unqork

Legacy Code Is the Problem, No-Code Is the Solution

Find out why today's most innovative enterprises are sidestepping code altogether with Unqork.

unqork

Contents

"Legacy code" is a bit of a catch-all phrase developers use to casually refer to any existing code they find themselves working on. While the term is somewhat ambiguous, the impact is all too real. Maintaining years—or even decades—of legacy code demands an outsized amount of developers' time, increases overhead costs, and inhibits an organization's ability to efficiently address challenges.

Case in point: Consider last year's unexpected rush on COBOL programmers. Many government and financial systems still run on COBOL[1], even though most universities no longer include this half-century-old language in their CS curriculum. This turned out to be a huge problem for several legacy systems, which needed to be updated in response to the COVID pandemic. In order to take advantage of the CARES Act passed by Congress, state unemployment systems needed to be quickly updated to accommodate new rules and a massive influx of recipients. The state of New Jersey, however, found itself stymied through their reliance on a comparatively ancient codebase, which led the governor to publicly plead for COBOL-literate volunteers to update its infrastructure.

The COBOL rush may have been a highly visible case study in the problems of legacy code, but even code based on contemporary languages can cause headaches and heartburn for developers and businesses alike.

As technologies, business processes, and workforces evolve over time, so must the code that supports them. The more complex and sprawling your digital footprint, the more challenging this becomes. Today' enterprise environments evolve so quickly that as soon as any code is sent to production, it's already legacy. The only way to avoid the challenges of code is to remove it altogether. In this eBook, we will explore how a new class of no-code development platforms such as Unqork can eliminate the challenges of legacy code.

[1] It's predicted that 70-to-80% of all business transactions worldwide are still based on systems written in COBOL.

The Consequences of Legacy Code

Modern-day developers are responsible for scaling functionality across multiple departments, devices, systems, and locations—which means maintaining hundreds of thousands of lines of code built over years, if not decades. Here are just a few of the challenges that enterprises face from an expansive legacy codebase:

Inflexibility

Today's architectures are expansive, disparate, and built on decades of deeply-ingrained legacy systems. Most pull data from or push data to external systems, storing data in one system, analyzing it in another, and using another to display it.

Integrating and meshing these systems together with convulited workflows makes it difficult and expensive to maintain and make updates/changes down the line. Attempting to build and manage within these complex digital systems using code directly contributes to reduced IT productivity. Despite all the iterative-improvement coding tools and organizational approaches, development productivity is getting worse, not better. With one study finding a 20% loss in developer productivity over the past decade. (For more, see "A Brief History of Enterprise Development" on page 14).

20%

The increase in hours to build enterprise software over the past decade

Technical Debt

With limited resources to address every problem in an expansive codebase due to changing technological or business environments, IT teams often must settle for kicking problems down the line with small workarounds. This is known as technical debt. And like all debts, the "bill" will eventually come due.

A very public example of that bill coming due can be found in the recurring effects of the millennium Y2K bug, which is still causing problems two decades later due to short-sighted fixes. Back in 1999, many developers "windowed" the dates in the codebase to fix 80% of various systems. This fix treated all dates from 00 to 19 as being 2000 to 2019, while anything beyond the "pivot date" would be considered a previous century. This fix was intended to be a short-term fix, which would surely be replaced before 2020. However, in many cases, the fix never came.

On January 1, 2020, many of the "fixed" systems rolled back to the early 20th century and read the "20" in dates as 1920. Utility companies reportedly produced bills with this incorrect date, and thousands of New York City parking meters rejected credit card transactions because of it. The video game WWE 2K20 also reportedly stopped working that day too, until the developer patched it a few days later. It led to a lot of funny news stories, but for the affected companies, it caused more headaches than laughs—not to mention the costs incurred. One 2018 report estimated that the cost of technical debt that year in the US alone was around a half trillion dollars.

The more code you maintain, the more potential you have to fall into technical debt. One analysis by the CAST Research Labs (CRL) analyzed found that the average-sized application of 300,000 lines of code has over $1 million worth of technical debt. That represents an average debt of $3.61 per line. This can be a huge amount of money over time and place companies at a disadvantage.

No-Code Vs. Technical Debt

Consider the plights of Enterprises A and B. They're similar-sized companies with similar goals and are in similar markets. Enterprise A decides to take a traditional code-based approach, while Enterprise B embraces no-code.

Using this framework referenced in this section, we can calculate how much additional technical debt Enterprise A incurs over time by relying on code. Meanwhile, over the course of 5 years, Enterprise B avoids more than $5.4 million in costs to address technical debt. This also means that profits and revenues grow because they can innovate and release new products to stay ahead of the competition.

	Enterprise A *Code-Based Approach*	Enterprise B *No-Code*
Assumed New Projects Per Year	5	5
Average LOCs Per Enterprise Application	300,000	N/A
Total New LOCs Created Per Year	1,500,000	N/A
Average Cost of Technical Debt Per LOC	$3.61	N/A
Cost of New Technical Debt Each Year	**$5,415,000**	**$0**

Maintenance Costs

As an organization's digital infrastructure—and its codebase—expands over the years, IT teams are tasked with untangling the web every time a change needs to be made.

Developers will tell you that legacy code takes up a large chunk of their time. One survey found that developers spend 30% of their time maintaining code or an average of 12 hours every week. A Stripe survey found that they spent an average of 17.3 hours every week on code maintenance.

Other studies have shown that developers spend 50% of their time learning the code they have to work with and their employers spend 60% of the development budget maintaining it. No matter how you look at it, that's a huge portion of your IT resources just fixing bugs and dealing with technical debt.

The more time you spend researching old ways of doing things or reaching out to retired developers for fixes to code you're stuck maintaining or fixing.

Add in the new lines of code you add every year, it can add up to several million dollars annually. In the U.S., companies spent $70 billion in 2003 on maintenance and nearly $600 billion in 2018. That's a 7x increase over just 15 years.

30%

Percentage of time developers spend maintaining code

60%

Percentage of development budgets dedicated to maintenance

7x

Increase in maintenance costs between 2003 and 2018

Risk

Like those COBOL-based applications or systems that are a honeycomb of applications and networks that are bolted on to each other (like airline management systems). Systems are so interconnected that one small problem can have a cascading effect.

Consider what happened in 2019 when a single piece of safety software grounded flights from at least five airlines, causing planes to be grounded nationally. And the previous week, another software application went down, crippling ticketing and boarding operations for another three airlines.

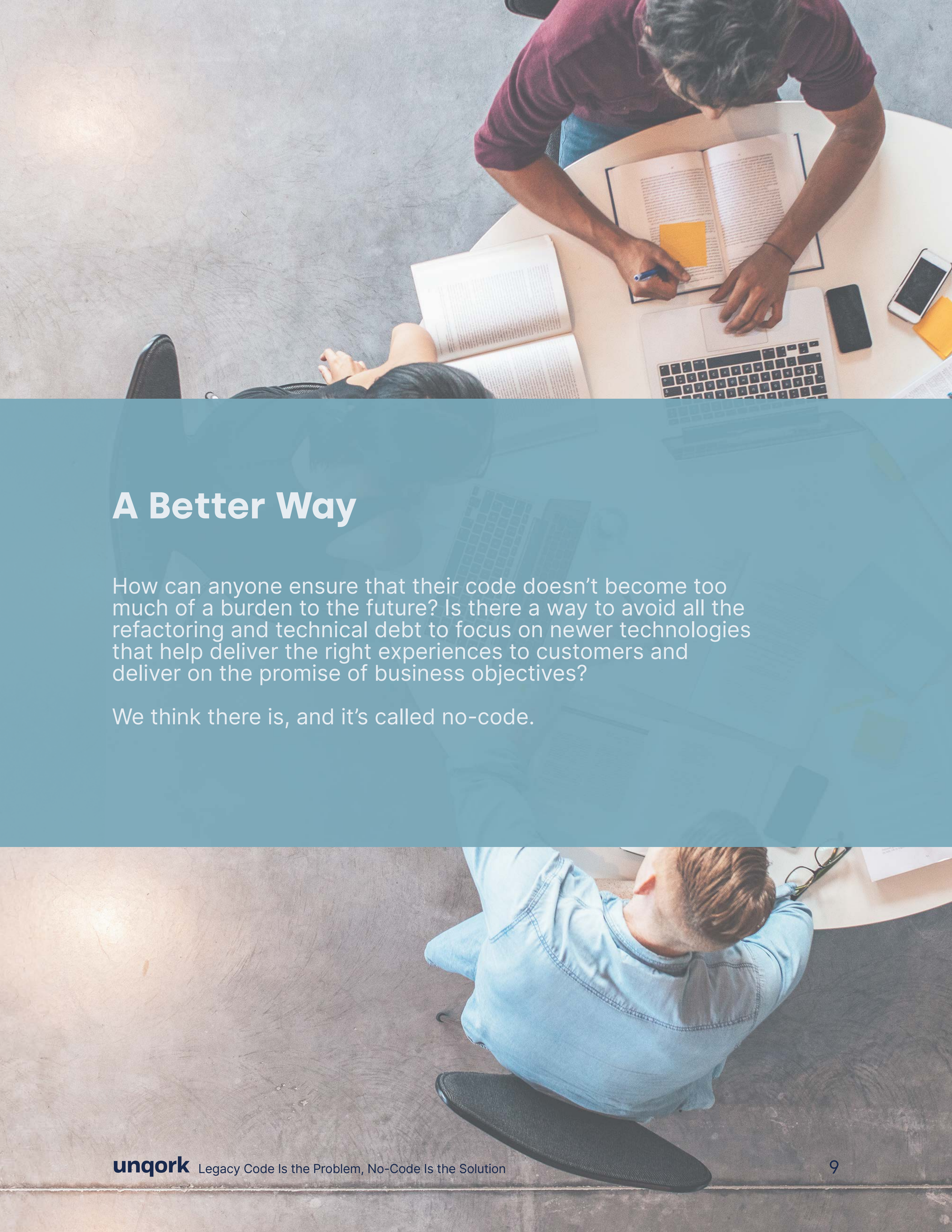
A sprawling infrastructure is inherently unwieldy, so when you add hotfixes to it that cause new issues, your problems suddenly become very visible to end-users and consumers. You're assured financial damage (according to Gartner, network downtime costs enterprises, on average, $5,600 per minute) as well as reputational damage, which is less quantifiable, but can impact your business for years to come. Factor in the 100-to-150 errors most developers average for every 1,000 lines of code (according to Watts Humphrey's book, *A Discipline for Software Engineering*) and you can see how your codebase is a potential timebomb.

100-to-150

The average number of errors for every 1,000 LOCs sent to production

$5,600

The average per-minute cost of network downtime for a large enterprise



A Better Way

How can anyone ensure that their code doesn't become too much of a burden to the future? Is there a way to avoid all the refactoring and technical debt to focus on newer technologies that help deliver the right experiences to customers and deliver on the promise of business objectives?

We think there is, and it's called no-code.

The No-Code Solution

No-code allows companies to build robust custom solutions without writing a single line of code. No-code platforms like Unqork provide a layer of abstraction between developers and the codebase, which means developers focus their attention on configuring application logic rather than dealing with their application's code or syntax.

To be clear, no-code systems do generate code, but users never have to invest any thought or resources into writing or maintaining that code—and that can make a huge difference. Here are a few highly impactful benefit that come from choosing a code-free paradigm:

- ✓ **Stop Producing New Code**: The most obvious benefit of an enterprise no-code platform like Unqork is that it eliminates the creation of new code that you will have to deal with, which means your team can focus its resources on building value-adding solutions, or maintaining/sunsetting existing legacy code without adding code that will eventually have to be worked on.
- ✓ **All Technical Updates Are "Outsourced"**: With no-code, the platform takes care of everything "under the hood" and eliminates the need for you to keep systems up-to-date because the no-code platform provider takes care of everything on the backend. You design the business logic, and the platform handles all the shifting technologies.
- ✓ **Eliminate Human Coding Errors**: No-code eliminates the chances of developer typos when working on the codebase because it removes the need to write any code—and in many cases provides guardrails which proactively avoids errors before they happen.

Users create functionality through the visual interface, and the platform handles the coding. The Unqork platform, for example, lets you build your app with configurable components and provides "guardrails" to proactively guide users away from errors.

- ✓ **Accelerated Development**: Building business solutions via a visual configuration-based platform is inherently faster than writing them out in code. This acceleration empowers businesses to hasten time-to-market and time-to-value for new solutions, as well as for any updates down the line. The average enterprise application can take up to a year to go from ideation to production using a traditional code-based approach; with no-code sophisticated applications can be created in a matter of weeks, or even days.

- ✓ **Flexible Workforce**: A study by Forrester found that while 74% of enterprises have a digital strategy in place, only 15% seem to have the right skills to deliver it. No-code can help mitigate the global IT skills shortage. While modern programming languages can take a year to learn, and several more to master, platforms like Unqork can be picked up in a matter of weeks.

  Since the platform handles so much of the "heavy lifting," companies can use less-experienced developers (or "Creators" as we refer to them at Unqork) to handle routine tasks, while more-senior developers can concentrate on applying their experience in more value-additive ways.

At Unqork, we believe the future of software development is based on your business and the people in it, not the code you create or maintain. We believe in using legacy code for good and preventing you from creating more in the future. Want to learn about how to transition your business from legacy code to no-code? Get in touch to see how easy it can be.

Unqork: The First Enterprise No-Code Application Platform

Unqork is the first enterprise no-code development platform specifically engineered for the world's most complex and regulated financial service environments. Our platform empowers companies to rapidly build and deploy sophisticated applications without writing a single line of code.

We are backed by some of the most disciplined investors in the world, including Goldman Sachs, Capital G, and BlackRock. In just three years, our technologies have been adopted by dozens of leading organizations (including Liberty Mutual, John Hancock, State Street, and the city of New York just to name a notable few).

Unqork is a cloud-agnostic SaaS platform, so customers can avoid cloud vendor lock-in and deploy applications with the provider of their choice. All customers operate in single-tenant environments, so there is never a mixing of data between Unqork clients.

Our clients can achieve unparalleled speed and flexibility in their development function while requiring a fraction of the resources. We can deliver these benefits through:

A unified SaaS platform: Unqork is a completely unified SaaS platform, which means it provides all the components and capabilities related to crucial areas like **compliance** (up-to-date regulatory and enterprise rules engines for FATCA, CRS, UK CDOT, Dodd-Frank, EMIR, and MiFID II, etc.), **security** (native encryption both in transit and rest, custom RBAC capabilities, and crowd-sourced penetration tests), and **application management** (SDLC governance, application versioning, and module management)[2].

A visual UI: Applications are built via an intuitive, visual User Interface (UI) featuring drag-and-drop components representing user-facing elements, backend processes, data transformations, third-party integrations, and a growing library of industry-specific templates.

Enterprise-grade standards: While there are several business-area-specific or consumer-level no-code systems on the market, Unqork is the only no-code platform designed specifically to build complex, scalable, enterprise-ready applications, which is why it's already being used by some of the world's leading organizations.

[2]While Unqork is a SaaS platform, our customers operate in single-tenant environments, which means there is never a mixing of client data between Unqork customers. Unqork is cloud-agnostic, so customers can avoid cloud vendor lock-in and deploy applications in the cloud of their choice.

Unqork allows enterprises to shift all their focus to addressing business challenges instead of technical ones. The platform takes on the “heavy lifting” and frees organizations to invest their resources building operational efficiencies and perfecting the client experience. This streamlined approach helps organizations achieve:

- **Accelerated speed-to-market:** No-code automates many high-volume development tasks so new applications can be built and deployed much faster. In many cases, applications that would take months or years to reach the market can be built in a matter of weeks, or even days.
- **The elimination of legacy code:** Code becomes legacy nearly instantly. With no-code, organizations only need to be concerned with building business logic, even if there is a technical change, the platform handles all that on the backend.
- **Ease of updates and maintenance:** Large enterprises can spend up to 75% of total IT budget maintaining existing systems. One of the reasons is the complexity of making a change in one area requires changes throughout the process. A no-code platform automates many of these cascading tasks and therefore reduces the complexity of making changes.
- **Business agility:** Whether it is a pandemic or disruptions of a smaller scale, no-code can provide organizations with a way to address events quickly.

Curious about how no-code can be applied within your organization? Get in touch to schedule a demonstration from one of our no-code experts.

Appendix

A Brief History of Software Development Productivity

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

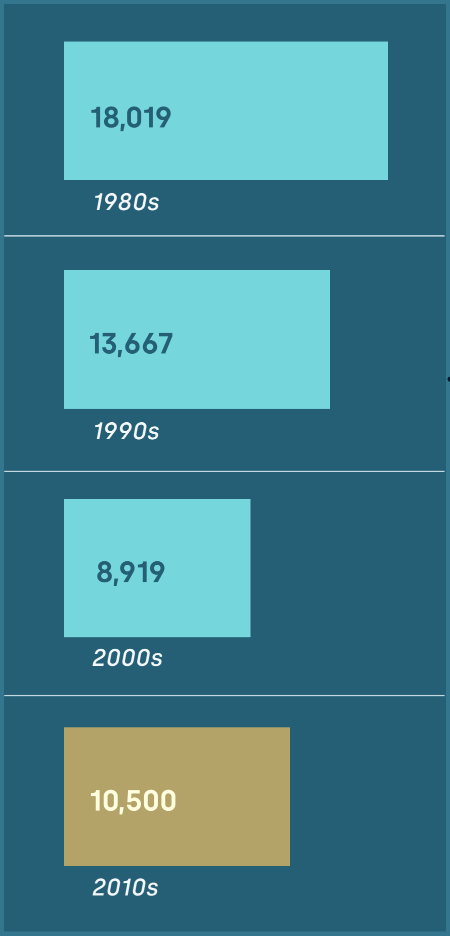
This isn't a new challenge. In fact, many years ago, the problem was even worse, as many of the stopgap solutions we rely on today didn't even exist yet! Let's take a quick look at how building applications has evolved over the last several decades:

- **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**,

.

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 Decelopment Database, 2019