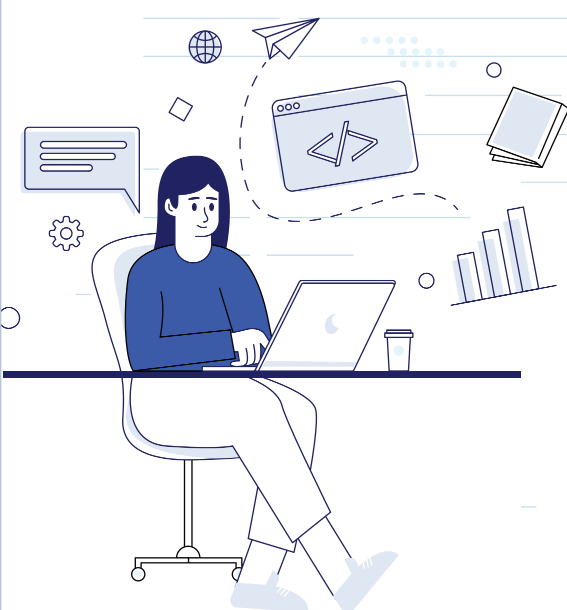
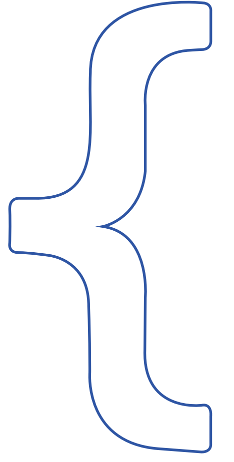
The

threads of low-code

{ Low-code past, present, and future }

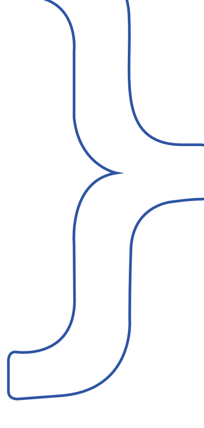


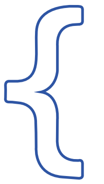


Low-code tools are an increasingly popular alternative to traditional hand coding. Despite its success, however, low-code continues to face skepticism.

To overcome this skepticism, it's important to understand the role low-code plays in today's business world, where it's been, and where it's going.

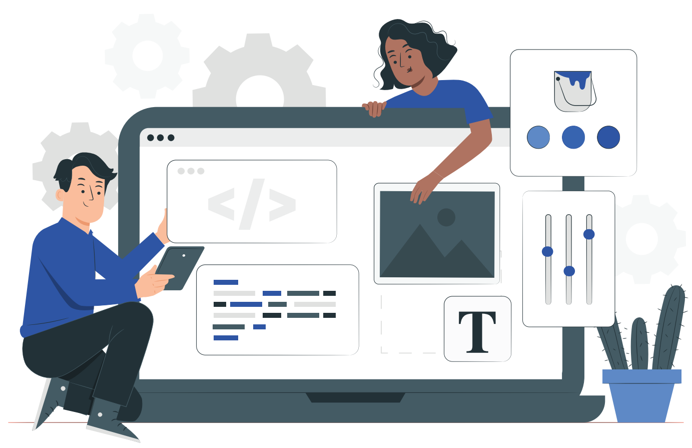
The best practices behind low-code, in fact, have been evolving for decades. Tying these historical threads together shines new light on the power and importance of low-code today and where it might go in the future.





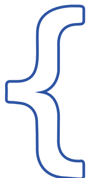
Precursors of modern low-code

With low-code (and its simpler cousin, no-code), a wider range of people (no longer strictly professional software developers) are responsible for application building. For this, they use a variety of visual metaphors, from drag-and-drop widgets to configuration wizards to Tinkertoy-like assemblies of boxes and lines representing process logic.

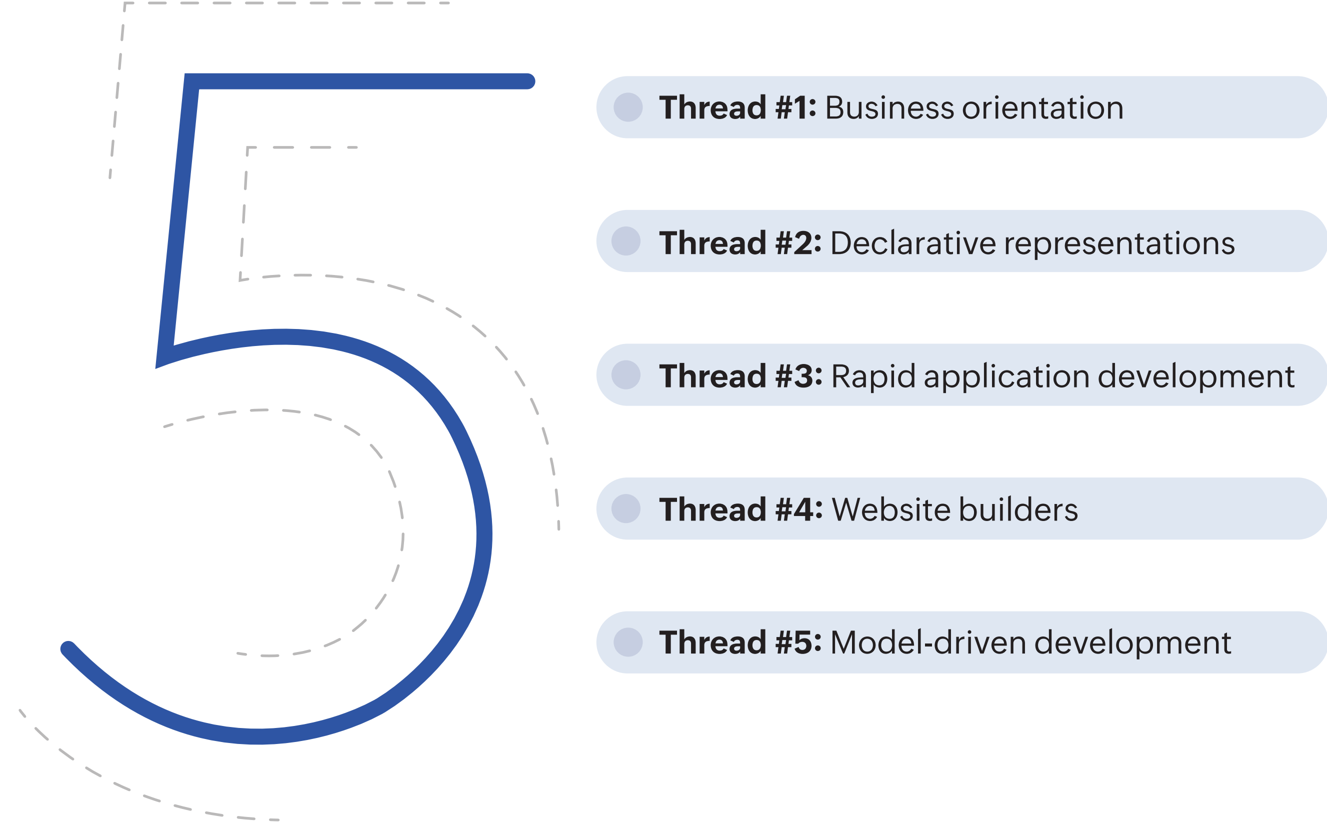


Low-code might seem like an unfair shortcut for those professional developers who spent long hours in college and many years of their careers building their hand-coding skillsets. Is it even possible to build fully functional enterprise apps with little more than drag-and-drop logic?

Such skepticism, while understandable, is unwarranted. In fact, modern low-code derives from a rich history of innovations dating back to the early days of programmable computers. When we take this historical context into account, it's clear that low-code is—and has always been—an important component of the modern software development landscape.



The five historical threads of low-code



When we say low-code, we mean various things, as its history illustrates. Here are five threads from the past we can tie together to understand modern low-code tools.

Thread #1: Business orientation

In the late 1950s, Grace Hopper and others realized that the computer programming of the day—object and assembler code—was too slow, technical, and hardware-specific to have wide practical application.

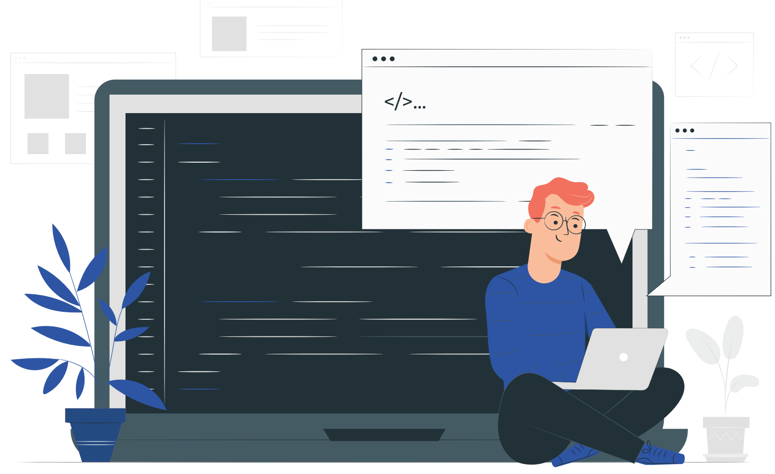
Hopper figured the world needed a business-oriented programming language—one that a broader population of programmers could learn and use. Her innovation: the Common Business-Oriented Language, or COBOL.

The "business-oriented" part of its name means that lines of COBOL read much like English—common across today's programming languages, but revolutionary in its day.

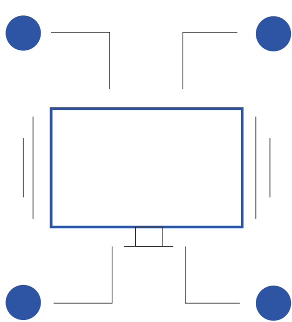
As client/server architectures came to the fore in the 1980s, programmers could place a greater emphasis on user interfaces, as the green screens of the mainframe era lacked sufficient usability.

These requirements led to the development of fourth-generation languages (4GLs), which emphasized the behavior of "fat client" user interfaces more so than earlier generations of programming approaches.

Today, low-code inherits the business-centric focus of COBOL as well as the user interface focus of 4GLs—fundamental requirements for modern application construction.



Thread #2: Declarative representations



All programming languages of the day (including COBOL and the 4GLs) followed an imperative programming model, meaning the computer read through each program one line at a time, following the instructions line by line.

The declarative programming approach contrasts with the imperative model in that the programmer describes the behavior they are looking for, and the underlying platform interprets that description behind the scenes.

Dating from the 1980s, the first wildly successful declarative programming language was the Structured Query Language (SQL). Similar to COBOL, SQL enabled the programmer to write English-like sentences (SELECT * FROM CUSTOMERS, for example) that described the information the programmer was looking for, leaving the specifics of how to return the desired result set to the underlying platform.

In the 1990s, the Hypertext Markup Language (HTML) was also a popular declarative language. Once again, the programmer used the language to describe how they wanted a webpage to look, and the browser automatically took care of rendering the page.

Declarative languages like SQL and HTML provide a remarkable combination of power and simplicity to programmers—a combination that today's low-code tools inherit. However, such languages also had their shortcomings: in particular, a limited ability to create arbitrary programming logic.

To address this need, the relational database management systems that executed SQL added stored procedure functionality, and the browsers that rendered HTML added JavaScript.

This pattern survives in the low-code world, and is, in fact, the reason low-code is "low;" there are always some situations where hand coding is the best approach for solving certain programming problems.

Thread #3: Rapid application development

Combine the text-based declarative approaches of SQL and HTML with the visual sensibilities of 4GLs, and the result is a new way of building applications more quickly. We called this generation of application construction tooling from the 1990s rapid application development (RAD).

RAD tools added declarative techniques to 4GLs, replacing some of the need to write code by hand by enabling programmers to create business logic by checking boxes and selecting values off of lists.

As long as the RAD platform boiled down all possible programmatic constructs to a list of items in a menu, the programmer could assemble a new application in minutes, instead of the weeks that traditional programming would take.

RAD was useful for streamlining easier tasks but wasn't able to simplify more difficult ones. More so, RAD tools typically yielded inflexible code that didn't play well in a business environment.

Thread #4: Website builders

Coming up with a RAD tool that simplified general programming proved unworkable, but building a tool that would generate HTML was much simpler.

As the world wide web came to the fore, so too did a generation of webpage building technologies. These tools focused on offering visual programming environments that would generate working websites—essentially, what modern low-code tools do today except limited to browser environments.

In many ways, these tools combined the innovations of all previous threads—the business -centric focus of COBOL, the visual environments of 4GL, the flexibility of declarative approaches, and the rapid development of RAD.

The world of the website builders, however, was the browser – thus limiting their scope and their broad applicability in the enterprise.

Thread #5: Model-driven development

To make the leap from building websites to general purpose applications, low-code platforms required a more sophisticated approach to representing arbitrary functionality.

The final missing piece of this puzzle was model-driven development (MDD). The fundamental idea of MDD was to represent the functionality of an application as a model—either a visual or metadata representation that the MDD tool could automatically render into running code.

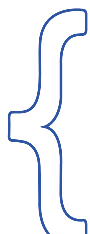
Low-code platforms were a successful marriage of the powerful, but overly technical MDD approach and the flexible, but limited website builder technologies. With the addition of MDD, application creators could model process and data flows as easily as modeling webpages, bringing the necessary enterprise context to such platforms.

Low-code today

Modern low-code platforms, therefore, are business-oriented, declarative, and model-driven rapid application construction platforms that use visual metaphors to both create user interfaces and the underlying programming logic that supports them.

All five threads outlined above are essential precursors for modern low-code platforms like Zoho Creator. The proof is in the execution.

True, low-code accelerates and simplifies application creation, bringing it to a wider range of individuals within the organization. But it also supports greater collaboration across business stakeholders and application builders, while fostering rapid, iterative approaches to building software.



British insurance agency "Horizon" (not its real name) leveraged model-driven development for rapid development and collaboration with the business.

Horizon leveraged a popular low-code tool to build their next-generation insurance platform.

The company found the benefits of the model-driven approach quite valuable, as it allowed its application builders to visually show the business what was going on quickly. In one or two days they could show the business what they were requesting.

Horizon found that many model-driven tools focused primarily on building user interfaces. However, Horizon also took a model-driven approach to the integration code behind the scenes, even for complicated applications that interfaced with third-party pricing and policy engines.

The model-driven approach opened up enterprise application development to a larger population of users, including tech-savvy business people with no prior programming experience.

What, then, is the future of low-code? It's always difficult to predict the future, but here are three trends that are likely to continue.

Trend #1: Low-code/no-code/pro-code convergence

The flip side of low-code is no-code. Low-code primarily targets professional developers, simplifying their work by taking "plumbing" tasks off their plate, while also facilitating collaboration with business stakeholders.

No-code, in contrast, is for business users, aka "citizen developers," who can build rudimentary applications with nothing but the wizards and drag-and-drop capabilities of the tools, with no coding more complicated than Excel formulas.

Recently, low-code vendors have been simplifying their interfaces while reducing the number of situations where hand coding would be necessary. Many such low-code tools thus became no-code in practice.

This convergence between low-code and no-code tools has impacted the hand coding, or "pro-code" corner of the market as well.

Modern pro-code tools shift the focus of the professional developer away from coding to a greater emphasis on engineering and architecture.

In other words, these pros spend more of their time on what it means to build a "good" application, where good means high quality, performant, and generally in alignment with business needs, including the need for greater agility.

Global nonprofit, "Peaceful" (not its real name), reworked several JavaScript apps without adding professional developers.

Global nonprofit Peaceful had many application development priorities but—as with all nonprofits—it had a tight budget.

The most urgent application for Peaceful was an invoice workflow app for its participating churches. Peaceful required churches to sign off on invoices and then send them to headquarters to get them paid.

Complicating matters was the fact that Peaceful employees, who were scattered over hundreds of churches and other facilities, were using corporate credit cards.

The team had already been working on an invoice app for SharePoint they had written in Angular, a popular JavaScript framework. However, this Angular app didn't have the workflow the team required.

Peaceful took a low-code approach that didn't require a complete rework of the Angular app. Their application builders took 70% of the code and turned it into a credit card approval system using low-code.

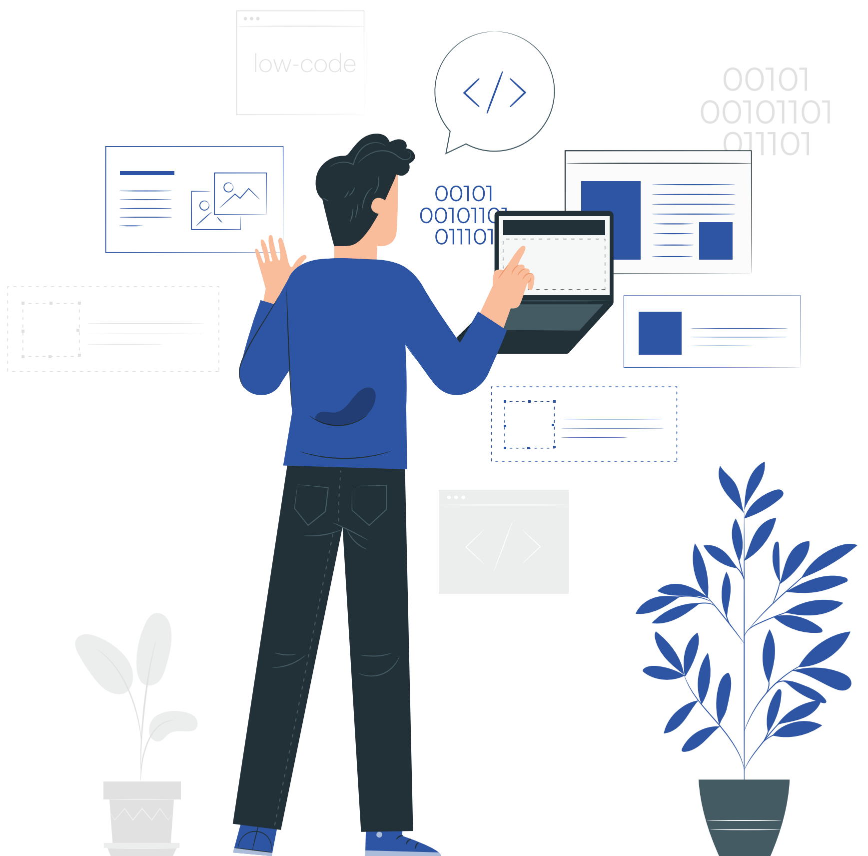
Trend #2:

Low-code everything: low-code becomes an adjective

This distinction between low-code, no-code, and pro-code is rapidly fading away. Instead of a separate market category, low-code is becoming a set of capabilities that products in other categories exhibit.

Robotic process automation (RPA), for example, is adding low-code capabilities. We're also seeing low-code data analysis tools, low-code business intelligence products, and the like. In fact, any tool that hitherto required hand coding for something can now benefit from low-code instead.

Furthermore, any application category whose products typically require customization can also benefit from low-code—think enterprise resource planning (ERP), service management, and other applications that typically require extensive customization to meet the diverse needs of various enterprises.



Trend #3: Intent-based programming

Regardless of the terminology or sophistication of the tooling, creating applications can still be hard work requiring a certain level of expertise. What we really want is to be able to express our intent for the application and let the software take it from there—an approach we might call intent-based programming.

Unfortunately, today's AI can't create an algorithm that satisfies a human's intent in any but the simplest cases. What we do have is AI that can divine insights from patterns in large data sets. If we can boil down algorithms into such data sets, then we can make some headway.

In spite of these advancements, humans will still be creating applications for the foreseeable future, albeit increasingly with sophisticated low-code tools.

The Intellyx take

Given the deeply intertwined historical threads leading to modern low-code, it's no wonder that the entire market is becoming mainstream. In fact, it could be argued that low-code is soon to become the predominant approach for building business applications, with hand coding falling to second place.

We may not quite be there yet, but there's no question that two years of a pandemic have shifted the scales toward low-code. Covid accelerated digital transformation initiatives—sometimes dramatically—and there was simply no other way to meet such rapidly changing needs other than with low-code.

For low-code to achieve a broad-based level of success, however, there must be a change in attitudes across the industry.

Developers must realize that low-code will make their lives easier while shifting their efforts to more valuable work. Business stakeholders should realize that low-code will give them increased influence over the software development process, thus leading to applications that better meet shifting business needs.

Low-code also opens up new opportunities for creating customer value, hence positioning the technology as a strategic enabler of digital transformation priorities. Such opportunities would never have arisen without the five historical threads that led us to this moment.

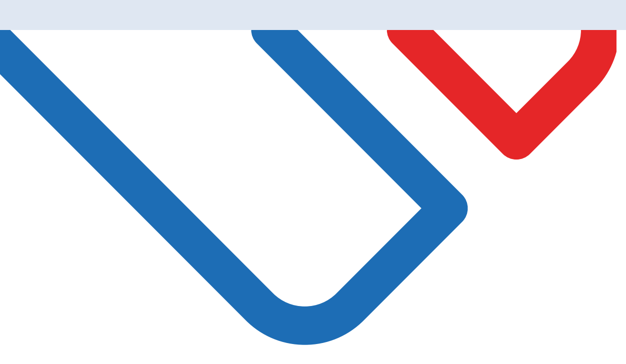
Copyright © Intellyx LLC. Zoho is an Intellyx customer. Intellyx retains final editorial control of this paper.

About the Author: Jason Bloomberg

Jason Bloomberg is a leading IT industry analyst, author, keynote speaker, and globally recognized expert on multiple disruptive trends in enterprise technology and digital transformation.

He is founder and president of Digital Transformation analyst firm Intellyx. He is a leading social amplifier in Onalytica's **Who's Who in Cloud?** For 2022, and he is ranked among the top nine low-code analysts on the **Influencer50 Low-Code50 Study** for 2019, #5 on Onalytica's **list of top Digital Transformation influencers for 2018**, and #15 on Jax's **list of top DevOps influencers** for 2017.

Mr. Bloomberg is the author or coauthor of five books, including **Low-Code for Dummies**, published in October 2019.



About Zoho Creator

Running a business is no easy feat, but we believe we can help.
Rapidly build custom applications that are a perfect fit for your business,
or choose from and modify our extensive range of pre-built apps. The best part?
You don't have to be a programmer. Just sign up, pick a plan, and start building!

zoho.com/creator

We'd love to talk! Reach out to us:
hello@zohocreator.com