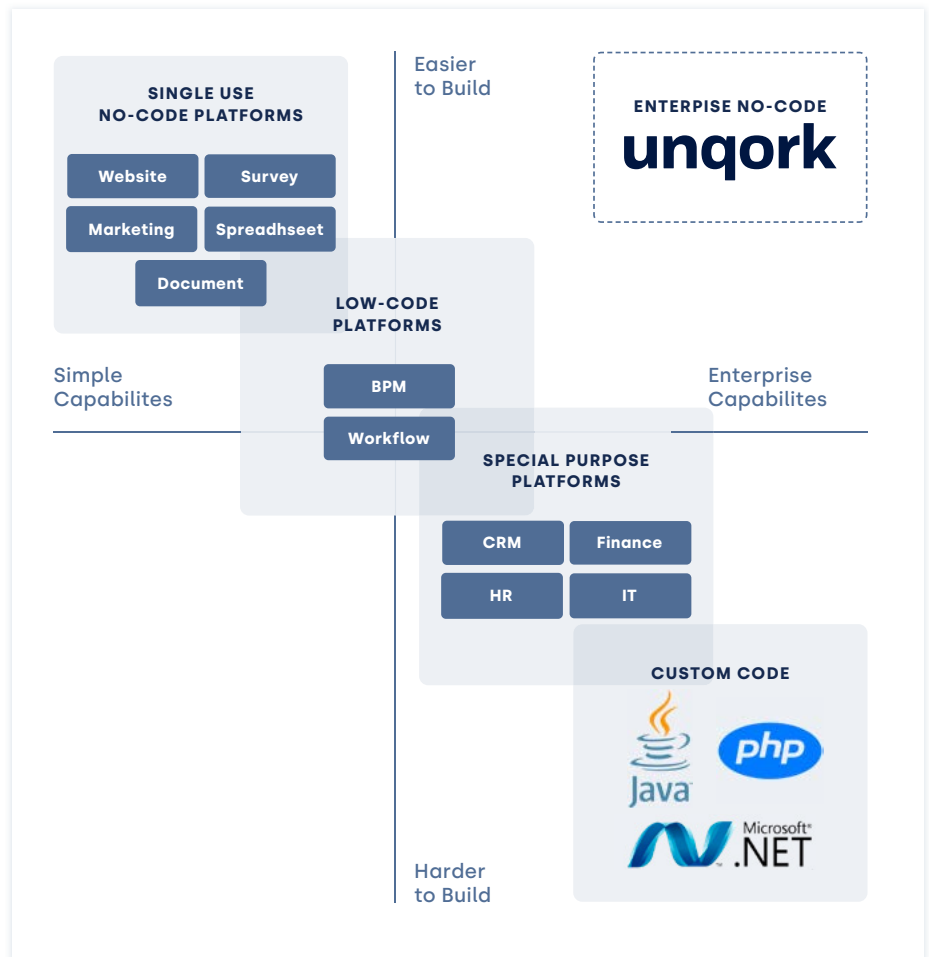
No-code (NC) is a category of cloud-based services that provide enterprises with a single unified platform to develop, run, and manage applications without the complexity of managing all of the parts themselves. These platforms were developed in the late-2010s with a focus on rapid, flexible application development for both seasoned developers and non-IT professionals (i.e., "citizen" developers).

Low-Code vs. No-Code: The Basics

	Low-Code	No-Code
History	Developed in the early 2000s to accelerate developer productivity vs. hand-writing custom code from scratch	Developed late 2010s with focus on rapid, flexible application development for both developers and non-developer
Primary use case	Most low-code platforms focus only on internal, back-end workflows; few have consumer-focused capabilities (e.g., styling, anonymous users)	Built for any type of application including both internal and external-facing software
Relationship to code	Scripting inserted in specific parts of the platform, designed to make writing code more efficient; platform user writes code	Substitutes code with configuration, configuration is then translated to code automatically upon rendering; platform user never sees code
Scripting language	Varies, some allow for commonly-used languages, others use their own proprietary language that must be learned	None, scripting is not allowed anywhere on the platform

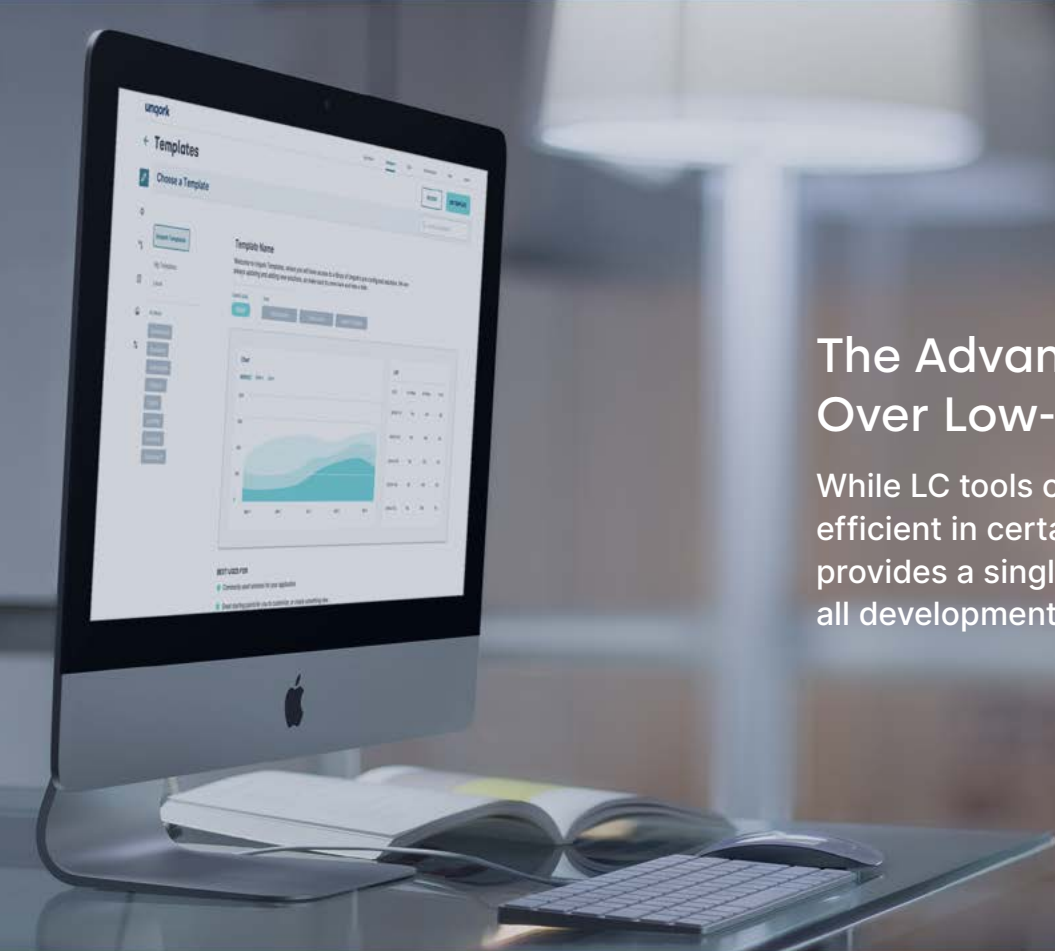
Before moving forward, we should highlight two important distinctions about NC.

One: several simple, consumer-grade “no-code” solutions allow users to, say, design a website or a survey sans any coding (e.g., Squarespace, Survey Monkey, etc). These are fine for limited applications with little need for advanced functionality. There are also “special purpose” NC systems that are only applicable in very specific areas like HR or Accounting, but can’t be used in other parts of the business. In this eBook, we are focusing on enterprise-level NC platforms like Unqork which are designed to build complex, robust applications for any industry, any role, or any user.



Two: the phrase “no-code” does not mean that there is no code anywhere in the platform—rather, NC provides a layer of abstraction between the user and the codebase. With no-code, organizations can focus all their efforts on configuring application logic, rather than dealing with code or syntax. The platform [takes care of everything](#) “underneath the hood” and automatically translates the user’s configurations into code upon rendering.

With NC, there are no languages to learn or master (proprietary or otherwise). Everything is handled by manipulating settings on different configurable modules that represent either user-facing elements on back-end functionality. Indeed, scripting isn’t even allowed on most NC platforms (because that’s kind of the point). This is a powerful distinction between LC and NC, which leads to many inherent advantages that we’ll detail in the next section.



The Advantages of No-Code Over Low-Code

While LC tools can help coders be a bit more efficient in certain development tasks, NC provides a single unified platform to handle all development tasks.

The intuitive code-free interface of NC means it can also be accessed by a wider-range of users. By eliminating code from the development process, NC platforms provide several distinct advantages over LC. Notably, NC is...



...way faster: With NC, organizations can focus their efforts on solving business challenges rather than coding ones. Consider that: 1) a typical hand-coded enterprise application can take 9-12 months to build, and 2) with LC, building an application of equal complexity can be sped up to 3-6 months. However, with NC, the process can be executed in just 2-3 months.



...far easier to use and results in improved collaboration: Learning to code is difficult, it takes years to become highly productive and efficient. NC's drag-and-droppable, module-based interface opens the development process to a wider spectrum of business users who can create/edit/approve parts of the application themselves. This enhanced collaboration can help bridge the divide between IT and business teams.



...able to bridge the IT skills gap: LC still requires organizations to keep coders with specialized knowledge on the payroll, which only becomes more expensive as the [IT skills gap](#) continues to widen. Since NC is easier to use overall, organizations can be less reliant on specialized (i.e., costly) coding skill sets.

We should note that NC does not eliminate the need for engineers. Rather, NC allows experienced engineers to spend their time building a greater number of effective, user-friendly custom applications. At the same, less complex software upkeep tasks can be relegated to less-experienced engineers who are just learning their way.



...far less prone to errors: Syntax can be very tricky and fragile—even for experienced coders. Every time you type code (which, to reiterate is still necessary with LC), mistakes can be made. When bugs pop up, they need to be investigated and smashed. By removing the need to write *any code*, NC completely removes the specter of human error.



...future proof: Any code you insert into your application instantly becomes tomorrow's legacy technology. In the case of LC, business logic is completely separated from technology upgrades. This means whenever the tools are upgraded, the system can fail. With NC, [any system upgrades](#) won't break any business logic designed within the system.

The Technology

	Low-Code	No-Code
Front-end development	Basic functionality can be configured in a visual editor, but complex operations (e.g. form data validation) requires scripting	Both basic layouts and and complex operations can be configured without the need for scripting
Back end processes and workflow	Application workflow can be configured visually, executing either pre-built modules or scripting	Application workflow can be configured visually, executing either pre-built modules or scripting
Integrations	Modern integrations done with configuration, but legacy systems or more complex data transformations require code	Modern and legacy integrations can be configured without scripting
Data transformations	Data transformations and logic requires complex code and data transformation	Conduct data transformations with a completely visual ETL tool that incorporates visual import

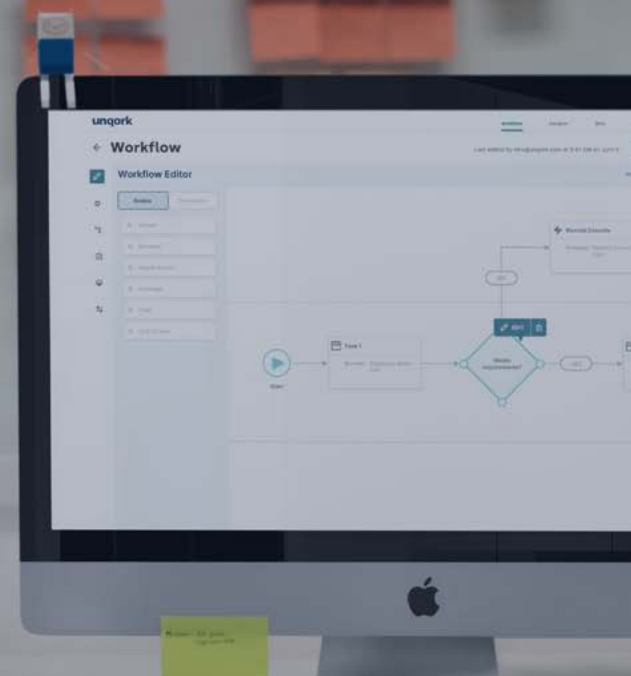


The Results

	Low-Code	No-Code
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

No-Code in Action

No-code isn't just some theoretical future application—it's making a very real impact right now. In this section, we'll explore a couple of real-world examples of how no-code was used to address challenges through the rapid development of robust digital applications.



CASE STUDY

A Wealth Manager Used No-Code to Digitize its Entire Client Lifecycle in Just 12 Weeks

With no-code, organizations are able to accelerate the development of custom software. Take this example from a global-leading wealth management firm, which used no-code to develop and deploy a robust value-creating digital system in just 12 weeks.

CHALLENGE

Facing a revenue slowdown due to low-advisor productivity and prolonged time to onboard and service clients, the firm's margins were under pressure due to the high operational cost and risk driven from manual processes and controls.

SOLUTION

Using Unqork's no-code solution, it only took 12 weeks for the wealth manager to build an end-to-end digital solution, fully automating client/advisor data capture, KYC, suitability, product selection, and account opening. The no-code solution was fully integrated with record keeping systems, as well as third-party services such as SFDC, DocuSign, and PLAID.

RESULTS

Not only was the application developed and deployed in record time, but it resulted in tangible business benefits.

- **Accelerated** client onboarding times by 60%
- **Reduced** operational risk by 70% with automated controls
- **Decreased** cost of operations and ownership by 40%
- **Increased** revenue by 20%

CASE STUDY

NYC Used No-Code to Mount a Rapid Digital Response to COVID-19

The impact on New York City by the initial wave of the COVID-19 pandemic was as deep as it was rapid. In order to respond adeptly to the crisis, the city needed to quickly develop a suite of robust digital tools that no one was planning—let alone even considering—just a month before.

The speed of traditional development methodologies (particularly those at the local government level) would not be sufficient to address the rapid pace at which the disease was rampaging through the city. In order to accelerate the development of four enterprise-grade digital portals to address the crisis, the city's COVID response team tapped the power of Unqork's no-code development platform.

- **COVID-19 Engagement Portal:** In just 72 hours, the city was able to build and deploy the [COVID-19 Engagement Portal](#). The Portal, which is available in 11 languages, allows residents to self-report COVID-19 data, which the city can use to map the impact of the virus and connect residents with critical services.
- **PPE Donation Portal:** As infection rates grew at an alarming rate in those first few months, the healthcare system found itself facing shortages of critical PPE, so the city worked with Unqork to rapidly build a [PPE Donation Portal](#) that allows individuals and organizations to donate much-needed medical equipment.
- **GetFoodNYC Delivery Portal:** Prior to the pandemic, millions of NYC residents relied on food pantries, soup kitchens and congregate meal programs at senior centers. Given the fast-moving economic

impact of the pandemic, the number of city residents who depend on these services expanded at a rapid clip.

To provide food for COVID-19-vulnerable and food insecure New Yorkers not currently served through existing food delivery programs, Unqork worked with the City to launch the [GetFoodNYC Delivery Program](#) which provided Taxi Limousine Commission-licensed drivers with the opportunity to earn money while making food deliveries to vulnerable New Yorkers.

"In just a few days, the Unqork platform helped us deploy applications to address a wide spectrum of needs—from health impacts, to hunger, to PPE for healthcare workers on the front lines."

-JESSICA TISCH, COMMISSIONER OF DOITT AND CIO FOR NEW YORK CITY

- **"Project Cupid" Marriage License Hub:** To ensure that citizens could still enter legally-recognized unions despite social-distancing barriers, the city worked with Unqork to develop a hub to process marriage license applications remotely. The [Project Cupid](#) platform digitized the entire process from application to identity verification to online fee payments to license generation.

The development process for these portals would have been far slower had they relied on traditional development methodologies. Since going live, these support hubs have enabled the delivery of over [40 million free meals to residents](#), accepted donations of essential medical supplies, and allowed NYC residents to self-report how they are impacted by COVID-19.

Business impact realized by Unqork customers



FINANCIAL INSTITUTION

Digitization of home loan application process

Developed a single application automating end-to-end loan origination across borrower home loan application, underwriting, offer, and acceptance processes.

- **Speed to Market:** 6 weeks to go from ideation to production, with only 4 resources
- Increased revenue capture potential & improved broker productivity



A TOP 5 RETIREMENT SOLUTIONS PROVIDER

Digitization of plan sponsor onboarding

Developed an end-to-end, digital self-service solution automating sponsor, plan, servicing, pricing, advisor, and TPA data capture.

- **Speed to Market:** 16 weeks to go from ideation to production, with only 4 resources
- Accelerated client onboarding times from 4 weeks to 3 hours



GLOBAL P&C CARRIER

Digital front office and policy administration

Developed an end-to-end digital solution fully automating intake, quote, bind, issue for no-touch and underwriter referral workflows.

- **Speed to Market:** 12 weeks from inception to production, with only 5 resources.
- Reduced average time-to-quote by 90% (real-time quoting)



GLOBAL INSURANCE BROKERAGE

Digitized broker and carrier sales operations/marketplace

Developed an end-to-end digital marketplace concept that enables brokers and carriers to manage the lifecycle of the sales prospect.

- **Speed to Market:** launched marketplace concept in 8 weeks
- Improved response time to customers by more than 50% resulting in higher sales potential



A TOP 5 LIFE INSURER

Digitized invasive medical questionnaire

Developed a direct-to-consumer, customer authenticated, self-service application (mobile native) that digitized entire interview process and underwriting.

- **Speed to Market:** Launched app from ideation to production in 8 weeks
- Reduction in turn-around time, from 45 minutes to less than 10 minutes

In Conclusion

With no-code, organizations can separate the benefits of code from the pitfalls of coding. This paradigm is more important than ever as traditional methodologies have proven themselves incapable of keeping up with the expectations of today's modern connected enterprise.

Here at Unqork, we believe the future of software development is based on configuration, **not code**. We believe that the elimination of code is the only way to bring development up to today's standards. Want to learn more about what no-code can do for your organization? [Get in touch](#) to see how we can work together.

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.

