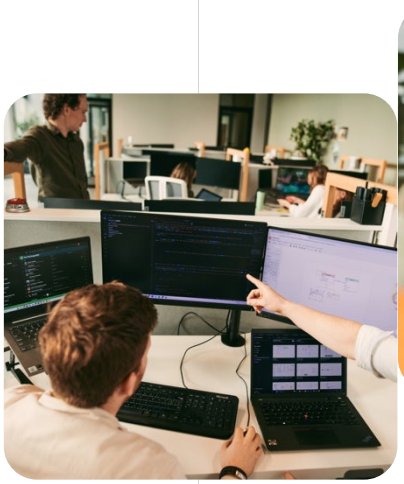
BEYONDER

PRO-CODE OR LOW-CODE



A Decision Guide with Insights from Our Experts

TABLE OF CONTENTS

01.

INTRODUCTION

In the rapidly changing world of software development, businesses are continuously faced with the challenge of selecting the most suitable development strategy that aligns with their specific needs and objectives. At Beyonder, where we specialize in both pro-code and low-code software development, we are frequently asked: “Which approach is the best choice for our project?”

The answer to this question isn’t always straightforward and depends on various factors, such as the complexity of the application, available resources, desired timeline, and future scalability. In this whitepaper, we aim to explore the nuances of both approaches to help businesses make informed decisions.

Pro-code refers to the use of programming languages like C#, JavaScript, or Python to build software. Low-code, on the other hand, involves environments like Mendix, OutSystems, or Microsoft PowerApps, which enable software development using prebuilt, “ready-made” building blocks and visually oriented construction.

Pro-code development provides flexibility and control, making it ideal for complex, customized solutions. Conversely, low-code development facilitates the rapid and cost-efficient creation of applications, even with limited technical expertise, making it attractive for a wide range of projects.

With our experience and expertise in both domains, we are well-positioned to analyze the advantages and disadvantages of pro-code and low-code approaches. We will also discuss various use cases to illustrate when each method is preferable, highlighting the key considerations businesses should weigh in their decision-making process.

Whether you are looking to launch a Minimum Viable Product (MVP) quickly or need a complex enterprise solution, this whitepaper offers valuable insights and guidelines to help you make the right choice.

02.

THE ORIGIN OF LOW-CODE

Low-code emerged from the need for faster and more efficient ways to develop software during a time when traditional development methods were often slow, costly, and error-prone. In the 2000s, companies like Mendix, OutSystems, and Appian began creating platforms with visual, drag-and-drop interfaces for application development, enabling both technical and non-technical users to participate in the process. These platforms allowed for the rapid creation, testing, and deployment of functional applications without requiring deep programming knowledge.

The movement was further strengthened by the growing demand for digital transformation, the need to bridge the gap between IT and business departments, and the increasing scarcity of experienced software developers. Consequently, low-code has become an appealing solution for many organizations seeking greater agility and innovation.

03.

WHAT IS LOW-CODE?

Low-code development is a modern approach to software development that utilizes visual interfaces, drag-and-drop tools, and minimal coding to build applications quickly and efficiently. Low-code platforms are designed to reduce technical complexity and shorten development time, enabling both professional developers and non-technical users (like business analysts) to create software.

3.1 | Features

Features of Low-code Development:

- **Visual Development Environment:** Low-code platforms provide a visual development environment where users can build applications by dragging and dropping components instead of writing full code. This makes the process more intuitive and accessible.
- **Faster Development Cycles:** By utilizing reusable components and pre-built modules, developers can quickly build prototypes and complete applications. This significantly accelerates the time-to-market.
- **Reduced Need for Technical Skills:** Low-code platforms are designed to be easy to use, even for individuals without deep programming knowledge. This democratizes development capabilities within an organization.
- **Automation and Integration:** Many low-code platforms offer built-in capabilities for automating workflows and integrating with other systems and databases, simplifying the development of end-to-end solutions.

3.2 | Use-cases

Low-code development is often chosen for projects that:

- **Rapid Prototyping and MVP:** When an organization needs to quickly launch a Minimum Viable Product (MVP) to test an idea or seize market opportunities. Limited benefits and a short lifespan; a fast, temporary solution.
- **Business Process Automation:** Low-code is well-suited for quickly automating repetitive business processes and workflows. Processes that can be described in a linear model, such as BPMN, are particularly ideal.
- **Applications with Limited Complexity:** For applications that do not require complex business logic or custom functionalities, such as a tailored HRMS, ERP, or CRM system—essentially applications driven by straightforward CRUD operations.

Low-code development offers an efficient and cost-effective way to build applications, particularly for projects with limited complexity and tight deadlines. It democratizes software development by making it accessible, to some extent, to a broader range of users, from professional developers to business users without extensive knowledge of software development.

04.

WHAT IS PRO-CODE?

Pro-code is the most common approach to software development, where developers use conventional programming languages such as C#, Java, Python, and others to build custom applications. This method contrasts with low-code platforms, which offer visual development tools and minimal coding to create applications.

4.1 | Features

Features of Pro-code Development:

- **Full control and flexibility:** Pro-code development offers developers full control over all aspects of an application, from the user interface to backend logic and database interactions. This allows for highly specific, tailored functionalities that meet the unique needs of an organization without compromise.
- **Use of traditional programming languages:** Pro-code projects are written in traditional, text-based programming languages. This requires in-depth knowledge of the language used, as well as its associated frameworks and libraries.
- **Flexible architectures:** Pro-code development enables the creation of more flexible software architectures, such as microservices, distributed systems, and applications that require high performance and scalability.
- **Robust integrations:** Pro-code offers far more possibilities for seamless integrations with other systems, databases, and external services, which is essential for many enterprise solutions. For instance, think of integrations with healthcare hardware systems like alarm systems or fall detection devices.

4.2 | Use-cases

Pro-code development is often chosen for projects that require:

- **Complex functionality:** Applications with intricate business logic, advanced algorithms, or specialized features that cannot be easily achieved with low-code platforms. For example, scheduling algorithms for childcare centers or predictive analytics in the pallet industry.
- **High performance:** Systems requiring optimal performance and scalability, such as real-time data processing, financial transaction systems, or large-scale e-commerce platforms.
- **Strict compliance and security:** Projects where strict adherence to security standards and regulations is crucial, such as in healthcare, finance, or government sectors. Examples include tenant-specific data separation, strict encryption requirements with

private keys, data backup requirements, or extensive audit trails for impactful decision-making processes.

- **Legacy systems:** When a system needs to work with existing legacy systems and architectures, pro-code provides the flexibility and compatibility required for such integrations. For instance, by supporting custom integrations on binary interfaces.

Pro-code development delivers ultimate flexibility and power for building customized applications. For organizations requiring specific, non-standard solutions or complex integrations, pro-code remains the preferred choice.

05.

TRENDS AND DEVELOPMENTS

The landscape of pro-code and low-code development is constantly evolving. Advancements are making pro-code development faster while granting low-code platforms greater flexibility. Below are the key developments broken down:

5.1 | Integration Platforms

For complex enterprise integrations, custom development has traditionally been essential. However, the rise of advanced integration platforms has significantly reduced this dependency, enhancing the flexibility of low-code development.

Integration platforms like Zapier, MuleSoft, and Microsoft Power Automate offer powerful capabilities to seamlessly connect different systems and services. These tools enable the integration of data and functionalities from diverse sources without requiring extensive programming knowledge. By utilizing pre-built connectors and visual workflow designers, both pro-code and low-code developers can quickly and efficiently set up integrations that were previously achievable only through complex, manually written code.

These platforms greatly expand the scope and capabilities of low-code applications. They allow businesses to connect existing systems, such as ERPs, CRMs, and other enterprise software, with new low-code solutions, enabling seamless data flows and process automation. This not only boosts operational efficiency but also facilitates rapid adaptation to changing business needs and technological innovations.

By leveraging integration platforms, organizations can combine the advantages of low-code development—speed, ease of use, and reduced costs—with increased flexibility and scalability. This makes low-code an even more powerful tool for developing robust, integrated solutions that meet the complexity and dynamics of modern business environments.

5.2 | Low-Code Cloud Development Platforms

The evolution of low-code platforms continues, with vendors striving to enhance ease of use and efficiency for their users. A significant trend in this space is the emergence of all-in-one

cloud-native solutions. These platforms operate entirely in the vendor's cloud, enabling developers to quickly and easily deploy environments without requiring extensive knowledge of cloud infrastructures. This boosts productivity and allows teams to focus on building applications instead of managing underlying infrastructure.

However, this shift to cloud-based, all-in-one solutions also introduces several challenges, including vendor lock-in and compliance issues:

Vendor Lock-In: Organizations that fully host their applications and data in a specific low-code vendor's cloud may become dependent on the vendor's technologies and services. This can limit flexibility when switching platforms or environments in the future and may result in higher long-term costs due to licensing and service fees imposed by the vendor. In contrast, pro-code solutions leverage abstraction layers like Docker, Terraform, and Kubernetes, and can utilize cloud-agnostic services (e.g., databases). This makes pro-code solutions less tightly bound to the cloud environment they operate in.

Compliance Challenges: Compliance issues arise when low-code solutions do not run within an organization's own cloud environment. This can be problematic for companies that need to adhere to strict regulations and data governance policies, such as GDPR, NEN7510, or industry-specific standards. Limited control over data storage, security, and access makes it harder to meet compliance requirements.

Organizations across various sectors are increasingly shifting from public to private cloud or on-premise solutions to address compliance and cost concerns. The cloud platform lock-in associated with many low-code platforms can hinder this transition.

By contrast, pro-code solutions offer greater control over compliance and costs through flexibility in deployment environments. Tools like Docker and Kubernetes ensure portability, while cloud-agnostic services enable a future-proof approach to development that aligns better with stringent regulatory requirements.

5.3 | AI in Pro-Code Development

The integration of Artificial Intelligence (AI) in pro-code development is significantly transforming how software is designed, developed, and managed. Here are some key ways AI is influencing pro-code development:

- **Automation of coding processes:** AI-powered tools such as code suggestions and automatic code generation accelerate the development process by helping developers write more efficient, error-free code. This reduces the time and effort required for manual coding, allowing projects to be completed faster.
- **Improved debugging and testing:** AI algorithms can identify patterns and predict potential bugs or vulnerabilities in the code. Advanced AI tools can automatically generate test scripts, detect bugs, and even suggest fixes, enhancing software reliability and quality.

- **Performance optimization:** AI can monitor and optimize application performance. By analyzing large datasets, AI identifies bottlenecks and provides recommendations for improvements, such as optimizing database queries or enhancing algorithm efficiency.
- **Intelligent documentation and support:** AI-driven documentation tools can automatically generate technical documentation based on written code, saving time and improving consistency. Furthermore, AI chatbots and virtual assistants can support developers by providing instant answers to technical questions and issues.

AI's impact on pro-code development is profound, automating tasks, increasing efficiency, and improving code quality. By strategically leveraging AI, pro-code solutions can be developed faster and with greater precision.

5.4 | Pro-Code Builders

Early tools like Visual Studio and Microsoft FrontPage laid the groundwork for drag-and-drop interfaces within pro-code environments, enabling developers to quickly design user interfaces (UIs) for Microsoft programming languages.

Today, even more advanced drag-and-drop builders are available that generate pro-code as output. Platforms like Webflow and FlutterFlow enable developers to design visually appealing, functional interfaces with minimal manual coding while maintaining the flexibility and power of pro-code. These tools produce efficient, maintainable code that integrates seamlessly into existing pro-code projects.

Additionally, solutions like Firebase and Hasura offer drop-in replacements for APIs, greatly accelerating backend functionality development. Design tools such as Figma are increasingly capable of generating front-end code, creating a seamless transition from design to development.

By leveraging these tools, developers can significantly speed up the pro-code development process, narrowing the productivity gap between pro-code and low-code while preserving the benefits of customization, control, and scalability.

5.5 | Recruiting Low-Code Talent

Low-code is a relatively new technology that is rapidly gaining popularity. Consequently, the demand for qualified low-code talent has grown exponentially, while the supply lags behind. Many professionals simply have not yet had the opportunity to specialize in low-code platforms.

Furthermore, low-code development requires a unique mix of skills. It is not only important to understand how specific low-code tools work, but also to have a solid foundation in software development and business processes. This combination of technical and business knowledge is rare, making it challenging to find suitable candidates.

In practice, a limited number of pro-code developers are willing to transition to low-code. Low-code technology is often not included in traditional computer science curricula, and many

pro-code developers identify strongly as code specialists. They may feel that low-code platforms offer fewer opportunities to fully express their technical expertise.

Organizations must address these challenges by creating tailored training programs and offering incentives to attract and retain low-code talent while bridging the gap between pro-code and low-code expertise.

06.

CONSIDERATIONS

The choice between pro-code and low-code is not binary. The considerations outlined below rarely yield a straightforward answer but instead provide food for thought and insights to guide the decision-making process. Additionally, low- and pro-code approaches can be combined, leveraging the strengths of both options.

In practice, we observe four main directions:

1. **No-Code**: Ideal for creating simple applications or automation tasks without programming knowledge. Often used by non-technical users or business analysts for workflows, forms, or basic data management solutions.
2. **Low-Code:** Suitable for projects requiring more customization and complexity than no-code can deliver. Visual development tools allow developers to quickly build applications using prebuilt components with minimal manual coding. It's useful for projects that require custom logic or integrations with external systems but don't need to start from scratch.
3. **Pro-Code:** The traditional approach where developers have complete control over the code and development process. Used for complex projects requiring custom functionality, advanced logic, performance optimization, or deep integration with existing systems. Pro-code offers the greatest flexibility but often requires more time and specialized programming expertise.
1. **Hybrid (Pro- & Low-Code):** By combining low- and pro-code, organizations can create solutions that leverage the flexibility of pro-code and the speed of low-code development. For instance, an administrative portal built in low-code with a sleek consumer-facing front-end in pro-code, or a scheduling portal in low-code combined with a pro-code scheduling algorithm. The trade-offs include the need for expertise in both approaches and potential impacts on running costs.

NOTE:

No-code is not directly comparable to low- or pro-code. It is primarily seen as a solution for relatively small, specific use cases that do not handle business-critical information. Examples include digital forms for HR processes or simple external intake workflows.

6.1 | Intellectual Property (IP) Rights

In low-code, a portion of the solution falls under the intellectual property of the low-code platform. Ownership and control over the foundational building blocks remain with the low-code vendor, which makes companies reliant on the vendor's strategy. This dependence can be a limitation and may reduce the organization's value, as part of the technology is not fully owned by the company.

Pro-Code: Pro-code provides full ownership of intellectual property rights. Organizations retain complete control over their source code, allowing independent management and customization of the software. This significantly enhances a company's value by enabling the commercialization and modification of its technology without restrictions. Additionally, it can serve as a valuable business asset protected by patents, copyrights, and trademarks. Full IP ownership makes a company more attractive to investors and potential buyers, showcasing innovation and strategic capabilities.

Example: For SaaS providers, intellectual property rights are a critical factor. This is especially relevant as many traditional companies transition from physical to virtual services, becoming increasingly dependent on digital infrastructure. Banks are a prime example of this shift.

6.2 | Vendor Lock-In

Low-Code: With low-code platforms, further development is constrained by the limitations and capabilities of the chosen platform. This creates a strong dependency on the vendor for future updates, new features, and ongoing support. Such reliance can become problematic if the vendor discontinues support, raises licensing fees, or imposes technological restrictions. This can result in unexpected costs and reduced flexibility, limiting the adaptability of software solutions to changing needs.

Pro-Code: Pro-code provides flexibility in technology and vendor selection. Organizations can choose from a wide array of programming languages, frameworks, and tools tailored to their specific requirements. This independence enables them to respond quickly to market changes and technological innovations.

Critical Note: While pro-code reduces vendor dependency, it's important to be cautious when selecting programming languages and frameworks. Established languages like C#, with over 24 years of stability, widespread adoption, and Microsoft's backing, pose minimal risks. However, the same cannot be said for newer, trend-driven front-end frameworks, which may lack long-term viability or support.

6.3 | Lifecycle

The expected lifespan of the software to be developed is another important consideration:

Short Lifespan: For projects with a shorter lifecycle, low-code is often more suitable due to its rapid development capabilities, allowing organizations to realize benefits sooner.

Long Lifespan: For software intended to be used over a longer period, the development roadmap will eventually stabilize. Over time, the cost advantage of low-code diminishes as licensing fees weigh heavier, whereas pro-code solutions can become more cost-effective since they do not require ongoing platform licensing.

Key Insight: A robust business case is essential to make the right decision. However, a general benchmark is that the break-even point between low-code and pro-code development occurs around 5 years in many scenarios. Beyond this point, pro-code often becomes more advantageous due to its lack of recurring licensing costs and its long-term adaptability.

6.4 | Time-Critical Processes

Low-Code: Low-code platforms often provide less control and precision, which can pose challenges for time-critical applications such as real-time data analysis, alerting systems, or stock market transactions where every millisecond counts. Limited control in these platforms may lead to delays and inconsistencies in process execution.

Pro-Code: Pro-code, on the other hand, offers full control over the application architecture. Developers can implement optimizations and customizations to ensure the timeliness and reliability required for time-critical applications. Pro-code provides the robustness and precision necessary for accurate and dependable execution of such processes.

Key Insight: The rule is straightforward: low-code is not suitable for time-critical processes. While achieving precise real-time processing is challenging even with pro-code, it offers mechanisms and components for accurate and fast event handling that low-code platforms cannot match.

6.5 | Multi-Tenancy (SaaS)

Low-Code: When an organization considers offering their software as a service (SaaS), supporting a multi-tenant architecture becomes essential. Low-code platforms often present limitations in this regard due to their restricted flexibility in handling multi-tenant structures. Features such as separated data storage per tenant, support for diverse organizational structures, or tenant-specific branding are typically not available out-of-the-box in low-code environments.

Pro-Code: Pro-code development provides complete freedom and precise control in designing and implementing multi-tenancy structures. Developers can build custom architectures tailored to specific customer requirements, including data isolation and security measures.

Key Insight: For SaaS projects requiring robust multi-tenancy, pro-code is the preferred approach, offering flexibility and scalability that low-code platforms cannot match.

6.6 | Budget

One of the most important considerations when choosing between low-code and pro-code development is the balance between initial development costs and ongoing operational expenses. The productivity gain is a factor of 2 to 5 for solutions that fall within the capabilities

of the low-code platform. It should be noted, however, that this primarily concerns the net development effort, as a proper development process for both pro- and low-code also includes architecture/analysis/UX components, coordination, and investments in QA.

However, these cost advantages are accompanied by higher ongoing operational expenses due to the licensing model employed by low-code platforms.

As a rule of thumb, low-code is not financially attractive for a (simple) single-app strategy. The licensing costs are often too high for this purpose. The ROI for low-code is quicker/easier to achieve when multiple solutions are built with it. Only then does the productivity gain offer real advantages.

6.7 | Time-to-market

In low-code platforms, development and implementation can be done much faster than with pro-code. This not only accelerates the development process but also enables organizations to quickly build prototypes, gather user feedback, and iteratively implement improvements. For companies that need to respond rapidly to market changes or launch new products and services under tight deadlines, low-code offers an efficient and flexible solution to gain a competitive advantage.

An alternative strategy to consider (when it comes to choosing a specific architecture) is to view low-code as an innovation platform. If the innovation proves successful and the specifications (MVP+) are largely clear, the solution can later be replaced with a pro-code implementation.

6.8 | User Base

Low-Code: Many low-code platforms use pricing models where costs increase as more users gain access to the application. Sometimes, distinctions are made between internal and external users. This can lead to rapidly escalating licensing costs as the user base grows.

Pro-Code: For applications that must support a large number of users, pro-code is often more cost-effective, as costs do not increase linearly with user growth. Instead, organizations can scale their infrastructure to meet demand, often achieving long-term cost efficiency. This flexibility can result in significant savings and provides a clear advantage for businesses with a large or growing user base.

Key Insight: When making a decision, consider the potential user base a few years into the future, rather than focusing solely on initial (MVP) releases. Pro-code is often the better choice for scalable solutions that anticipate significant user growth.

6.9 | User Experience (UX)

Low-Code: Low-code platforms provide standard components and templates that ensure a consistent and functional user experience. However, these limit customization and creativity. Low-code often results in less unique user interfaces that may not fully align with specific branding or user expectations. Key limitations include:

- Difficulty creating native-feel apps with smooth animations and transitions.
- More "website-like" interactions with limited functional feedback.
- Static design systems that may not fully meet modern accessibility standards (e.g., WCAG 2.1+).

Pro-Code: Pro-code development offers complete freedom to customize every aspect of the interface, from advanced animations to bespoke layouts. Developers can create a unique, brand-aligned UX using off-the-shelf component libraries designed to meet high accessibility standards. These libraries serve as a strong foundation for dynamic and modern design systems, enhancing user engagement and satisfaction by delivering a distinctive, well-tailored, and aesthetically pleasing experience.

6.10 | Changing Business Processes

Low-Code: In dynamic environments where business processes constantly evolve or are influenced by external factors, low-code offers greater flexibility. Low-code platforms are designed to enable rapid adjustments and iterations, allowing organizations to respond quickly to changing market conditions, customer needs, or internal process improvements without lengthy development cycles.

Pro-Code: While adjustments in pro-code take more time and require specialized developers, changes are better controlled during development and testing. Pro-code benefits from mature tools, such as **Git**, for version control, enabling developers to:

- Review, branch, and track changes.
- Ensure that all modifications are reliable and correctly implemented.

For organizations prioritizing robustness and reliability, pro-code ensures that every change is thoroughly validated before deployment. This rigorous approach makes pro-code the preferred choice for environments where reliability is critical, even if agility is sacrificed.

6.11 | Complexity

Low-Code: Low-code platforms excel at quickly and efficiently building applications with relatively simple or moderately complex logic. Their visual development tools and reusable components make it easy to model common business processes and workflows. However, when applications require more complex logic—such as advanced algorithms, detailed data processing, or specialized integrations—the limitations of low-code platforms become apparent.

In these cases, developers often resort to custom code or even transition entirely to pro-code solutions to meet functional requirements. This can result in hybrid scenarios where both low-code and pro-code are employed, potentially increasing development and maintenance complexity.

Key Challenge: There is no clear definition of "complexity" in software; it often only becomes evident when symptoms arise, such as:

- A drop in developer productivity due to a loss of project clarity.
- Performance issues stemming from the software's size or design.
- The need to offload logic to external services because graphical low-code structures are insufficient for complex operations, such as advanced data manipulations.

Key Insight: Anticipating these aspects beforehand is crucial to making an informed technology choice. Proper assessment of the application's complexity should be part of the planning phase to avoid misalignment with the platform's capabilities.

6.12 | Compliance and Traceability

Low-Code: Low-code platforms often include built-in security and compliance features. However, these may fall short of meeting the stringent requirements of industries such as healthcare, finance, or government. Challenges may include:

- Limited data governance and infrastructure control.
- Inadequate support for detailed audit trails.
- Difficulties integrating with systems like Security Operations Centers (SOCs) or accessing comprehensive trace logs.

Pro-Code: Pro-code offers full control over the architecture and implementation of compliance and traceability measures. Developers can design custom solutions tailored to meet specific requirements and regulations, such as GDPR or NEN7510. This includes:

- Comprehensive data governance policies.
- Detailed logging and monitoring capabilities.
- Robust security measures.

Key Insight: For organizations that prioritize strict regulatory compliance and complete traceability, pro-code ensures these needs are met effectively. This makes pro-code a reliable choice for industries where adherence to regulations and robust auditing capabilities are non-negotiable.

07.

CONCLUSION

The choice between low-code and pro-code (high-code) is not always straightforward and involves nuanced considerations. A well-informed decision requires balancing many factors, including compliance, speed, solution lifespan, and complexity. The future of software development further complicates these choices as pro-code productivity continues to rise, while low-code limitations diminish.

Low-code is becoming increasingly mainstream, with existing platforms evolving and new, potentially more specialized platforms emerging. This technology will keep improving, offering more advanced functionalities, better integrations, and enhanced usability. These advancements will enable both technical and non-technical users to develop and adapt

applications quickly. Additionally, there will be a growing focus on governance and security to meet stricter compliance and data privacy demands.

Pro-code development will remain focused on complex, custom solutions requiring specific functionalities and scalability that are difficult to achieve with low-code platforms. While low-code will take on more routine and less complex application development tasks, pro-code will continue to be indispensable for critical systems, advanced integrations, and applications requiring high performance and precise control. Furthermore, with the application of generative AI in the development process, pro-code development will accelerate significantly. Hybrid solutions, combining the speed and flexibility of low-code with the robustness and depth of pro-code, will become more common. High-code developers will increasingly collaborate with low-code platforms to create solutions that leverage the strengths of both approaches.

At **Beyonder**, we focus on both low-code and pro-code. Our expertise allows us to provide tailored recommendations for each use case, ensuring the right balance between the two approaches. We'd be happy to discuss your specific needs—let's meet for a cup of coffee with one of our experts!

APPENDIX: DECISION MATRIX

	CONSIDERATION LOW-CODE	CONSIDERATION PRO-CODE
Intellectual Property (IP)	Part of the technological solution falls under the IP of the low-code platform.	Full ownership of IP rights.
Independence from Technology/Vendor	Further development is only possible within the constraints of the low-code platform.	Flexibility in technology and vendor selection.
Solution Lifespan	Longer lifespans make the licensing model less favorable.	Shorter lifespans make development costs disproportionate.
Time-to-market	Low-code enables faster development cycles and quicker realization.	Pro-code development timelines are often 2-3 times longer than low-code.
Budget	Lower initial development costs, higher running costs due to licensing.	Higher initial development costs, lower running costs over time.
Number of Active Users	Direct impact on licensing costs as the user base grows.	User base size primarily affects infrastructure costs.
Transitioning to SaaS (multi-tenancy)	Multi-tenancy complexity is harder to achieve in low-code platforms.	Full freedom to design and implement multi-tenancy structures and complexities.
Changing Business Processes	Changes can be implemented quickly, sometimes even by business users themselves.	Changes must go through the development team but are always validated through a controlled process.
User Experience (UX)	Limited by the set of components available in the platform.	Full freedom, enabling branding-aligned and customized experiences.
Logic Complexity	Complex logic may require partial reliance on pro-code.	Well-suited for complex algorithmic and logic-heavy applications.
Compliance and Traceability	Limited measures based on platform capabilities, e.g., data governance, custom infrastructure, or regulations.	Full control over implementation, enabling robust compliance and traceability measures.
Time-Critical Technical Processes	Lacks control over timing and sequence, making it less suitable for time-sensitive solutions.	Provides control over timing through complex architectures and algorithms in pro-code solutions.

BEYONDER