Katherine Kostereva | Burley Kawasaki

NO-CODE PLAYBOOK

DESIGN

GO-LIVE

EVERYDAY DELIVERY

Your ability to compete, thrive, and grow depends increasingly on software innovation. Your business depends on it. Your customers demand it. Your employees embrace it. If you don't find innovative new ways to leverage software to enable your business processes, you're at a significant competitive disadvantage against those who will.

Katherine
KOSTEREVA

CEO, CREATIO

As a CEO of Creatio, Katherine is passionate about the freedom to automate any business idea in minutes. Katherine has been named one of the Top 25 SaaS Influencers, Top 50 SaaS CEOs and Top 50 Women Leaders in SaaS in 2018-2021.

Burley
KAWASAKI

FOUNDER, TACHYON SOLUTIONS

Burley brings proven leadership experience in product strategy and R&D in the areas of no-code, low-code, cloud, application platforms and enterprise solutions. Burley was named one of the Top 25 Software Product Executives.

Creatio

NO-CODE PLAYBOOK

The No-code Playbook is a vendor-agnostic guide that empowers teams to deliver business applications of any complexity with no-code.

The Playbook provides guidance on how to organize efficient IT and business collaboration and deliver game-changing results leveraging the potential of the no-code approach while staying compliant with governance requirements.

2022

Content

INTRODUCTION TO NO-CODE

Chapter 1 – Introduction to the Playbook 6

Chapter 2 – What is No-code 12

Chapter 3 – Principles of No-code Development 24

Chapter 4 – No-code Roles and Responsibilities 32

Chapter 5 – Application Matrix 44

DESIGN

Chapter 6 – Introduction to Design 56

Chapter 7 – Stage 1: Business Use Case 66

Chapter 8 – Stage 2: Options Analysis 80

Chapter 9 – Stage 3: Design and Prototyping 90

Chapter 10 – Stage 4: Project Assignment 102

GO-LIVE

Chapter 11 – Introduction to Go-Live 110

Chapter 12 – Stage 5: Prototype-to-MVP 118

Chapter 13 – Stage 6: Feedback Loop 128

Chapter 14 – Stage 7: Governance Checks 136

Chapter 15 – Stage 8: First Release 146

EVERYDAY DELIVERY

Chapter 16 – Introduction to Everyday Delivery 154

Chapter 17 – Stage 9: Feedback Collection 162

Chapter 18 – Stage 10: Incremental Improvements 170

Chapter 19 – Stage 11: Everyday Delivery 176

Chapter 20 – Stage 12: Application Audit 182

FINAL TOUCHES

Chapter 21 – Building Your Center of Excellence 190

Chapter 22 – Making No-code Your Strategy 202

Acknowledgements 210

“Change is the only constant in life”

Heraclitus

Introduction to the Playbook

01

Introduction

The advent of digital technologies has vastly increased the pace of change in our business environment. New business models and competitive strategies that once took years or decades to emerge are now being conceived and launched in weeks or months. Put simply — your ability to compete, thrive, and grow increasingly depends on keeping pace with the latest digital innovations. Your business depends on it. Your employees embrace it. Your customers demand it. If you don't find innovative ways to adopt new digital solutions to enable your business processes, you're at a significant competitive disadvantage against those who will.

Yet this highlights a growing challenge — the supply of software talent is not unlimited.

The gap between supply and demand of software developers is increasing. According to analyst firm IDC[1], the overall developer population was 26.2 million in 2020, roughly half of whom were full-time developers. This may seem like a lot, but businesses of all sizes now want to use software as a competitive advantage, leading to an explosion in the demand for new digital apps. As a result, the supply of developers is being vastly outstripped by the market demand for new software applications. This puts enormous pressure on IT, which is hard-pressed to keep up with the backlog of requested projects for new business applications.

Even for those businesses lucky enough to already have access to larger teams of software developers in-house, it's increasingly difficult to retain top talent. The lure of "work-from-anywhere" opportunities and increasing competition for scarce software roles have contributed to the "Great Resignation," with millions of developers leaving their jobs to seek new employment options elsewhere. Professional software developers are increasingly finding new careers and career models altogether (including freelancing and gig economy marketplaces) that make it even more difficult for traditional enterprise businesses to compete for software talent.

Given the developer talent shortages, many businesses now worry about the impact on their ability to innovate. In a survey of more than 1,000 C-suite executives, 61% said they believed access to developer talent is a threat to the success of their business — an even bigger threat than lack of access to capital, according to the research conducted by Harris Poll on behalf of Stripe[2]. Furthermore, they found that developers on average spend 42% of their time simply maintaining and debugging existing code, further reducing their ability to put energy into sources of new software innovation or differentiation. This all results in immense pressure on businesses to find and retain software developer talent.

[1] Worldwide Developer Census, 2020: Infrastructure Developers Dominate Part-Time Developer Segment

[2] The Developer Coefficient: Software engineering efficiency and its $3 trillion impact on global GDP

At the same time, business teams are now discovering they don't need to wait weeks and months (or perhaps years) for their development requests to be addressed by IT. They are adopting a "Do-It-Yourself" approach to software development and speeding up application development by building apps themselves using no-code. By democratizing the ability to develop software using visual and intuitive "drag-and-drop" tools, no-code enables a new range of nondeveloper roles within an organization to take on the building of software applications. While IT and software developers are still critically important for certain types of apps, the use of no-code has now significantly increased the talent pool potential inside most organizations by allowing employees within the business itself to take on or assist in development tasks.

About This Book

As you might imagine, the business world has responded positively and rapidly to the powerful "no-code" industry shift — it's the answer to much pent-up demand by business functions for new and alternative solutions for building software applications. However, as with any shift, it can be challenging to embrace it efficiently if you don't know exactly how or where to start. That's where this No-code Playbook comes in!

What is the Playbook about?

This Playbook provides practical, no-nonsense guidance for developing enterprise business applications. While this book may interest readers building other types of applications, most of the considerations and practices are optimized for the needs of enterprise app development. It is a hands-on guide intended for anyone embarking on or navigating through a no-code journey. It's an overview of the end-to-end lifecycle for those who want to learn tips and tricks for success.

For whom did we write this book?

It is written as a guide for business and IT stakeholders who are looking for real-world guidance in building no-code enterprise applications and scaling the no-code approach in-house. The playbook is especially targeted at IT/digital and operations executives (VP and C-level leaders) as well as practitioners delivering automation projects.

How is the Playbook structured?

The Playbook is structured in three parts for ease of reading and understanding:

The Introduction Chapters

(Chapters 1-5) initiate readers into the foundational concepts used throughout the Playbook. They are important to read as a primer, and we advise starting here. Even if you have some exposure to no-code, it's worth skimming to make sure you haven't missed some important principles.

The No-code Lifecycle Chapters

are organized around the key phases of a no-code project. First, the Design Phase (Chapters 6-10) covers planning for your project; followed by the Go-Live Phase (Chapters 11-15), which addresses building and releasing your initial app. Finally, the Everyday Delivery Phase (Chapters 16-20) covers the process of ongoing enhancements and evolution of your no-code app.

The Concluding Chapters

(Chapters 21-22) close with more advanced topics and tie together our overall discussion of no-code from throughout the Playbook.

Final Takeaways

Software development is in the midst of the no-code revolution, democratizing the process for knowledge workers and, for the first time, allowing nondevelopers to build apps. However, if you're new to no-code development, it can seem intimidating, and you may not know where to start. This Playbook provides all the guidance and practices you will need to get off to a fast start and successfully deliver on your project outcomes.

However, the no-code space is still new and sometimes overhyped. Understanding what it really is — and what it isn't — is key to realizing the expected benefits and successfully starting your journey. We'll discuss what it is in the next chapter.

“Software is eating the world”

Marc Andreessen

What is No-code

02

Introduction

When entrepreneur and investor Marc Andreessen famously said, "software is eating the world," he was referring to how certain emerging innovators like Netflix, Amazon, Pixar, Spotify, and Pandora were quickly transforming whole industries. These disruptive software companies were leveraging the power of software to completely rewire business models, establish new rules for competitive landscapes, and fundamentally change how customers think about consuming goods and services. Think about the societal impact of on-demand video streaming, same-day/next-day package delivery, or anywhere/anytime music streaming. These software pioneers were disrupting and forever changing customer expectations about daily life.

Yet, when we look at today's business landscape, the impact of software has expanded far beyond Andreessen's earliest predictions. Now, it's recognized that software isn't restricted to a few special disruptive startups — with many firmly believing that every business is a software business. This democratized view is embraced by businesses of all sizes and across all industries and sectors. Organizations, big and small, are using software to enable sweeping digital transformation initiatives. They've been able to put software at the heart of their business strategy because of dramatic advances in cloud computing, which has significantly lowered the barriers and costs of adopting software-enabled innovation.

The cloud has enabled an explosion of applications and services, with some predictions estimating that there will soon be as many digital apps and services created with cloud-native approaches as developed in the last 40 years combined. The majority of these new apps will address industry-specific digital transformation scenarios and result in new competitive requirements being defined. This increased appetite for apps takes IT well past its breaking point — it's just not possible to simply "work harder" and expect IT to deliver independently on these many requests for apps. So, it's no surprise that the trend toward DIY software development (as discussed in the last chapter) has emerged in response to these demands.

The Journey Toward No-code

In many ways, the trend toward DIY software development began as part of the rapid expansion of Software as a Service (SaaS) over the past decade. The popularity of SaaS is clear within the business function as a way to realize faster access and use of applications. Unlike traditional custom software development, with SaaS, you can instantly and immediately provision software on-demand. Furthermore, it removes the complexity and risk of hosting, managing, and updating your software. That complexity is still there, but it's now the responsibility of the SaaS provider to take care of these details and ensure the continued reliability of the app. For many businesses, this value proposition is compelling and has led to the rapid explosion of SaaS to address the needs of nearly every business process or industry vertical.

However, while compelling in terms of speed, availability, and maintainability, packaged SaaS may not always be the appropriate answer as your business may not be satisfied with shrink-wrapped, "off-the-shelf" solutions. To compete, the business function often finds that it needs more highly customized and tailored business processes. In those instances, a SaaS application may be insufficient (just as everyone may not want to live in a cookie-cutter house identical to their neighbors). Every business that deploys SaaS apps will look pretty much the same to their customers, so businesses that are seeking to differentiate themselves are usually focused on finding new sources of technology innovation.

Think of no-code in many ways as offering the "best of both worlds" between SaaS and custom software development. It provides a balanced approach offering the speed and simplicity of SaaS applications but with the ability to customize with many of the same benefits of custom-built software.

Defining No-code

So, what exactly is no-code? Simply put, no-code platforms allow nondevelopers to participate in the application development process through visual drag-and-drop tools. Users can visually compose the forms, workflows, and data needed to build an application without understanding a programming language or having formal software development training. This has the potential to vastly expand the supply of talent by enabling millions of nondevelopers with the ability to address application backlogs. It still requires deep knowledge of the business process or domain as well as the ability to think analytically and logically about problem-solving, but no-code development does not require formal training in software development.

Now, we should acknowledge that it's a bit of a misnomer to say there's no code — a lot of code had to be written to build the no-code platform! However, it's the responsibility of the no-code platform vendor to write and maintain this code and to keep up with the latest trends in innovation. This is much like the responsibility of the SaaS provider to maintain all of the software they host and manage for you "as a service."

Finally, as we define no-code, it's important to note that we will focus on using no-code for developing enterprise applications. While the techniques of no-code abstractions can apply to a wide variety of areas — including building marketing websites, setting up e-commerce sites, defining business intelligence (BI)/business analytics dashboards, or training machine learning (ML) models — those types of solutions are not the focus of this Playbook. We've focused instead on the use and benefits that no-code provides to meet the application backlog needs of the enterprise. This addresses many of the most common enterprise use cases ranging from back-office applications, customer-facing applications, workflow apps, digital forms, employee collaboration apps, and more.

Benefits of No-code

So, let's dive into some of the benefits of no-code. As it turns out, there are many reasons to be excited about its use:

Faster to start

As noted earlier, by taking ownership of parts or all of the development, the business can avoid some of the usual delays when waiting for IT development resources to be assigned. This allows projects to be started more quickly.

Faster to finish

Once you've started your project, the productivity advantage of visual assembly also results in faster development cycles. In a recent study, some 71% of respondents cited faster app development as one of the major reasons for choosing no-code tools[1].

Improved alignment

It's common for IT to spend months or years working on a development project without realizing that the goals or market conditions may have changed. Shorter development cycles and visual language of no-code development enable easier and continuous collaboration and alignment between the business function and IT.

Increased agility

The business function can also respond to change more readily, whether it be to react to new competition or capture new opportunities in the market. Furthermore, the business function can also pivot its strategy quickly because of the flexibility no-code development provides for ideating innovations. You will have the agility to rapidly introduce new strategies, new products, or new services.

[1] No Code Census 2020

Debunking the No-code Myths

However, while the business value is real, the benefits can be overhyped similar to any emerging innovation — and it can be difficult to separate fact from fiction. So, let's explore some of the myths and discuss what's real and what's not.

MYTH #1:
NO-CODE IS MERELY VENDOR HYPE AND ISN'T NEW

Like most myths, this is grounded in some reality. The idea of using visual tools for software development has indeed been around for a long time. The principles behind visual computer-aided software engineering (CASE) tools were first explored as early as the 1970s. However, while the principles of visual drag-and-drop software assembly have been around for a while, the vast majority of early attempts still involved relatively specialized and complex development and architectural knowledge to set up, maintain, and use these toolsets — and they were fairly complex to apply in practice. This all contributed to low usage, except for a brave set of early adopters.

As a result, business users resorted to applying other "homegrown" tools, such as spreadsheet macros, scripting languages, or Microsoft Access databases. While these apps were relatively easier to build, they nearly always began to show performance issues, expose security flaws, or were difficult to maintain as they became more broadly used. Only recently (since the mid-2000s) has the combination of ubiquitous cloud computing, advances in software platforms, and broad adoption of modern application programming interface (API) protocols begun to truly address many of the historical challenges of software engineering in a manner that is enterprise ready. So, while the promise of no-code may have been around for decades, the act of making it simple, easy, and scalable to enterprise needs has been a relatively recent development.

MYTH #2:
NO-CODE IS GOING TO PUT SOFTWARE DEVELOPERS OUT OF WORK AND REDUCE SOFTWARE JOBS

This is demonstrably untrue. If you're a software developer reading this, don't worry — your job is safe! To help keep pace with the increasing appetite for new digital apps, it's essential to include the business function and nontraditional developers in the development process. Given the projected explosion of cloud-native digital apps and services mentioned previously, it will take all of the DIY developers (and more) to just help keep up with meeting this insatiable app demand.

Also, there are plenty of app scenarios that will still require software developers to work with no-code teams (as part of fusion teams that will be outlined in more detail in Chapter 4) because the innovation around new software development languages and frameworks will not slow down.

Software developers will always be needed to push the boundaries of innovation because they are the ones who explore advancements and create the development frameworks or libraries before they are released as standardized components of no-code tools. Typically, there is an adoption and maturation cycle, where development approaches get tested and matured first by software developers until market demand is established. At that point, they will usually be offered as mainstream, prebuilt no-code components.

MYTH #3: LOW-CODE AND NO-CODE ARE SIMPLY VARIATIONS OF THE SAME APPROACH

This is untrue, but many articles often mention the terms "low-code/no-code" in a single breath, leading readers to believe they are simply different flavors of the same approach. However, while they may both use visual abstractions to reduce the complexity of software development, low-code and no-code are two distinct platforms offering different benefits and are designed for different users.

Low-code platforms have evolved from previous concepts of "Rapid Application Development," and they don't attempt to replace programming fully. Instead, they lower the amount of code that must be written (hence the name). Typically, this means that the user of low-code is still a developer of some type, albeit perhaps a more junior one. One of the major benefits of low-code is the ability to make less experienced developers productive easily — they can quickly adopt a standard framework and tools that avoid many of the common pitfalls or complexities that one can face with professional development. However, at the core, there is an inherent learning curve and complexity to low-code, as it still essentially offers a value proposition to IT that enables the development of custom software. Knowledge of proper application design and architecture and some lightweight coding knowledge (typically, JavaScript or some scripting language) is needed to build a low-code app.

No-code platforms, in contrast, are intended to be used by nondevelopers, which means that they attempt to fully remove the need for coding when building apps. No-code users typically sit outside of IT, usually inside a business unit or functional team, which will be explored in greater depth in Chapter 4. As such, the benefit of the no-code platform is that it speaks the language of the nondeveloper — typically focusing on allowing the no-code creator to model the industry domain (e.g., objects or rules) and business process (e.g., forms, workflows) using familiar terms and visual models. There can be a learning curve with the tools — as there is with any new software — but users do not need to learn any scripting or programming languages to finish building the application.

Given the differences in target users and the expectations of programming knowledge, this means that platforms tend to focus on either low-code or no-code, but not both.

MYTH #4: NO-CODE WILL GET OUT OF CONTROL (PROMOTING SHADOW IT) AND SHOULD BE STOPPED

This is very untrue. However, given the software development shortage and a significant backlog of apps unmet by IT, it's understandable that IT would worry that the business function may charge ahead with building apps that may be completely insecure, noncompliant, unreliable, and risk data loss or leaks. It's also a common fear that the explosion of apps built with no-code platforms may result in inconsistencies and wasted investments and create a nightmare to update and maintain.

However, as the saying goes — "if you can't beat them, join them" — and this is one fight that IT will likely lose if it tries to police the use of innovation. Rather than fighting the business function of building new apps, no-code platforms offer solutions for building apps while simultaneously implementing controls and governance to ensure proper use. Modern generations of no-code platforms offer the full range of governance and reporting capabilities needed to ensure that apps will have the ability to be monitored for compliance, security, and maintainability. By giving a standard set of tools for building apps that are business-friendly, it encourages the use of a standard platform that is "blessed" by IT and that can be consistently governed.

So, this myth would be true if your use of no-code is not properly managed, but we'll provide the right no-code methodology steps as well as checks and balances to ensure proper governance, which will be covered in more depth in later chapters. This whole Playbook, in fact, is aimed at enabling the enterprise to embed secure no-code development processes into their standard practices and governance models and to help IT and the business function collaborate more effectively together.

MYTH #5: NO-CODE IS ONLY FOR SIMPLISTIC APPS, YOU NEED SOFTWARE DEVELOPERS FOR MISSION-CRITICAL APPS

No-code platforms indeed result in lowering the cost of building apps, which means that many lightweight app scenarios, especially ones that might not have been previously considered because of scarce development talent, can now be justified. It's also true that the speed and lower costs of app building can result in greater exploration or ideation activities. This often results in building opportunistic apps to experiment with new ideas. These apps may start out simple to test viability and business value quickly.

However, not all mission-critical scenarios demand professional developers. An application's level of business criticality has more to do with the selection of the business process and domain. No-code apps can be used to automate business and mission-critical processes. Suppose that the mission-critical app relies on systems of record (and most do). In that case, it may be that software developers are involved in the project to help establish some of the initial APIs and data integrations or to set up some of the more sophisticated components. But this doesn't preclude business stakeholders from building or maintaining significant parts of the app (we will explore more on this idea of combined teams of business and software developers in Chapter 4).

MYTH #6: NO-CODE PROJECTS FOLLOW THE SAME APPROACH AS TRADITIONAL SOFTWARE DEVELOPMENT

While no-code definitely should apply some learnings from traditional software development, such as Agile or DevOps practices, it would be a mistake to simply treat no-code the same as other ways of development. It's important to tailor the development practices to take advantage of the unique strengths of no-code platforms, which intentionally insulate and abstract you from many details that can trip you up with traditional development. Furthermore, the fact that no-code brings nondevelopers more directly into the app-building process also means that you should expect a different set of skill sets and backgrounds to be part of a no-code team.

Tailoring the methodology used for projects is a key first step, and we'll discuss this further in Chapter 6 when we present the No-code Methodology Framework. This no-code methodology is a scalable model that allows no-code technology to be made available to a much larger population of knowledge workers. The framework will give you the confidence to bet on this approach.

MYTH #7:
NO-CODE PROJECTS CAN'T BE COMBINED WITH TRADITIONAL SOFTWARE DEVELOPMENT

There's a myth that no-code projects are "closed" and somehow conflict with the use of traditional software development techniques. Some believe that a no-code approach constrains you to only using no-code tools and, therefore, may not be suitable for more complex types of enterprise applications. However, as outlined above, no-code development can address a wide spectrum of application types, from simple to mission-critical. The fusion team approach, that we will discuss in Chapter 4, allows participants with different skills and from different domains to distribute tasks and collaborate on building and launching apps. Fusion teams include both no-code creators and software developers and represent a very efficient and synergetic way of delivering enterprise-grade applications using no-code.

Final Takeaways

If you don't find innovative ways to leverage software to enable your business processes, you're at a significant competitive disadvantage against those who will.

There is a lot of hype about no-code development in the industry, but the benefits are real. Understanding the reality of no-code is essential to ensuring successful outcomes and return on investment.

Regardless of the hype, the opportunities that no-code development presents are exciting! Before we dive in, though, let's discuss some of the essential principles of no-code development that every practitioner should understand.

"The things best to know are first principles and causes, but these things are perhaps the most difficult for men to grasp, for they are farthest removed from the senses"

Aristotle

Principles of No-code Development

03

We'll cover a lot of ground in this Playbook to make sure you understand no-code strategies, how to organize for success, what methodologies to follow, etc. However, before we get into these details, let's step back and examine the "first principles" of no-code development. First principles are essential and, as observed by Aristotle, they are sometimes the most difficult concepts to grasp. It's easy for the busy reader of this Playbook to get so immersed in the "how" of no-code that we forget to first understand the "why." So, we are going to begin with a discussion of the three core principles of no-code development:

Principle #1

Use no-code to gather the requirements and prototype on the fly.

Principle #2

Everything that can be developed with no-code, should be developed with no-code.

Principle #3

Deliver to end users as fast as you can.

These three principles are the foundation for the rest of the Playbook. The No-code Lifecycle that we'll be detailing from here on out will build upon these core principles. So, it's important to address them first as everything else will extend from there.

Principle #1

USE NO-CODE TO GATHER THE REQUIREMENTS AND PROTOTYPE ON THE FLY

This first principle is about streamlining the upfront stages of the software development lifecycle that precede actual development. In traditional software projects, the requirements collection and Design Phase usually account for a big time and effort investment.

Part of the reason why custom software projects take significant effort and time is because of the sheer amount of documentation that must typically be created (and reviewed) by both business and technical stakeholders, and this work is time-consuming and complex. These documents can often be fairly technical and use specialized notations to support custom development because they are meant to facilitate the translation from the original business intent down into detailed, lower-level concepts (ultimately being translated into lines of code by developers later in the software lifecycle). The sheer volume of documentation can also result in "losing the forest for the trees," making it easy to obscure the overall understanding of the application and overlook important gaps or missed requirements.

Also, the need to produce large amounts of documentation in traditional custom development often introduces costly defects due to translation errors. A typical custom software development lifecycle involves multiple people in each stage. It starts with a business analyst whose job is to meet with the business and document the business requirements. These requirements are then translated by a solution architect to design the overall solution and then translated again by developers to create detailed specifications. Finally, they're translated one last time by testers who develop testing plans to ensure the software aligns with the requirements. There is a risk of introducing errors into the documents at every stage of the translation process. Errors introduced early in the process — while gathering business requirements for example — create exponential waste because each subsequent stage develops materials based on the early error.

No-code development takes a very different approach. In contrast, using the no-code tools to capture the requirements and design is inherently a more efficient and accurate process. Each of the major elements of the functional specification can be described visually, such as capturing the design of user interface (UI) form layouts, business process flows, and business logic. Using no-code tools is a highly efficient way to capture the specification and facilitate a more direct and effective way to review and gather feedback from stakeholders and end users. Business stakeholders do not have to be technology literate — they can be shown working prototypes of the no-code application very early in the lifecycle, making it easy for them to understand, navigate, and provide feedback to the no-code development team. This improves the efficiency of the process and results in higher quality feedback.

Finally, all of the time spent during these no-code activities does not result in "throwaway" documentation. As you use the no-code platform to build your prototype, you are creating both a specification and the working application. This results in greater efficiency and productivity. It also means that as you change and iterate on the underlying model, the specification and app stay synchronized throughout the lifecycle. This is a huge advantage over having to maintain both the specifications and the application code and keep them updated in tandem.

Principle #2

EVERYTHING THAT CAN BE DEVELOPED WITH NO-CODE, SHOULD BE DEVELOPED WITH NO-CODE

The second principle is about minimizing the complexity of your overall solution architecture by adopting a primary architecture of no-code. While there are many options available for designing your overall architecture, pursuing too many options can introduce unnecessary complexity — just because one can choose from many options does not mean that one should! Also, teams can fall often into the behavior of sticking with what they already know. Companies with heavy expertise in software development tend to overuse code and apply it everywhere as it's a learned behavior that is difficult to unlearn. In contrast, the goal of the no-code approach should be to break this cycle and use coding only as needed as a part of the fusion team approach.

Simplicity is good. Sometimes there is a temptation to identify many possible technical alternatives in the solution approach in the quest for completeness. However, understand that every time you introduce more options and custom code, it comes with an inherently higher cost of longer-term maintenance and support. Over time, it becomes more difficult for developers to understand the original design choices. When new developers join, it will take them longer to review the original design specifications and solution components if they're built using multiple programming tools or languages. They'll also need to understand more layers in the overall solution architecture. While this may have seemed advantageous to the trained software architect who initially designed the solution, it can make the overall updating and evolution of the app more costly and complex. There are countless technology solutions that are so complex that only the original developers fully understand the solution. As any company with an aging workforce will tell you, losing the original knowledge of a solution sometimes prevents you from making future changes out of fear of breaking it.

Betting on a no-code approach across your application gives you a more unified and streamlined architecture that will ultimately be simpler and easier to maintain and support. New no-code creators will have a faster onboarding time as they come up to speed on the application. Your no-code team will also be more self-sufficient because they are able to evolve and support the application themselves — they don't need to worry about finding specialized skill sets or depending on IT to provide development support. This ultimately reduces the total cost of ownership and accelerates your ability to change and evolve the application.

Note: this principle should not be taken to an extreme to mean that we are recommending against any use of third-party software or custom-developed components! That is not the case, it's just important to have a clear and rational decision framework for anything that is added to the solution. We'll present such a framework later in Chapter 8 when we discuss the Options Analysis stage of the methodology, which helps guide your selections based on your requirements. We're recommending that you take the simplest approach possible by striving to make no-code the primary underlying architecture.

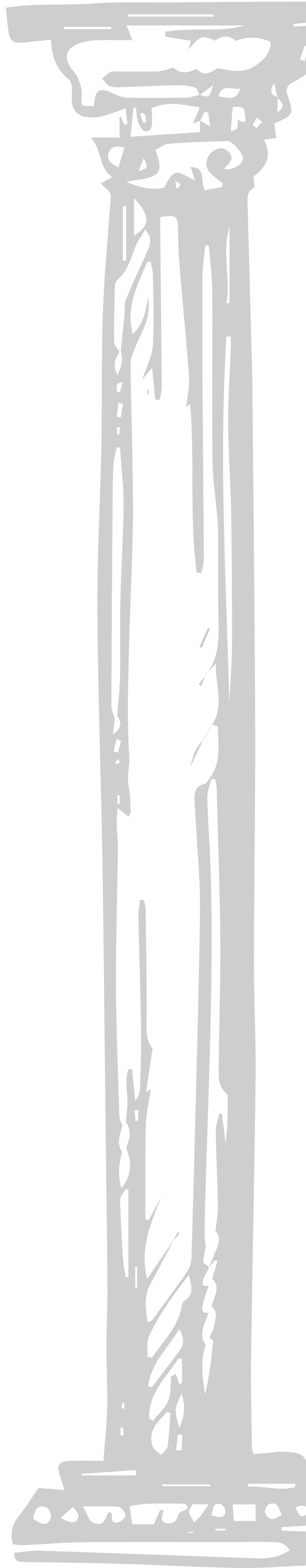
Principle #3

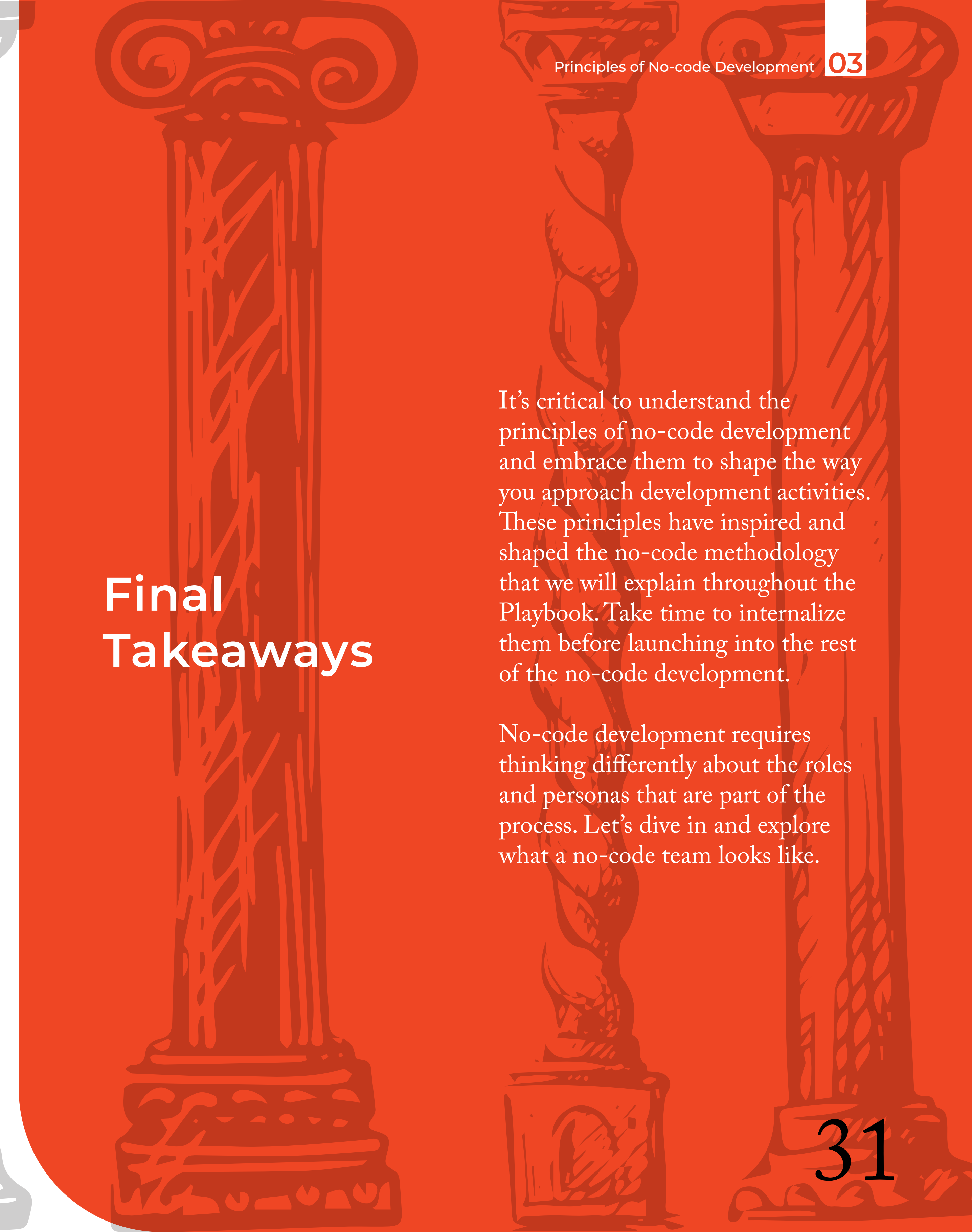
DELIVER TO END USERS AS FAST AS YOU CAN

The third and final principle is about speed — and avoiding the trap of aiming for perfection. Classic software development methodologies have often attempted to maximize end user value by stuffing as many features as possible into the first release. This is partly driven by stakeholders' honest desire to "have it all," but it is sometimes also driven by concerns about the timeliness of getting to the next update. If you're concerned that the next release may take weeks or months, then you'll push hard to sneak all you can into the first release.

In contrast, no-code does not wait for perfection all at once but instead realizes it over time. No-code aims to release features quickly to the end user — even if it is a very tight and minimal solution. The market and competition move quickly, and it's better to release something impactful, relevant, and timely — even if it's a highly-targeted subset of the scope — than it is to delay and try to take on a massive set of functionality all at once. This will be discussed during the Design and Go-Live Phases of the lifecycle.

Also, while it's important to deliver value to the user, it doesn't have to be done all at once. Instead, choose the minimum possible scope that will unlock the business value, then continue to keep adding incremental features to end users quickly in small updates. This will continue to move you closer to the original longer-term strategic vision with the advantage of providing more adaptability (you can evolve and course correct with every incremental step). This approach also usually results in more satisfied and engaged stakeholders as they will see your commitment to rapidly responding to feedback. Ironically, this will make stakeholders better at prioritization, and they will be less anxious about deferring items in the backlog as they build confidence that it will not be a long wait for the next rounds of updates. This approach is referred to as "Everyday Delivery" and will be further outlined in the Playbook.





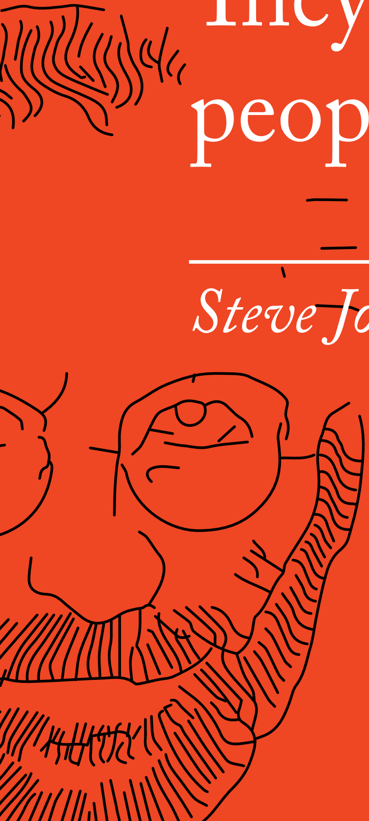
Final Takeaways

It's critical to understand the principles of no-code development and embrace them to shape the way you approach development activities. These principles have inspired and shaped the no-code methodology that we will explain throughout the Playbook. Take time to internalize them before launching into the rest of the no-code development.

No-code development requires thinking differently about the roles and personas that are part of the process. Let's dive in and explore what a no-code team looks like.

“Great things in business are never done by one person. They’re done by a team of people.”

Steve Jobs



04

No-code Roles and Responsibilities

Building software — even building software using a no-code platform — typically requires a diverse set of individuals and skills, all working together to achieve the goal. A single person could perform the smallest no-code projects but, for most applications, it requires assembling a team. Some of these team contributors may be full-time, but a lot may be part-time as people in the business function are often wearing multiple hats. So, it's important to be clear and focused about what roles you need and what their responsibilities will be. This allows you to set clear expectations and secure any needed resources.

There is also a range of skill requirements based on varying levels of application complexity. You might be building a simple app for capturing suggestions or a mission-critical app that the business function depends on for automating its most important processes. Most likely, it will lie somewhere in between. As you look at the different types of apps, you'll face choices for how to organize your team to support different levels of complexity.

We'll explore both topics in this chapter. First, we'll walk through the three primary approaches for organizing no-code teams and discuss what they look like from an organizational perspective. Next, we'll outline the key roles that are part of these teams on a no-code development project, and we'll explore how these "Delivery Models" and roles tie together to support effective and multidisciplinary delivery.

No-code Delivery Models

Let's start by discussing some of the common delivery models for organizing your teams. A delivery model is simply a suggested configuration for a no-code team structure designed to support different levels of complexity and scale when building no-code applications. We will introduce three distinct delivery models in this section. Note, the process of actually picking which delivery model is right for your application should be guided by a decision framework called the "Application Matrix," which will be outlined in the following chapter. For now, we will simply define the models.

DIY delivery

The simplest delivery model is what we'll refer to as the "Do-It-Yourself" approach, where all the primary roles of the no-code project are contained within a team sitting inside a single business unit. This team typically has a single overall project sponsor overseeing them. This makes the business function highly autonomous and in charge of its own destiny as they do not depend on groups in IT or other organizations to form and execute a no-code project. There may be just a single no-code team. There may also be multiple teams in organizations that have a high level of no-code maturity and are building no-code apps for a large number of business units. The size of the team is less important than where the participants come from. In this model, all the roles should come from the same functional business unit that sponsors the app.

Considerations

- This is the simplest model for organization and execution, leading to efficiency and streamlined operations.
- Team roles may be easier to resource and operationalize by the business function because of reduced dependencies.
- Offers a clear definition of accountability and priorities within a single organization unit.
- May be constrained by the availability of skills and people in that group.
- Not having access to more technical development skills may lead to roadblocks.
- It will be challenging to embed a new no-code application into a legacy ecosystem without proper technical skills.

Center of Excellence (CoE)

The next delivery model is the "Center of Excellence," or CoE. The CoE is typically owned and led by a single overall cross-functional CoE leader. It has skilled knowledge workers whose mission is to maximize efficiency through consistent definition and adoption of best practices for no-code development across the organization. This approach typically does not get formed immediately but emerges after the organization has started building some no-code expertise from multiple projects. It then becomes attractive to standardize and centralize some no-code expertise and skills to leverage across teams. When this happens, it's often part of an overall digital transformation initiative. As the CoE is formed, it may or may not have direct full-time staff. Sometimes, the CoE leader can have direct employees working under their supervision or they can operate in a matrix environment working with no-code creators from other business units.

Resourcing a no-code project using the CoE model results in a matrix organizational model. Most of the key roles will still be driven by the business unit, but some specialized no-code roles may sit within the CoE. However, even the CoE resources will typically have a dotted-line reporting relationship with the overall business unit owner as a part of the development project.

Considerations

- Makes the most efficient use of scarce resources across the organization by providing team members with a centralized and shared basis.
- Accelerates learning and adoption of best no-code practices across projects.
- Fosters a higher degree of no-code components reuse across teams within the organization.
- Offers higher complexity of governance because there will be a matrix of shared accountability across both the business unit and the CoE.
- Requires additional budgeting by the organization to fund the centralized team.

Fusion team delivery

The last delivery model we'll introduce is the "fusion team." This delivery model represents a multidisciplinary team with members coming from the business function and IT to collaborate. Typically, this model is used when you have greater technical requirements and complexity that require software developers to build some components of the no-code application. The software developers may still sit somewhere within the business function — perhaps a business function unit within the IT group — or they may be part of a centralized corporate IT function. They may also be tapped to provide expertise in specific technical areas, such as security or DevOps. Regardless of where they sit within the organization, there is shared accountability for the no-code app being built.

The fusion team model usually results in a matrix organizational model. Most of the key roles will still be driven by the business unit, but some specialized no-code roles may sit within an IT group. However, typically software developers who may be supporting the project from IT will have a dotted-line reporting relationship to the overall business unit.

Note: the concept of a fusion team is based on research[1] from the analyst firm Gartner. You can reference more of their research on the topic.

Considerations

- Provides access to more technical development skills required by certain projects.
- Blends technology skills, analytical skills, and business domain expertise.
- Offers shared accountability for the no-code solutions being built.
- Offers higher complexity of governance because there will be a matrix of shared accountability across both the business unit and fusion team.
- Requires additional budgeting by the organization to fund more technical talent from IT.
- Slower and more resource dependent; requires more time to align the IT and business functions.

[1] What Are Fusion Teams, Gartner

No-code Roles and Responsibilities

Regardless of which delivery model is selected, there are a set of defined roles that are typically present in any no-code project. We briefly define each below, outline their major responsibilities, and discuss how they fit into one or more of the delivery models.

Role

No-code stakeholder

The no-code stakeholder is a business function role — often a senior functional leader — who acts as an executive sponsor on behalf of the business unit. Typically, they are chartered with articulating the no-code app business vision and requirements and working with the no-code project team to review and approve business use cases, prototypes, MVPs, changes, etc.
The no-code stakeholder will typically have a good understanding of the business process and have sufficient authority to make decisions related to functionality. Finally, while the stakeholder does not need to be deeply technical, they should understand the technology and its ability to impact the business.

Responsible for:

- Defining and approving the business vision and requirements.
- Providing overall direction and feedback that shapes design activities.
- Providing feedback directly back to the no-code development team during demos.
- Working with the no-code team to set priorities for the backlog of micro use cases.
- Final approver who indicates the app is ready to deploy to production.
- Working with the business function, CoE, and IT teams to procure and assign resources to the no-code project team.

Delivery models:

- Applies to all delivery models.
- In the CoE or fusion team scenarios, the multidisciplinary teams should still have a dotted-line reporting to the no-code stakeholder who is ultimately responsible for the project direction and success.

Role

No-code business architect

The no-code business architect is a business role often filled by a more senior expert. The architect is responsible for setting the most suitable design approach. The architect decides on the right option to deliver the project and the combination of reusable components that will be needed. They also make sure the solution aligns with the business value. They are usually directly involved in the no-code project and own more challenging design or development tasks.

Responsible for:

- Working with the no-code stakeholder on defining requirements and design and ensuring alignment of the options and business needs.
- Providing subject matter and process domain expertise that shapes design activities.
- Participating directly within the no-code development team in the app building process; may take on more challenging components.
- Coordinating with IT and Operations to plan, schedule, and conduct Governance reviews.
- Coordinating with Operations once the app is approved and ready to deploy to production.
- In fusion team delivery scenarios, coordinating with IT software developers to build out custom software components of the no-code application.
- In fusion team delivery scenarios, coordinating with IT to build or validate integrations with other systems and APIs.

Delivery models:

- Applies to the CoE and fusion team models. Might be engaged in the DIY model.
- The no-code business architect typically sits within the business function itself. However, the CoE may also have a number of architects who can provide subject matter expertise in more specialized areas. They may also help the CoE drive overall standards and best practices, such as fostering, harvesting, sharing, and reusing of components across teams.

Role

No-code creator

The no-code creator sits within the business unit and has a strong understanding of the business domain. Importantly, they also possess strong user empathy. They sit within the no-code project team and have been trained in using the no-code tools. They are responsible for the system configuration and quality activities using no-code capabilities.

Responsible for:

- Working with the no-code business architect and no-code stakeholder to contribute to requirements and designs activities.
- Participating directly within the no-code development team to build the app; responsible for owning specific work items (micro use cases) in the backlog and implementing the use of the no-code tools.
- Defining and executing quality assurance (QA) activities to validate end-to-end user scenarios.
- Producing user documentation, training, or enablement related to the no-code application.
- Working with the no-code business architect to develop feedback mechanisms and collect user and system feedback.

Delivery models:

- Applies to all delivery models.
- There will be one or more no-code creators on the team based on the size and scope of the no-code application. For larger applications, there may even be multiple teams of no-code creators who are working on the application in parallel.

Role

No-code CoE leader

The CoE leader is a cross-functional leadership role that helps drive efficient prioritization and management of capacity and people resources leveraged within a shared CoE.

Responsible for:

- Driving the adoption, sharing, and reuse of best practices for no-code across the organization.
- Driving the adoption, sharing, and reuse of components — often from some type of shared internal repository or marketplace — across project teams.
- Supporting the no-code stakeholder in procuring and assigning resources to the no-code project team (especially around specialized skill sets).
- Responsible for recruiting, training, certifying, and enabling no-code creators.
- Responsible for managing the process, applying correct governance policies, and promoting the no-code culture and strategy.
- Helping to manage coordination and conflict resolution across teams.
- Can help coordinate the planning and scheduling of governance reviews.

Delivery models:

- Applies to the CoE and fusion team models.

Role

Software developers/QA

Software developers and QA resources participate (if needed) as part of fusion teams to help build and test more complex technical components (e.g., new architecture services and custom integrations). They have been trained in custom software development tools, skills, and methodologies.

Responsible for:

- Directly supporting the no-code development team to build the app; works with the no-code business architect to own more technically complex components of the overall solution.
- Defining and executing quality assurance activities on their components.

Delivery models:

- Applies to no-code fusion teams.
- These are typically IT roles, either sitting in corporate IT or perhaps business unit IT. They are typically not directly part of the no-code project team, but they should still have a dotted-line reporting to the no-code stakeholder who is ultimately responsible for the project direction and success.

Role

Approvers (IT/Operations)

Approvers participate as needed at certain stages of the lifecycle and they are responsible for advising on team compliance, governance, and security criteria. They also provide final governance checks and approval for Go-Live. They have been trained in governance and provide subject matter expertise to support the no-code project team.

Responsible for:

- Directly supporting the no-code development team in planning and executing governance reviews (typically in coordination with the business architect).
- Directly supporting the no-code app release deployment with the final release activities and post-release monitoring or support.

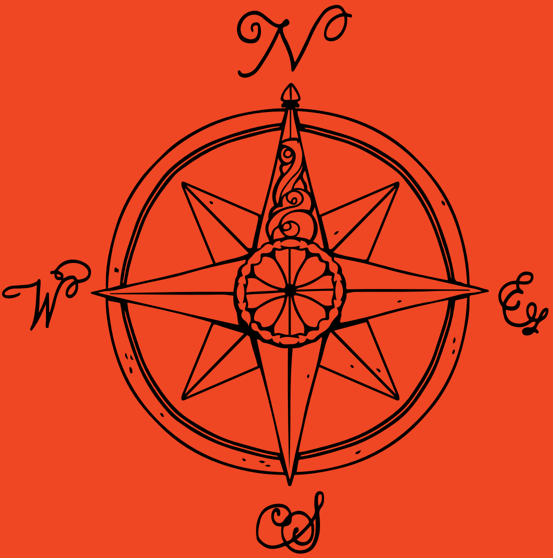
Delivery models:

- Applies to the CoE and fusion team models.
- These are typically IT or Operations roles, usually sitting in a shared corporate capacity — often representing Legal, Compliance, IT, Security, human resources (HR), and other administrative functions. They tend not to be directly part of the no-code project team, but approvers should still have a dotted-line reporting to the no-code stakeholder who is ultimately responsible for the project direction and success.

Final Takeaways

Building your no-code team is critical, so start by identifying the appropriate delivery model that fits best. Next, understand the responsibilities associated with each role so that you can recruit team members best suited for carrying your project forward. While you may be able to use no-code yourself, you're not going to realize the broader outcomes if you do it alone. Put care into your team selection, so your no-code project is set up for success.

Now that we've discussed the various roles and delivery options, which one should you choose? Let's look at a critical decision framework called the Application Matrix that will aid with picking the appropriate delivery model and help scale many key activities within the No-code Lifecycle.



“Set your course by the stars, not by the light of every passing ship”

Omar N. Bradley

Application Matrix

05

There was a time when all sea-traveling captains entrusted their lives to the stars to chart their path. With simply a sextant and a clear view of the night sky, they could sail their way around the globe! Now, of course, we have access to much more advanced technology. We are used to (and, perhaps, dependent on) having phones with global positioning system (GPS) receivers in our pockets, allowing us to adeptly navigate unknown cities or countries confidently. Yet, as advanced as modern GPS is, its approach to finding your exact location — referred to as trilateration — is not so dissimilar from how a sextant was used long ago. Instead of triangulating off stars, a GPS receiver measures data from three orbiting satellites to form an accurate position of its location on the Earth's surface.

In this chapter, we're going to introduce an important framework — the Application Matrix — that will act as a GPS of sorts in charting your course through the No-code Lifecycle. However, instead of anchoring our position based on readings from three satellites, we'll take a reading on the starting point for your no-code application by assessing three complexity dimensions — specifically measuring the Business, Governance, and Technical complexity. The result of using this framework will give you the basis for adapting the use of the No-code Lifecycle to meet your needs. It will help you chart your course — both during the initial release and for the subsequent continual evolution of your no-code application.

Introducing the Application Matrix

It's important to avoid a "one-size-fits-all" mentality regarding no-code. The approach you take to design and build a simple no-code app may not be sufficient to handle the complexity as the size/scale of your app grows. For example, the requirements of an employee feedback app would likely be very different from those of a business-critical solution like invoice management or digital lending. Conversely, if you only applied the approach that works for the largest and most mission-critical projects across all apps, you would overwhelm small project teams with too much process and prevent them from moving with speed and agility.

This means that different types of no-code apps require different approaches to skill sets and methodologies. Therefore, it's important to have a way to customize the methodology to meet the scale of your project needs, and the Application Matrix addresses this need. We're describing it upfront in the Playbook because it will play a key role, and we will be referring to it in various chapters.

The Application Matrix evaluates your no-code project from three different dimensions: Business, Governance, and Technical. We will outline some of the suggested criteria you should assess as a starting point for each dimension. We will also provide a few examples to illustrate the use cases. However, like any framework, it is meant to be customized and tailored to your specific needs. You should ultimately internalize this within your no-code team (or within the No-code CoE) and adapt it to meet your needs precisely.

Dimension

Business complexity

The Business complexity dimension helps assess where your app sits on the spectrum of business process and organizational complexity. It will also assess how standardized the app requirements may be or whether there will be a lot of variations that must be accounted for in the requirements and design of the app. For each of the following criteria, you should assess them on a scale of Simple, Medium, or Advanced.

Suggested assessment criteria:

Process complexity

How complex is the business process? Is the process simple and consistent, or does the process include many different exceptions, steps, rules, and multilevel nested processes?

Business critical use case

How critical is this to business operations? If the app fails, can the business function continue to operate? Will downtime have a significant business impact (lost revenue, etc.)?

Cross-departmental usage

How broadly across the business is the app used? Is it used within a single department/unit or broadly across the business? Is it used by partners or customers outside the business?

Regional requirements

This assesses how much variability the app may have when used by different regional user populations. For example, tax rules or regulations may vary widely by region or geography.

Language requirements

Is this a single-language app, or does it have multilingual requirements? Are there any requirements that will require significant localization of the UX (e.g., adaptation of the flow or layout)?

No-code apps that score Medium or Advanced Business complexity should anticipate a more sophisticated approach to design, along with more complex operational and deployment/enablement requirements. Medium complexity apps may also benefit from support by the No-code CoE. We'll highlight these considerations in later No-code Lifecycle chapters.

Dimension

Governance complexity

The Governance complexity dimension helps assess where your app sits on the spectrum of requirements for compliance with external laws, guidelines, or regulations imposed by external entities or internal audit groups within your organization. It will also assess the complexity of specific security and data governance considerations. For each of the following criteria, you should assess them on a scale of Simple, Medium, or Advanced.

Suggested assessment criteria:

External compliance

This includes generally accepted accounting principles (GAAP), Sarbanes-Oxley Act (SOX), Health Insurance Portability and Accountability Act (HIPAA), and General Data Protection Regulation (GDPR). Does this fall under one or more established laws, guidelines, or regulations imposed by external governments, industries, and organizations?

Internal compliance

Does this fall under the governance of internally enforced checklists, policies, or controls? What are the business risks if this fails to meet an internal audit? How complex are the internal access controls?

Security requirements

How securely is the information being accessed? What is the business risk if the information is leaked or compromised due to internal or external attacks?

Data governance

How is corporate data being managed and secured? Does this contain sensitive, proprietary, or confidential business data? Does it contain customer personally identifiable information?

No-code apps that score Medium or Advanced Governance complexity should anticipate a more sophisticated design and deployment/operational procedures approach. They will also require more advanced planning during the governance review stages. Medium complexity apps may also benefit from support by the No-code CoE. We'll discuss these considerations in later No-code Lifecycle chapters.

Dimension

Technical complexity

The Technical complexity dimension helps assess whether your team may require assistance from professional developers or other specialized technical resources. This also will assess whether the app requires additional options beyond the no-code tools (e.g., third-party components and integration with packaged apps). Assess each of the following criteria on a scale of Simple, Medium, or Advanced.

Suggested assessment criteria:

Coding requirements

Is custom code required for this project? (e.g., for custom controls or extensions outside of the no-code platform).

Complexity of integrations

How complex is it to set up an integration with the system? Do existing connectors exist, or are custom integrations required? How many external packaged apps or add-ons are part of the solution?

UX/UI complexity

How complex is the UX of the app? Is it a simple web form/portal, or does this require more sophisticated omnichannel experiences (e.g., native mobile and voice)?

Scale of user transactions

On average, how many users will the app have? Will this be used infrequently throughout the week, or is it an app that is used daily/hourly?

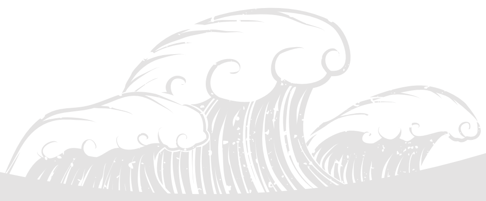
No-code apps that score Medium or Advanced Technical complexity should anticipate a more sophisticated approach to design. Medium complexity apps may also benefit from support by the No-code CoE while Advanced Technical complexity may require a fusion team approach. They will also require more advanced planning during deployment and operations activities. We'll review these considerations in future chapters.

Picking the Delivery Model

Once the aggregate complexity by dimension has been assessed, you will be able to select the appropriate overall delivery model. The three delivery models discussed in the last chapter will be applied again here: DIY delivery, CoE delivery, and fusion team.

Types	Simple	Medium	Advanced
Business complexity	DIY	CoE	CoE
Governance complexity	DIY	CoE	CoE
Technical complexity	DIY	CoE	Fusion team

No-code apps rated Simple across all three complexity dimensions can be readily owned and delivered via the DIY team approach. No-code apps with Medium/Advanced complexity usually require involvement from the No-code Center of Excellence. Finally, the no-code apps with Advanced Technical complexity will likely be delivered using the fusion team approach.



To help solidify these concepts, let's walk through a few examples that put the Application Matrix into action.

Example #1

Team feedback solution

This first no-code app is a feedback-capturing solution built to capture internal feedback and suggestions within the commercial teams of a shipping company. It's a simple app with only a few screens. It replaces the paper forms previously used by the company. The assessment using the Application Matrix is as follows:

Business complexity

Overall Assessment / Delivery Model: Simple – DIY

The Business complexity is rated Simple overall because the process and use case criticality are simple.

Criteria		Requirement
Process scope/complexity	→	HR function
Business critical use case	→	Can be captured manually if needed
Cross-departmental usage	→	Single department (Sales)
Regional requirements	→	No variations – global process
Language requirements	→	Multiple languages (English, Spanish)

Governance complexity

Overall Assessment / Delivery Model: Simple – DIY

It was assessed with Simple Governance complexity because it is an internal app that contains little or no sensitive company data, meaning it is low risk.

Criteria		Requirement
External compliance	→	No external regulations
Internal compliance	→	Low business risk
Security requirements	→	Internally accessed only
Data governance	→	No sensitive information

Technical complexity

Overall Assessment / Delivery Model: Simple – DIY

Technical complexity is also rated Simple given that the app could be built without any custom code or integrations and has simple technical requirements.

Criteria		Requirement
Code development requirements	→	No code needed
Complexity of integrations	→	One integration with standard API
Number of users and transactions	→	100 users
UX/UI complexity	→	Simple web form

Assessment:
This app represents a good example of one well-suited for the DIY delivery model. Its Simple complexity along all three dimensions is definitely within the no-code team's skill sets, even if it is their first no-code project.

Example #2

Field inspection solution

This next no-code app is a field inspection solution built by a manufacturing company for its inspection employees who will be visiting sites and completing checklists to validate completeness and quality. Because of different equipment categories, it will have roughly 15 different workflows depending on the type of equipment.

Business complexity

Overall Assessment / Delivery Model: Medium – No-code CoE

The Business complexity is rated Medium overall because of the complexity of the inspection process, the criticality of the use case, and the need to support regional variations.

Criteria		Requirement
Process scope/complexity	→	Inspection function
Business critical use case	→	Impacts manufacturing processes
Cross-departmental usage	→	Single department (manufacturing)
Regional requirements	→	Some variations (NA, EMEA, and APAC)
Language requirements	→	Single language (English)

Governance complexity

Overall Assessment / Delivery Model: **Medium – No-code CoE**

It was assessed with Medium Governance complexity because it contains essential manufacturing data and sensitive company information representing a medium risk.

Criteria		Requirement
External compliance	→	No external regulations
Internal compliance	→	Medium business risk
Security requirements	→	Internally accessed only
Data governance	→	Sensitive company information

Technical complexity

Overall Assessment / Delivery Model: **Simple – DIY**

Technical complexity is Simple. All needed capabilities can be built using the no-code visual tools and by adding existing components.

Criteria		Requirement
Code development requirements	→	No code needed
Complexity of integrations	→	Three prebuilt integrations
Number of users and transactions	→	250 users
UX/UI complexity	→	Mobile and desktop

Assessment:
This app is well suited for the No-code CoE delivery model. The no-code team would benefit from CoE involvement to provide additional expertise throughout design activities and extra support when the team is ready for governance reviews. The Technical complexity is low, given that there is no need for custom coding of components or integrations.

Example #3

Financial advisor solution

The last example is a financial advisor app used by a regional bank in the United States. It's used inside the branch by bank wealth management advisors, account managers, and other staff to advise their clients on investment strategies and products. This is an advanced solution with complex workflows and the need to be integrated with core banking systems.

Business complexity

Overall Assessment / Delivery Model: Medium – No-code CoE

The Business complexity is rated Medium overall because of the complexity of the financial advisory processes, the criticality of the use case, and the cross-departmental uses.

Criteria		Requirement
Process scope/complexity	→	Wealth management
Business critical use case	→	Impacts customer service
Cross-departmental usage	→	Cross-department
Regional requirements	→	One region (US)
Language requirements	→	Single language (English)

Governance complexity

Overall Assessment / Delivery Model: Advanced – No-code CoE

The Governance complexity is assessed at Advanced because the app manages highly sensitive customer financial data, falls under specific banking industry laws and regulations, and has strict data governance requirements.

Criteria		Requirement
External compliance	→	Federal Deposit Insurance Corporation (FDIC) & other industry regulations
Internal compliance	→	Medium business risk
Security requirements	→	Internally accessed only
Data governance	→	Highly sensitive customer information

Technical complexity

Overall Assessment / Delivery Model: Advanced – Fusion team

Technical complexity is also Advanced because of the complexity of integration with the core banking system. It also has complex omnichannel requirements as it should be deployed across desktop web and tablet form factors within the branch.

Criteria		Requirement
Code development requirements	→	Custom risk assessment logic
Complexity of integrations	→	Custom integration with bank core
Number of users and transactions	→	250-plus users
UX/UI complexity	→	Web and tablet interfaces

Assessment:
This app is well suited for the fusion team multidisciplinary approach. The no-code team would benefit from CoE involvement to provide expertise throughout design activities, and both no-code creators and software developers can work jointly on the configuration process.

Final Takeaways

Scaling your approach to building your no-code app is key to success. You should select the appropriate delivery model, which will help give you the right set of roles and talent for your team. Also, take time to tailor your methodology appropriately — you don't want to underestimate the complexity, which could lead to greater project risks or unnecessary complexity in the project. Let the Application Matrix be the guide to chart your course!

Now that the introductory concepts have been covered, we're about to discuss the methodology that should be followed for your project. Let's begin with the first phase: The Design Phase.

"Alice:
Would you tell me, please, which way I ought to go from here?

The Cheshire Cat:
That depends a good deal on where you want to get to.

Alice:
I don't much care where.

The Cheshire Cat:
Then it doesn't much matter which way you go.

Alice:
... So long as I get somewhere.

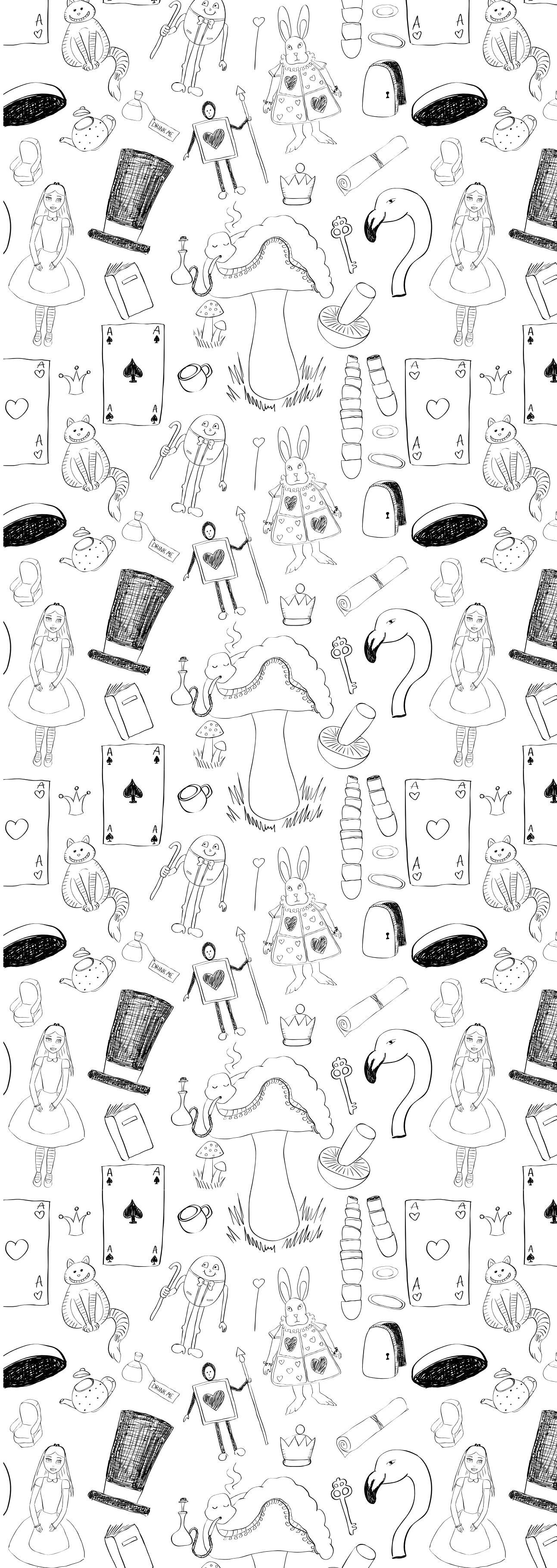
The Cheshire Cat:
Oh, you're sure to do that, if only you walk long enough."

Lewis Carroll,
"Alice's Adventures
in Wonderland"

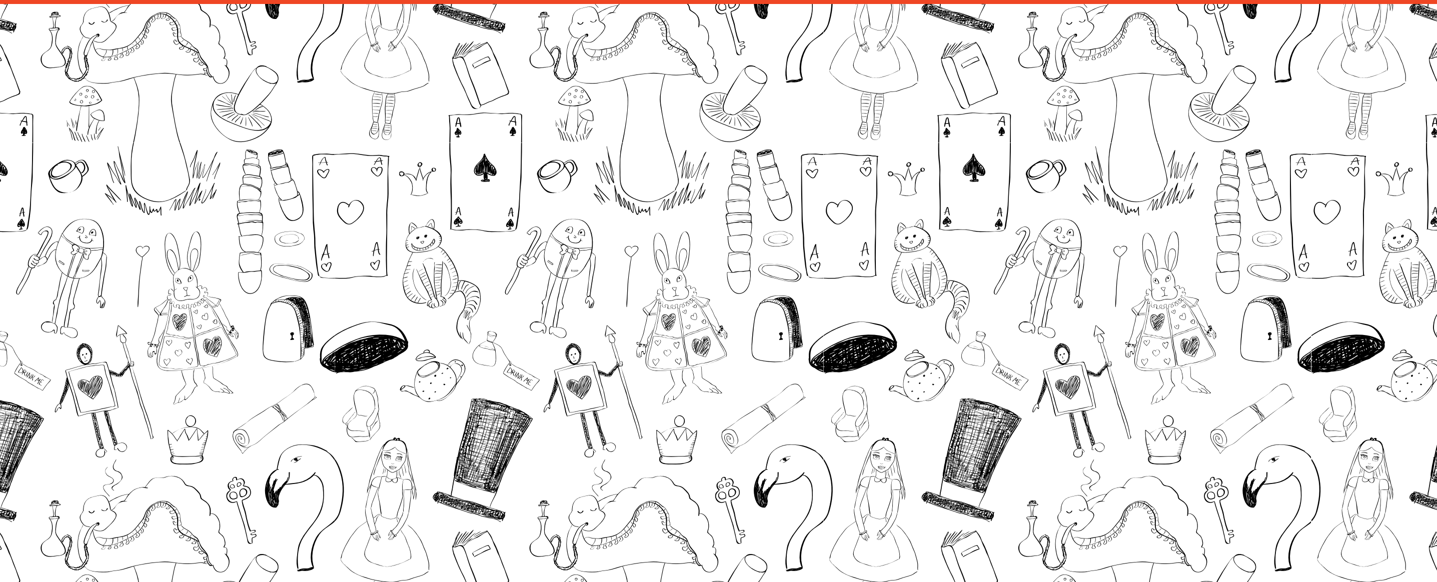
06

Introduction to Design

Much like Alice in the famous fairy tale "Alice's Adventures in Wonderland," you may be tempted to start charging down the "no-code rabbit hole" by immediately building your first app without really having a clear plan of where you're heading. Because it is simple to build no-code apps, it's easy to immediately get immersed in the details of app building and lose sight of the bigger picture. But you will regret this later in your journey when you get bogged down in the complexities of project delivery without a clear view of the essentials needed to ensure successful business outcomes. Or worse, like Alice, find yourself at the wrong destination, and end up with an app that doesn't fulfill your needs.



The Design Phase is critical because it helps define the plan for your application and ensures you are building the app correctly.



No-code Design Differences

You might ask, “Isn’t this just like any other software development project?” It’s true that the concept of application design is not new. Design activities of some fashion have been part of the software lifecycle since computer programming was first invented. However, to make full use of the no-code benefits outlined earlier in the book, it’s important to recognize some fundamental differences in the Design Phase of a no-code project compared to a custom software development.

Traditionally, design activities in custom software development involved a high degree of technical notation and abstraction that required advanced technical skills that most business users don’t possess. This was needed to translate the business requirements into specifications for the technical components and architecture that a programmer used to start writing code. For example, a skilled business analyst might have used Agile “user stories” and specialized tools to design the user interface and capture the workflow. However, these tools require another person to “translate” those designs into detailed specifications that are useful to a developer. This translation process was often complex, time-consuming, and prone to transmission errors.

That’s why design work has often been done by developers or technical architects. However, the roles are different in no-code projects (as we explored in Chapter 4). Business stakeholders and end users are more directly involved in the design itself because no-code tools and processes are more accessible to nonprogrammers. For lower-complexity applications, the business doesn’t need any support from IT at all! This has a fundamental advantage in terms of improving the alignment of the design with the business needs.

Another difference from traditional software development is the efficiency of the design process. Typical software development depends on creating intermediate design specifications and technical documents that serve as the “bridge” between the business function and developers. While software design specifications and tools have improved greatly over the years, they still all suffer from similar challenges. Design documents often suffer from the “telephone game” problem — introducing transmission errors at an early stage of their development. They are laborious to create in the first place and even harder to keep updated — commonly many design documents become out of date almost as soon as the first version is released and the application begins to evolve. With no-code, the work of the Design Phase is to use the no-code tools directly as much as possible. This results in a “living” design model — the design and the implementation are updated simultaneously over time from a shared model.

Another inherent challenge of traditional development projects is maintaining the accuracy and fidelity of the original business requirements when it is communicated and rolled out across the team. You may start with a clear and simple definition of high-level business needs, but these quickly become

obscured as more levels of detail are defined. Each step in the design process must, at some level, translate the original intent into a different lower-level set of more detailed constructs. This is very much like the classic party game, where one person tells a story to another person. As the story gets repeated multiple times, errors in translation begin to inevitably creep in. You may end up with something that, at the end of the project, bears no resemblance to the original intent. With no-code design, there is a singular definition of the application — defined and updated throughout the lifecycle using the no-code tools. This improves accuracy and understanding by eliminating handoffs and reinterpretation. All stakeholders across the business function and development can share a common view without requiring translation into intermediate documents or semantics.

Finally, one of the challenges with any software design is that it must be aligned (and continually realigned) as the needs of the business change. Ongoing changes are a given in the rapid and dynamic environment in which most enterprises compete. This may be continuous improvement of business processes, innovation of new products or services, or responding to a new competitive threat. As the pace of business increases, the software applications that your business depends on must also be continuously updated, creating the risk of introducing misalignment. The business function may have already adapted to some external process change, but the software development teams don't yet fully understand the intent, which means the app doesn't fully reflect the present state of the business. With no-code projects, bringing together a commonly shared view across business and development helps improve the business alignment with the application itself.

The Four Stages of the No-code Design Phase

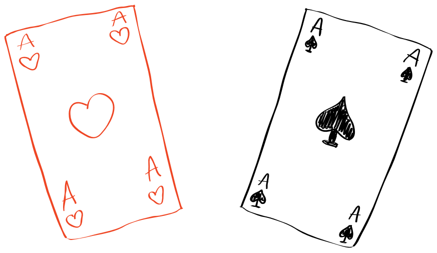
It's time to examine in more detail what a No-code Design Phase looks like. While it may sound daunting if you lack a technical background, don't be alarmed! It's quite straightforward and composed of four key activities. Each of these will be described more fully in the following chapters, but let's briefly discuss the essential stages:

Stage 1

Business Use Case

The first stage defines the highest-level business requirements for the solution you're building and outlines the criteria for business success. During this stage, it's important to keep the right level of detail by focusing on the "what" and the "why" but not the "how." While it's tempting to get into details such as defining the application UI or specifying inputs/outputs or technology components, the Business Use Case stage should focus solely on higher-level requirements and business processes. In Chapter 7, we will give examples of appropriate levels of requirements and define a simple template you can use to capture and understand the business requirements and their evolution over time.

This stage should also consider the full-picture definition of the business. It's important not to start constraining yourselves too much about how this might be broken into releases over time. That will come later, but for now, maintain a broad view of what the solution needs to be according to business needs.



Stage 2

Options Analysis

The second stage helps start to decompose the overall solution vision and shape the fundamental building blocks of the solution by selecting the right combination of packaged software, custom development, no-code development, templates, and components. This particular stage can seem overwhelming at first, given the potentially large number of options available, but we'll help you simplify this process with a basic decision framework. This will help minimize the number and complexity of elements in the overall solution and reduce the ongoing effort required to maintain it.

Stage 3

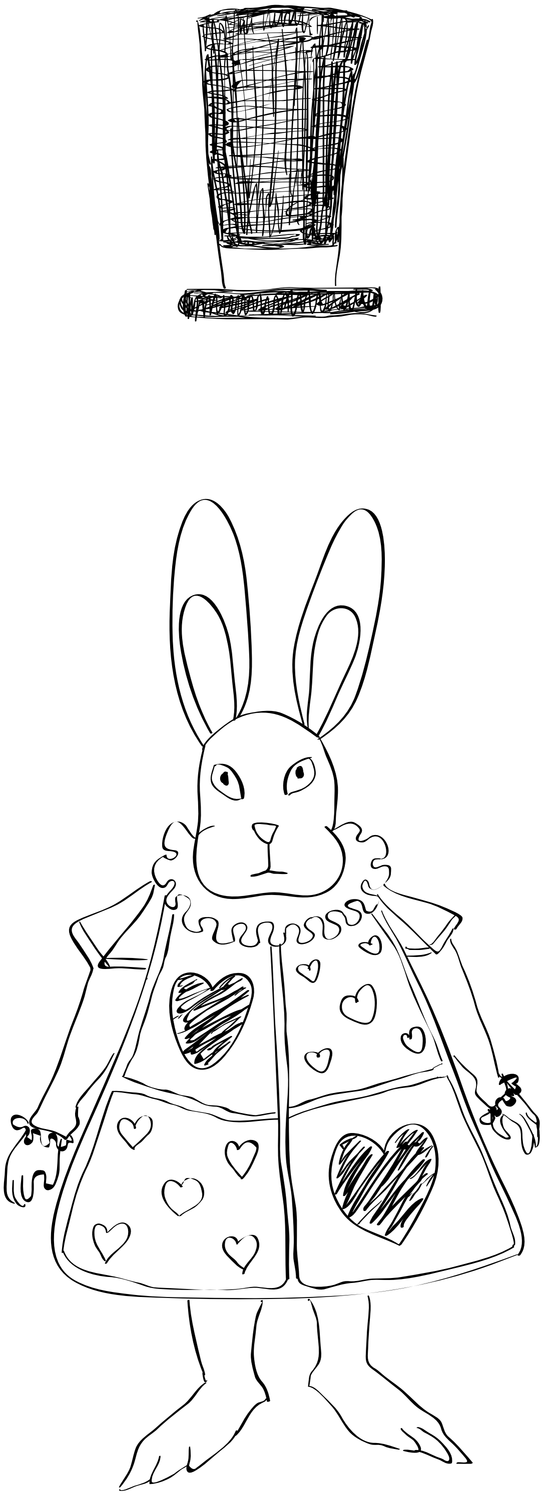
Design and Prototyping

This stage looks broadly at the vision identified by the business use case and uses visual prototyping techniques to quickly ideate and imagine the breadth of the solution by defining user experience (UX)/ user interface (UI), workflow, analytics, etc. Importantly, work activities in this stage will be performed using the no-code tools themselves — not intermediate document specifications that are later turned into code. The focus of the prototyping should be breadth over depth — you'll flesh out rapidly what appears to be a working iteration of the solution within the no-code environment itself. It will seem to have most of the desired business processes, forms, etc., but the depth of the functionality and logic may be incomplete or stubbed out. The scope of this stage is still the broadest possible view of the solution (in alignment with the Business Use Case). Do not constrain your project by scoping it down into releases yet.

Stage 4

Project Assignment

The last stage decomposes the vision into smaller use cases/apps/components. This is where you begin to define your target roadmap by focusing on the smallest possible initial scope that delivers business value — what is often referred to as the Minimum Viable Product (MVP). The MVP must deliver a minimum amount of utility quickly without being bogged down by the fullest view of the features and functionality that will be added over time (these will be added incrementally through a set of rapid updates). Think of this as the minimum number of features required to make the first version of the product useful. As this MVP release is defined, you will make key decisions on prioritization, what must be delivered first, the sequencing of dependent features, etc. Finally, you will apply a key framework called the Application Matrix (introduced in Chapter 5) that determines the right delivery model based on assessing the complexity of the application. The identified delivery model will guide the assignment and organization of the required resources (e.g., budget, roles, system environments, etc.) needed for the first release of the application.



Freedom from Failure

The Design phase may sound like a sequential activity, but it should be largely iterative — activities performed later in the Phase may uncover new ideas or opportunities that should trigger you to iterate on earlier thinking. This is typically true as you prototype the vision and design — you'll think of things that may inform changes to the business process. Unlike classic software development, no-code design allows the business vision to be influenced at times by the "art of the possible" and to respond to new or innovative ideas. This is a very powerful concept that will encourage innovation — no-code offers the ability to make changes to design easily and quickly while immediately validating a working model with business stakeholders. This means the cost of experimentation is lower, compared with traditional software development approaches, which encourages the use of ideation and iteration techniques and a boldness that comes with a "freedom from failure" approach.

This may sound like a lot, but it's quite straightforward. Each of these stages will be explored in the following chapters, along with guidance, sample templates, and practical tools that can be used at each step along the way.

Final Takeaways

While software development can be daunting and fraught with risks and challenges, no-code design techniques can set up business leaders for success in designing their software solutions. At first, the business function may be reluctant to perform this design work themselves. Yet the power of no-code makes this an ideal set of activities for business stakeholders and users to own and drive — it puts them fully in control of shaping the vision and approach. It helps to ensure alignment with business requirements and priorities. Finally, having the business function own the Design Phase will most often result in breakthrough innovations and deep alignment with the core business strategy.

In the next few chapters, we'll unpack each of the activities in the Design Phase. We'll start by examining the most underappreciated (and usually overlooked) stage in the design process — beginning with a clear understanding of the Business Use Case.

"Beauty is in the eye of the beholder"

Margaret Wolfe Hungerford

Stage 1

Business Use Case

07

clea sim

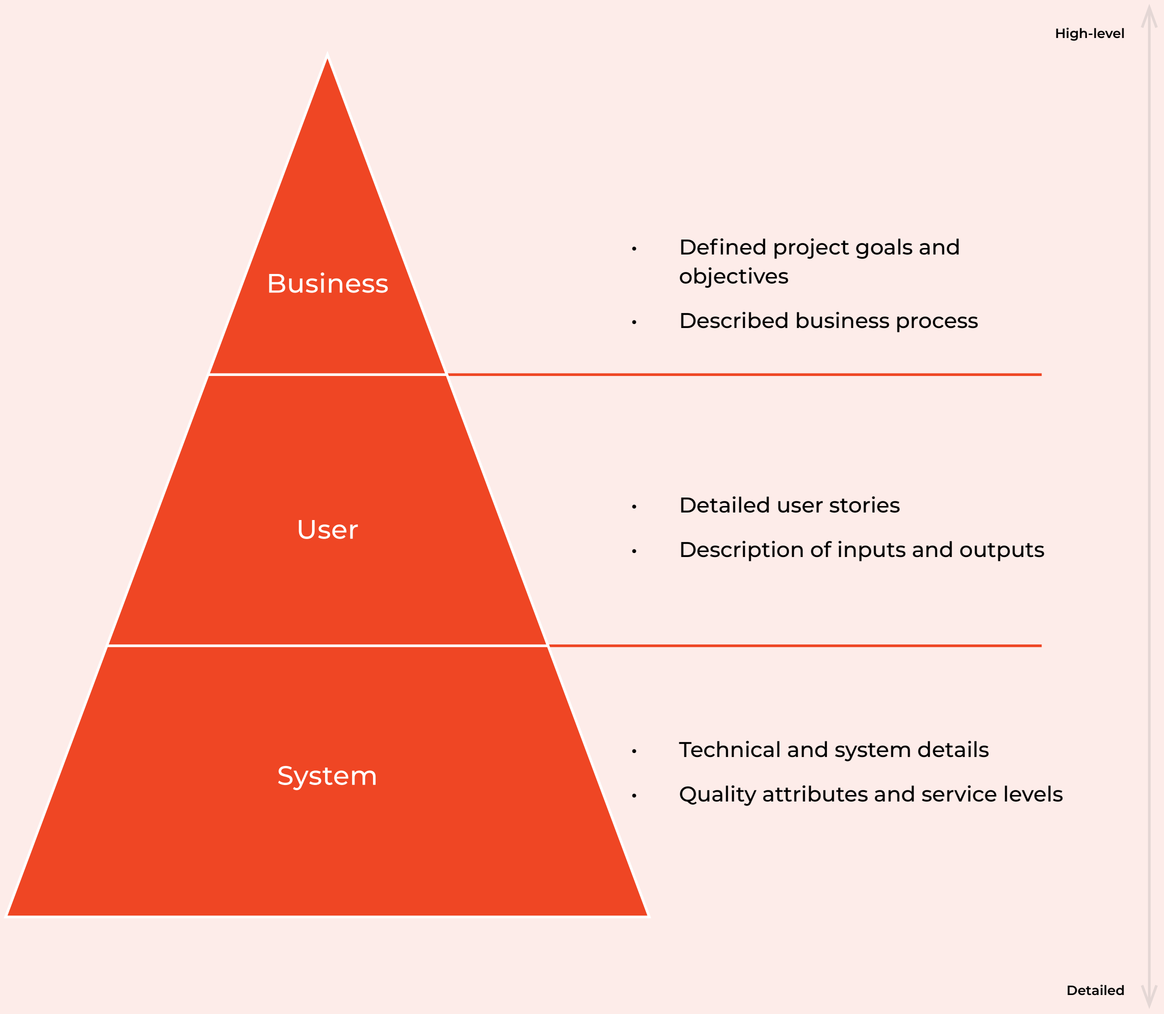
The quest to meet and hopefully exceed stakeholders' expectations when developing a new application can be like the quest for true beauty — it's always seen in the eye of the beholder. And in most cases, there will be plenty of beholders. Ask 10 different people within your organization what they expect from the app, and you'll likely hear 12 different opinions! With all the possible different ideas and perspectives on what constitutes "success," it's easy to get caught up in seeking to please all stakeholders. If you do this, you risk losing focus on the high-priority requirements that must be met for your app to be successful.

Therefore, it's critical to begin with a clear and focused understanding of the essential business requirements before you dive into designing and building your app. This is the intent of the first stage of the no-code methodology — the Business Use Case, which will define the intended business requirements and outcomes. The business use case should be in clear and simple language and defined by the designated business sponsor of the application.

It may be tempting to rush through this stage in the excitement of building apps. This can be especially true for no-code apps, given how easy and simple no-code tools are. It can seem easy to start defining the application by diving straight into the rapid construction of forms and workflows. Yet, even with no-code, you will still fail to deliver on expectations if you haven't thought through the business requirements upfront. So, it's important not to skip this stage. Make sure you have captured the essential business vision. Most applications that fail to meet business needs fail at this stage in the process, so pay careful attention to the business use cases.

Understanding the Development Requirements Pyramid

The term "requirements" is often overloaded with many loose meanings. It can cover a broad range of statements about what an application must do. Throughout any application development project, there is typically a broad spectrum of requirement statements gathered — from high-level business statements ("what" are the outcomes that must be achieved) down to highly detailed technical statements ("how" should the application be built). To help provide clarity on what we intended for the Business Use Case stage, refer to the Development Requirements Pyramid.



- Defined project goals and objectives
- Described business process

- Detailed user stories
- Description of inputs and outputs

- Technical and system details
- Quality attributes and service levels

The Pyramid divides requirements into three levels: business, user, and system.

- The business requirements focus on the project goals and objectives. It defines the business process at a high level that is being automated or transformed by the application. Sitting at the top of the pyramid, this level is the primary focus of our Business Use Case stage.

- The user level of requirements sits in the middle and begins to capture more detailed information about the functionality your app will need. It's typically expressed through a user's eyes. In traditional software development, these detailed requirements are often developed using intermediate design artifacts, such as user stories or user journeys, to describe both user flows and inputs/outputs of the app. This is not in scope for the Business Use Case stage, but we will discuss how similar considerations are captured later during the Design and Prototyping stage.

- The system level of requirements is at the base of the pyramid and addresses nonfunctional and system requirements, typically describing the technical details of the application, such as platforms supported, APIs, and required environments as well as quality attributes and service levels (e.g., response time and performance/throughput). This is not in scope for the Business Use Case stage either because when you're developing a no-code app, the no-code platform typically abstracts these considerations. The system requirements typically undergo a thorough assessment during the software selection process for your no-code tools.

At its most simplistic level, the business use case should define the following: no-code stakeholder, business processes, process use cases, process consistency, and success criteria. We'll briefly look at each of these key elements as follows.

No-code stakeholder

The no-code stakeholder is the primary business owner of the requirements and is the person ultimately accountable and responsible for representing the sponsoring business function or unit. There should only be one business owner, who will act as the ultimate viewpoint when it comes to making any decisions on requirements or priorities.

Business processes

The best way to frame the business requirements for a no-code app is by describing the highest-level definition of the business processes that will be addressed. Any application ultimately will either automate a new business process or digitize an existing one. Starting with the business process frame of reference enables you to describe the app in a way that is easily mapped to the business function.

Process use cases

A business process will typically consist of multiple subprocesses. Getting more granular in your description is an important part of defining the business scope. You will want to describe the business vision by decomposing the impacted process into smaller units, often referred to as process use cases, which will be addressed by the app. Do not yet start to artificially narrow the scope down to the initial release. Keeping the business vision broad is important at this stage. We will discuss how to address scoping the first release later in the Project Assignment stage in Chapter 10.

Best practice tip:

Drilling into successful techniques for process decomposition is outside the scope of this book. Thankfully, however, there are a lot of resources available to help if you are new to this activity. Most business analysis training or handbooks will include tips and practices for process decomposition as a core concept.

Process consistency

While a high-level process may seem globally consistent at first, as you decompose it into more granular process use cases, you may often find variations in implementations across the organization based on regional or business unit differences. The business use case should describe where known variations exist and discuss the desired target state for the business function (do they remain highly autonomous or should they become entirely globally consistent?). This will be further analyzed during the Design and Prototyping stage.

Success criteria

Lastly, how do you measure success? When will you know the initiative has been completed? The no-code stakeholder should specify this because they will most likely be held accountable to these success criteria by their leadership. Getting clear and measurable on the success criteria is critical as this will be essential when defining the initial release and the app's ongoing evolution.

The participants contributing to this phase will usually come from within the business function itself to ensure that it's as close to the front line and operational function as possible. Typically, the key roles (discussed in Chapter 4) who will contribute to the work activity include the no-code stakeholder and the no-code architect. It may also include a no-code creator as a way to engage some members of the actual development team into the early definition of the project scope. These resources will almost always be directly from the business group (function or unit) that has chartered the application and should have deep domain and subject matter expertise about the business process and functional requirements.

Real-world no-code example:

Territory Management Application

Let's put this in action by presenting a short example based on a use case for a no-code application built by a fast-growing outsourcing company. The company sought to accelerate its revenue growth with a new solution for automating territory management for enterprise customers.

No-code stakeholder

The primary owner within the business is the VP of Sales who is responsible for growing the sales funnel with qualified enterprise opportunities.

Business processes

The company builds its sales pipeline through multiple sources, including inbound lead generation, channel partners, and outbound activity. The territory management approach is meant to streamline and boost the outbound activity by introducing a well-structured process focused on selected enterprise accounts. The process will be executed by the enterprise sales team. Each sales rep will receive 30 target accounts with up to 20 associated personas. The goal of the enterprise rep is to generate and qualify sales leads by matching personas with the company's possible solution use cases.

Process use cases

Territory management can be further decomposed into these use cases:

- Ability of the sales team to assign accounts to other teams and individual reps.
- Access to the assigned accounts and the ability to segment them based on different criteria.
- 360-degree view of the assigned account.
- Ability to upload and view the associated contacts (including needed details) within the account with the ability to specify role and engagement strategy.
- Ability to manage engagement cadences with the account — plan, execute, and report activities (calls, emails, messages, and meetings) per each associated and validated contact.
- Proactive alerts (marketing touches, responses, news, and scoops) associated with the account.
- Ability to generate leads and opportunities within the account.
- Ongoing access to engagement-based key performance indicators (KPIs) in different dimensions — teams, individual reps, territories, etc.

Process consistency

In this example, the process followed by each of the regional sales teams (The Americas, EMEA, and APAC) has a fair amount of variability depending on regional priorities, assigned verticals, and strategies. It was decided in this case that the teams would move to this new territory management app to standardize around a globally consistent process across regions to gain greater efficiency and ease overall organizational sales execution.

Success criteria

The success criteria for the new no-code app were centered around increasing the overall volume of the qualified pipeline. This included increasing:

- Annual recurring revenue generated from the territory.
- Pipeline volume created from the territory.
- Number of opportunities promoted within the territory.
- Number of sales qualified leads generated from the territory.
- Conversion rates.

Tips on Best Practices

● Let the no-code stakeholder explain the business use case in a simple way with a focus on expected value and business goals.

● Stay at the right level of detail by focusing on the "what" and the "why" but not the "how"; focus on higher-level requirements and business processes and avoid discussing detailed recommendations regarding the foreseen solution (e.g., what fields or objects shall be used, etc.).

● Understand the workflow and outcomes first.

● Provide real-life examples to support the business requirements.

WHAT?

Final Takeaways

It's easy to be impatient and push to get started quickly. However, as John W. Bergman famously said, "There is never enough time to do it right, but there is always enough time to do it over." Likewise, it may not seem as if there is enough time for a proper business use case, but this upfront investment will help eliminate a lot of downstream errors and wrong decisions later.

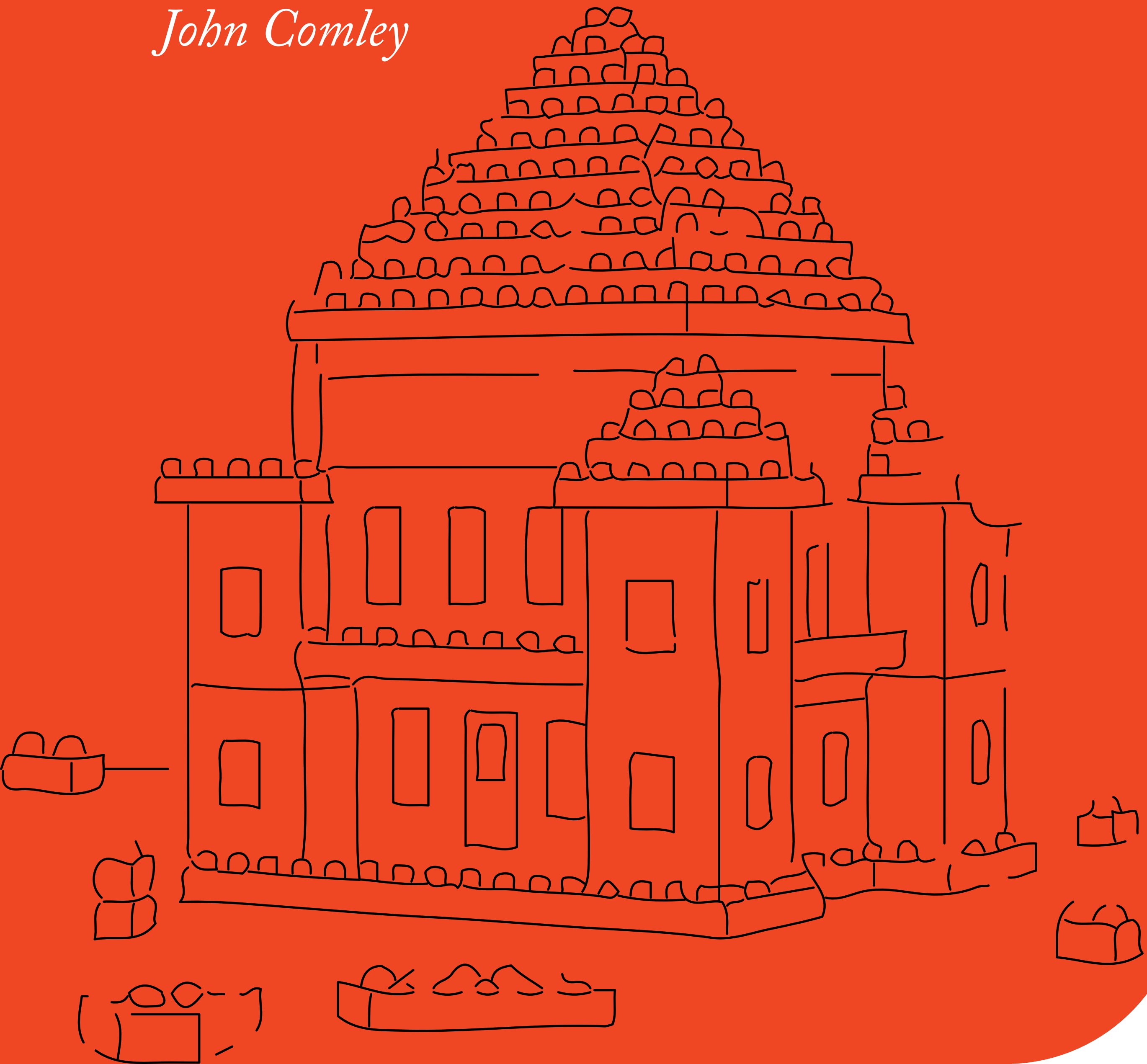
The Business Use Case stage can be the most important part of the entire lifecycle because it helps identify your target and how you will measure success. Resist the temptation to skip or rush through this process. Instead, thoughtfully identify as clear of a definition as possible — this will act as the "true North" that will keep the team on track throughout the project and the evolution of the app.

As you start turning the business use case into a high-level architecture for your application, you'll need to start making some key choices. How do you pick the right components? In the next chapter, we will explore how to conduct an effective analysis of your options.

WHY?

“The end is the final page of a book. In LEGO®, it’s not a thing.”

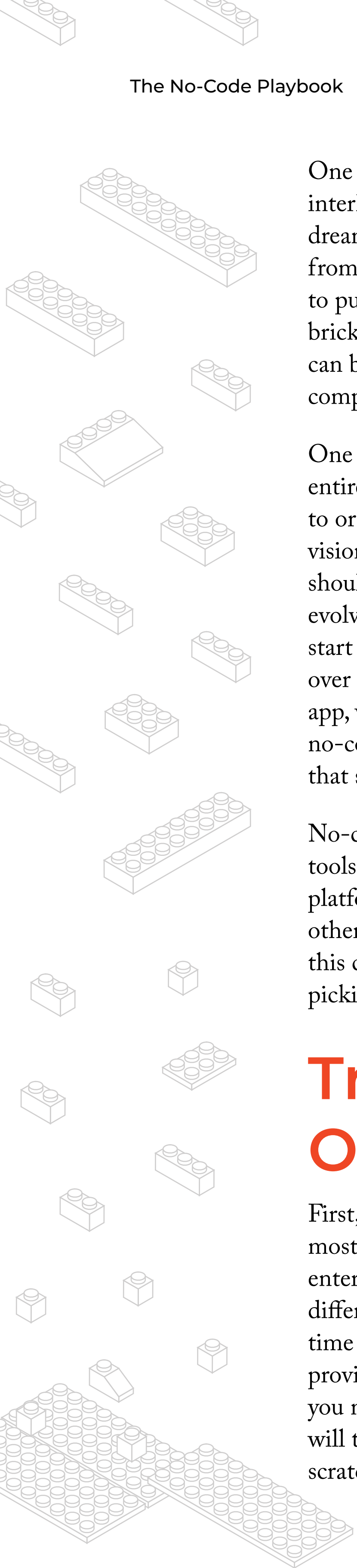
John Comley



Stage 2

Options Analysis

08



One of the most popular children's toys of all time is the brightly colored, interlocking LEGO® bricks. The ability to rapidly construct anything you can dream up — whether it be vehicles, buildings, trucks, or spaceships — from a set of standardized pieces make it fast and fun for any child (or adult!) to pursue their dreams of being a builder. No-code apps are a lot like LEGO® bricks in that regard. They democratize the construction process — anyone can be a LEGO® builder — by allowing you to assemble from preconstructed components instead of having to manufacture your project from scratch.

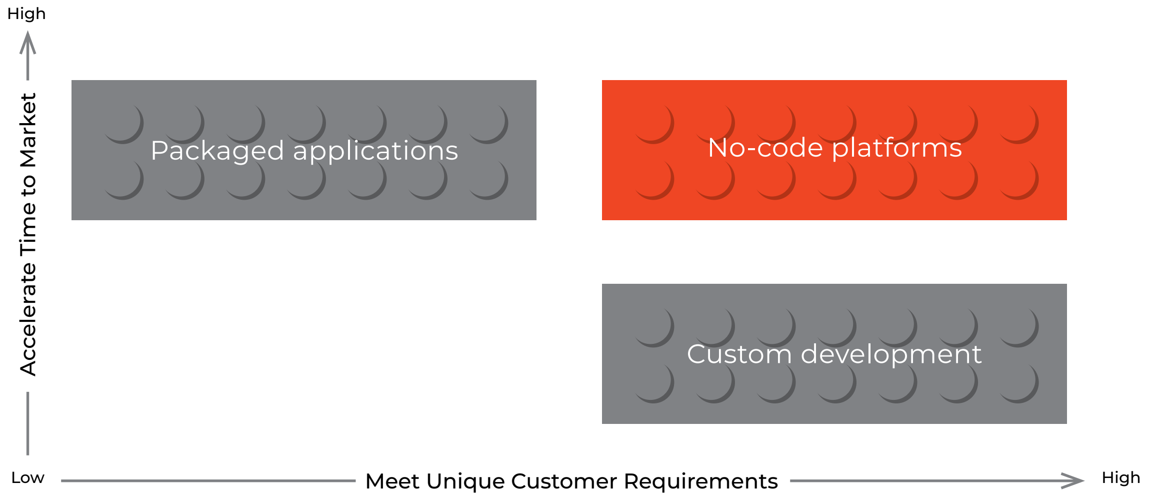
One of the powerful attributes of a LEGO® project is that you are never entirely done. There is always an opportunity (and desire) to continue to add to or modify your construction with new or different pieces to get closer to the vision in your imagination. This can also be a metaphor for how no-code apps should be built — they are often rarely ever fully "done." Instead, they keep evolving and adapting along with the changing needs of your business. You can start simple and then extend your initial vision by adding and replacing blocks over time. Instead of this evolution becoming a risk to the stability of your app, which is the case for a lot of traditional software applications, evolving a no-code app is safe (and perhaps inevitable) as it is based upon an architecture that supports this style of graceful evolution.

No-code solutions can be built entirely from "blocks" within the no-code tools. You can also integrate building blocks from outside of the no-code platform, such as integrating custom development code or integrations with other systems. Don't be overwhelmed by the various assembly options. In this chapter, we'll discuss an Options Analysis framework you can rely on for picking the right blocks for your solution.

Traditional Development Options: "Buy" vs. "Build"

First, let's start by reviewing the traditional development options available to most enterprises: typically, it is a choice between "buy" and "build." In most enterprises, there is a mix of both as each option helps optimize for slightly different outcomes. Packaged applications ("buy") help you accelerate your time to market but may constrain you to fit within a defined process or UX provided by the application vendor. Custom development ("build") will help you meet even the most demanding customer requirements, but the process will take you longer as it comes with the inherent risks of building from scratch.

You can think of no-code as the third alternative — the one that offers the best of both worlds in many cases. This can be visualized with the following diagram:



When you use no-code development for building software, you can accelerate your time to market by using configuration tools, prebuilt components, and templates. In the meantime, you can meet and exceed even demanding enterprise-grade requirements by leveraging the extensibility of the platform.

The addition of no-code enables a powerful new approach to a style of architecture typically referred to as "composability." Gartner uses the term "composable enterprise"[1] when referring to this approach of architecting modern systems built for adaptability. A composable architecture is typically built around a standard foundation — based upon a no-code platform — that allows you to add/change components both easily and quickly over time. Usually, prebuilt components are provided by the no-code vendor or its community. Composable architectures use no-code as the "glue" to assemble components and provide an overall architecture that is highly resilient and adaptable to change.

Prebuilt components allow for a tremendous amount of reuse across the platform. Such components are usually available through marketplaces maintained by prominent no-code vendors. The marketplaces allow specialized developers to build connectors, extensions, and even complete applications that the platform community can use (for example, see Creatio's marketplace: marketplace.creatio.com). This allows specialized knowledge to be packaged for quick reuse by a nontechnical audience, and it offers greater speed of delivery and agility of the no-code solution. Does your application need to obtain data from your accounting system, but you don't have the needed skills? Don't worry, someone has likely built a connector that will allow you access through the no-code platform.

[1] The Future of Business Is Composable, Gartner Keynote

Options Analysis

KEEP IT SIMPLE

For enterprises used to simple “buy” and “build” choices, the introduction of a composable architecture approach provides more flexibility and more options to choose from — but, understandably, more options can also seem overwhelming. How do you choose the best path when there are many possible ways to build a solution to a problem? To address this dilemma, we present a simple decision framework for Options Analysis to help guide the selection process.

First, we generally suggest embracing a philosophy of simplicity. Try meeting your business requirements using the fewest different component technologies needed. While it’s good to have choices, including too many types of differently built components can often raise the complexity of maintenance and evolution. A simple, consistent, and coherent architecture will generally lead to lower support and maintenance costs in the long run. Having a simpler architecture will also typically be easier to change and more resilient to the future evolutions that will occur.

Next, use the following questions to begin identifying the major elements of your solution:

Question #1

How standardized is your business process?

If you need to automate a standardized business process without too many specialized needs — identify a packaged application if one exists. Packaged applications are a great fit for standardized business processes (e.g., accounting or payroll systems) because they give you access to prebuilt functionality. They can also help ensure you comply with existing process standards for your industry. However, be sure to verify that the packaged software will meet your needs without requiring too much customization. Otherwise, you may encounter higher application maintenance costs and upgrade efforts.

Question #2

How much customization is needed to meet your business needs and provide a competitive advantage?

If you have many unique requirements and need high flexibility in process change management — either because of unique business requirements or because the solution differentiates your business, then, you should use no-code to build your app. Preferably you should start — if possible — with a ready-to-use template or prebuilt components. By taking advantage of pre-built apps and components, you can lower the development cost while responding more quickly to business or market demands.

Question #3

How technically complex is your use case?

If you need a complex use case (with highly specialized requirements), including the development/changes of a legacy system (for which it makes no sense to use the no-code platform), then you should go for custom development. An example might be an application that has a unique architecture or performance requirements. In this case, custom development should be considered as a possible option. However, many typical use cases do include these types of requirements. We usually wouldn't recommend custom development for traditional enterprise workflow automation processes (e.g., sales, operations, finance, etc.).

Question #4

Are there specific subcomponents that could be integrated?

After the core solution architecture has been selected, you'll want to revisit the other elements of your application to see if there are narrow use cases that can also be covered by third-party solutions and integrated with the core application. For example, the sales tax calculation use case can be automated by the third-party module or APIs integrated with the no-code solution (e.g., tax compliance software like Avalara).

Let's illustrate the concept by applying this simple decision framework to a few examples.

Example #1

Client accounting solution for a midsized business services firm

Use case

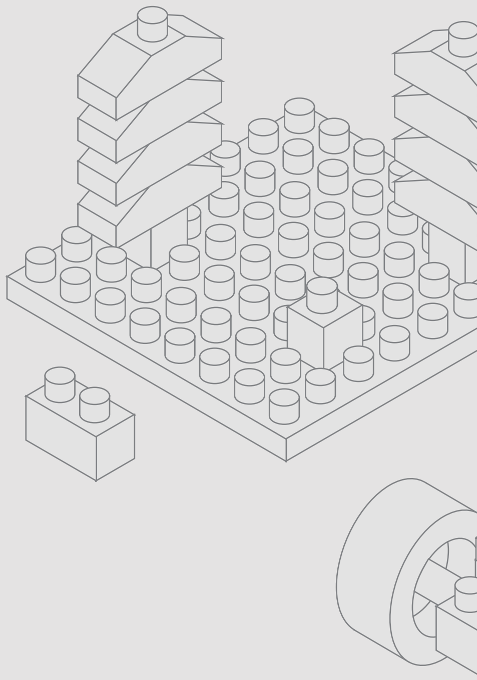
A consulting firm is seeking a cloud-based solution to automate back-office processes, including accounts receivable (A/R), accounts payable (A/P), general ledger (GL), and billing.

1. **How standardized is your business process?**
   In this example, the back-office processes are fairly standard and conform to the U.S. generally accepted accounting principles (GAAP) accounting principles.

2. **How much customization is needed to meet your business needs and provide a competitive advantage?**
   Little customization is needed. This is a solution that everyone in the market can access, a commodity that will not differentiate our position in the market.

3. **How technically complex is your use case?**
   The solution complexity is low. The requirement to be cloud-based is no longer a technically complex requirement given the broad availability of software-as-a-service (SaaS)-based accounting solutions.

4. **Are there specific subcomponents that could be integrated?**
   Yes, there are likely a few add-on modules or components needed for specific functions.



Conclusion

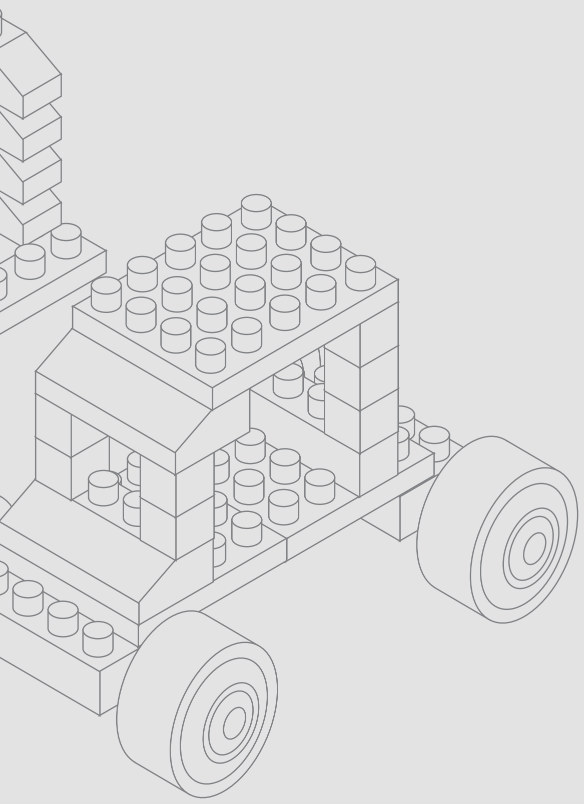
In this case, packaged software is likely the best approach for the firm given the availability of standardized SaaS accounting solutions that would meet its needs. Additionally, the firm can draw on the SaaS vendor community for key extensions that may be needed to support consulting needs (e.g., client time tracking and reporting) without significant integration effort. No-code is probably not a great fit here to begin with, but it may be considered in the future if some of the processes require a high degree of customization as the business evolves.

Example #2

Spare parts tracking extension for a legacy ERP solution

Use case

For 25 years, a large manufacturing organization has been using its custom-built enterprise resource planning (ERP) solution. The legacy system has covered most of the company's use cases, but business leaders need to introduce a new feature to track spare parts within orders as well as product management data. The legacy system has limited integration capabilities, and extending it requires a lot of coding and tribal knowledge.



1. **How standardized is your business process?**
   The process is quite unique and is dependent on the types of products that the company produces and delivers to the market.

2. **How much customization is needed to meet your business needs and provide a competitive advantage?**
   The functionality will require heavy customization because it is fully dependent on the existing capabilities of the legacy solution, its objects, and business logic.

3. **How technically complex is your use case?**
   The use case itself is not especially complex. However, it represents a part of a more complicated order and product management process automated via the legacy ERP. Thus, it will require a deep knowledge of the existing solution and software development resources.

4. **Are there specific subcomponents that could be integrated?**
   No, this is not currently required.

Conclusion:

Assuming the company is not ready to make a significant investment to replace its legacy ERP, the best approach is to build a custom-developed module on top of the current ERP solution.

Example #3

Digital lending solution

Use case

Reviewing and approving bank loans is a time-consuming and expensive process. A bank wants to automate the lending process to increase efficiency and create a better customer and employee experience. The organization has a unique approach for capturing digital loan applications, verifying them, and then processing loans through underwriting. Once complete, a customer can use a secure portal to authorize the loan execution process digitally and via electronic signature.

1. **How standardized is your business process?**
   The bank has a medium customized process that ensures the application's completeness and risk reduction through the underwriting process.

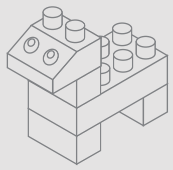
2. **How much customization is needed to meet your business needs and provide a competitive advantage?**
   High customization is needed. This is a market differentiator and will directly impact the effectiveness of the bank's go-to-market success.

3. **How technically complex is your use case?**
   The solution complexity is high. The solution requires several integrations with the core banking systems, secured portals, etc.

4. **Are there specific subcomponents that could be integrated?**
   Yes, there are likely a few add-on modules or components needed for specific functions.

Conclusion

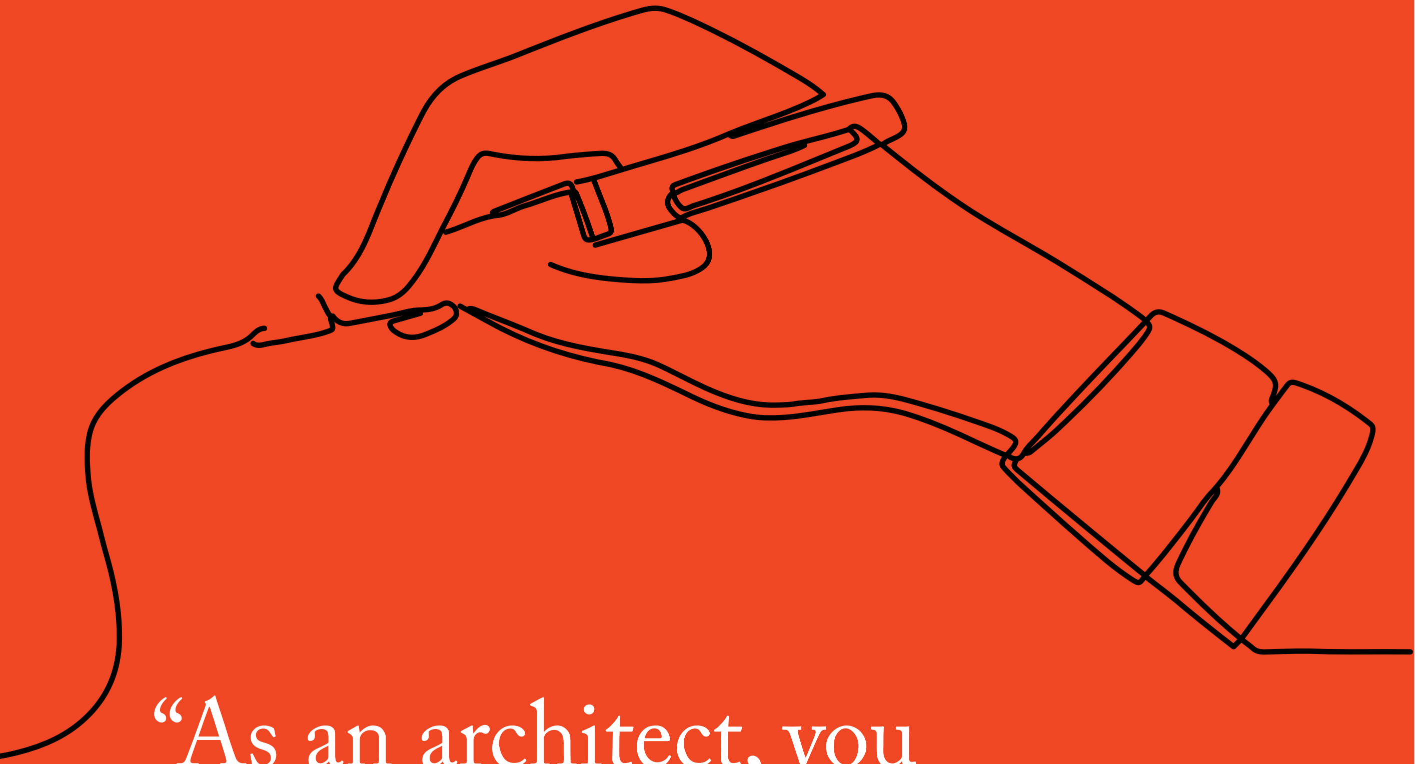
In this case, a no-code composable architecture is likely the best approach, given the medium customization and high complexity needed to support the bank's differentiated loan approval process. The bank may go ahead with the no-code app using a prebuilt template for lending and a number of connectors for integration with the core banking system, document management solution, and e-signature. The no-code app will automate an overall workflow and streamline how data is entered into the banking system. It will tightly integrate with other online services to support specific functions, such as credit reviews, resulting in a simplified and efficient user experience. This approach enables a highly differentiated process while still offering speed to market by using third-party components where available. The no-code template and connectors in this scenario were available from the vendor's marketplace, which allowed the bank to accelerate development by starting with pre-built and reusable functionality.



Final Takeaways

The organizations that are best set up to thrive in today's fast-paced world are able to make changes quickly. By adopting a composable architecture approach using no-code platform capabilities, you'll be designing an environment built for change. When your business is dynamic and ever-changing, you need an architecture that allows you to meet your evolving needs with plug-and-play components. As you embark on this journey, apply the Options Analysis framework to help simplify your overall approach and develop a solution that is easily maintainable and extensible to your evolving needs.

Now that you've selected the right approach to your solution components, you're ready to start actively designing. What no-code design best practices should be followed and how "deep" do you go?



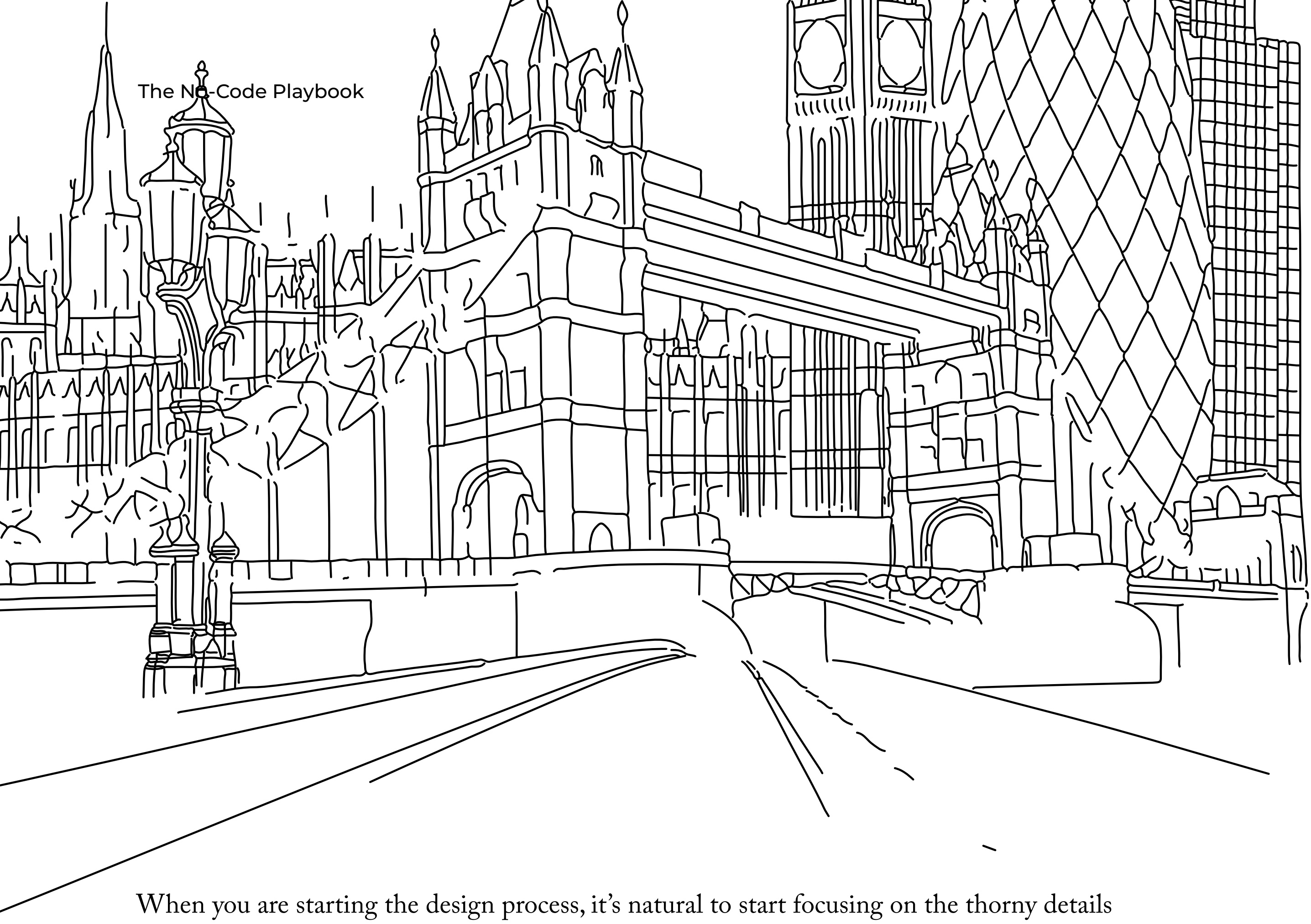
“As an architect, you design for the present, with an awareness of the past, for a future which is essentially unknown”

Norman Foster

Stage 3

Design and Prototyping

09



When you are starting the design process, it's natural to start focusing on the thorny details of trying to solve the immediate business challenges. It's tempting to want to go deep into designing the specific features that you believe will represent the first release. But design (like architecture) must balance near-term needs with future thinking — you must keep in mind your longer-term aspirations and business vision (identified in the business use case) to guide the evolution over time. So, it's critical that the Design and Prototyping stage of the lifecycle focus on the whole solution — you're defining the long-term vision and architecture, not merely a few of the immediate parts.

Another key tenet of this stage is that you should try to leverage the power of the no-code platform to define the design of the application directly. Instead of putting the functional requirements into an intermediate document and then translating it into code at a later stage, the functional requirements and design process in no-code development are much more efficient and streamlined. A no-code architect should define an application with all the needed parameters using the no-code tools (e.g., fields, dashboards, UX/UI, and workflows) and make changes on the fly when presenting and evolving a prototype. Unlike traditional prototyping, with no-code, you're building the software itself. It's not a simple low-fidelity wireframe or some type of "throw-away" clickable prototype — you're building a working iteration of the final application.

The Design and Prototyping stage should be performed using the no-code platform visual tools themselves. This allows for rapid ideation cycles where the design effort more directly and immediately can be tested with end users. While the exact number and type of tools will vary depending on which vendor you select, there are typically at least five major design activities performed during this stage:

UX/UI design

Designing the user experience, typically workplaces, and visual forms.

Integration design

Designing how the app will connect to other systems of record or databases, using application programming interfaces (APIs) or connectors.

Workflow and logic design

Designing business rules and processes (both human and system processes) using a visual workflow designer.

Dashboards and analytics design

Designing the pages or reports that will provide analytical insights.

Each of these will be briefly discussed in the following sections.

UX/UI Design

Usually, the first step in the prototyping process begins with defining UI elements for the input of business data, typically by providing a visual forms builder. The capabilities of the forms builder may vary, but they will usually offer some sort of "drag-and-drop" design (from a standard palette of UI controls or "widgets") onto a visual canvas that allows you to design the form and its required elements. More sophisticated no-code tools will already have the needed components to provide optimized and consistent UX, including templates and recommendations. They will also have a larger library of controls, and some may even offer the ability to find and import controls from an external marketplace for even greater levels of customization and flexibility.

The data model is usually connected to the UI, and advanced no-code platforms allow you to combine data modeling with UI design. This is especially powerful during this Design and Prototyping stage as it allows you to iterate very quickly without having to separately define the data model. Another powerful benefit of this stage is the ability to rapidly build the UI within the no-code tools themselves; there is no need for external prototyping tools, such as Figma, Adobe XD, etc. as the prototyping is done seamlessly within the no-code platform.

Beyond just defining the form, this step of the design process also typically includes some implicit business process steps associated with the UI. The form data will nearly always require some business rule validations before you can submit the form. This usually also includes the required inputs/output events (or triggers) that may be linked to the form and the chaining of multiple forms into a mini workflow (if the input steps will not all fit into a single form). The forms may also be dynamic, with defined rules or conditions that may cause optional or dynamic parts of the forms to be displayed. All these additional layers of form design help make the input easier and more intuitive for the end user, but they require more detail and definition during the design process.

The UX of the no-code app is often (but not always) a composable UX (i.e., one which represents data from multiple other systems). The benefit of a composable UX approach is that it provides a simpler experience for end users by allowing them to enter data into the app once. Users do not have to navigate to multiple systems and reenter the same data multiple times. This can provide great improvements in productivity and speed of data entry.

Best practice tip:

During the Design and Prototyping stage, don't try to capture every entity or fully define all attributes. "Close enough" will be sufficient at this stage! The focus is not on completeness but on capturing enough of the business domain so that you can validate the essential business vision with your stakeholders. You can iteratively come back and extend/evolve the data model during the MVP stage, so don't sweat the details at this point!

Workflow and Logic Design

Many no-code tools will offer the ability to define the business process, but this is an area where there is a fair degree of variance in capability. On the simple range of the spectrum, the no-code tool may simply provide the ability to chain multiple forms together to express navigation through the application. On the other end of the spectrum, more sophisticated no-code tools will offer a full-blown business process designer that visually describes the business process, typically in some type of process notation. Some may offer support for industry standards like Business Process and Model Notation (BPMN). BPMN-enabled designers are great as they represent common knowledge across different stakeholders and can be easily read by different groups/users.

During the Design and Prototyping stage, it's usually fine to focus on the "happy path" process (i.e., focus on the default process featuring no exceptional or error conditions). Later, during the next phase, we will come back to the design activities and add in more variant and exception paths but, for now, the simple process will help test the vision. It is important, however, to also think about more than just workflow and incorporate business rules and additional logic because it might be useful for the stakeholder.

While the high-level business process may be simple to understand, process definitions can sometimes quickly become more complex during the design process. The top-level business process may call many lower-level subprocesses to address specific variations or common activities within a process. The nesting of processes is not technically required — you could define all the steps as a single, large process — but proper nesting of processes is encouraged as it will ultimately improve both the ease of understanding and reuse of your process definition. For example, a top-level order processing workflow may call a nested inventory picking workflow to ensure products are reserved and picked for the customer before returning to the next step of the top-level workflow. Nesting prevents you from creating one monolithic workflow that is hard to understand and maintain.

The ability to describe a business process visually is very important as it allows the business stakeholders to describe requirements in a language that easily matches the way they think about their business function — a sequential flow of decisions, actions, and results. This allows them to provide feedback more readily on both the accuracy of the current process (if you are automating a process that exists), and it helps them envision TO BE processes that you may be introducing.

Best practice tip:

During the Design and Prototyping stage, focus primarily on the top-level workflow (or perhaps the first-level nested workflow). You may want to temporarily skip over the prototyping to define exceptions and system processes for now; they will be needed in the MVP solution, but you can come back and iteratively add these details once the business function has validated the vision. You probably want to acknowledge (and perhaps list) the variant processes as it's important for the stakeholder to define how standard the processes should be across the organization, but fully defining these using the no-code tools may not be required during the prototyping work.

Integration Design

The integration design is an essential part of nearly every no-code tool. This allows your no-code app to work with the business data from other applications, and it helps your app fit into an existing set of IT applications and data sources. However, the integration design capabilities will vary tremendously across no-code tools. Some may be limited to only working with a few data sources, such as working with spreadsheets or database tables. Capable no-code platforms will offer a broader selection of ready-made integrations (usually referred to as "connectors" or "adapters"). Sometimes, they will even offer a broader marketplace where you can discover and reuse prebuilt integrations that are offered by either the vendor or by their community.

During prototyping, try to keep the integration simple — this stage doesn't typically require full-blown data integration but just an illustration of the possibility. You will find that representational state transfer (REST) and simple object access protocol (SOAP) services are often the best way to set up easy data exchange, or you may be able to use a prebuilt connector. In the next phase, you may need to get into more complex integration design activities. For example, you may have systems for which there is no off-the-shelf connector — it may be because it's a custom application or because it's a less commonly used system (and the no-code vendor or their community hasn't yet seen sufficient market demand to build a connector). To handle these scenarios, typically no-code platforms offer some type of Software Development Kit (SDK) for building custom connectors if one doesn't already exist. However, this will require additional custom development skills to be part of your project team. It also may require coordination with the no-code vendor to operationalize the connector into your runtime environment if your platform is running in a cloud environment.

Some additional notes on integration design:

- It is usually highly recommended that no-code tools DO NOT map directly to another system table or data source. This creates a dependency on the underlying system, which can cause the no-code mappings to "break" if the other system is upgraded or changes its data model. Instead, most no-code tools will use a standard API for any system to enable them to be more resilient when the applications are upgraded.
- While the no-code platform may offer prebuilt integrations, this does not remove some of the fundamental complexity of internal application integration. Your IT department may have heavily customized or extended the existing systems, which may require additional logic to be added to the integrations or mapping. This is where a fusion team may be required.
- You may need to consider data migration requirements as part of this activity as well. This may be needed if the integration is more of a one-time load of data files into the system or if you are replacing a legacy system (and need to preload it with key reference data). For migration/import of data, you should attempt to find the easiest path; for example, a common approach is to import existing data through Excel files, which are usually supported as a file format across a wide range of systems.

Best practice tip:

During the Design and Prototyping stage, you want to limit the integration efforts — you can either deploy a ready-to-use connector from an available marketplace or set up a simple REST interface, or even configure a mock-up of the integration to visually showcase the stakeholder how it might be working. This can help you avoid unnecessary complexity and possible delays at this stage.

Dashboard and Analytics Design

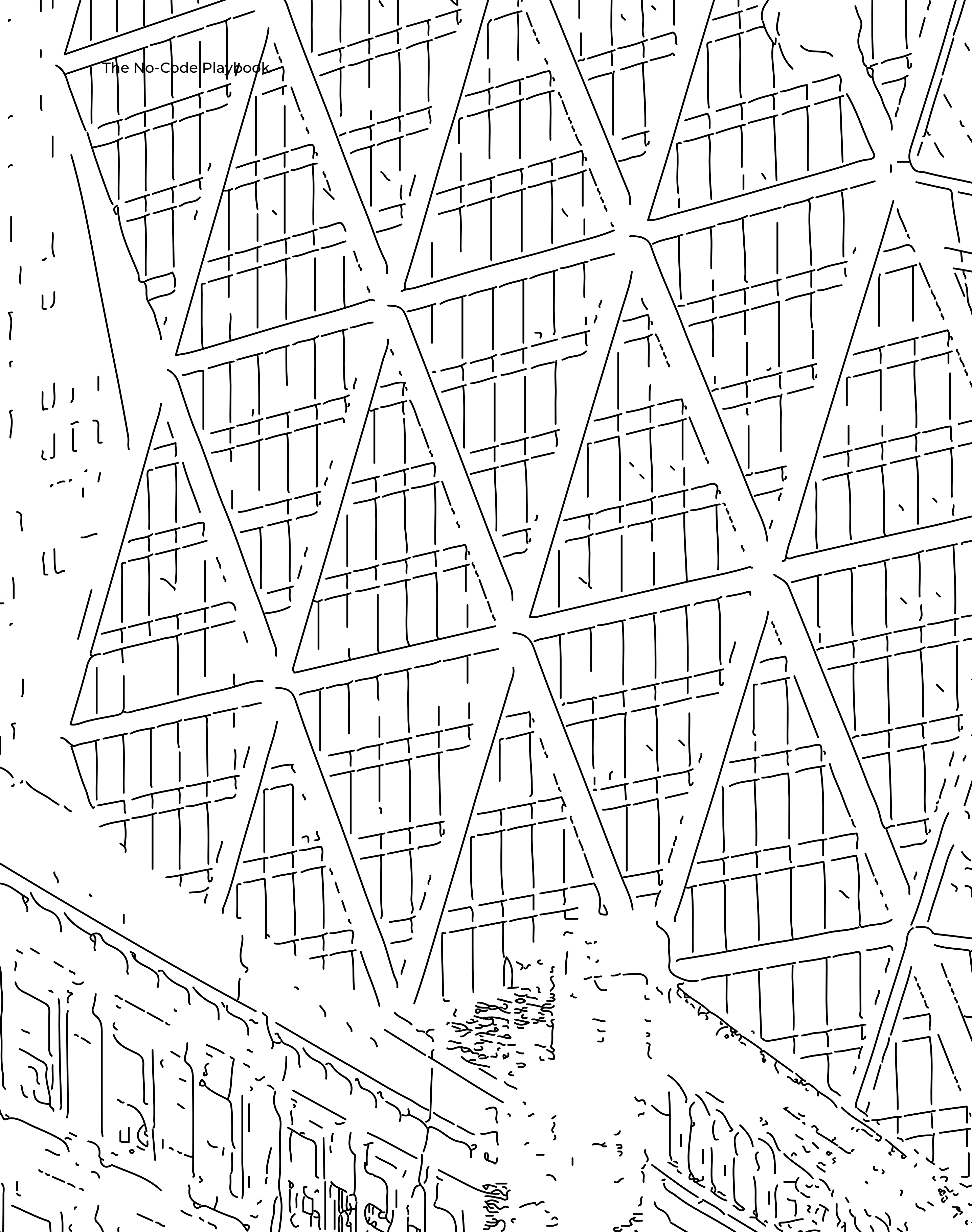
This design activity guides the prototyping of the pages or reports that will provide analytical insights. The no-code tool will typically offer some way to visually compose a dashboard or report, often through drag-and-drop of prebuilt components into a dashboard page. This will be similar to the UX/UI forms design but will leverage prebuilt visual widgets (e.g., bar charts, line charts, and highlight tables) to assemble into a dashboard page. There will be some way to configure these widgets to connect them to data from other systems. There may also be a set of prebuilt components available through the vendor's marketplace that can offer a rich set of visualizations and data sources.

Dashboards are important as part of your no-code app design to help employees to monitor the performance of the business use case at once. With just one UI form, the user has a more limited view of the process. Having a dashboard allows users to view multiple visualizations at once and understand the business from many different angles. Dashboards also make it much easier to compare data on the fly as they can group together related data visualizations into a single page that allows users to quickly get a sense of how those visualizations are related in real time. This allows your users to make connections in the business data that can drive new insights and enable taking more holistic actions to improve the process.

Finally, dashboards help your no-code app make better use of the data that is already collected, making it contextual, holistic, and actionable. Having embedded dashboards within your no-code app can help every aspect of your users' operations by enabling them to become more data-driven and allowing them to leverage that data to drive insight and focus on the right areas. One good place to start is by focusing on targeted key performance indicators (KPIs) that are meaningful to your business process. KPIs are the metrics that will be used to determine whether a process is performing successfully. This helps the average user focus on the top improvement areas and helps improve overall levels of impact and efficiency.

Best practice tip:

Start simple during the Design and Prototyping stage. Set up easy-to-digest dashboards and KPIs that highlight the result of the use case. At this stage, it's important to simply show the vision of how a dashboard may be used; it's not necessary to have accurate data that correctly matches the process. For prototyping, you should consider stubbing out data or using simpler data that may be easily available. You'll come back to data during the Go-Live design activities. However, for this stage, it's important simply to show the ability to demonstrate the type of data and how it can be analyzed. The accuracy and completeness of the data can be addressed down the road.



Final Takeaways

As outlined above, the intent of this effort is to focus on breadth not depth — you are constructing the “frame” of the app not designing every last detail. Don’t worry, these details will be added to the app iteratively as it evolves into the MVP stage. The important focus of this stage is to provide a “close enough” representation of the business vision that you can validate with your business stakeholder.

This iterative approach to the no-code Design and Prototyping stage has the following benefits:

- Early prototyping helps you thoroughly test and evaluate the design. Users typically respond better to seeing (what appears to be) a functional app and providing input and feedback.
- It’s much faster and cheaper to make changes based upon feedback early in the process before you have started formal development.
- This fluid approach to prototyping and iterating encourages innovation and more “out-of-the-box” thinking, as the cost of exploring and ideating is much lower with no-code.
- It can help you with understanding costs, issues, and risks, allowing you to make more accurate estimates. Having a prototype of the full scope will let you more accurately size and scope what can fit into the initial MVP release.

If used correctly, this powerful approach to no-code design will help you prepare for the “unknown future” by allowing you to experiment, collect feedback, and iterate rapidly to ensure alignment with the business vision. Now that we’ve validated this vision, let’s change gears in the next chapter and begin to define the successful steps that will get you to your destination.

“The odds of hitting your target go up dramatically when you aim at it”

Mal Pancoast

Stage 4

Project Assignment

10

Despite constant evolution and innovation, custom application development projects can still be risky and may fail to meet the expectations of the business function. While there are many reasons for this, it's well understood that longer and larger projects present higher risks including scope creep, budget/schedule overruns, and in the end — missed business opportunities. Trying to be overly ambitious when building software is a recipe for disaster! So, while the earlier stages of the No-code Lifecycle looked broadly at the business vision and requirements, and prototyped a broad view of the overall solution, this stage will now narrow our focus. Here, you'll get a clear view of the achievable "target" that you want to aim for in the first release.

The Project Assignment stage defines the target scope of your no-code app that you will be building in the next phase (the Go-Live Phase). At a high level, it includes the following activities:

- Decomposing the business use case into smaller use cases.
- Selecting and confirming which of these use cases will be included in the initial Go-Live scope.
- Ensuring that the scope aligns with any timeline constraints established by the business function.
- Defining the necessary roles and participants in the project (using the delivery model suggested by the Application Matrix described in Chapter 5).
- Preparation for enablement of the release (including identification of power users, scheduling of governance checks, planning enablement, selecting the right set of environments, etc.).

We'll be exploring each of these activities briefly in this chapter but, first, let's start by digging into the concept of a "Minimum Viable Product" (introduced in Chapter 6).

Minimum Viable Product (MVP)

In software development, the approach of defining a narrow initial release is often referred to as selecting an MVP. It is a concept taken from the book "Lean Startup" by Eric Ries. The approach stresses the benefits of defining the smallest scope possible for an initial software release. This approach allows you to quickly get the software in front of actual users to begin learning how they use the product and validate its business value. MVP emphasizes time to market over striving for perfection. It allows you to change direction before too much investment has been sunk into the current approach. By taking this iterative approach to development, you can use early feedback from real users to continuously enhance your software rather than investing time into building something your users don't really want.

The MVP approach has some pitfalls. Some development teams may overemphasize the "minimum" in their scope and release versions that fail to deliver business value due to their limitations. When adopting MVPs, it's critical that your scope definitions deliver enough value to warrant the rollout to end users. Remember the adage, "you never get a second chance to make a first impression." If your MVP lacks sufficient value or capabilities, users may not stick around long enough to wait for it to evolve into something better.

With that defined, let's walk through some of the key steps to perform in the Project Assignment stage.

Step #1

Decomposing the business use case

You should start with the overall definition that you first outlined in the business use case stage. This probably already included some definitions of smaller subprocesses that enable a broader business vision. During the Design and Prototyping stage, you likely further decomposed the overall processes into smaller subprocesses, prototyped with the no-code tools. This list of subprocesses provides a candidate inventory that could be part of the MVP but, for now, they are too broad to include in the initial Go-Live release and will need further prioritization. For example, thinking back to our Territory Management solution, the MVP may include the ability to assign accounts and track key account data points, including associated contacts, as well as capabilities to execute core engagement cadences. Meanwhile, other features, such as integration with a data enrichment tool, more sophisticated workflows, and an advanced dashboard can be delivered after the initial MVP release.

Step #2

Selecting and confirming the Go-Live scope

You should work with the no-code stakeholder to prioritize your inventory of candidate subprocesses based on the business impact and value to users. The work you have performed during the Design and Prototyping stage will have helped educate and inform your business stakeholders about the benefits by visibly demonstrating the value that will be released with the no-code app.

At this point, attempt to work with the no-code stakeholder to define how far down the prioritized list you must go to deliver sufficient initial business value. Typically, the MVP scope should include at least one end-to-end subprocess. The key will be to identify the smallest set of individual subprocesses/business tasks that, when implemented, cover the most important part of the business requirements.

Best practice tip:

It's recommended to assign a stack-ranked prioritization of subprocesses, rather than simply classifying each as "small," "medium," or "high" value (often referred to as "T-shirt sizing"). Forcing a ranked prioritization is more difficult and may require a lot of discussions but, ultimately, it will be essential when applying later constraints while finalizing the release definition.

Step #3

Applying timeline constraints

In traditional software development, the timeline is often estimated either by a bottom-up estimate of how long it will take (common with waterfall development) or as a predefined structure of the release cadence (common with Agile, which will start with some defined sprints/releases). However, this works only in cases where the development team can arbitrarily set its own timetable without having any influence or constraints imposed by the market or business. Alternatively, we recommend that it's often better to start with understanding the business timetable — there is usually some defined timeline linked to the agreed success criteria as stated in the Business Use Case stage (e.g., having the app support the launch of a new product or service, or enabling the rollout of a process change). It is important to understand external drivers and factors that will influence scoping of the process and use cases.

Based on the timeline constraints, revisit the candidate set of subprocesses you prioritized and selected in the prior step. If you cannot fit the desired scope into the timeline, it may require a further narrowing of the scope to fit within the calendar constraints.

Step #4

Identify the appropriate delivery model

Now that the target scope is mostly defined, you should select the correct delivery model using the Application Matrix framework presented earlier in Chapter 5. This is important as it will help define the types of resources the project needs: Can the business team deliver this app independently? Is more support needed from a fusion team, including software developers (to build more complex components), or from the No-code CoE if one exists? Selecting the delivery model is also the key to sizing up or down the remaining stages in the no-code methodology, as some stages (in particular the "governance checks" in Chapter 14) may need to be scaled depending on the complexity of your needs.

Step #5

Defining roles and participants

With the delivery model set, you should start identifying the roles you'll need for your project. A few considerations to keep in mind:

1. Multiple roles can be played by a single individual, especially on very small projects. However, beware of having people wear too many hats. Also, the no-code stakeholder should be independent of the full-time members on the development team because it's important to have a sufficiently senior stakeholder who isn't biased by the day-to-day tactical efforts.

2. Roles can be played by part-time members assigned to the project, but make sure that all individuals assigned to the project are able to spend a sufficient amount of time with the no-code team.

3. At times, it may be difficult to get sufficient resources assigned with the correct amount of bandwidth — especially resources from the business function, which typically already has full-time accountabilities to the organization in other areas. Ultimately, it's the no-code stakeholder's responsibility to make the case to management for freeing up the necessary resources to make the no-code project a priority.

Step #6

Future project enablement

Besides the individuals involved in the development effort, what other resources are needed to enable your project? This may typically include, but is not limited to the following:

1. **Power users.** While you may have some users already integrated into your no-code development team, you should plan to identify and recruit a set of "power users" (those who are highly proficient with the current business function/process and represent individuals who would give you more advanced and richer feedback). It's important to have these power users test the app prior to going live, typically as part of a user acceptance test (which will be discussed in Chapter 15).

2. **Governance checks.** In a similar fashion, it's recommended to identify and allocate resources of the governance team to test all the compliance and regulatory requirements; this won't be performed until the Governance Checks stage (described in Chapter 14), but you'll want to secure their time in advance proactively. Keep in mind that the types and depth of governance checks vary based on your application complexity, so use the Application Matrix to select and scale as needed.

3. **User enablement planning.** It's also a good idea to start proactively building the user enablement plan so it's not left to the last minute. It's important to build the enablement content and schedule training and walk-throughs for application users. Too often, this is compressed late in the project as an afterthought. Start by assigning someone early in the project to own building the plan. They will work alongside the no-code development team as the app is built.

4. **No-code environments.** There will need to be some set of no-code cloud environments to support the development effort. There are typically at least two environments — development (where the daily core no-code building and testing take place) and production (where the "live" application will exist). This list will vary somewhat, and you may decide to add or remove environments depending on the complexity of the app and the number and size of teams. For example, more complex enterprise deployments may add additional quality assurance (QA) environments for more comprehensive testing efforts and often a preproduction environment (a close mirror to production, typically used for a user acceptance test). The Application Matrix can help you with the approximate sizing of environments.

5. **Release strategy.** Finally, after all of your team members have been assigned to these tasks, you can begin defining a roadmap and schedule for both MVP and subsequent releases. Identification of parallel vs. sequential delivery is another important step. Seek opportunities to deliver multiple apps and/or components in parallel.

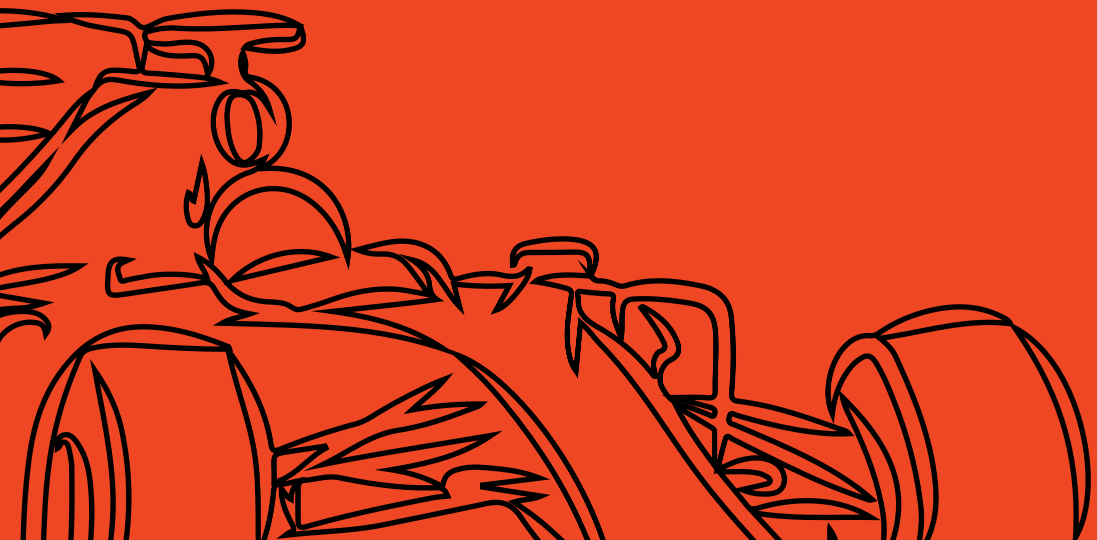
Final Takeaways

When you focus on perfection and completeness, you risk trying to "boil the ocean" by including too much in the first release. It's key to mitigate this risk by staying "on target" and focusing on an MVP that delivers "just enough" features to test value early and adopt an incremental approach to adding functionality over time. Yet, you can't build incomplete feature sets — each step of the journey must still be incrementally valuable, usable, and delightful. That is the art of defining the MVP release.

Now that you've scoped and assembled the MVP release — with the right team and resources — it's time to move to the next phase of the No-code Lifecycle. You're now ready to begin the Go-Live Phase!

“If everything seems under control, you’re not going fast enough”

Mario Andretti



Introduction to Go-Live

11

The Go-Live Phase begins the actual development of your no-code app. Part of the power of no-code is that it abstracts you away from a lot of the low-level details of traditional software development, which allows for a greater ability to step back and focus on what matters — releasing the MVP app to your users as quickly as possible!

So, we're going to start with the basics. In this primer, we'll introduce the key concepts and provide important context about the approach as well as tips and best practices to navigate you through this phase. The following chapters will then dive into more specific details on each of the individual No-code Lifecycle stages. Keep in mind that the stages we'll outline include what is relevant to mission-critical enterprise applications. However, the steps may be streamlined significantly if your apps are not complex. Please remember to use the Application Matrix to assess the level of complexity of your apps. In most cases, especially when the applications don't include critical requirements, Go-Live should happen in days or weeks (not months or quarters).

No-code Go-Live Differences

1 Let's start, however, by taking a fresh look at the no-code development specifics and discussing how they vary from traditional development at this stage of the cycle. First, in no-code, the business function acts as a direct participant in the development itself, rather than simply being an external stakeholder specifying requirements to the developers. The common software development methods tend to focus primarily on software developer roles and don't consider business personas as being direct participants in the team (or even contemplate the fact that the business function might be a developer). In contrast, the business function sits directly in the driver's seat in no-code projects — authoring the requirements and driving the design activities. The business function will also play the lead role (and often the sole role) in the development by placing no-code creators and no-code architects directly onto the team itself. When the business function plays a larger role, you typically have greater direct knowledge and understanding of the business domain seeded into the development itself. This often results in improved efficiency and accelerates the creation of differentiating intellectual property. It also increases the overall speed of getting your no-code application live and into the hands of end-users.

2 Secondly, no-code offers a unique opportunity to gather user input much earlier in the process. In traditional development projects, you often must wait for user feedback until the end of a sprint or until you have a functioning application, which can be fairly late in the development process (perhaps during user acceptance testing). This often results in delays because you may receive key feedback late in the project and spend time reworking functionality or adding missing features. In contrast, with no-code development, you can capture user feedback early and iteratively (as you will nearly always have a working app, even at the start of the project), providing the ability to incrementally course correct and accelerate your progress.

3 Finally, there tends to be more fluid movement between ideation/design/build activities because of the inherent power and ease of collaboration with visual no-code tools (the idea or design becomes the app itself, not simply a throwaway documentation artifact). This typically results in a more effective rapid collaboration between the no-code team and the business stakeholder. It also means that the feedback you collect from end users during the building process doesn't have to be put into a future backlog that may not receive attention for weeks or months later. Rather, your no-code teams can often take the input they received during daily/weekly feedback sessions and address it the same day. Note that scope management is still important, and you need to ensure you're applying the same criteria for defining the MVP as discussed in Chapter 10. You can often address smaller feedback relevant to the MVP almost immediately.

Breaking It Down

Let's start to examine in more detail what the Go-Live Phase actually looks like. As with the Design Phase, this is composed of four key activities. Each of these will be described more fully in the following chapters, but let's briefly discuss the essential stages:

5 | Prototype-to-MVP

Stage 5

Prototype-to-MVP

This begins the process of building the MVP functionality defined in the prior stage. We will continue the philosophy that we should use the no-code visual tools to perform as much of the work as possible. This means that throughout the design and building of the MVP app, the activity will continue to be captured visually inside the no-code platform (as opposed to making specifications in separate design documents). This results in a very efficient, lean, and iterative approach. At this stage, you are taking the loose prototypes defined in prior stages and fully fleshing them out so that they fulfill their intended purpose. The result is a completed solution. This may include completing the data model, workflows, and all the required user forms as well as building any required reports, etc.

We will manage the process using Kanban as an Agile methodology. This allows you to visually track and manage the flow of work through a series of development stages. Unlike frameworks, such as Scrum, Kanban can be layered incrementally on top of your team or processes. It also allows you more flexibility to release updates when they are ready vs. having to adhere to strict sprint or release planning structures that take a fair amount of training and experience to get right. Finally, we'll discuss the important topic of software testing, which should also be adapted and streamlined because the no-code environment abstracts much of the traditional need for unit or infrastructure tests and allows the quality focus to stay at the business and user requirements level.

The no-code creators will take on a significant leadership role as the building begins. These creators will primarily come directly from the business group (function or unit) that has chartered the application or from the business analytics group. For more complex projects (as assessed using the Application Matrix) that involve a fusion team delivery model, software engineers may be integrated into the project as well.

o-Live

6 | Feedback Loop

7 | Governance Checks

Stage 6

Feedback Loop

As discussed earlier, we'll walk through how to capture feedback throughout the development process using an approach to continuous feedback. This will build on the earlier feedback you captured during the Design and Prototyping stages. The feedback becomes more specific and focused on the current release use cases as the focus is shifting from broad design thinking to a more concentrated view for the MVP. We'll also discuss how to select the right stakeholders to give feedback, the right frequency of feedback, and how to efficiently approach change management of user feedback.

Stage 7

Governance Checks

This is a critical step of the review process to ensure your app has successfully passed the applicable checklists and is ready for production release. This typically includes reviewing the following:

1. External compliance checklists to assess compliance with external laws, guidelines, or regulations imposed by government institutions, industries, and organizations.
2. Internal compliance checklists imposed by internal audit teams or committees to enforce adherence to rules, regulations, and practices as defined by internal policies and access controls.
3. Security checklists to protect your corporate information resources from external or internal attacks.
4. Data governance checks to assess how sensitive corporate data is managed and secured.

-Live

8 | First Release

Stage 8

First Release

This last stage is where the application is released to production to end users. The release process is typically straightforward in modern no-code platforms — they adopt the “continuous deployment” philosophy and use automation to deploy features quickly and seamlessly across environments in an on-demand fashion. However, there will be variations in the number and type of environments as well as in the specific steps of the continuous deployment workflow. The scale and complexity of the release will be driven by the Application Matrix, which helps define the appropriate level of sophistication needed. Finally, there are associated operational/support readiness activities and end-user onboarding/enablement activities that will be needed for the first release of the application.

This is a lot to cover, but we’ll step through each of these stages in the following chapters, and offer guidance, examples, and practical tips that can be used along the way.

Final Takeaways

Don't lose sight of the destination when building no-code software: releasing your MVP as quickly as possible. One of the key benefits of no-code development is the ability to streamline the steps of traditional software development so that you can build your app faster for your end users. No-code platforms offer the opportunity to reimagine key parts of the software development process. Rather than simply following a traditional software development approach, embrace the potential of no-code and take full advantage of its power by going live with an exciting app in days or weeks, not months.

In the next few chapters, we'll take a lap through each of the activities in the Go-Live Phase, beginning with the first stage in building the no-code application: Prototype-to-MVP!

"Don't let perfect be the enemy of good"

– *Voltaire*

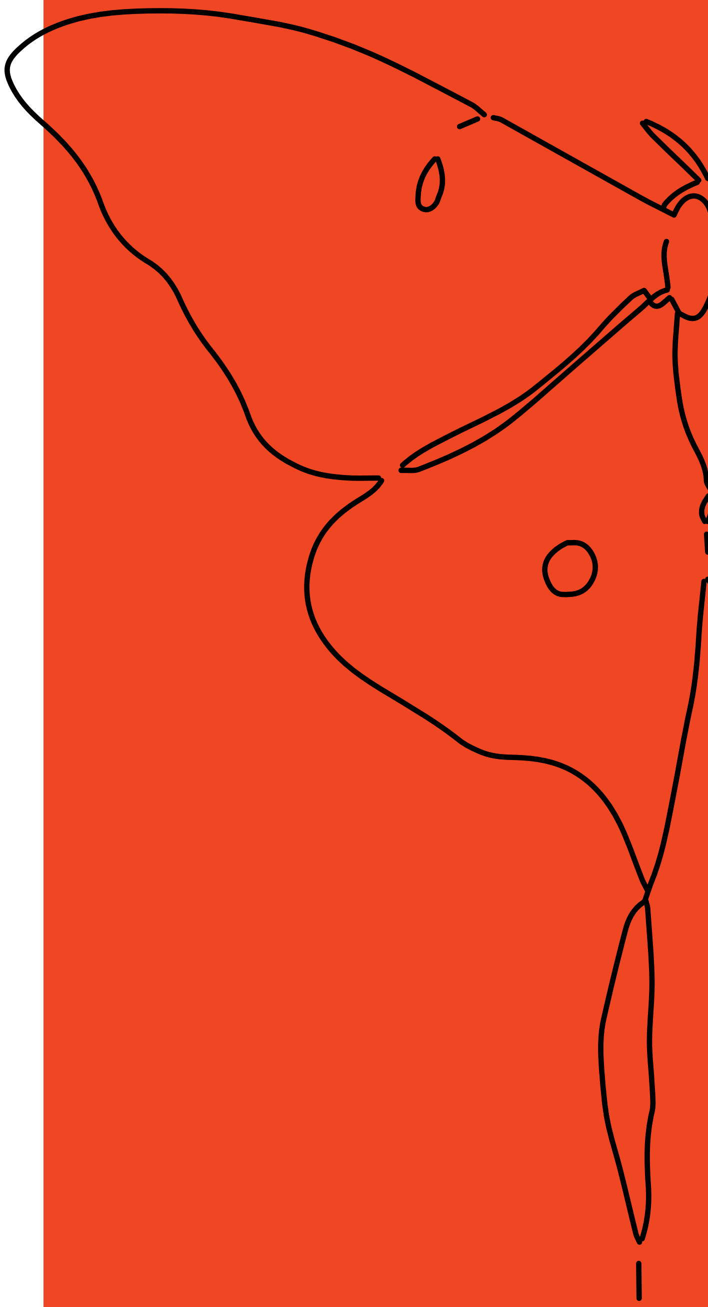
Stage 5

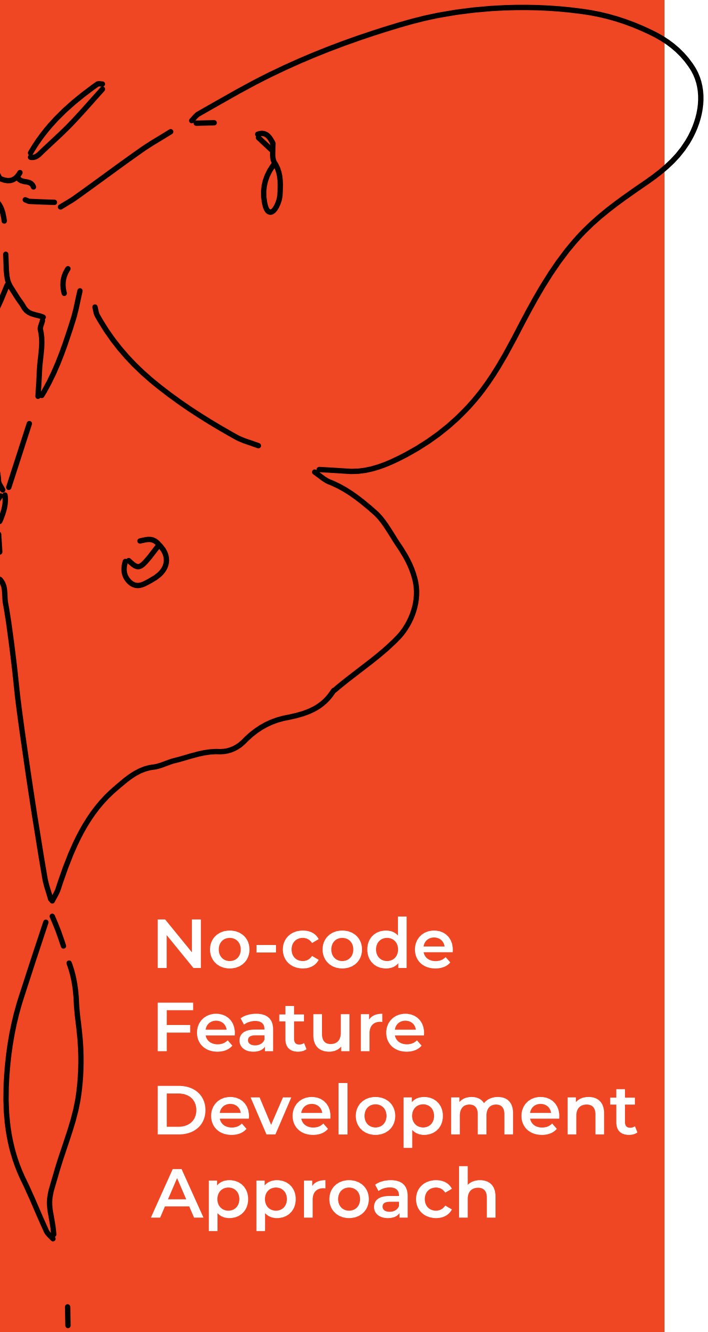
Prototype-to-MVP

12

Classic software development methodologies sometimes overfocus on managing risk and uncertainty, leading to many additional steps in development and QA on the quest to strive for perfection. If you're building software to power a rocket ship that flies to the moon, this is completely understandable! It can take years to ensure the software's completeness and perfection. However, in the business world, the pressure to avoid disruption (think Uber, Netflix, etc.) demands greater speed and requires the ability to release apps and features swiftly so that you can respond to competitors or innovate your business processes faster. This demands a focus on meeting the needs of the business and maintaining a fast pace. Your initial release should focus on key capabilities that can be deployed quickly to enable the business to move forward. Perfection can be achieved in later stages following the everyday delivery approach that we will describe later.

No-code development helps meet these pressing challenges by accelerating the development of both the initial release and enabling rapid and continual updates. The Prototype-to-MVP stage starts by focusing on the rapid delivery of the initial Go-Live release, which was defined by the prior Project Assignment stage.





No-code Feature Development Approach

We continue following the philosophy that we will leverage the no-code visual tools as much as possible. This means that the MVP design steps will be performed visually inside the no-code platform (as opposed to making specifications in separate design docs). Unlike traditional software development, we aren't starting from scratch. First, no-code platforms provide a significant amount of the underlying infrastructure. This allows you to focus on the business functionality that you want to build, rather than worrying about building technical and application frameworks. You also accelerate the process by "inheriting" the working prototype that was built during the earlier Design and Prototyping stage. We start by extending the available prototype, which allows us to save a lot of time and minimize the risks of misalignment.

Another benefit of working from a prototype is that needed features can typically be built almost immediately and incrementally on top of the existing app prototype, allowing you to avoid lengthy delays waiting for a reviewable version of the app. As new micro use cases are added, you can usually publish the working prototype for quick review and feedback. This enables a much more immediate and continuous feedback cycle throughout the building process.

MVP Structure

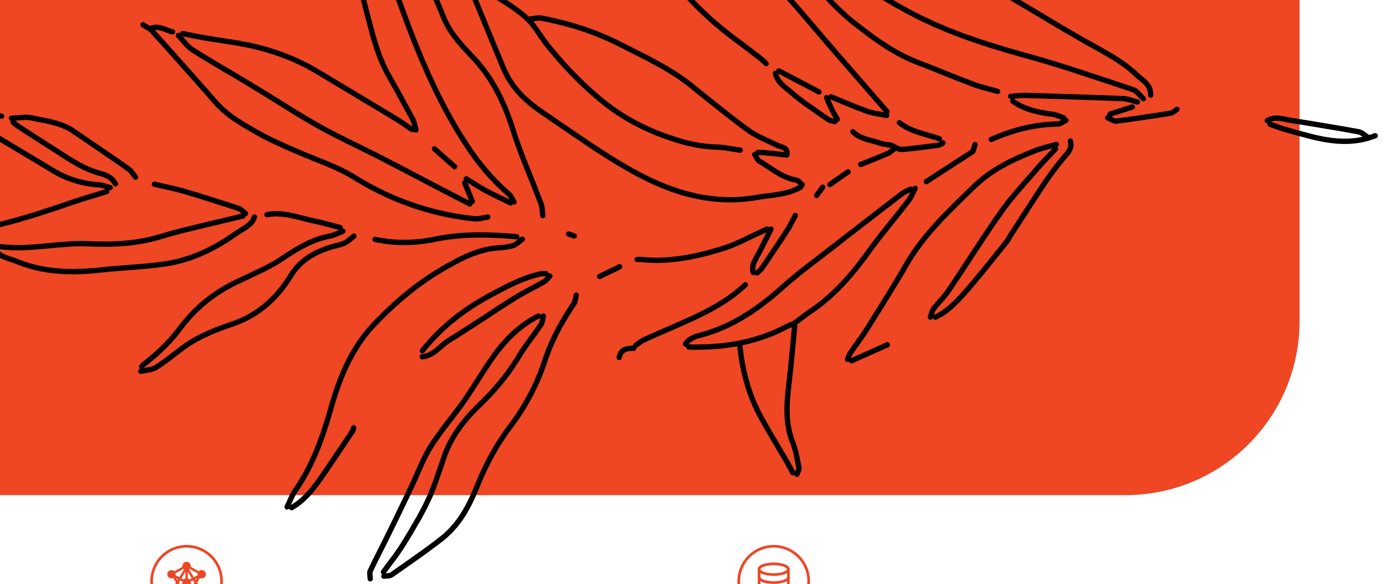
What is included in the no-code feature development? Similar to before — during the prototyping phase — it will typically focus on these key areas:

UX/UI

The basic UX will have been established by the prototype but will now be extended by adding individual features/micro use cases. This will add more depth and completeness (by adding forms or screens needed to complete a full process) and desired usability enhancements (e.g., streamlining the user flow throughout the application by combining steps or automating the population of fields). It will also add additional business rules and validation logic needed for the MVP. Finally, the UX will also likely undergo some form of brand/UX review and may require updates to ensure consistency with overall corporate standards. Also, during the earlier prototype stage, we mentioned that data models should not be in focus.
At this stage, it's time to establish the right data structure and the necessary dependencies between the objects.

Workflows and business logic

You will extend the business processes (both human and system processes) that were defined in your prototype by adding more detail around subprocesses and tasks. This will also typically introduce more nuances around "edge" conditions and process variations that might be suggested by different departments or business units. Error conditions must also be anticipated to handle cases where there are failures or process exceptions. Allow these to be handled gracefully.



Integrations

During prototyping, the internal systems that needed to be integrated with the no-code app are often "stubbed" for simplicity but, in this phase, the actual integrations will be introduced. If data is accessible in systems with preexisting connectors or modern APIs, this may be simply a no-code configuration. Often, however, this may require some participation by software engineers as part of either a fusion team or perhaps sourced through the Center of Excellence.

Dashboards and analytics

At this point, you need to extend the dashboards and reports you've built in the prototype so that they include real data and cover key aspects that are needed to analyze the performance of the business use case. Keep your focus on adding key dashboards with defined success criteria to the MVP as this helps streamline the adoption of the app, specifically by the leadership team.

Keep in mind that while the no-code tools will be used to define the core of the no-code application, they may be integrated with other components of the solution that you identified during the Options Analysis stage (e.g., with either packaged application functionality or with custom-developed components). This will introduce the need to use other development tools as part of a fusion team approach. However, always remember that the goal is to utilize no-code capabilities wherever it's possible. Professional software development tools should be used only in cases when no-code development cannot be applied.

Kanban progress tracking

The ability to break down larger functionality into a stream of small features/micro use cases is powerful but, to be successful, it requires a deliberate approach to manage the flow of work effectively. We recommend the Kanban approach, which we believe is optimally suited for no-code development. Kanban is one of the common flavors of modern Agile software development, but its origins trace back to advances in lean manufacturing pioneered by Toyota in the late 1940s. It is a "just-in-time" approach to managing complex workflows. There are a lot of primers on Kanban available for review. We don't want to spend time duplicating these guides, but we do want to make sure you understand one of Kanban's core practices: The use of visual boards that breakdown work into a set of "cards" and "columns" (e.g., requested, in-progress, done). This is a highly visual approach to managing and tracking work items that allows the development team to pull work from a defined backlog (to-do list) and easily track and communicate updates. Kanban also provides both the development team and key business stakeholders with visibility into the scope and progress at any time, which facilitates collaboration and discussion about priorities.

Many software development teams may be familiar with Scrum (a type of Agile methodology). However, we recommend Kanban over Scrum for no-code development for the following reasons:

- Kanban builds a continuous "push" delivery model where teams release value as soon as they are ready, while Scrum organizes work in Sprints and defined Release Trains.
- Kanban also offers the flexibility to use the method on top of existing workflows, systems, and processes without disrupting what is already in place. Scrum can require adopting a nontrivial number of new systems and processes, which can be overwhelming for a business-led no-code development.
- Finally, Kanban minimizes the need for practical experience and specialist roles (e.g., Scrum master, product owner), which also makes it easier and faster to adopt by business teams.

Software development cycles

(If needed for a fusion team model)

We keep emphasizing that you should be focusing on using no-code tools to fully exploit the power of the platform. Using no-code tools wherever possible drives greater productivity and lowers the total cost of support and maintenance. You'll want to revert to custom development only when you have specific needs that call for the fusion team model. Don't forget, the Application Matrix will have identified your appropriate delivery mode type. You may have included software engineers on your team (as part of a fusion team delivery model) or you may seek software engineer support from the No-code CoE.

Depending on the size of the custom software components, this may dictate some of the processes that the software engineers may follow. For small development scenarios, it may be easiest to simply have them join the cadence of the no-code team and use the Kanban Method outlined earlier. For larger software development projects, the software engineers may identify the need to follow other traditional software development methods, such as Scrum, to manage their work before it gets integrated into the no-code app.

Scope and change management

As you further build MVP features, it's common to continue to elaborate on lower levels of depth of use cases and identify additional requirements (based upon the feedback techniques that we'll discuss in the next chapter). However, while it's good to be responsive and agile, you must also be disciplined throughout this process of managing scope and applying change management practices. Clearly identify and track new feature requests as they are identified and maintain a focus on deciding whether they are in for the MVP.

Here are suggested criteria for deciding which features should be included in the MVP:

- **Timeline.** In the prior stage, you established the timeline for the MVP and allocated resources accordingly. It's recommended to adopt a timeboxed approach for hitting this date. As you identify and elaborate scope items that could possibly exceed an expected MVP timeframe, the no-code team should reprioritize the defined scope and target fewer features and shorter time-to-market.

- **Business value.** The end users should be able to start using the MVP capabilities and receive the expected business value as outlined by the business use case. Any newly identified features that are not critical to supporting these business outcomes should be deferred to the backlog for future consideration.

- **Crystal clarity of the use case**. The ability to execute is important, and the MVP should include only those capabilities that are fully aligned, clarified, and don't require additional research and approval. Vague and not fully confirmed requirements should be taken out of scope and deferred to the backlog until they are clarified or matured.

The most important criterion in the no-code development process is "aggressive" decision-making and prioritization processes during the MVP scope. If wrong decisions are made or incorrect prioritization is allowed, the MVP release timeline could possibly be extended to months, dramatically increasing opportunity cost. When you are working on the scope, it's critical to focus on the capabilities that can be released within days or weeks to deliver value to the business. So, while analyzing each feature, the no-code team should be asking themselves the following question: Can the business get value without it?

If the answer is "yes," then consider it out of the scope for the MVP release and move it into the backlog for later consideration. The team should also focus first on features that are easy to implement relative to the business value they provide. Prioritize features that are low-cost to implement and provide high business value.

Software testing

Testing is important to ensure the quality of software, but the approach to your testing methodology does change a bit with no-code. Traditional software testing methods anticipate that many software defects can be introduced during the low-level coding process and set quality gates and validation steps to identify and remove them. Defects will certainly still need to be removed from a no-code app, but the abstraction layer of the no-code platform tends to provide guardrails that help prevent you from making lower-level technical defects.

During the current Prototype-to-MVP your focus should be on testing end-to-end scenarios that validate the end-user features and user journeys to be delivered in the app. Depending upon the number and complexity of the integrations, you may also need to invest time in API testing, especially for newly created data exchanges. Note that the user acceptance test is important but not in scope during this stage. It will be covered later as part of the "First Release" discussion.

What happens to unit testing? Well, in no-code, traditional unit testing activities are not typically needed. Unit tests are short program fragments that are written and maintained by the developers that exercise some narrow part of the product's source code and check the results. The outcome of the unit test either passes or fails. Developers will typically write many unit tests to cover the application modules or features, grouped into an overall test suite. This can require a significant amount of development effort but is a key part of maintaining the quality of a custom app. In no-code, however, there are few (if any) original "units" of code. Instead, each of the smaller components of the app is usually a prebuilt unit that is supplied by the no-code platform and assembled into higher-level functional components (e.g., forms, workflows, etc.). This allows teams to put focus on validating business rules and end-to-end user scenarios rather than having to unit test for minute defects that were accidentally introduced during coding.

Likewise, technical infrastructure and environment testing are also abstracted away from the application developer. You do not need to worry about running these test scenarios because once the no-code platform is established, it typically hides all the infrastructure and most of the environmental configurations away from the developer.

Finally, one last difference in no-code testing is that it tends to also be performed more incrementally, rather than waiting until the entire MVP is complete. You don't defer the end-to-end testing to the very last minute. No-code encourages conducting scenario testing as soon as they are ready to be validated. This doesn't eliminate the amount of testing, but it often allows you to spot defects earlier in the process.

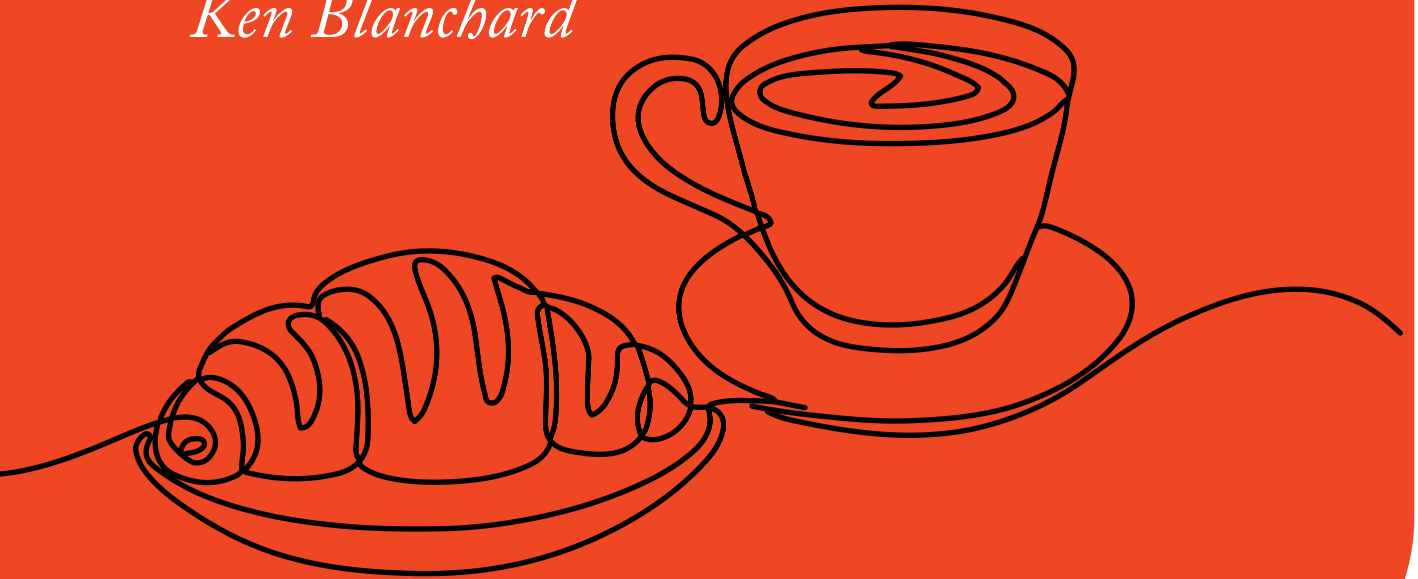
Final Takeaways

This was a lot to cover, but “don’t lose the forest for the trees.” The most important thing to take away is to stay focused on releasing the MVP as quickly as possible. It doesn’t have to be perfect — it shouldn’t be because the quest to find perfection will distract you from the core mission. Instead, it will help if you embrace the power and speed of no-code platforms — take advantage of them by adopting an efficient, lean, and iterative approach that fully uses its unique strengths in accelerating development. Speed is essential to keeping up with the demands of the business, and a no-code platform can help you address these needs.

Now, concurrent with the MVP development discussed in this chapter, you should also be collecting feedback. We’ll explore this parallel activity in the next chapter.

“Feedback is the breakfast of champions”

Ken Blanchard



Stage 6

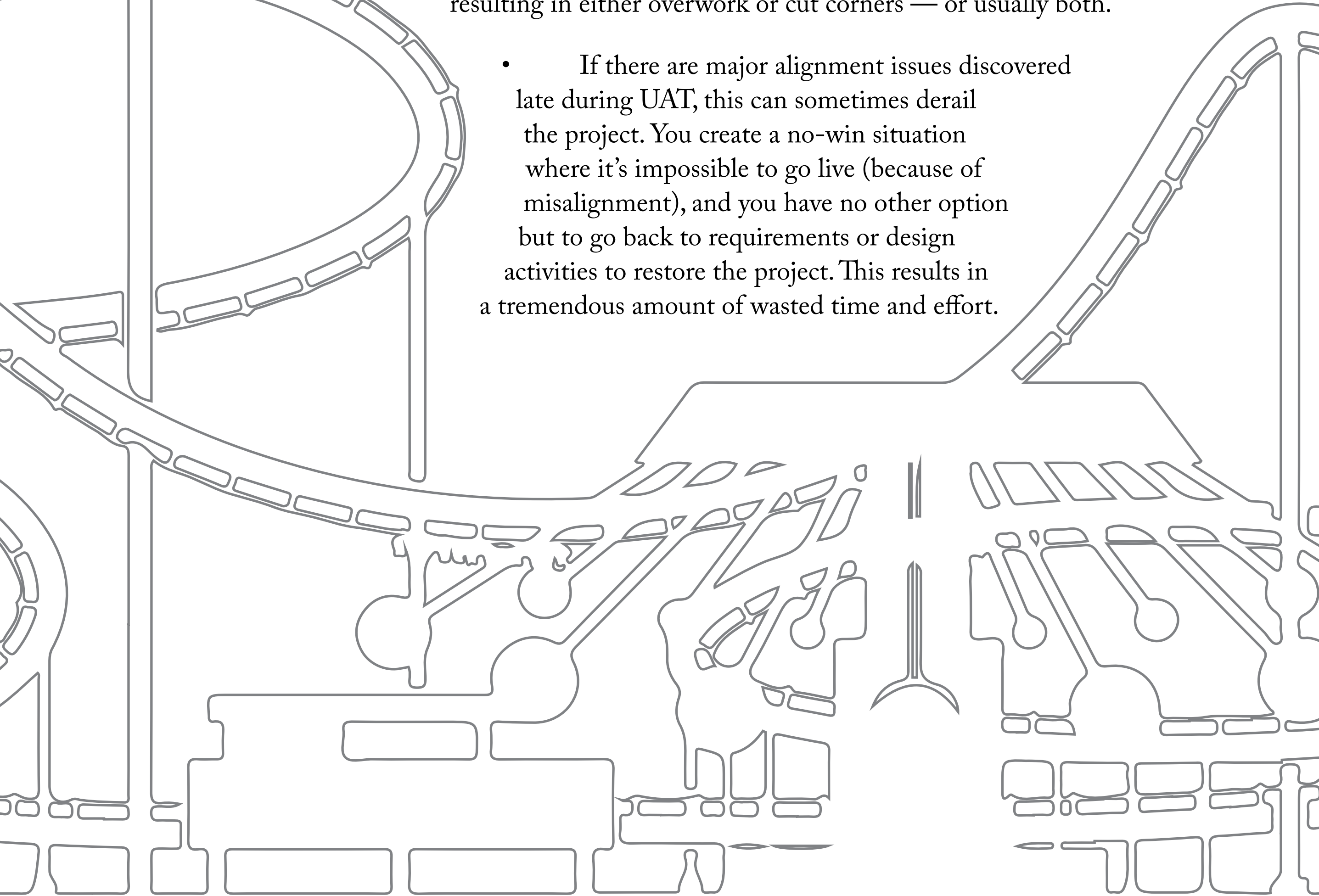
Feedback Loop

13

When it comes to being successful in business, the impact of collecting (and responding to) customer feedback cannot be underestimated. Continuous focus on listening to your customer and responding to their needs will drive higher levels of customer satisfaction. The same is also true for being successful with your no-code application — your ability to listen (and respond) to quality feedback from your users and stakeholders will significantly improve your odds of exceeding expectations.

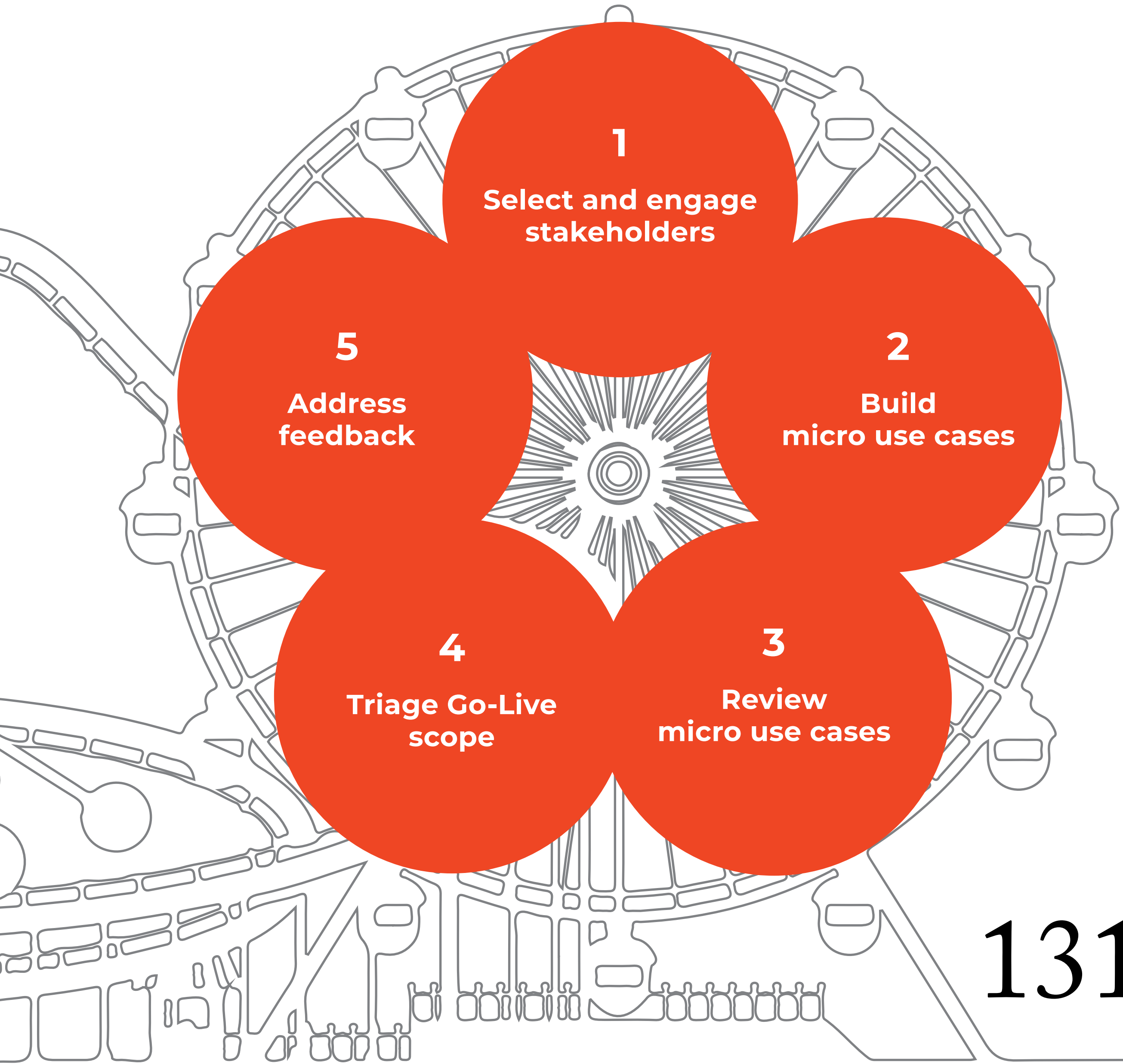
Now, in traditional software development, user feedback is typically part of a user acceptance test (UAT), which comes at the very late stages of the development lifecycle. This classic approach to UAT has some typical challenges:

- It delays user feedback until too late in the cycle, when it can be expensive to address. There is always pressure on the need to "go live," resulting in user feedback being ignored or deferred.
- UAT is always rushed and time-compressed. If there are delays in finishing the software, it may delay turning the software over to the UAT, but there's still pressure imposed on the business function to release the software, resulting in either overwork or cut corners — or usually both.
- If there are major alignment issues discovered late during UAT, this can sometimes derail the project. You create a no-win situation where it's impossible to go live (because of misalignment), and you have no other option but to go back to requirements or design activities to restore the project. This results in a tremendous amount of wasted time and effort.



Rather than delaying feedback to the end, no-code allows you to take advantage of a more continuous feedback model (as shown in the following figure). Instead of feedback only happening at one point in time, we collect constant interactions and approvals by the stakeholder to ensure alignment throughout the development process. This may happen as often as a few times a day! So, this stage actually runs concurrently with the prior "Prototype-to-MVP" stage.

We'll briefly touch on each of these steps and provide guidance on performing these as effectively and efficiently as possible.



Step #1:

Select and engage stakeholders

One important consideration is selecting both the most effective and also the right number of stakeholders to include in the feedback process. While more feedback is always good, identifying the wrong stakeholders (those who lack the right domain knowledge or perhaps are not authorized to make decisions) can waste a lot of the team's time. Also, the complexity of reconciling user feedback increases geometrically when there are more "cooks in the kitchen." So, try to avoid having extended teams of stakeholders to define and approve the scope as this usually leads to a more complicated process and an extended timeline.

Therefore, it's recommended to minimize the number of stakeholders involved in the MVP. Ideally, you'd nominate a single stakeholder/decision-maker who would represent the user population. This will streamline the decisions and prioritization of scope during both the initial MVP and also during the ongoing feedback prioritization. The stakeholder you select should have a good understanding of the business process and also have sufficient authority to make decisions related to the functionality. Finally, while the stakeholder does not need to be deeply technical, they should have an understanding of technology and its ability to impact the business.

Note that while you may limit the number of official stakeholders, it is highly recommended to include users directly on the no-code development team. This provides a way to properly embed a lot of practical and operational knowledge of the business function into the no-code development itself, ultimately improving quality and increasing the odds of better alignment with the no-code stakeholder.

Steps #2 and 3:

Build and review micro use cases

As discussed in the last chapter, building the no-code app can be done highly incrementally as you add new micro use cases into the application. This allows you to "work in" review sessions throughout development. Reviews with your no-code stakeholder to demo the current version of the prototype can be performed on a completed micro use case without a need to wait for either MVP or sprint completion, which is the case for Scrum. As soon as the use case/ features are completed to a point that they can be demonstrated, you can start to collect feedback in parallel with continuing development in other areas. Generally, you should plan for feedback sessions at least a few times per week to keep up with the needed pace of no-code development. The feedback session should focus on reviewing the prototype and discussing business requirements for micro use cases.

We also previously discussed the use of the Kanban Method to manage the building of new micro use cases. In addition to helping organize and structure the building process, the Kanban Method also provides benefits for continuous feedback. The Kanban Method is all about promoting transparency, encouraging feedback, and holding regular review meetings, making it ideal for Feedback Loop activities since your stakeholder will have full visibility of the work being reviewed and where it fits within the broader MVP scope.

Step #4

Triage Go-Live scope

The success of the Feedback Loop stage lies in constant collaboration between the no-code team and stakeholders. It's possible to collect feedback from the user and tweak the current no-code app almost immediately with these capabilities to gain alignment and progress with the development.

That said, as you progress, it's also essential to stay focused on the MVP and not veer too far off track. Prioritize any new requests for capabilities that arise during the feedback sessions with the same approach you used to define the MVP. If something is critical to the MVP and can be accommodated in line, then it's best to address it immediately. However, all requirements that have not been included in the MVP should be placed into the post-MVP improvement backlog that will be reviewed and processed in the Incremental Improvement stage.

Step #5

Address feedback

The process of collecting and addressing feedback doesn't stop at the initial release. You'll continue to address feedback that you receive from end users in production and follow a similar process for responding to it. This will be discussed more in the Feedback Collection stage (in Chapter 17).

Feedback Loop Benefits

There are many benefits of this continuous feedback loop model — let's briefly review a few key perks:

- **It enables addressing problems more quickly.** If you identify missed requirements or incorrect use cases using this approach, the no-code team can focus on resolving issues while development is still underway. This reduces the risk that problems will continue to proliferate and snowball, and you'll be better positioned to avoid negative downstream effects.

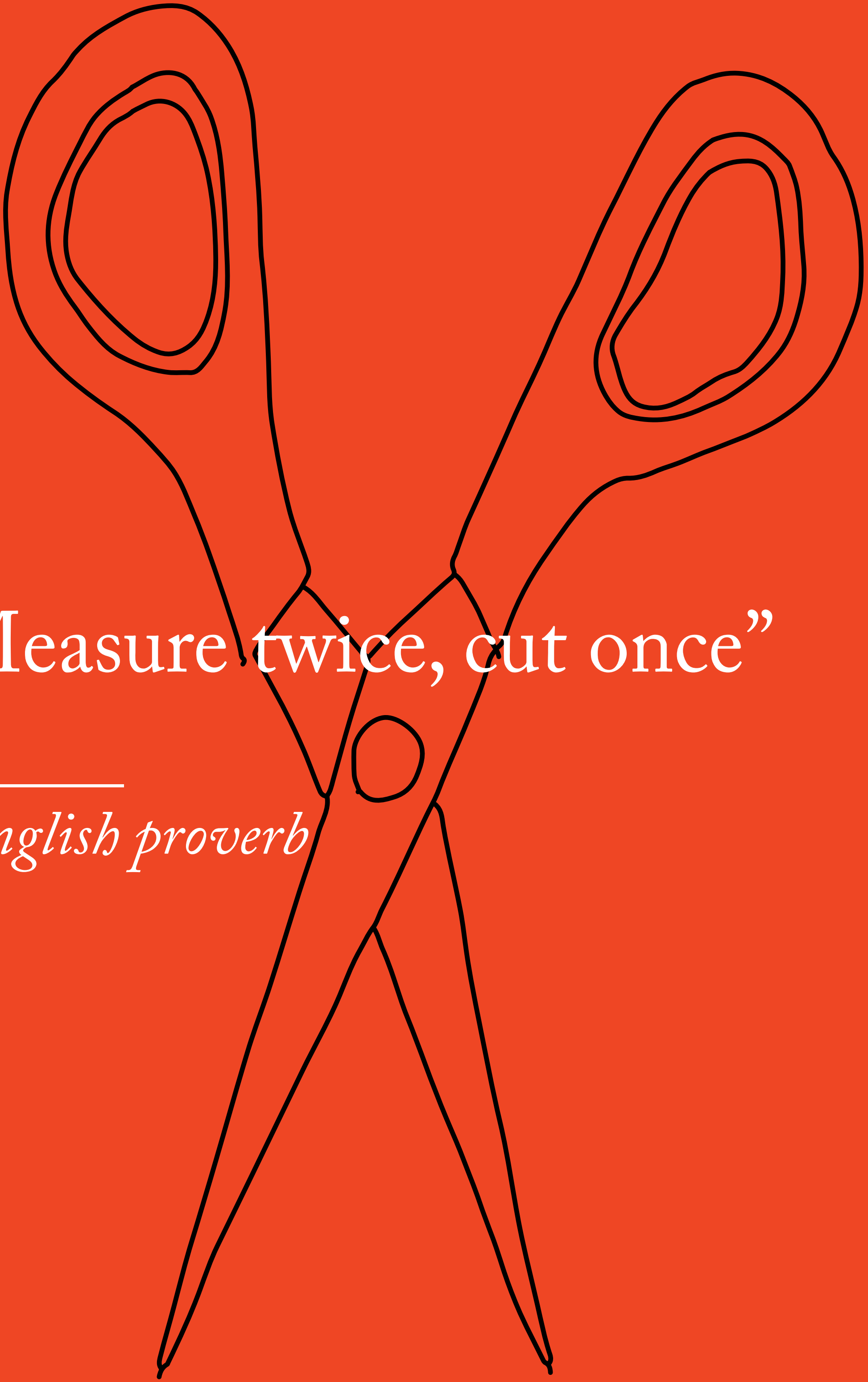
- **Improved (re)alignment with the business process.** If you've done a good job with the earlier Business Use Case stage by documenting the core business vision and success factors, then you're off to a good start as it relates to alignment with the target state process. However, even with this alignment, it's still possible to "drift" as the project moves into greater complexity and detail. You may find that you're not completely addressing the highest-priority use cases in the way that was intended. To ensure you're delivering on the needs and expectations of the business function, you must adapt and realign your understanding of the business priorities and scope as you go, which requires collecting continuous feedback and making course corrections.

- **Getting frequent feedback encourages engagement.** You want your no-code stakeholder (and the business function more broadly) to be deeply committed to the project. It's essential they feel a part of the project team, even though they may not be in the development activity itself. Getting frequent feedback from your stakeholder — and genuinely incorporating it into your daily changes — strengthens communication and collaboration in both directions. It also demonstrates how much the no-code team values their input in the project delivery.

Final Takeaways

In no-code development, gathering feedback should be an ongoing and continual process. This allows you to respond to feedback more readily and ensures alignment with stakeholder needs. You won’t know for certain what the user wants unless you ask — and the sooner (and more often) you ask, the more successful you will be!

You’ve finished development and revised it with a continuous feedback loop — you’re almost ready to release. However, before you deploy, it’s important to ensure the app has met the required governance and compliance reviews. We will discuss this in the next chapter.



“Measure twice, cut once”

English proverb

Stage 7

Governance Checks

14

Just because no-code allows for rapid, incremental updates does not give you a pass to ignore governance requirements. Proper technological governance is essential for maintaining security and governmental compliance at all times. The cost of skipping governance checks can have significant impacts on the business's function — becoming noncompliant can result in fines, settlements, business disruption, productivity, and revenue loss. Furthermore, the damage to a business's reputation can be irreparable. So, as the proverb goes, "measure twice, cut once" to make sure the cut you are making is the one you want! Make sure you have thoroughly reviewed your application for governance requirements before releasing it into the wild!

Now, let's begin by identifying and reviewing some of the more common types of governance you will encounter:

1. **External compliance.** Checklists to assess compliance with external laws, guidelines, or regulations imposed by external governments, industries, and organizations.

2. **Internal compliance.** Checklists imposed by internal audit teams or committees to enforce adherence to rules, regulations, and practices as defined by internal policies and access controls.

3. **Security.** Checklists to protect your corporate information resources from external or internal attacks.

4. **Data governance.** Checklists to assess how sensitive corporate data is managed and secured.

External Compliance

We'll start by looking at external compliance reviews. These are reviews conducted by a designated entity to assess whether your application complies with the laws, guidelines, and regulations set by external governments, industries, and standards organizations. Usually, this will apply to applications that contain sensitive customer data, especially healthcare-related information or financial data (e.g., credit cards and bank accounts). The external auditor will usually work with someone in IT or Operations, so be sure to build time for these reviews during the Project Assignment stage.

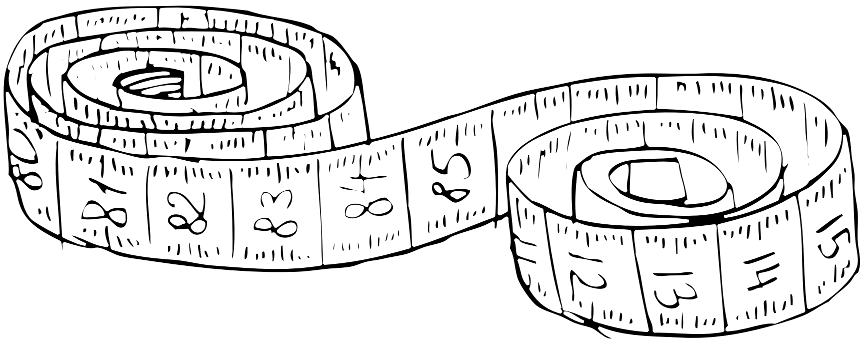
The range of external compliance standards can be quite varied depending upon industry or geography. The following is not a complete list but provides a representative set of examples of compliance requirements that may be applicable to your application:

General Data Protection Regulation (GDPR)

If you are building an application that will be capturing and manipulating the personal contact information of European Union (EU) citizens (for example an application used by sales and marketing teams), you need your application to be approved by the Data Protection Officer.

Health Insurance Portability and Accountability Act (HIPAA)

If you are in the healthcare vertical and the application is touching patient-sensitive information, the no-code team should be working with a relevant approver who is ensuring that the appropriate data privacy and security regulations are being met.



Payment Card Industry Data Security Standard (PCI DSS)

If you are building an application that will be touching sensitive financial information like credit card data (e.g., customer case management application in financial services).

Know your customer (KYC) and anti-money laundering (AML)

If you are building an app where a financial institution will be onboarding new clients, you may be subject to regulations that require you to ensure that no monies you are receiving have come from criminal or terrorist activity. Your compliance officer will need to ensure that checks are performed to specifically verify the identity of your customers and investors together with their financial activities and any risks they may pose.

These are just some of the more common external standards or regulations that are specific to your industry. Some of these may apply at the platform or data center level (like SSAE 16) but not require being reviewed per individual application. In other cases, app-specific reviews may be required. So, it's recommended you collaborate with IT, your IT security team, or with the CoE, if applicable, early to identify the relevant external governance checks that will be applicable to your app, when they must be performed, and begin planning early to prepare.

Internal Compliance

Internal compliance reviews develop an independent assessment of the effectiveness of an organization's risk management, processes, and general governance. Unlike external compliance, these reviews are not mandated by an external entity or legislation. Instead, they are the organization's own way of performing internal quality measurement and management. The goal is to collect accurate information internally about the team's performance, governance, and risks.

Here are a few examples of internal audits you may encounter:

Management audits

These focus on assessing whether a team or the company as a whole is hitting targets related to the goals set by management. These audits are typically conducted by an internal audit team and could include a review of the team's quality control processes or compliance with company legal policies or diversity standards.

IT audits

These focus on the infrastructure, technology, and systems that an organization has put in place. These may be assessed by IT or by an external entity. This type of internal audit may help you prepare for some of the specific security or data governance reviews discussed in the later sections. An example of IT audits could include reviewing internal systems for their use of sensitive customer data or sensitive HR-related information. These audits might also be used to identify data conflicts between different teams.

One very important part of an IT audit is user access management. Having the right set of user access permissions defined for your app can be challenging given changing roles and responsibilities and the regular onboarding and offboarding of employees. An IT audit (typically an internal assessment by IT) will define controls for user access management to both protect the organization from risk and help decrease costs and inefficiencies. User access management controls help ensure that users only receive authorized access privileges, meaning they only have access to what's required to do their job. Should their role or responsibilities change, their permissions should be promptly revoked.

Operational audits

These have the widest focus of the internal audit types as they are concerned with assessing the efficiency and effectiveness of your organization's internal controls. Typically, the auditor focuses on high-risk areas that present a threat to the company if something goes wrong. For example, operational audits might review whether you've provided too much (or incorrect) information to the customer as part of a sales process. They could also look at whether teams are accidentally not complying with how invoices and payments should be reconciled. All of these are examples of internal audits that can be performed in-house if the person or team conducting it is trained in the field. Your internal auditors need to be able to gather evidence and reach their conclusion free of outside influence. Thus, internal compliance checks are typically performed by an internal audit team that is independent, objective, and free of influence from the team or department being assessed.

Security

Security risks and cyberattacks are increasingly becoming a board-level priority for most organizations, especially with security threats increasing because of the pandemic and the shift to hybrid work environments. Approximately 80% of security and business leaders say their organizations have more exposure to cyber threats today as a result of remote working[1]. It's more imperative than ever before to prioritize security reviews and take preemptive steps to protect against significant security threats.

Typically, an organization's chief information security officer (CISO) and/or security department will have defined a standard collection of processes and technologies that work together to help strengthen a company's overall security profile. Adherence to these standards will be assessed during a security review. A security review should be a collaborative process between the security team and the no-code team to identify security-related issues, determine the level of risk associated with those issues, and make informed decisions about risk mitigation or acceptance.

While there are well-defined security checklists for software development, such as the "OWASP Top 10" guidelines[2], they are not as applicable for no-code development as many of the common security mistakes can be prevented and automatically abstracted away by the no-code platform. To be clear, proper security practices must still be followed, but this is the responsibility of the no-code platform vendor. No-code platforms can significantly improve adherence to security guidelines since they enforce a more standard way of building and deploying software. They remove the opportunity for developers to accidentally write insecure code. Instead, no-code development reduces the risk of insecure apps by enforcing more consistent usage and app design patterns than traditional software development. Note that additional security reviews will be required the first time a no-code platform is implemented to validate the security profile of the platform. But subsequent use of the no-code platform to build individual apps will likely be streamlined because they will follow a consistent pattern.

However, even if the no-code platform handles some of the more common and basic security practices for you, it's still critical to work closely with IT and, if applicable, the CoE to understand what reviews and checklists still apply and ensure early scheduling with the appropriate security team to avoid delays.

[1] Beyond Boundaries: The Future of Cybersecurity in the New World of Work, Forrester

[2] OWASP Top Ten, OWASP Foundation

Data Governance

Data governance is the final category of governance. Data governance is a defined approach to data and information management that is formalized within an organization as a set of policies and procedures. These governance checklists encompass the full life cycle of data — from acquisition to use and disposal. Data governance checklists are essential to control how sensitive corporate data is managed and secured. This will include considerations on data governance, access rights, quality, and managing risks around data loss.

Many organizations have recently gone through initiatives to comply with the GDPR. While this is a regulation specific to the European Union, many global companies are using it as a framework to help drive better customer data management practices.

The no-code team will typically work with data owners and the data governance group to review how your organization's data policies will impact the app they're building. Such reviews are typically focused on security and privacy protection, data quality, access, sharing, dissemination, and security and risk management.

Governance Considerations

Regardless of the type of governance check, there are some common considerations and best practices to keep in mind:

Properly scoping the effort

Many of the governance checks (especially internal ones) may require varying levels of effort and time commitments based on the criticality of your application. Some apps may not require many reviews while others may need significant time investment. The exact scope and necessary governance checks should be defined using the Application Matrix during the Project Assignment stage. At that point, the no-code development team should engage the required roles for each type of check. Later, when you are preparing for the initial release, the no-code team should confirm that none of your iterations with new capabilities have impacted the Governance complexity requirements. They can do this by applying the Application Matrix to the completed MVP app (as it has iterated).

Planning ahead

Once you've identified the applicable governance checks, then the no-code project team (usually supported by IT or the CoE) will work with an internal audit team or external auditor to perform the review. The frequency varies: most require annual certification, but others may be dictated on a different schedule. Reviewing for governance should not be an afterthought in the development process. While the final reviews for governance may need to wait until after the app is completed (near the end of the Go-Live Phase), you should have already scheduled the required reviews and started collaborating with the designated approvers during the Project Assignment stage.

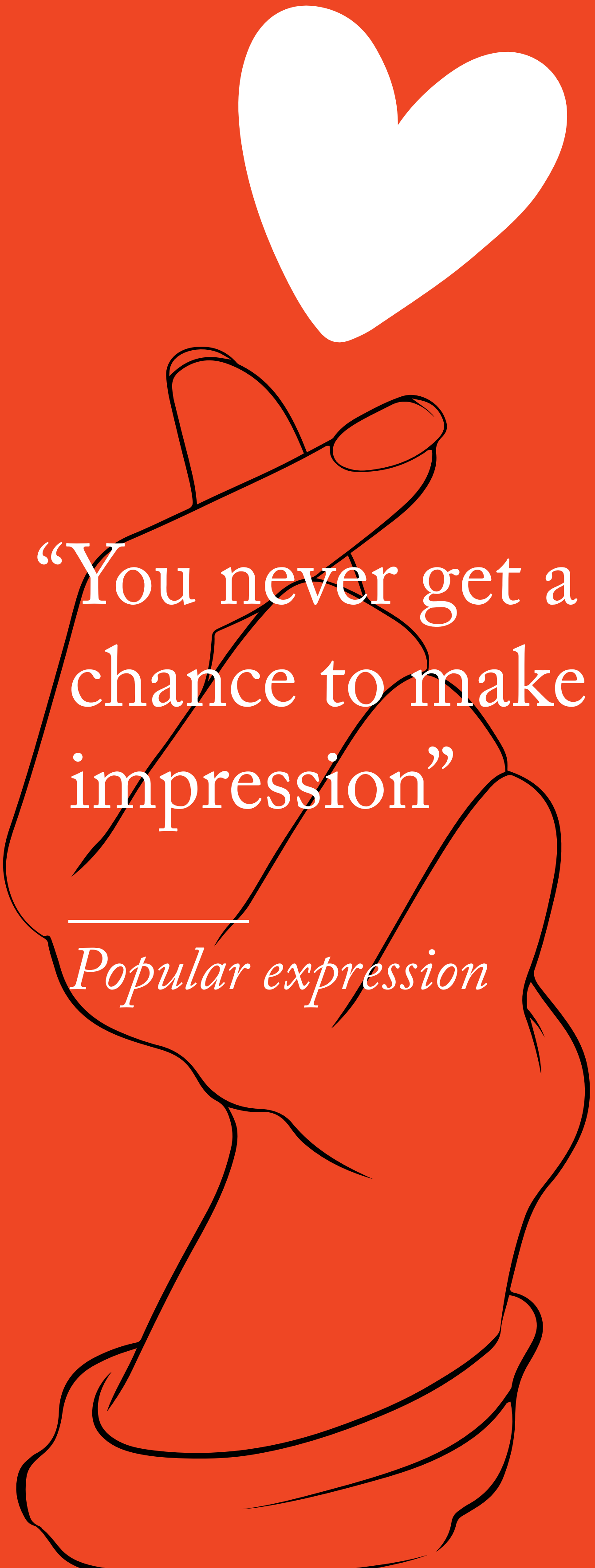
Early collaboration

One of the other benefits of scheduling governance early is that it facilitates very early collaboration with the appropriate approver stakeholders (typically in IT or Ops) to anticipate and mitigate significant issues. Ideally, you should proactively be consulting with the identified Governance team from the beginning of the Go-Live Phase, so that you streamline and reduce downstream issues found later in the lifecycle.

Final Takeaways

Keeping up with the speed of the business is important but so is ensuring the proper governance of the application. Don't move so quickly that you fail to meet security, compliance, and data governance checks. Speeding through these compliance checks could result in significant business penalties and consequences, potentially eclipsing the gains achieved through the no-code solution. While you may feel pressure from stakeholders to release the app as soon as possible, doing it without heeding governance could result in a business loss that could be painful and expensive to remediate. Don't overlook the essential governance checks that will measure your success.

Congratulations! Now that you've finished your governance reviews, you're ready to release to production! But wait, don't forget to consider the essential final activities of a successful release that we discuss in the next chapter.



"You never get a second chance to make a first impression"

Popular expression

Stage 8

15 First Release

Congratulations! You've passed the governance checks, and you're ready to deploy the initial MVP release of your no-code app! You're almost done, but don't forget some of the most essential final considerations. You moved fast and have almost completed the journey of building your first app, so don't stumble at the final stage. Taking these final measures to prepare for release will help set up your app for success and ensure it makes a good first impression with your users.

The stages we covered in the No-code Lifecycle have ensured the application has met all of the stakeholder, user, and governance requirements. However, there are still some final steps to prepare the business function to use the application, including the following:

- Documentation and application guides
- Deployment
- Validating environments
- Final user acceptance
- Support and monitoring
- User onboarding and enablement

We'll briefly review each of these and offer some tips and guidance on each.

Documentation and application guides

Documentation and application guides can vary widely depending upon the complexity of the application. For a simple app, it may be enough to have a walkthrough guide for a first-time user, whereas for a business-critical enterprise application, you may need full-blown documentation that describes all features, functions, and operations. Also, consider that you may want to put more user support and online documentation into the app itself through in-app guides, hints, tips, and walkthroughs to support the user navigation. If the app is simple and intuitive with support for users as they get going, they may not need to rely on traditional external documentation.

It's worth noting that the building of documentation is often started late during traditional custom software development as the teams may not have access to working software until late in the development cycle. In contrast, with no-code development, there are early "working" versions of the app even during the prototyping stage. While the app will progressively evolve over time, you can begin planning the development of needed content even at the early stages of the lifecycle.

Deployment

With modern software development, there has been a trend toward more complete automation (referred to as “continuous deployment”) of the deployment activities to take manual steps and points of error out of the equation. Most no-code platforms depend on continuous deployment approaches as part of their automation (this set of functionalities is often called Application Lifecycle Management or ALM) to allow for more rapidly moving no-code applications through environments. When an operator needs to move a no-code app to production, this usually involves nothing more than approval and a click of a button.

However, moving an app into production still must be performed with care and in concert with appropriate validation steps. Deployment should be released from quality assurance/pre-production to the production environment only after the final acceptance testing has been performed. Then, following the release to production, it’s important to still verify the solution changes operate as expected before significant users or customers are onboarded.

Preparing and verifying environments

Until now, most of the focus has been on development or QA activities as you’ve largely been working in a small number of nonproduction environments. But it’s important to carefully plan out your needs regarding the types and numbers of environments so that you can respond to change. Once the full set of environments is provisioned, you’ll need to test and verify that the environments are ready.

Typically, you may have several of the following environments provisioned, although the exact number and configuration will vary based upon the application complexity.

Sandbox (SANDBOX)

This is an environment used for quick experiments and demos that won't affect the work of other developers on the team. The sandbox can be used for early ideation and prototyping before the MVP release officially starts.

Development (DEV)

This is where the primary development for the current MVP release is performed. You may have multiple teams depending on the size of the application but, ideally, they are still all working in the same environment.

Quality assurance (QA)

As feature development is completed, it may be moved into a separate QA environment to allow for more detailed and controlled testing (e.g., integration or system testing).

Staging/preproduction (PREPROD)

Once testing has been completed, the app will often be moved into a controlled staging environment that closely mirrors production. Staging will mimic similar volumes of data, and system resources will be sized to mimic production response times. This is often used for final performance simulation and user acceptance testing.

Production (PROD)

This is the actual production environment, where end users will directly be onboarded and use the live application.

You should use the Application Matrix as a general guide to determine which (and how many) of the environments may be required:

- "Simple" apps may only need two environments (DEV and PROD). With simple complexity apps, you might use a single environment for both building and testing and then deploy it into production when you are ready for Go-Live.
- "Medium" and "Advanced" apps typically represent more business-critical applications and may require additional environments (e.g., multiple QA environments to support parallel releases to be tested at once, a PREPROD production mirror). Having more environments adds some incremental complexity, but it also allows for more flexibility in testing — you can conduct final UAT, smoke testing, and regression testing in controlled environments before being released without impacting the development underway. It also allows fixes to be made rapidly in PREPROD and released without having to worry about the accidental release of work-in-progress changes.

Final user acceptance

As discussed earlier in the Feedback Loop Chapter, users should be involved throughout the process to supply continuous feedback. Incorporating user feedback along the way makes the final end user approval a less scary event (e.g., will it fail to reach signoff?) and ensures that most of the buy-in and alignment has happened along the way. However, final user acceptance and stakeholder approval should still be attained prior to the release. Consider this a final check to make sure that the no-code stakeholder and business function are indeed ready for the app to go live. Note that this is especially important when the Application Matrix has assessed business complexity as “medium” or “high.”

The focus of the final user acceptance should be to validate the highest-level business requirements (as defined in the business use case) and obtain approval to release. This is not meant to be a substitute for lower-level functional or user testing, which should have been validated during the prior Feedback Loop stage. This will help eliminate issues resulting from data or environmental inconsistencies.

Be very careful to manage expectations on feedback gathered during this final user acceptance effort. The primary focus is to validate that the app is ready for deployment in production, not to gather more feedback on possible enhancements or improvements. If any critical “release blocker” defects are identified, it may be necessary to hold the release so that they can be fixed. However, at this point, the bar should be very high for accepting any new requests. It’s essential to triage the feedback and only accept the most critical items at this late stage. Everything else goes to the post-MVP backlog and will be addressed as part of the continuous improvement cycle.

Support and monitoring

Your organization will typically adopt Information Technology Infrastructure Library (ITIL) and IT Service Management (ITSM) methods and practices as the overall model to provide support for the application. However, depending on the level of application complexity, your organization might be leveraging all 10 processes of ITSM or just have one part (i.e., an individual providing ongoing help for simple apps). Monitoring is a form of proactive support where an organization can set up a relevant process to monitor critical indicators of the application's health to respond to events in real time. Such indicators may include time to execute critical actions, the opening of pages, and other triggers.

Note that, unlike custom development, no-code platforms should abstract the individual app teams away from having to manage the health and operations of the underlying platform infrastructure. This would include things such as maintaining a disaster recovery plan, performing backups of system data, ensuring service redundancy, monitoring loads, and stress testing. This is critical at the no-code platform level and will likely be supported by IT working closely in concert with the no-code vendor. However, the individual no-code teams should not have to perform these activities for each app that is built on the no-code platform.

User onboarding and enablement

Once a new no-code app is deployed, you're ready to give users access to the app! Some of the user setups may be done automatically, for example, existing employees or customers can just log in and go. In other cases, users may need to take some action the first time they use the application. However, even if user onboarding is mostly automatic, you may want to control the pace of bringing new users to the new system. Consider whether phased or staged onboarding is needed to give the business function time to gradually adjust to using the new solution. If a phased approach is desired, you can break up deployment by region or team, for example. If some remaining defects exist in the no-code app, having a controlled onboarding ramp allows for easier fixing of initial issues before significant numbers of users are live on the new app.

Even if the software is working perfectly, process or operational changes may be needed. For example, you may learn from the earlier users that enhancements to training or procedure documents will help them be more effective in their daily use of the no-code app.

Support teams may not be fully ready to handle the volume of inquiries about using the new application. Giving them an initial period of light usage can help with training and ramping up knowledge about the new app.

As end users are onboarded, there is also usually some associated enablement activity required to support effective adoption. Besides traditional training and enablement techniques, it is recommended to adopt a model of continuously retraining the users based on the "Everyday Delivery" approach (more frequent smaller enablement is more effective with an app that is frequently being updated). It won't be needed to set up training every time when new functionality gets deployed. It will depend on the size and the impact of the change. However, for frequently changing apps we would suggest setting up regular monthly "what's new" reviews. Also, it is suggested to use certifications and testing to confirm their ability to fully utilize the system specifically for complex enterprise-grade apps.

Final Takeaways

Releasing your app to production can be exciting, but don't overlook the final critical steps. To be successful, you need to secure signoff from the business function and prepare both users and technical environments for the release. The final details matter — and how you introduce your app to your users will make an impression that colors how they judge the success of the app. It's worth the extra effort to follow these final steps.

You've done it! You've successfully finished the journey to MVP. However, now the real work begins as you manage the continuous evolution of the app. We'll discuss this as we start the final Everyday Delivery Phase of the lifecycle.

“To succeed in this world, you have to change all the time”

Sam Walton

Introduction to Everyday Delivery

16

No-code Differences

The ability to continually evolve is critical for your business and, therefore, your no-code app as well! Now that you've delivered the first release, this isn't the end. In fact, this is just the beginning. It's critical to follow up on the first go-live release with a disciplined approach to enable constant feature evolution. You don't need to wait to batch your evolution into a big release. Ideally, your updates should be delivered to users as soon as they are ready. Response to user requests can be delivered in small, quick, incremental updates on a frequent basis — perhaps daily. In this chapter, we're introducing the final phase: "Everyday Delivery."

The Everyday Delivery Phase provides value delivery in small, quick, continual updates. This addresses some of the common challenges of traditional software development. Previously, software development projects often attempted to comprehensively document all requirements upfront and then spent months methodically building and testing each of the required features until the fully completed application was ready. This lifecycle is organized in sequential waves — analysis, design, coding, testing, and implementation. Hence, it is referred to as a "waterfall" approach. Waterfall lifecycles maximize the potential value to the stakeholder by saving up features into one big release (perhaps quarterly or annually). However, the risks of the final release delivery can be higher, and the sequential handoffs make it difficult to adapt to changes that may occur during the development releases. It also means that end users must sometimes wait a fairly lengthy period to receive enhancements they may have requested.

Agile methodologies for custom development have popularized taking a more incremental approach, whereby you break down larger releases into smaller releases of features. Depending on Agile's flavor, each release's duration varies. The Scrum version of Agile typically defines shorter "sprints" of two to three weeks in duration. However, not all sprints may be releasable to end users as the features built in each sprint may be a part of a broader user scenario that takes more time to fully complete. So, Agile can often lower delivery risks, improve quality, and also address the challenge of responding to changes since the team can adapt and change the plan during each sprint. However, it does not necessarily build out features any faster than traditional approaches. Users will usually get value delivered sooner, but they will still have to wait until the next sprint or iteration is complete.

Everyday Delivery builds on concepts from Agile but does not force you into a strictly defined release duration. Instead, you can release when you are ready and, ideally, as fast as you can. You can get feedback from your stakeholders and end users, and you can respond more quickly. Development is significantly accelerated thanks to the power of no-code capabilities. As a result, new features can be built and released more quickly. This creates an opportunity to release new micro use cases to end users in small incremental steps — ideally every single day.

Breaking it down

Let's examine in more detail what Everyday Delivery Phase actually looks like. As with the previous two phases, it is composed of four key stages. Each of these will be described more fully in the following chapters, but let's briefly discuss the essential steps.

9 | Feedback Collection

10 | Incremental Improvement

Stage 9

Feedback Collection

This stage ensures you have a model in place to collect regular feedback from your stakeholders and end users so that in later stages you can respond to their input quickly, systematically, and incrementally. We will provide guidance on how to adopt an approach that encompasses both user feedback (backed by data) and system feedback so that you aren't letting your no-code team get too biased by "the squeaky wheel" or infrequent users.

We will offer a brief review of some of the most common feedback collection techniques — from standard techniques like email surveys and focus groups to more advanced ones. We'll review typical mistakes and suggest best practices on how to engage and interact with different groups.

Stage 10

Incremental Improvement

This stage starts with developing a constant stream of incremental improvements based on the feedback collected in the prior stage. We will discuss it using a five-step continuous improvement model to guide a systematic approach to developing the enhancements. Proper planning and definition are important to make sure the work is decomposed into micro use cases. Once defined, the use cases will be developed through the continued use of Kanban to manage the stream of work. We'll also discuss important considerations around software testing and governance that apply to this stage.

livery

11 | Everyday Delivery　12 | Application Audit

Stage 11

Everyday Delivery

This stage is all about delivering ongoing releases of functionality as soon as they are ready for deployment to end users. Unlike traditional waterfall or Scrum methodologies, the use of the Kanban Method promotes release when ready. We'll outline the important components of the Everyday Delivery approach and offer key tips/practices. Properly defining and scoping enhancements into small, granular updates will help with both dependency management and conflict resolution. We'll also discuss considerations to keep in mind when planning user rollout and enablement.

Stage 12

Application Audit

This final stage provides ongoing support and management for the developed no-code applications to ensure their optimal health and fitness over time. This stage includes a range of key activities including:

1. Auditing app performance to remove feature "bloat" and ensure proper usability and system responsiveness.
2. Auditing the app to watch out for obsolescence of specific features or even the end-of-life of the app.
3. Reviewing changes in the broader business process or organization that may require changes to the app.
4. Auditing for changes in internal or external governance requirements that may introduce changes into the governance processes of the application.
5. Reviewing app components to assess for opportunities to harvest functional or technical components that can be reused more broadly across other apps.

This is a lot to cover, but we'll step through each of these stages in the following chapters, along with guidance, examples, and practical tips that can be used at each step along the way.

Final Takeaways

No-code platforms offer the opportunity to constantly deliver value in smaller increments. Just as we used a different approach for the Go-Live Phase, it is important to embrace the methodology for the Everyday Delivery Phase tailored specifically for no-code platforms. You'll be in a better position to win a competitive advantage when you can constantly respond to user feedback, customer interests, and market changes.

So, let's dive into more details, beginning with the first stage in the Everyday Delivery: Feedback Collection.

"We can't just sit back and wait for feedback to be offered, particularly when we're in a leadership role. If we want feedback to take root in the culture, we need to explicitly ask for it."

Ed Batista

Stage 9

Feedback Collection

17

The first Go-Live release is a huge milestone, and it's easy to "take your foot off the gas" and allow things to coast, but now is the time when you need to keep the momentum going strong! This starts by collecting feedback. Feedback is an important stage as this is the first time when the no-code app will likely start having daily usage by real end users who are fully vested in the app working because it enables them to perform their job function. They will let you know when the app isn't working right! View all types of feedback — even if it is critical — as a good thing because it will help you continue to improve and enhance the app.

It's now a great time to start building a foundation for achieving perfection and make your app state of the art. By collecting and accommodating feedback from multiple dimensions, you empower yourself with data and information to tailor the application to perform at its best. The feedback provides you with actionable areas of improvement and opens up a lot of "blind spots" that you now can address. Getting a balanced approach to feedback collection is important. Here are some tips and best practices for establishing an efficient process.

Feedback Basics

First, let's start by identifying three types of feedback inputs that we should be focused on collecting at this stage:

Stakeholder feedback

The feedback from the no-code stakeholder is essential as they are ultimately chartered with defining success for the app, which should have been outlined during the Business Use Case stage. How do they personally view the app based upon direct use? What feedback have they received from the teams using the app? Keep close to the no-code stakeholder — and perhaps cross-functional leaders in other groups that use the app — to keep a pulse on their feedback.

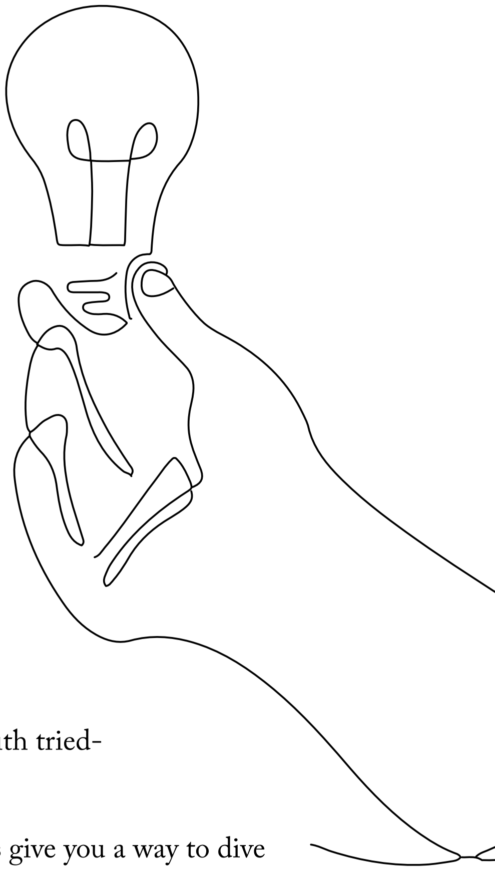
End user feedback

You need to look beyond the no-code stakeholder, however, and also gather feedback from the frontline users who use the app frequently (ideally daily). By listening to end users, you can collect a much higher volume of feedback and higher quality of insight since they are vigorously using the app.

System feedback

Finally, extracting and collecting system data from monitoring the app is important because it will help provide a highly objective view of UX and performance, and it may help you identify feedback in areas that could go unnoticed at first by end users. It will also help prevent feedback bias that may be skewed to a subset of highly vocal end users.

You'll want to proactively develop a process for collecting a blend of data from each of the above feedback types (we'll review some of the more common techniques and tools in the next section). Relying on just a single form of feedback can result in blind spots in your understanding of true system issues or areas for improvement. The feedback you collect from different sources will also help you validate/test the accuracy of the feedback.



Feedback Techniques

The two most common forms of gathering feedback usually start with tried-and-true classics: email surveys and focus groups.

- **Email surveys** give you a broad way to collect feedback from your no-code app user base. Usually, they are easy and low-cost to prepare and distribute, especially when using common self-service survey-building tools. However, getting a high response rate may be a challenge given that your user base is focused on getting their job done and may not be able (or willing) to pause whatever critical tasks they are performing to give you feedback. If you send out a survey, keep it short — ideally, less than 7 or 8 minutes in length — focused, and personalized.
- **Focus groups** give you a way to dive deeper into your end user's feedback, and they are a good complement to email surveys. A well-run focus group will give you richer qualitative feedback, helping you understand the "why" behind other feedback you've collected. It's important to select the right set of users, though — identify a cohort of experienced users who will give you a rich source of insight.

Here are some additional techniques that you should consider to receive a more diverse and well-rounded feedback on your no-code app.

	User Shadowing	**UX Questionnaire**	**Feedback Widget**	**Net Promoter Score**	**Usage Analytics**
Applicability	Depth research on customer satisfaction	Breadth research of customer satisfaction	Reporting bugs and collecting real-time feedback	Researching customer satisfaction and looking for improvement opportunities	Gathering actual usage of used features and user activity
Advantages	Greatest level of data about actual user behavior	Unobtrusive, user-friendly, and may include diverse questions	Activated by users; gives users a place for leaving feedback	Simple, one-click design; quick follow-ups available	Unobtrusive; does not require user explicit action
Drawbacks	Time-consuming, requires effort to set up and conduct	Long forms are usually unattractive to users	Targeting is limited to pages where the widget is implemented	Can be skewed to really happy (or annoyed); does not provide a lot of detail	Requires more work to synthesize and interpret

- **User shadowing** is a qualitative feedback technique. It is conducted on a small scale using a defined sample of users who are observed (often by someone from the UX team) while using the no-code app just as they would in real life. The user is observed for a set period (ranging from 30 minutes to a few hours, depending upon the scope of the process). Ideally, the observer tries not to interfere with the user to avoid deviating from their natural practices of using the no-code app.

- **UX questionnaires** complement other types of user surveys. They focus on app usability and seek to understand the user's attitudes and preferences. Was using the no-code app the first time easy? Which parts of the app did they find confusing? This type of survey should avoid diving into the depth of features and, instead, focus on impressions of usability. Often, a UX questionnaire is structured around user personas with feedback being grouped and analyzed with a cohort of similar user types.

- **Feedback widgets** are one of the most convenient methods for gathering real-time, quality user feedback. Placing a feedback widget into the app itself — often at the far right or bottom of a form — allows you to interact with your users within the app, which helps improve both the completion rate and relevancy of feedback.

- **Net Promoter Score (NPS)** is a widely used market research metric first created by Bain & Company to predict customer loyalty. However, you may also find it helpful to use this approach for internal employee surveys. NPS surveys avoid survey fatigue by focusing the feedback on just a single question that is quick and easy to answer. It helps you keep your fingers on the pulse of how satisfied users are with the experience and whether they would recommend the app to others, such as coworkers for an internal app.

- **Usage analytics** gather empirical data on actual app usage. This is important because it helps avoid subjective bias in the feedback you may collect from other sources. Usage analytics can often be collected from reporting provided by the no-code platform or sometimes through other analytics add-ons. This can give you accurate information on actual feature adoption, product friction points (i.e., areas where users get slowed down or halt entirely), product stickiness (how frequently users return to the app), and overall user engagement.

No one of these feedback strategies will give you a complete picture, so it's recommended to adopt a multipronged feedback approach to how you gather, analyze, validate and gain insight into the real-world use of your no-code app.

Key Mistakes to Avoid

As you implement various programs of feedback collection, be aware of these common pitfalls:

Avoid the "squeaky wheel" syndrome

The loudest users may not be the best or most representative of the user population. Apply a structured approach to analyzing the overall feedback (from all three feedback types) to prevent input from a few users biasing your view on priorities or decisions.

Avoid infrequent app users

While collecting usability feedback from casual users is important, they will probably not give you the same depth of insight on functionality as "power users" who have demonstrated frequent activity.

Avoid moving too slowly

Note that the intensity of your feedback collection (and deployment of updates) plays a significant role at this stage. The pace of gathering and responding to feedback should be equal to the Go-Live cycle. It may include multiple weekly sessions to collect and analyze the inputs and get relevant decisions and approvals.

Final Takeaways

Feedback collection is essential to evolving your no-code app, but it's important that you're gathering the right feedback — through a combination of stakeholder, user, and system feedback — while taking a varied approach to input collection. Having a complete view of the areas of improvement for your no-code app is powerful and is enabled by a disciplined approach to continuous feedback collection and using that insight to evolve the application rapidly.

Now, what do you do with all the feedback you've collected? Let's discuss this next!

“Continuous improvement is better than delayed perfection”

Mark Twain

Stage 10

18 Incremental Improvements

It's tempting to focus on achieving perfection all at once — but that is an unachievable goal. Instead of trying to address all requirements, expectations, and suggestions, it's better to strive to continuously improve daily based on real-world feedback. This allows you to make progress toward your goal while also continuously learning and adapting along the way. If you've done a good job collecting recommendations in the previous stage, you now have a mountain of input from your stakeholders, users, and the application itself (system data). You've prioritized it carefully and know the most essential items to tackle first to deliver enhancements to your app.

Consider the example of a young plant growing in a garden. If you only examined the plant every few weeks (or perhaps months), you would notice big differences. It may appear to be an entirely different plant each time you looked at it. You might think the growth was happening in big spurts. However, in reality, the plant grows every day — it's constantly taking in sunlight and nutrients from the soil and growing in small amounts (usually imperceptible to the eye). It never remains static. The plant is adapting to its environment and constantly growing and expanding in response to its surroundings.

Like a living organism, your no-code app also will be more successful if it takes continual, rapid steps to adapt to the needs of the business function. Using the incremental improvements philosophy can also make it possible to significantly exceed the expectations of your stakeholders because you will be able to respond to their requests continuously and more rapidly.

In this stage of the lifecycle, you should focus on the following simple steps:

Step #1
Analyze and decompose use cases

Step #2
Prioritize and approve micro use cases

Step #3
Review design considerations

Step #4
Build/test enhancements

Step #5
Review governance checks

Step #1:

Analyze and decompose use cases

You've set the groundwork for this at the Feedback Collection stage. However, as you gather more feedback across various channels, the detail and completeness of feedback become essential — it's important to analyze it to ensure users are sufficiently documenting their responses and experiences in a way that is clear and reproducible. You'll want to tie any recommendations or decisions back to very specific and observable feedback and data, not just some general perceptions.

Just as in prior stages, it's also very important to fully decompose the recommended improvements into micro use cases. It will help minimize the dependencies across work items and allow the micro use cases to be worked on and released independently.

Step #2:

Prioritize and approve micro use cases

Prioritization of the feedback is also super important at this step. It's expected that you have received a lot of input, and it's vital to organize the right decision-making process based on the value of capabilities. The easiest way to test your prioritization accuracy is to check if the selected micro use cases allow you to achieve the application's business goals (and ideally specific KPIs you've identified during the Business Use Case stage). Also, while you have a backlog of items based on your prioritization from the last phase, it's important to understand that the user feedback you're receiving from the live app will likely give you new and perhaps unexpected learnings. So, you must maintain an open perspective on the backlog as you may decide to change priorities.

Finally, in your haste to act on feedback, don't forget to have the no-code stakeholder authorize key changes before you move any further. Not all feedback is equally important in priority to the business function, so it's essential to review it with the no-code stakeholder. Just as your no-code stakeholder helped you prioritize the initial MVP release, they also need to help with the prioritization and approval of the ongoing enhancements.

Step #3:

Review design considerations

Sometimes changes in the original no-code app design may be necessary after the first MVP is released. That's OK! Needs may evolve and change. Just make sure you think through the implications those changes may have on the design. This is especially true with more significant changes to the process or data models — those are areas that may have ripple effects across the application.

As the team moves through the process, you may encounter new requirements that can trigger returning to the Options Analysis stage and reviewing assumptions. For example, to satisfy the new requirements, you may need to extend your app with marketplace add-ons or connectors. These should be managed carefully because introducing new options into a production app may result in additional costs or other impacts.

Step #4:

Build/test enhancements

After you have prioritized and approved a set of items to address from the backlog, the next step in the incremental improvement cycle is to begin using the no-code tools to build these enhancements. The Kanban Method can continue to be used as a framework for managing work items post-release, as it provides transparency to stakeholders about ongoing progress.

While this step has many similarities to the Prototype-to-MVP stage, here are some additional tips and practices to keep in mind:

- Don't forget to transfer all items you parked during the Go-Live Phase into a unified backlog with all the requirements, including the new ones. You've been managing the scope of MVP carefully up until now, so you have been building a lot of backlog items for the future — don't lose these requests! It will encourage user engagement (and more feedback) when they see the feedback that they shared during the initial MVP addressed promptly post Go-Live.

- It's important that new changes — even small ones — do not break existing capabilities within the app. Make sure your software testing approach addresses new features and revalidates end-to-end scenarios that test existing end user features and user journeys already delivered in the no-code app. This is important to prevent unwanted regressions in features or unexpected changes to app behavior.

- Automation of API testing may also be worth considering if there is a risk of the other applications changing unexpectedly. Those changes could result in significant impact and downtime if they occur while your no-code app is in production. While API automation requires effort, this can also be a worthwhile investment to catch and prevent software defects in no-code integrations (especially for integrations that may be used in mission-critical apps or across multiple apps).

Step #5:

Review governance checks

New requirements may influence and change the application's complexity level, which should trigger relevant actions around governance checks as per the Application Matrix. The new requirements shall be analyzed using governance complexity questions. Please note that new governance checklists may need to be applied. For example, the additional functionality for capturing credit card data for a customer case management app may trigger a need to undergo PCI DSS compliance. Make sure you proactively review and assess if the no-code backlog contains items that may result in new external or internal compliance verifications:

- The roles-based access policies for your no-code app will need to be reviewed periodically to ensure they are appropriately enforced and controlled. It's not uncommon for an app to start with simple user access permissions because the initial MVP functionality may be made available to all users at the start. However, as more specific features are added — especially to address power user requests — it may be decided that some features should be restricted, which may require additional logic and rules to be enforced within the app. As always, you need to be careful with access rights to sensitive data including the ones that might be available for external users outside your organization.

- New security vulnerabilities may be created if you add new integrations with additional systems or expose new user interfaces to new user types (e.g., a self-service portal to new customers or business partners). In general, you should periodically review the backlog of planned enhancements with your security team to proactively identify areas that may require new approvals.
- Data governance checks are still vital to review how sensitive corporate data is managed and secured. As you continue to evolve the no-code app and add new features, it's not uncommon to begin collecting new types of data, which may trigger new data governance requirements.

There is a balance to strike here. You need to ensure that appropriate quality and governance checks are applied without letting the speed of responsiveness grind to a halt!

Final Takeaways

No-code tools enable a mode of continuous incremental improvement that will help you evolve and succeed. As discussed in the last chapter, it's important to maintain the intensity of your actions at this stage. The pace of responding to feedback should be equal to the Go-Live cycle and to build confidence with users that their feedback is being heard and addressed. Nothing discourages user feedback like the perception that their feedback is "going nowhere" or that there will be long delays in seeing results. So, take care to deliver quick and timely updates post-release.

Embrace this model to make your stakeholders and users happy. However, keep in mind not to get too distracted by chasing immediate perfection! Instead, focus on delivering incremental improvements every day. This will be the focus of the next chapter.

“Our highest priority is to satisfy the customer through early and continuous delivery of valuable software”

Principles behind the Agile Manifesto

Stage 11

Everyday Delivery

19

One of the key principles behind Agile is the notion of continuous delivery of value. Yet, too often in traditional software development, we get stuck in having to conform to defined release sprints or iterations. A key differentiation of the no-code approach is accelerated time-to-market. The modern platform capabilities empower no-code teams to deliver sufficient outcomes daily to end users.

The spirit of the Everyday Delivery stage is striving to provide rapid updates to the end user and maintaining a high velocity and ongoing improvement cycle. The deployment can be based on a specific feature or set of features without a need to connect it to a specific sprint deadline or other formal milestones. To establish this mode of continuous delivery, it's critical to appropriately decompose and scope your work items so that you can rely on the higher degree of deployment automation provided by no-code platforms. This allows for quick, small updates to be pushed to production frequently (perhaps daily) while maintaining higher levels of quality than traditional "big bang" software releases.

Let's dive into discussing the Everyday Delivery stage by defining what it is and what you should focus on at this stage.

Defining Everyday Delivery

As the name implies, the goal of the Everyday Delivery stage is to deliver value "every day." At first, this may seem obvious — isn't delivering value the goal of any software development team? Ideally, yes, but we all know it can be incredibly difficult to release enhancements daily, especially when using traditional software development methods. Delivering value at speed requires a fundamentally different approach. But once you adopt the approach that allows daily enhancements, you'll see how it results in incredible advantages, including the ability to change and innovate at a whole new level.

Components of Everyday Delivery

Let's start understanding the common elements that enable this breakthrough release strategy:

No-code tools

As might be expected, no-code tools are at the foundation of the Everyday Delivery strategy. No matter how powerful traditional custom development tools are, just the act of writing custom code means that daily releases are unlikely (lower-level coding and unit testing alone impedes this pace of delivery). So, the speed and productivity you receive from no-code tools facilitate this rapid Everyday Delivery approach.

Continuous deployment

In Chapter 15 (First Release), we initially introduced the approach of fully automating the movement of no-code applications across environments and into production. This remains key to Everyday Delivery as any manual steps in the deployment process will result in additional time, effort, and risk.

Fast rollback

A "rollback" is the ability to revert an environment to an earlier known state if you find that a change is not working as expected. This allows you to undo a problematic set of updates so that users can continue using the application as it was previously. At the same time, you troubleshoot and correct whatever problem was introduced. Rollback can typically be addressed either by the deployment automation in the no-code platform (if it can undo or reverse deployments) or possibly through a dedicated staging environment (mirroring the last production environment before a change is being released). Fast rollback gives you the confidence to move quickly in the Everyday Delivery approach because you know you can always return to an earlier configuration if needed.

Best practice tip:

Here are additional tips and practices for establishing the Everyday Delivery approach.

Appropriate definition and scoping are essential.

It's difficult to maintain a speed of release if the scope of your features is large and cross-dependent on many other features. Focus on micro use cases that can be deployed almost immediately.

Make sure you're factoring in other dependencies.

You need to consider whether other teams are developing components or related features that your application will need and plan accordingly. For example, suppose your feature is ready for delivery and the other remaining features are not done yet. In that case, the team needs to find a way to separate incomplete features (so they can be delivered in the future) and focus on deploying the completed ones.

Use the Kanban Method to track micro use cases.

Using Kanban (explained in Chapter 11) gives you fine-grained control and visibility over the work as it flows through the team. It also provides more flexibility in choosing when to release — Kanban does not enforce a set amount of time required for a release. Each team can release value as soon as they are ready.

When delivering features, follow the same sequence and engage the same set of checks and environments as you did during the Go-Live Phase.

Just because you are moving fast does not mean you get to skip steps! As outlined in the previous chapter, the use of appropriate software testing and governance checks are still required to have confidence in the release.

It's important to remember that Everyday Delivery means users will see changes to the app more frequently. Thus, you should train and enable current users on new features and workflows. Be sure to communicate any impact that updates may have. Here are some user rollout considerations:

- The no-code team should continuously update and retrain end users on the newly-delivered capabilities. It is recommended that you establish a consistent and engaging process for informing and training users on these changes. Avoid overreliance on email, which can be missed or not read as new users join the organization.
- The new feature changes can fall into one of the following categories:
  - System changes/invisible changes (usually don't require informing/training).
  - Changes that are intuitively understandable (require minimal informing/training).
  - Drastic changes that impact the flow and experience (require an effort for informing/training).

- These first two categories may require updates via prompts within the app. More drastic changes that require understanding a new flow or experience should be assessed and enablement as necessary. This could range from a short video demo to in-person instruction.

- The release of a new feature should be in sync with the overall end user experience and impact on their process. Don't change the app before instituting any released process changes (or vice versa). All the changes should be coordinated to be rolled out together, along with applicable user enablement.

- Scoping of feature size is key here as well. Smaller "bite-sized" updates are typically easier for users to consume and lessen the change management impact. Consider aligning more significant changes into batches where user enablement and process changes can be rolled out.

Final Takeaways

No-code platforms offer the opportunity to adopt a model of Everyday Delivery — allowing you to deliver features to your end users daily. This increases user satisfaction (by consistently and rapidly responding to their feedback) and improves user adoption (by reducing the impact of massive changes, instead introducing a smaller set of more easily adopted updates). However, it's important to correctly scope and decompose for a successful release and apply the right model for managing user enablement and rollout.

Now, we're nearly done — we're about to finish off the No-code Lifecycle with the final stage: The Application Audit stage.

“What gets measured gets done, what gets measured and fed back gets done well, what gets rewarded gets repeated”

John E. Jones III

Stage 12

Application Audit

20

Like any living organism, apps will take on a life of their own. They will evolve and change over time. While an app may have perfectly fit all business and organizational needs initially, as the business process changes (and you continue to release daily updates), it should be reassessed periodically to understand whether it is still delivering on the expected outcomes. Therefore, it's essential to establish an effective approach and cadence for measuring the impact and evolution of your no-code app, as well as provide support and management as needed to optimize its health and fitness over time. We'll refer to this as the Application Audit stage.

Initially, the Application Audit is likely performed by the no-code team and is reported to the no-code stakeholder. However, as the number of no-code apps increases and as you develop more business-critical apps, the audit function generally will become part of the responsibilities of the Center of Excellence. This will help look across the organization to measure and improve overall levels of efficiency, effectiveness, and business impact. It will also help identify reusability opportunities (a key concept we'll address shortly). However, regardless of who performs the audit, it's important not to miss any key steps. So, we will briefly introduce some of the key activities that should be included and some considerations for each.

Auditing App Performance

We probably have seen "bloat" happen in our personal health — maybe you eat too many unhealthy snacks, perhaps some unnecessary desserts, and you start putting on some unplanned (and unwanted) weight! This happens with apps as well — over time, it's not unusual for applications to experience "feature bloat" as more requirements are slowly added to the application. The impact may not be that unwieldy at first, but after a while, the aggregate impact of adding requests may result in additional (and perhaps unnecessary) complexity.

Complexity due to aggregate features can manifest itself in the app in several ways:

- First, it can increase the perceived complexity of the existing UX. The app might have been very simple and intuitive at MVP, but now forms or workflows have evolved and expanded and may have many more fields or steps than needed. Reviewing the UX and evaluating parts of the user journey that may need optimizing or refactoring for simplicity are important.
- Adding new features may also begin to impact the responsiveness of the app. It's important to continue closely monitoring performance metrics around the application (e.g., launch times, display times, and moving between pages) and identify areas that may require specialized performance tuning and optimization.

- It may not always be the visible feature complexity that starts to slow an app — it may also be a result of the additional user or data growth that happens over time. These volumes may put unplanned stress on the database or application services. It could be necessary to archive older data or take on more technical optimization at a database level to ensure that performance stays "fit" as your volumes grow.

Your team that manages operations should usually be able to help with the system performance tracking by providing health metrics on performance or downtime as part of their standard reporting. Tracking over time should help identify if there are possible concerns to address. Usability performance problems are sometimes harder to spot. Some of the techniques discussed in the Feedback Collection stage (e.g., user shadowing and UX questionnaire) can be good early warning signs that you may need to refresh or streamline the end-user experience.

Auditing Obsolescence

Another important consideration is feature obsolescence. Over time, some features in the app may no longer be applicable (given changes in the process or requirements) and may need to be removed or refactored. It's easy to forget to continue to review the app periodically for these types of outdated features, and it's also easy to let things go unused for periods. However, the cleanup of unused or unneeded features will help keep the app "fresh" and usable for the user. The audit team needs to analyze all the redundant features (clutter) that have become irrelevant. The ongoing feature cleansing process should be applied. This is important to reduce the "noise" for end users and keep the user experience efficient and productive.

Obsolescence will ultimately also happen to the no-code app itself. It's also OK for apps to run their natural course. Eventually, any application will reach "End of Life" (EOL). At this stage, it's important to retire or decommission the app, which is a natural part of the lifecycle and makes the path clear for new apps and workflows.

The review and decommissioning of obsolescent apps may sometimes be sporadic if left to the no-code team, especially if the original no-code team has moved on to other projects or parts of the company over time. For this reason, the evolution of your apps across the portfolio is often managed/enforced by the CoE. They should be responsible (when existing) for systemic review of the application portfolio to maintain health and relevancy.

Organizational Changes

Another key type of change that should be anticipated is linked to the environmental factors surrounding the app itself. The business may have changed, the process may have changed, or both, so it's important for the no-code app features to be reviewed for needed updates or, potentially, for feature removal. While not always easy to spot, the no-code team (or CoE) will need to monitor to see if no-code app changes may be required.

There may be changes in operational processes that are touching automation. The design of the no-code app has made assumptions about what operational workflow occurs. Therefore, as process change occurs, it may fundamentally change the organization of the workflow and UX within the app as well. It may be necessary to make changes to align better with the new operational processes and procedures.

Changes in organizational structures may also have unintended impacts on the no-code app. Changes to reporting or organization boundaries may trigger user workflow or navigation updates. Splitting teams into another organization may require updates to user access permissions. So, while app changes are often not considered when an organizational change is made, they must ultimately be synchronized so that the app supports the team(s) and employees who will be using it.

It's recommended to review the scheduled proactive audits of the process/organization and not simply rely on reacting to changes as they occur. Depending upon the business complexity (as guided by the Application Matrix), you should schedule more or less frequent reviews to ensure that the app stays aligned with the process and organization. It is advised to schedule most Simple apps for annual reviews, and Medium/Complex apps should likely be checked quarterly depending on the pace of changes they may be facing.

Auditing Governance Changes

In Chapter 14 (Governance Checks) we discussed identifying and proactively scheduling the appropriate governance reviews as a part of the lifecycle or as part of an annual cadence. However, while this has been planned upfront (in concert with IT or Data Governance teams), don't overlook the possibility of changes in external or internal requirements that may trigger you to reassess the type and frequency of checks. For example, there may be changes in federal regulations enacted that make it mandatory to comply with a new set of rules. Or perhaps your internal Data Governance organization adds a new data privacy policy that may require limiting access to data types. Regardless of whether this is externally or internally triggered, the no-code development team will need to review the app for possible new checklist reviews.

As with the process/organization changes, it's advised to collaborate with the Governance or IT teams and have a proactive schedule for routinely assessing which internal or external governance requirements may have changed. Governance checks may need to be done more often considering regulations and legislative requirements changes. The necessary frequency may vary widely depending upon your industry or business environment, as will the cost or impact of accidentally missing new governance requirements. So, you'll need to determine the right cadence for assessing major new external or internal governance updates. Generally, it's advised to set the appropriate audit cycles for governance based on the different complexity/criticality levels as per the Application Matrix.

Auditing Component Reusability

As discussed earlier during the Design Phase, one of the strategic benefits of using no-code platforms is embracing the composable architecture. As no-code apps are built, there will likely be various components developed in each app that fill another need and can be suitable for reuse. So, during the Application Audit stage, you should formalize a process that identifies functional or technical components that might have broader benefits and impact on the organization if reused.

This type of audit can be performed by the no-code team, but as adoption grows across the organization, this will most likely be part of the CoE's responsibility. As part of the CoE charter, it is recommended that you include the following activities to provide a strategic role in reviewing/curating components:

- The CoE should take a leadership role across teams to establish a standard process for assessing the applicability of reuse for functional and technical features.
- The CoE will help promote broader visibility/discovery of available components through establishing an internal catalog or marketplace-type function.
- To effectively identify, curate, and operationalize the component reuse, it may be required to introduce new roles/responsibilities (typically within the CoE) and training around component design, curation, and reuse. The CoE should help with the enablement of teams to assist with the initial use of shared or reusable components.
- The CoE may provide individual application teams additional assistance to harvest/repackage for broader reuse. This can help address concerns that individual teams may not have the staff/budget/skills to invest in the additional effort needed to create a reusable or generic version of a component.
- The CoE should establish standard reporting and metrics to track reuse effectiveness. It's important to avoid a "Field of Dreams" approach (i.e., "if you build it, they will reuse …") — don't assume it will happen but adopt a data-driven approach to measuring the impact of component sharing and the success of demand generation/reuse activities. This will help target areas where you need to provide additional direct assistance or enablement of no-code teams.

Final Takeaways

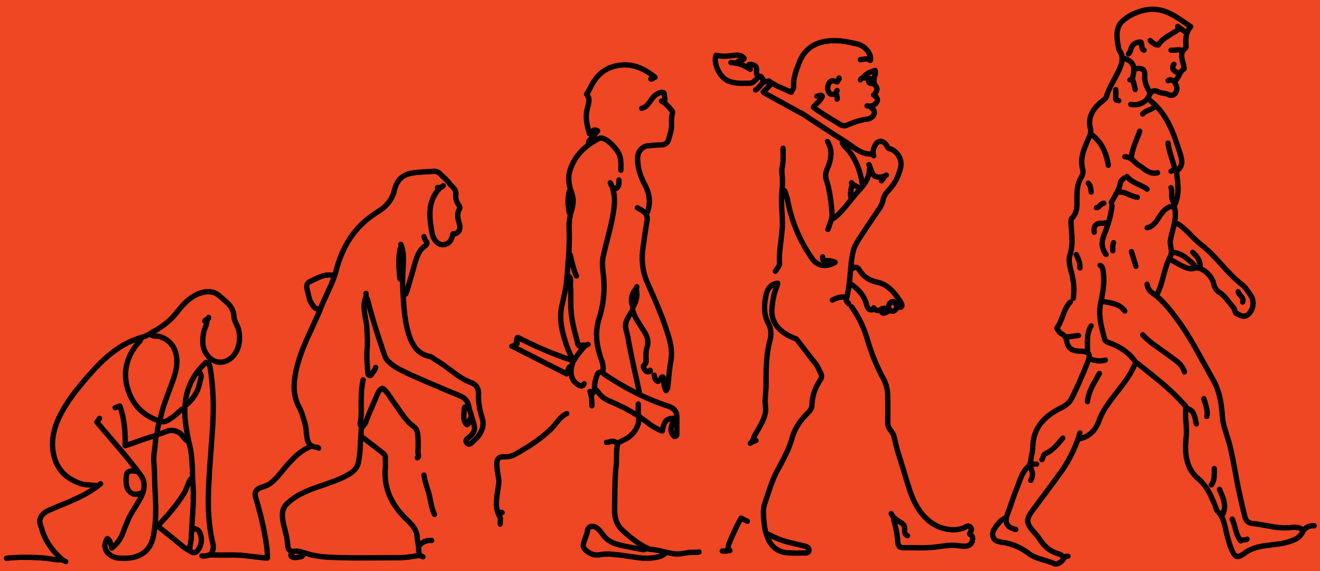
No-code apps have a unique lifecycle as they evolve over time. As your no-code usage develops and deepens, it's important to have efficient practices in place to maintain your applications.

These final measurement steps are key to any continuous improvement cycle, and the Application Audit stage ensures that the app stays relevant to the business.

That's the last stage in the No-code Lifecycle! Let's summarize now a few more important final considerations to plan for — beginning with how and when to build your Center of Excellence.

“It is the long history of humankind (and animal kind, too) that those who learned to collaborate and improvise most effectively have prevailed”

Charles Darwin



Building Your Center of Excellence

21

Establishing the No-code Center of Excellence (CoE) is an evolutionary approach in most organizations. Rarely is the CoE created and set up all at once with the initial no-code project. Instead, it's more often a result of organic growth and expansion when project teams start to gain success and experience in using no-code across different parts of the organization. Evolutionary steps toward no-code maturity allow the organization to build the business case for broader investment based upon proven success, allowing the CoE to grow and adapt. This incremental approach to forming the No-code CoE aligns with the "Everyday Delivery" philosophy. Start small, target a focused delivery of an initial CoE capability (think "MVP"), and then collect feedback on how it is performing so that you can evolve it over time. You can expand the CoE incrementally as you continue to deliver more no-code projects. In this manner, the CoE will grow and evolve in alignment with your successful use of no-code.

We have previously outlined that the No-code CoE is a go-to option for many application scenarios. The Application Matrix allows you to define where exactly you can apply it. In addition to that, the CoE approach is invaluable to driving efficiency, effectiveness, and overall impact of your no-code development practices. It can also play a significant role in radiating the no-code culture within your organization.

This chapter dives deeper into the No-code CoE and outlines how to establish one inside your organization. We'll look at the typical evolution of the CoE and common levels of progressive maturity. We'll outline seven of the key capabilities to focus on when you are defining the CoE. However, as always, this should be merely a starting point and guide that you can tailor to your unique needs and requirements.

The Maturity of Your No-code CoE

While everyone's evolution and expansion of the CoE may look slightly different, these are the stages you can typically expect to see:

Stage 1

Stage 2

Stage 1

This is the starting point for your No-code CoE journey. It often starts by establishing an initial small team of one or two individuals who may be part-time. The initial core team begins to assist the organization by defining the charter of the CoE and establishing sponsorship within the business and IT groups. The team also develops the initial definition of CoE roles and begins to implement standard processes. At this point, CoE engagement is usually opportunistic with individual no-code project teams. Each team will be engaged and addressed individually as needs arise.

Stage 2

At this stage, the No-code CoE identifies and clearly defines critical roles, and starts to formally assign full-time resources. As a result of successfully delivering a few simple apps, the CoE is able to further formalize the responsibilities of team members and define a repeatable engagement model for how the CoE should work with individual project teams. Then, the CoE establishes a lightweight guidance model and basic mechanisms for feedback collection and auditing for reuse.

Stage 3

Stage 3

The No-code CoE is executing on its charter and has delivered a robust set of initial projects, providing expertise to their project teams. By now, the CoE maintains an extensive library of best practices, coordinates reuse around shared components, and guides teams to develop high-quality solutions that perform well and are scalable. Delivery best practices and assets are available and used by multiple business units. Processes are in place to measure results, and business outcomes and successes are reported.

Stage 4

Stage 4

The No-code CoE is engaging proactively, consistently, and broadly across the organization, effectively measuring results, time to value, and business outcomes. It now provides support for the full spectrum of apps ranging from simple to mission-critical solutions (and has operationalized the Application Matrix to help teams know when they need to scale). The No-code CoE expands its library of Subject Matter Expertise (SME) services and best practices. It has implemented formal processes for enabling a composable architecture, including establishing processes for identifying, curating, and reusing components, and measuring the success of reuse.

These defined maturity levels are meant as a guide to help internal stakeholders set the right expectations around the CoE and how it effectively enables and supports the organization. You want stakeholders to be excited about the vision and potential for the CoE, but they should also be realistic about where you are in the phased evolution journey.

Framework for No-code CoE

When forming your CoE, it's helpful to begin with a framework that addresses the dimensions of People, Processes, and Technology.

People

A core part of the No-code CoE is providing the expert resources that support the rest of the organization. The CoE resources will offer a deep knowledge of the business processes, technical expertise, and no-code delivery methodology.

Processes

Building a repeatable and scalable model for no-code delivery across the organization is built around the formalization and sharing of prescriptive best practices. The CoE will play a key role in continuous learning from each project's success, applying the most successful practices, and monitoring the industry.

Technology

Delivering transformation with no-code at scale requires a capable, reliable, and secure platform for no-code development. It must meet current needs and, importantly, be able to accommodate future demand. The CoE will play a key role in establishing and evolving the technology platform, including managing shared environments and operational know-how.

Supporting this high-level framework, your CoE should invest in a set of important capabilities that enable and support each of these dimensions:

Dimension	Essential CoE capabilities
People	• Form a strong foundational CoE team • Establish a lightweight, consistent governance model
Processes	• Evolve clear process best practices and design principles • Continuous process improvement
Technology	• Establish reliable, secure, and scalable technology environments • Curate library of internal reusable components • Advocate no-code as part of your digital transformation strategy

These key CoE capability areas are briefly described in the sections below.

Capability #1:

Form a strong foundational CoE team

The key roles typically found in a foundational CoE are described in Chapter 4. It is worth repeating, however, that the CoE will not own all the roles directly but will typically provide resources in a matrixed organizational model — i.e., most of the key roles will still be driven by the business unit. Still, some specialized no-code roles may sit within the CoE. However, the CoE resources usually have a dotted-line reporting relationship with the overall business unit owner as a part of the development project.

Regardless of where the resources sit, the CoE plays an important role in managing the processes of recruiting, educating, certifying, and developing no-code creators and architects. The CoE should define the target profiles, including the specification of competencies and responsibilities for each role. Cultural fit is critical too: it is important to recruit individuals for the CoE who have the right technical and functional expertise and embrace the right cultural values around innovation. They must also be comfortable taking risks. The no-code project teams will be moving with speed and focus on Everyday Delivery, which requires an appetite for some calculated risks and a passion for driving innovation in new ways. With no-code, you can experiment and try new ideas. The team must be comfortable with trying new techniques and evolving its practices through continuous improvement.

This CoE team will also commonly emerge after the organization has started building some no-code expertise from multiple projects. Then, it becomes attractive to standardize and centralize some no-code expertise and skills so that they can be leveraged across teams. You should identify some of the individuals from early projects and recruit them into the CoE. Their expertise and learnings as early no-code project adopters will be invaluable to the organization at large.

Capability #2:

Establish a lightweight, consistent governance model

In Chapter 14, we discussed the importance of implementing governance practices. While the CoE is not required for these activities, it can play an important role in helping the individual no-code delivery teams navigate through these processes confidently, especially for teams that may not have prior experience in these internal and external governance practices. As the volume of no-code projects grows, the CoE will also play an important role in helping these governance activities scale and be performed consistently across the organization.

The goal of the CoE should be to act as an enabler for teams to efficiently and quickly conduct governance reviews. What can they do to help remove friction and uncertainty about the governance process? The No-code CoE should proactively start guiding individual teams early in the governance process starting at the Design Phase. This will help improve quality by ensuring that governance considerations are reviewed and addressed during the design activities, and it prevents the need for extensive (and expensive) rework later in the process.

This CoE governance model should also facilitate the auditing activities outlined in Chapter 20. As the no-code application portfolio grows, the CoE should help provide a consistent mechanism for reviewing the application portfolio for performance and obsolescence as well as updates for process and organization changes.

Capability #3:

Evolve clear process best practices and design principles

This No-code Playbook provides a starting point for building no-code apps. However, as you select a specific no-code vendor and gain experience in successfully delivering projects, you will start to develop additional guidelines and templates for how to build solutions. Each team that delivers a no-code project will doubtlessly add some new ideas or variants to the organization's knowledge base, so it's an opportunity for the CoE to continue to evolve and improve by harvesting best practices from each project and sharing them across the organization. As a result, you'll achieve greater impact and efficiencies because you won't be reinventing the wheel with every project.

It's important, however, to recognize that practices and templates must still be adapted to the needs of each team or project. Each project will have unique needs and, therefore, will not use the same practices. Common best practices should be adopted and adapted to match the specific complexity of the application (using the Application Matrix) and align with the culture of the team or business group, evolving as the group matures in its use of no-code.

Finally, don't overlook the importance of evangelism and communication of best practices.

Capability #4:

Continuous process improvement

The CoE should create an iterative approach to evolving best practices to ensure everything is completed the right way, at the right time, and with the right roles.

In Capability #1 above, we discussed the importance of fostering a culture of individuals and teams that embrace an innovation mindset and are comfortable taking risks. This "growth mindset" is essential for your ongoing efforts to improve practices. Outside the core CoE team, you should also look for ways to promote and reward this behavior; consider having the CoE give out rewards and recognition for the project teams or individuals who have contributed breakthroughs to core practices. Encourage the project teams that took risks with a new approach, even though it might have been unconventional. Building up a continuous improvement culture takes time, but with proper encouragement and enablement from the CoE, it becomes infectious and will begin to gather its own momentum.

Capability #5:

Establish reliable, secure, and scalable technical environments

In Chapter 15 (First Release) we described the typical set of no-code environments (e.g., DEV, QA, and PROD), how they are used, and the use of the Application Matrix for sizing according to project requirements. The no-code team itself is often responsible for its release and deployment activities, but it can sometimes run into more complexity, especially as the number of environments expands. This is where the CoE may play an important role in facilitating the underlying no-code platform and environments.

First, the CoE is likely responsible for the centralized monitoring of the no-code platform itself, including coordinating with the vendor if any updates are being applied (e.g., routine security or platform updates or new feature updates as the vendor adds capabilities). While these updates are typically automatic, the CoE may play an important role in scheduling updates (avoiding critical milestones for the business) and may also play a role in ensuring that training or enablement is rolled out to the project teams.

Second, the monitoring of the individual environments may be supported centrally by the CoE for more complex and mission-critical apps. This may help the individual project teams ensure that appropriate levels of high availability, reliability, and performance are achieved by properly tuning or configuring the no-code platform environment.

Finally, if any issues are encountered in production, the CoE may play a centralized role in working with the no-code vendor to create tickets or troubleshoot. Again, this may not be as essential for smaller no-code apps. Still, more mission-critical apps will need to have a clear service-level agreement (SLA) in place for any issue resolution and escalation, including working with the no-code vendor to deliver enterprise-grade support and service.

Capability #6: Curate library of internal reusable components

We've previously discussed the benefits of a composable architecture and using the Application Audit stage of the lifecycle to identify components suitable for harvesting and reuse. While reuse can sometimes occur based upon self-organized sharing between projects, the effectiveness of component reuse goes up exponentially if you make this a centralized function.

In Chapter 20, this was reviewed in length, but a few key points bear reinforcement with a focus on the role of the CoE:

- An important element of your component reuse strategy is facilitating collaboration and knowledge sharing across teams about the benefits of reuse and encouraging the exchange of ideas. The benefits and business value of component sharing must be promoted broadly across the organization to help drive the changes in behavior, instead of each team or project continuing to operate independently.
- The CoE should establish a standard set of practices during the Application Audit for assessing the applicability of reuse — this will include evaluating both functional and technical features of possible reusable components. This will help reduce the concerns that individual teams may not have the staff/budget/skills to invest in themselves by providing supplemental help needed to create a reusable or generic version of a component.

- Keep in mind that effectively identifying, curating, and operationalizing component reuse may require the introduction of new roles/ responsibilities — typically within the CoE. Usually, this is the responsibility of a no-code business architect. They should help develop internal training around component design, curation, and reuse and communicate it to the rest of the no-code project teams. The CoE business architect should also help enable teams to assist with the initial use of shared or reusable components.

- The CoE should help promote broader visibility/discovery of available components by establishing an internal catalog or marketplace of components. You should also look to the no-code vendor's marketplace (if one exists) to identify standard templates or components that can be reused from there.

- Don't forget the feedback and monitoring loop. The CoE should establish standard reporting and metrics to track reuse effectiveness. A data-driven approach is essential in helping identify which components may warrant supplemental investment and where to direct additional assistance or enablement.

Capability #7:

Advocate no-code as part of your digital transformation strategy

Many organizations will also be taking steps to define and operationalize their digital transformation strategy. Don't think of no-code as a separate or distinct agenda, instead, connect it directly to digital transformation. No-code can operationalize and scale the building of digital applications across your enterprise. While it's possible to pursue digital transformation with traditional custom development techniques, you will find that no-code can dramatically accelerate your ability to bring new ideas for digital innovation to the market. Explicitly advocating no-code as part of the digital transformation toolkit will help you achieve quick wins. It can also help build momentum and add additional resources to support the adoption of your no-code platform and tools.

Final Takeaways

There's a lot to think about when establishing your No-code CoE. Don't try to do it all at once. Instead, embrace an evolutionary approach, and adopt the "Everyday Delivery" philosophy to build out your CoE. Just like developing no-code apps, your approach to building your CoE can be to deliver an MVP and then evolve rapidly based upon your learnings. Investing in your No-code CoE will take time, but it will eventually help accelerate the delivery of value, and your no-code teams will realize even greater efficiency and productivity.

"The essence of strategy is choosing what not to do"

Michael Porter

22 Making No-code Your Strategy

You have started down your no-code journey with plans to build just one or two initial applications, and you used the No-code Lifecycle to deliver the project successfully. Congratulations! However, once you move further into building no-code apps, you may find that you are suddenly a "victim of your success," with additional requests springing up across different parts of the business function. This is a great problem to have, of course, but it also means you now need an overall no-code strategy to govern how you will prioritize the organization's efforts and manage the ongoing expansion of your no-code development projects. Trying to handle one no-code project after the next may not yield the best results.

So, let's step back. The No-code Lifecycle that we have discussed so far began with the definition of the business use case. In this chapter, we present what activities may need to precede this to help provide a strategy for ensuring that your no-code projects target the areas with the largest return on investment.

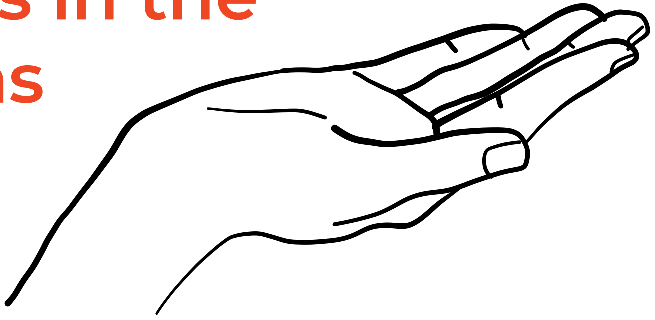
Step #1:
Educate and Engage

Step #2:
Group and Expand

Step #3:
Qualify and Prioritize

Educate the no-code vision → Engage the organization → Group ideas by type → Expand and elaborate → Qualify business idea → Prioritize the pipeline

We discuss each of these major steps in the following sections



Step #1:

Educate and engage

Think of this first step as an internal evangelism and marketing program that promotes the potential of no-code across your organization. Many business groups will not yet understand the concepts of no-code or the benefits of this approach, so you'll want to engage across the organization to educate them on the no-code vision and start to propose ideas for new areas of innovation. This may include the following:

Internal no-code roadshow

Plan to spend time taking your first no-code app around to other teams and showing them the power of no-code tools. They will most likely be amazed at how much was built in so little time. Taking a demo-led approach can be highly effective as groups often need to see the impact of no-code to fully grasp the power of what has been accomplished. Meeting internally with key business and technology leaders is important to gain buy-in and generate excitement. Then you can start collecting their ideas on possible no-code apps.

No-code showcase

It may not be practical to meet with the entire organization team-by-team, so you should consider other "viral" approaches to getting the word out. Putting together a simple "showcase" of case studies that link the no-code app with key metrics (e.g., how quickly it was built, benefits realized) can be an effective strategy that scales efficiently. Publish the showcase on one of your internal portals and promote it when it's shared across different teams. Developing strong word-of-mouth about the apps will create buzz!

Internal no-code hackathon

A powerful (and fun) approach is to be creative and experiential by staging a hackathon to encourage different teams to try no-code for themselves. No-code hackathons are facilitated events where you encourage small teams (or individuals) to learn new no-code skills around a facilitated one- or two-day project. Training is an important part of a hackathon to give participants a base set of skills. It is also important to pair them with mentors who can help design simple apps (typically, mentors are part of the CoE).

Note that an internal hackathon requires a fair amount of internal preparation and a clear sponsor to plan and manage the event, typically coming from the No-code CoE. Planning a hackathon is beyond the scope of this Playbook but there are good resources (like this guide[1]) that cover tips and guidance in more detail.

Best practice tip:

You'll want an easy way to capture ideas that get submitted across any of the above channels, so why not create a no-code app for conducting an innovation survey? Having a no-code app for surveying ideas is a fun and visible way to showcase no-code development as part of the strategy and process for engaging the entire organization. You can likely find an app already in the no-code vendor's marketplace (like this[2]) that allows you to easily conduct the survey.

[1] Hackathon Guide

[2] Conducting surveys for Creatio, Creatio Marketplace

At the end of this first step, the key success criteria are that you've generated a lot of internal excitement and collected many ideas for possible no-code app innovation. However, the ideas will likely be somewhat raw and unstructured, of different sizes and types, and may duplicate and overlap. You need to structure them before moving forward — which is the point of this next step.

Step #2:

Group and expand

This next step will begin to apply some structure across the portfolio of app innovation types. We propose grouping and aligning the apps into three distinct strategic categories, which help cluster applications based on their strategic impact on the business function.

Systems of Record

These apps will typically use no-code to extend or augment established packaged software or homegrown systems that support core business processes and data. The strategic focus of these types of apps is typically on efficiencies, often by eliminating unnecessary data reentry in multiple systems. They may be rolled out broadly to large numbers of users across the organization as they are part of core business processes. They may also handle sensitive data and be subject to different governance requirements.

Systems of Differentiation

These apps will use no-code to enable unique company processes or industry-specific capabilities. These apps need to be updated more frequently to accommodate changing business practices or customer requirements.

Systems of Innovation

These are new no-code applications built rapidly to address new business requirements or opportunities. These can sometimes have shorter lifecycles and be more focused because they may address areas that are being explored (often at a departmental level) and have not yet been addressed by more traditional packaged applications.

This is modeled after principles of Gartner's Pace Layered Application Strategy[3] but applied to the no-code approach. Recognize that all three of these application types are important and represent different no-code opportunities and benefits. By grouping your app ideas into these strategic categories, you realize a balanced approach to your no-code strategy.

As you group items by innovation type, you will likely find that the ideas are unevenly fleshed out. Some will have more detail, others will have less, and some may even have parts of the business vision that are unclear. At this stage, you would work with each group to review their submission and flesh out more detail as needed. Work to add more insights to the idea and specificity about expected outcomes.

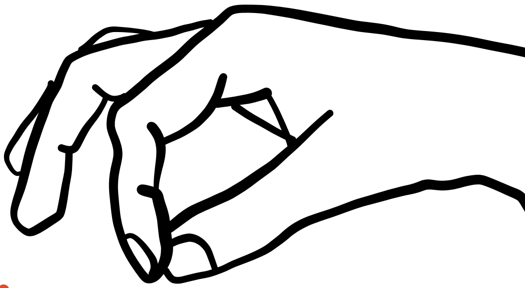
[3] Accelerating Innovation by Adopting a Pace-Layered Application Strategy, Gartner

Step #3:

Qualify and prioritize

Most sales organizations use the concept of a "sales funnel" to describe the customer journey through the sales process, stretching from early-stage brand discovery to final purchase. Visualizing the process as a funnel helps represent that sales opportunities are more qualified as they pass through the stages in the process. Opportunities are fewer the further you move down the funnel, but they become more highly qualified. This concept of a prioritization funnel should also apply to your pipeline of no-code apps as they flow through the strategic framework. The number of ideas will get smaller as you progress, but this is actually OK — the ideas that are making it into later stages are more highly qualified and represent higher-impact innovation opportunities for your organization.

Let's review the no-code strategy framework stages again using this view:



Step #1: Educate and Engage		Step #2: Group and Expand		Step #3: Qualify and Prioritize	
Educate the no-code vision	Engage the organization	Group ideas by type	Expand and elaborate	Qualify business idea	Prioritize the pipeline
• Communicate the vision • What are we trying to accomplish? • What does a no-code app look like? • What are the strategic benefits?	• Meet with the business teams • Publish the showcase • Plan and execute a hackathon	• Group by innovation type • System of Record • System of Differentiation • System of Innovation	• Work with groups to flesh out idea • Expand by adding more insights and expected outcomes	• Review submissions • Qualify if business outcomes are well understood • Qualify if submissions are sufficiently defined	• Council reviews and approves • Best ideas to move forward to the Business Use Case stage

By the time your no-code ideas have progressed through the pipeline, you will have identified a set of highly-qualified and impactful ideas! Take this list and prioritize it so that you end up with a stack-ranked list of the top ideas. An important factor for your prioritization should be how well the ideas align with and support the key elements of your business strategy and goals. You should also assess the ability of the no-code ideas to improve your competitive position, which can help you outpace other competitors in the market, such as by being able to launch new products or services faster to help attain more market share.

Now, you are ready to take each of these and review it with your executive leadership that has sponsored no-code at your enterprise. Have them review the list and get explicit approval to move forward with the top ideas and develop the business use cases.

Final Takeaways

Preparing an overall strategy for your no-code app portfolio is a more advanced concept after you have successfully built the momentum of different groups using no-code. But investing in a strategic framework that enables the continued inception and collection of ideas helps you group and align your portfolio and, ultimately, prioritize and approve the best ideas to move forward, which is important as your portfolio scales.

Acknowledgements

This book would have never been created without the thoughtful leadership of Andie Dovgan, Creatio's Chief Growth Officer. Thank you for driving the creative process forward, your never-ending optimism and trust in our team, your bright ideas, and your selfless support at every step of the journey.

Huge thanks to the research and development (R&D), innovation, and project leaders. Lena, Ivan, Vitaliy, and Anton, your immense experience in the no-code space, deep understanding of every concept outlined in the book, and ability to leave no stone unturned have inspired us!

The team that has orchestrated the process, edited, corrected, and, finally, published the book by investing great effort, time, and sleepless nights — you are our heroes! Vlad, Kate, and Julia — without your hard work, we wouldn't have accomplished this mission.

Constantin, Nina, and the team of designers — your talent for bringing ideas to life through art is truly special. We're big fans of your creations. Thank you for being a major part of this journey!

Thank you, Ginny, editor and reader advocate, for your brilliant mind. Thank you, Brandon, for your insightful comments and feedback.

The whole Creatio Community — your excitement about the book, plus your contributions and collaboration, made this book even better than we could have ever imagined!

Business book

Katherine Kostereva
Burley Kawasaki

The No-Code Playbook
2022
1st edition

Chief Editor: Andie Dovgan
Editor: Virginia Holden
Corrector: Mark Baker
Art Director: Konstantin Nesyn

The No-Code Playbook is a vendor-agnostic guide that empowers teams to deliver business applications of any complexity with no-code.

The No-Code Playbook includes illustrations from the Freepik website designed by Sky and Glass, zoni39007, macrovector, mulyasriwahyuni, katerina51, rawpixel.com, collayart, and Freepik; from the Shutterstock website designed by Vectorium, Naci Yavuz, Polina Erofeeva, Orfeev, and ursulamea; from the Envato website.

ISBN 979-8-218-06204-0