estenda

Build vs. Buy in the Age of **AI Software Development**

A Strategic Decision Framework
for Healthcare Technology Leaders

Executive Summary

The decision to build software internally or buy commercial solutions has long been a core strategic consideration for technology leaders. In healthcare, digital health, and life sciences, this decision is further complicated by regulatory requirements, complex integrations, and the need for long-term system reliability.

Artificial intelligence is reshaping this landscape by accelerating software development. AI-assisted tools enable faster prototyping, code generation, and iteration. However, these capabilities primarily reduce time to initial functionality, not the effort required to deliver production-ready systems.

As a result, the build versus buy decision is evolving into a new model: Strategic Build.

In this model, organizations do not rely solely on internal development or off-the-shelf software. Instead, they build critical systems in partnership with experienced engineering teams, enabling greater control while ensuring compliance, security, and long-term maintainability.

AI changes the economics of software creation, but it does not reduce the responsibilities of ownership. Organizations must still address validation, cybersecurity, regulatory compliance, integration complexity, and ongoing lifecycle management.

This whitepaper examines how the build versus buy decision is changing in the age of AI and outlines when a **strategic build approach** is most appropriate for organizations operating in regulated environments.

How AI Changes the Build vs Buy Equation

Artificial intelligence is reshaping the traditional build versus buy decision, but not by eliminating the need for engineering discipline. Instead, it is changing where effort and risk are concentrated. Historically, the build versus buy decision was framed primarily as a tradeoff between speed, cost, and flexibility. Purchasing software enabled faster deployment, while building provided greater customization and control.

→ **AI is altering this balance.**

AI-assisted development significantly reduces the time required to produce initial application functionality. Teams can generate code, prototype features, and iterate on designs far more quickly than in previous development models.

However, this acceleration applies primarily to **time - to - first - version**, not to **time - to - production**.

Production-ready systems, particularly in healthcare and regulated environments, still require:

- **validated requirements**
- **secure implementation**
- **integration with complex ecosystems**
- **traceable documentation**
- **ongoing lifecycle governance**

In many cases, these requirements become more important as AI accelerates development, because more functionality can be created faster than it can be governed.

As a result, the core decision is shifting.
The question is no longer simply whether to build or buy software.

It is whether an organization should own or rent the capabilities that are critical to its operations, compliance posture, and long-term differentiation.

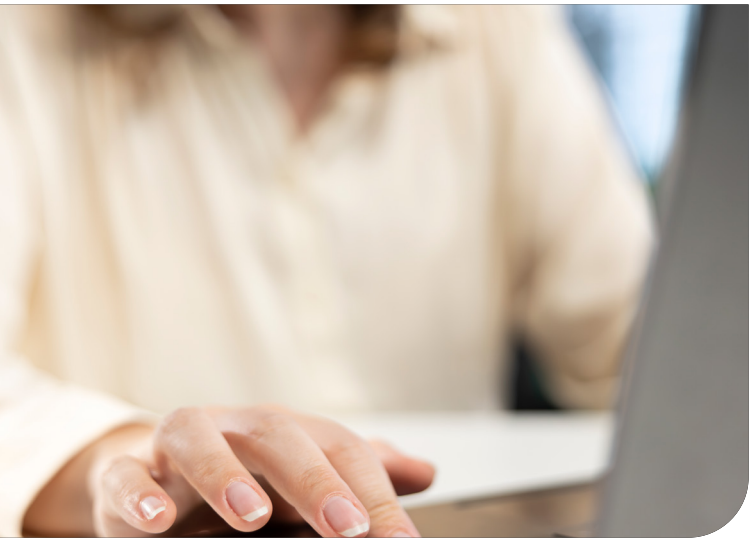
How AI Changes the Build vs Buy Equation

For many years, organizations evaluating new technology platforms have approached the **build versus buy decision** as a tradeoff between speed, cost, and flexibility.

Purchasing commercial software products has historically been attractive because it allows organizations to deploy capabilities quickly without building development teams or managing complex engineering projects. Commercial solutions typically include vendor support, regular updates, and established product roadmaps.

For many common business functions such as accounting, payroll, or customer relationship management, purchasing software can be a practical and efficient solution.

However, off-the-shelf software also introduces important limitations. Commercial products are designed to meet the needs of a broad customer base rather than the unique operational requirements of a specific organization. As a result, organizations often encounter challenges when attempting to customize these systems to fit specialized workflows.



Historically, many organizations favored purchasing software because the cost and time required to build custom systems were high, while many enterprise needs were sufficiently standardized. AI changes that calculus by lowering the cost of software creation—but not the cost of ownership.

Build vs Buy: Core Tradeoffs

Factor	Buy Commercial Software	Build Software
Deployment Speed	Faster initial deployment	Longer development timeline
Customization	Limited to vendor capabilities	Fully tailored to workflows
Integration	Vendor controlled integrations	Designed for internal ecosystem
Control	Vendor roadmap determines evolution	Organization retains control
Competitive Differentiation	Limited	Support Proprietary Capabilities

Additional Challenges in Regulated Industries

In regulated sectors such as healthcare and life sciences, these limitations can become particularly significant. Software systems frequently need to support specialized clinical workflows, integrate with medical devices or research systems, and maintain strict documentation and traceability requirements.

Another factor influencing the build versus buy decision is the long-term cost of maintaining internal development capabilities. According to data from the U.S. Bureau of Labor Statistics, maintaining internal engineering teams requires sustained investment in software developers, quality assurance professionals, and related technical roles.

In regulated industries, the required expertise extends beyond software engineering. Organizations must also maintain capabilities in:

- **cybersecurity**
- **regulatory compliance**
- **validation and verification**
- **operational monitoring**

These realities have historically led many organizations to favor purchasing software whenever possible.

However, as software becomes increasingly central to product innovation and digital transformation, organizations are recognizing that owning critical software platforms can provide strategic advantages, particularly when those platforms support proprietary capabilities or specialized workflows.

AI Is Changing Software Development

Artificial intelligence is transforming many aspects of software development. Over the past several years, new tools have emerged that allow developers to generate code, automate documentation, and accelerate early-stage prototyping.

For technology leaders, the significance of AI is not that it eliminates the need to build software. It reduces the cost of building differentiated systems—particularly when used within disciplined engineering processes. This shift makes custom development more accessible, but it does not reduce the complexity of delivering secure, compliant, and production-ready platforms.

Common Applications of AI in Digital Health Software Development

AI tools are increasingly used to support several software development functions:

- clinical workflow prototyping
- user interface and patient engagement design
- code generation and code review
- automated testing and test case generation
- requirements analysis and documentation
- validation and verification support
- interoperability and API integration support
- data quality and anomaly detection
- operational monitoring
- security risk analysis and vulnerability detection

By reducing the time required for repetitive tasks, these tools allow developers to focus more on system architecture, product design, and complex problem-solving.

However, while AI tools can accelerate development tasks, they do not eliminate the need for experienced engineering oversight.

Research evaluating large language models has shown that generative AI systems can produce plausible but incorrect outputs, a phenomenon commonly referred to as hallucination. Studies examining AI-generated scientific content have identified measurable error rates in generated references and documentation.

Additional research examining high-stakes professional tasks demonstrates that language models may produce confident responses even when the underlying information is inaccurate. These limitations are particularly important in regulated industries such as healthcare, where system behavior, documentation, and validation evidence must be accurate and auditable.

AI-assisted development can therefore improve productivity, but organizations must still design robust architectures, implement secure development practices, and establish governance frameworks that ensure reliability, security, and regulatory compliance over time.

AI-Generated Software Is Not Automatically Production Ready

The rapid adoption of generative AI tools in software development has created the perception that production-ready applications can be generated quickly with minimal engineering oversight. While AI-assisted development can accelerate coding tasks, research indicates that AI-generated outputs still require rigorous validation and governance before they can be used in production environments. AI compresses time-to-first-version. It does not compress time-to-production.

Studies examining the deployment of AI systems in healthcare emphasize that AI capabilities must be continuously monitored and evaluated throughout their lifecycle in order to maintain safety and effectiveness in real-world environments. In regulated environments, the standard is not whether software can be generated, but whether the resulting system can be validated, audited, maintained, and safely operated over time.

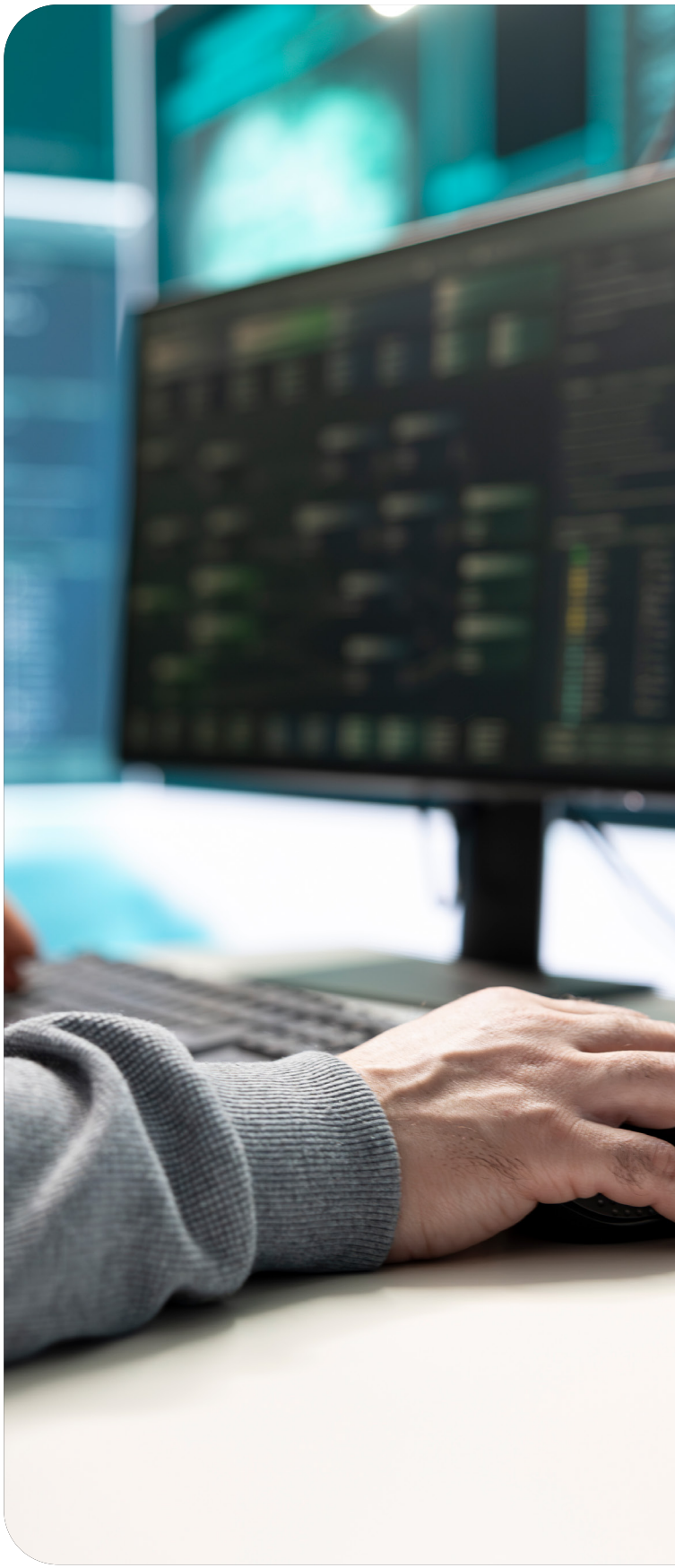
Implications for Regulated Software

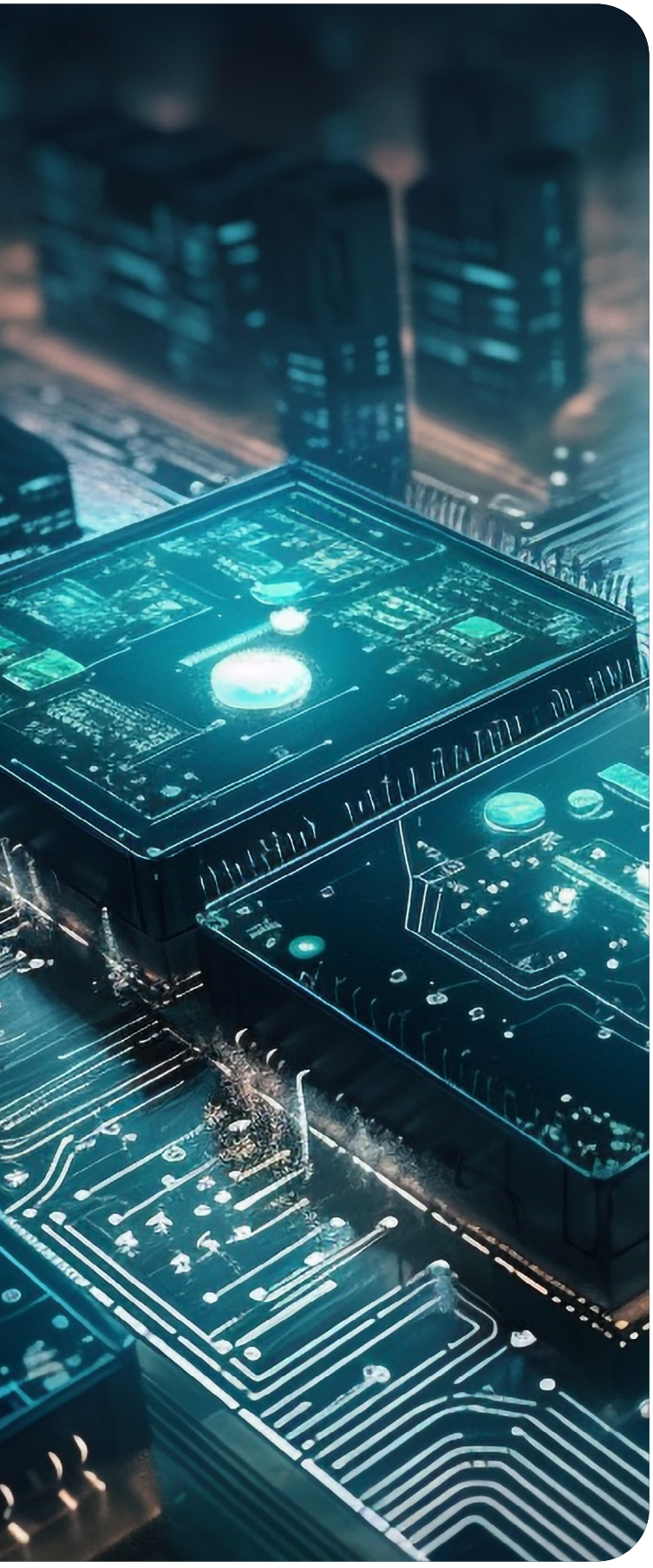
For organizations building regulated systems, these limitations have significant implications. Software used in healthcare, medical devices, and life sciences must meet strict requirements for:

- **traceability**
- **validation and verification**
- **documentation and auditability**

When AI tools are used in the software development process, the outputs they generate such as code, documentation, or test artifacts must be treated as part of the regulated system lifecycle. This means they are subject to the same expectations for validation, review, and traceability as any manually developed component.

Regulatory expectations from agencies such as the U.S. Food and Drug Administration emphasize lifecycle governance, documentation, and auditability of the final system. As a result, organizations must ensure that the use of AI in development does not compromise their ability to produce complete, verifiable, and compliant engineering artifacts.





Architecture and System Complexity

Another challenge arises from the complexity of modern software architectures. Production systems must operate reliably within environments that include distributed services, data pipelines, security controls, and integrations with external platforms.

AI-generated code typically focuses on localized functionality rather than system-level architecture. As a result, generated components must be carefully reviewed and integrated within broader architectural designs.

Human engineering oversight, therefore, remains essential to ensure that generated components function reliably within secure and scalable environments.

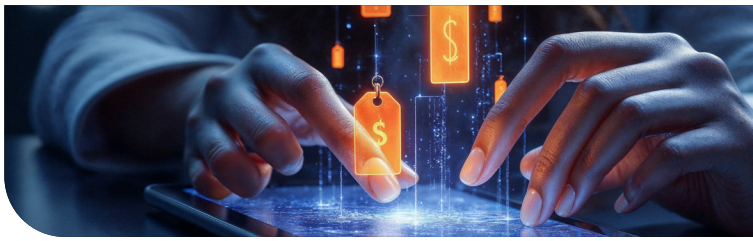
AI-generated software can be a valuable accelerator when used within disciplined development processes. However, organizations that rely on AI outputs without implementing appropriate review, testing, and validation practices risk introducing hidden technical debt and operational vulnerabilities.

AI Changes Cost Structure, Not Ownership Cost

AI has a meaningful impact on software development economics, but its effects are uneven across the lifecycle.

While AI reduces the time and expense of coding and prototyping, it does not significantly reduce the costs of operating and governing production systems.

In regulated environments, many of the most resource-intensive activities occur after initial development.



Impact of AI on Software Cost Structure

Cost Category	Impact of AI	Implication
Initial development	Often Decreases	Faster prototyping and feature delivery
Validation and verification	Shifts earlier and may increase	Regulatory burden remains
Compliance and documentation	Drafting effort may decrease; governance burden remains	Required regardless of development method
Integration	Some tasks accelerate; complexity remains	Ecosystem complexity persists
Cybersecurity	Risk profile expands	More dependencies and risks to manage
Operations and DevOps	Monitoring efficiency may improve	Systems must be continuously maintained
Maintenance and upgrades	Some maintenance tasks may accelerate; debt risk remains	Lifecycle cost dominates total cost

→ **This distinction is critical.**

AI reduces the cost of creating software. It does not reduce the cost of owning mission-critical software.

For organizations in healthcare and life sciences, where systems must operate reliably within regulated environments over long periods of time, lifecycle costs remain the dominant factor in technology strategy.

FDA Quality and Lifecycle Requirements

In the United States, the Food and Drug Administration (FDA) has introduced the Quality Management System Regulation (QMSR), which aligns U.S. device quality requirements more closely with the international ISO 13485 standard for medical device quality management systems.

According to the FDA, these regulations emphasize documentation, traceability, change management, and lifecycle governance for software systems used in regulated medical products.

Cybersecurity Expectations

Regulatory guidance from the FDA also highlights the importance of cybersecurity by design in medical device software.

Manufacturers are expected to incorporate cybersecurity controls throughout the development lifecycle and provide detailed documentation describing security architecture and risk management practices as part of premarket submissions.

Data Protection Requirements

Healthcare organizations must also comply with the HIPAA Security and Privacy Rules, which require organizations to conduct risk analyses and implement safeguards to protect electronic protected health information.

Guidance from the U.S. Department of Health and Human Services makes clear that these obligations apply regardless of whether software systems are developed internally or purchased from external vendors.

Implications for Build vs Buy Decisions

These regulatory realities have important implications for technology strategy. Purchasing software does not transfer regulatory responsibility to the vendor.

Organizations remain responsible for ensuring that systems meet applicable compliance requirements and that sufficient documentation and evidence exist for regulatory review and audits. This includes conducting appropriate due diligence to verify that vendors have the necessary qualifications, quality systems, and documentation to support regulated use.

As a result, many healthcare and life sciences organizations find that systems supporting regulated workflows require carefully engineered architectures and governance frameworks that align with their specific regulatory obligations, regardless of whether components are built internally or sourced from external vendors.

Regulatory Considerations for Software Development in Regulated Environments

1. FDA Quality and Lifecycle Requirements

The FDA's **Quality Management System Regulation (QMSR)** aligns with ISO 13485 and requires:

- documentation and traceability
- change control
- lifecycle governance
- validation and verification

2. Cybersecurity and Premarket Requirements

FDA guidance requires **cybersecurity by design**, including:

- secure development practices
- threat modeling and risk analysis
- documented security architecture
- cybersecurity evidence in submissions

3. HIPAA Compliance

The HIPAA Security and Privacy Rules requires:

- risk analysis
- safeguards for protected health information
- ongoing security management

These obligations apply to both built and purchased systems.

4. Implications for Build vs Buy

- Buying software does not transfer regulatory accountability
- Organizations must validate, document, and secure all systems
- Purchased solutions may not fully align with the specific regulatory workflows required by the organization.

Regulated systems often require custom architectures and governance frameworks, making strategic build approaches more viable for compliance-critical platforms.

Cybersecurity Risks in Modern Software Development

Third Party and Supply Chain Risk

Modern software systems, whether built or purchased, rely on third-party components such as APIs, open source libraries, cloud services, and vendor integrations.

According to the Verizon Data Breach Investigations Report, third-party involvement contributes to approximately **30 percent of security breaches.**

Common exposure points:

- external APIs
- open source dependencies
- cloud services
- vendor integrations

These dependencies are not inherently negative, but they introduce risks that must be actively managed. In purchased solutions, limited visibility into underlying components can make these risks harder to assess and control.

AI-assisted development introduces additional cybersecurity considerations. Generated code may include insecure patterns, dependency usage may expand rapidly, and AI-enabled features introduce new risks such as prompt injection, data leakage, and model misuse. These factors increase the importance of secure software development lifecycle practices rather than reducing them.

Third Party and Supply Chain Risk

The NIST Cybersecurity Framework and Secure Software Development Framework (SSDF) define:

- **risk management and governance**
- **secure design and coding**
- **continuous monitoring**
- **vulnerability management**

Process Debt and Technical Debt

As software systems evolve, organizations accumulate both technical debt and process debt, both of which can significantly impact long-term maintainability and operational risk.

Technical debt refers to compromises in code quality, architecture, or system design that increase the cost of future changes.

Process debt refers to weaknesses in development practices, including insufficient documentation, inconsistent testing, unclear requirements, or lack of governance.

AI-assisted development can accelerate the accumulation of both forms of debt if not properly managed.

Because AI tools make it easier to generate code quickly, organizations may produce functionality faster than they can validate, document, or integrate it within a coherent system architecture.

Over time, this can result in:

- **fragmented system design**
- **inconsistent coding patterns**
- **insufficient test coverage**
- **reduced traceability**
- **increased maintenance complexity**

In regulated environments, process debt can be particularly problematic, as missing documentation or validation artifacts can directly impact compliance and audit readiness.

Implications for Build vs Buy

- Vendor-based systems may introduce opaque dependencies and limited visibility
- Security responsibility remains with the organization
- Custom-built systems allow stronger implementation of security by design

Cybersecurity requirements often favor controlled, well-architected build strategies over black-box vendor solutions.

The Hidden Cost of Software Ownership

Software systems require continuous operational investment regardless of whether they are built or purchased.

Ongoing Responsibilities

- security patching
- infrastructure management
- monitoring and observability
- integration maintenance
- regulatory updates

Operational Reality

Organizations often struggle to maintain secure and up-to-date systems over time.

DevOps research shows that high-performing teams rely on:

- repeatable deployment processes
- continuous monitoring
- incident response capabilities

Implications for Build vs Buy

- **Buying software does not eliminate operational responsibility**
- **Vendor systems still require integration, monitoring, and compliance oversight**
- **Total cost must include lifecycle operations, not just acquisition**

Strategic decisions must account for long-term ownership costs, not just initial implementation.

Using AI to Build Software

Where AI Adds Value

AI can support several stages of the development lifecycle:

- **code generation and refactoring**
- **test generation and coverage analysis**
- **documentation drafting**
- **developer productivity tooling**
- **rapid prototyping of new features**

These capabilities can significantly improve development speed and enable teams to iterate more quickly.

The Limits of AI-Driven Development

Production systems require more than functional code. They must operate within architectures that address:

- **security and data protection**
- **system scalability**
- **regulatory compliance**
- **operational reliability**

Industry frameworks reinforce this requirement. According to the National Institute of Standards and Technology (NIST), the Secure Software Development Framework defines practices across requirements, design, implementation, testing, and incident response, emphasizing that security and governance must be embedded throughout the lifecycle.

Recent extensions to secure development guidance further highlight the need to manage AI systems throughout their lifecycle, including model evaluation, data governance, and misuse risk mitigation.

Integrating AI into Disciplined Engineering

These frameworks reflect a broader principle. AI capabilities are most effective when integrated into structured development pipelines that include:

- **secure coding practices**
- **validation and verification processes**
- **continuous monitoring**
- **controlled deployment workflows.**

When implemented within disciplined engineering environments, AI-assisted development can improve productivity while maintaining the reliability and compliance required for enterprise systems.

Why Strategic Build with the Right Partner Matters

For organizations operating in regulated industries, building software entirely in-house can be difficult to sustain without specialized expertise.

Software platforms supporting healthcare and life sciences products must address a combination of engineering, regulatory, and operational requirements that extend beyond traditional application development.

The Challenge of Internal-Only Development

Maintaining capabilities across engineering, compliance, and operations represents a significant organizational burden.

This includes not only development resources but also ongoing responsibilities related to security, validation, and system monitoring.

Research on software lifecycle economics shows that maintenance activities account for the majority of total system costs, reinforcing the importance of long-term sustainability in software strategy.

For regulated organizations, internal-only development can quickly stretch teams across too many specialized responsibilities, including:

- **architecture and engineering**
- **cybersecurity**
- **validation and verification**
- **regulatory documentation**
- **DevOps and monitoring**
- **maintenance and support**

This is why strategic build should not mean building alone. The stronger model is to keep internal ownership of the product vision, roadmap, and business priorities while using an experienced engineering partner to provide the specialized capabilities needed to build and sustain the platform.

The Strategic Build Model

As a result, adopting a strategic build approach is increasingly a winning strategy for organizations that need differentiated, reliable, and compliant software.

Rather than simply delivering code, experienced partners help establish repeatable and sustainable software delivery capabilities.

In this model, partners contribute to:

- system architecture and design
- secure development practices
- regulatory-aligned documentation
- validation and testing frameworks
- scalable DevOps processes

A Practical Framework for Build vs Buy Decisions

As AI changes the economics of software development, organizations need a more structured approach to evaluating build versus buy decisions.

The following framework provides a practical set of decision criteria.

1 Differentiation

Is the system key to your competitive advantage or intellectual property? If yes, building or owning the system is often the preferred approach.

2 Regulatory Compliance

Does the system require traceability, validation, or regulatory oversight? If yes, controlled development and governance become essential.

3 Complex Integration

Will the system need to integrate deeply with internal systems, devices, or data environments? If yes, custom architecture may be required.

4 Rate of Change

Will the system evolve rapidly as workflows, models, or requirements change? If yes, building enables greater adaptability.

5 Operational Risk

What is the impact of failure, downtime, or misalignment with business needs? If high, greater control and ownership are typically warranted. The more a system scores highly across these dimensions, the stronger the case for a strategic build approach.

How Estenda Helps Organizations Build the Right Way

The value of this approach is not only in accelerating development, but in reducing the long-term risks associated with architecture decisions, regulatory compliance, and lifecycle management. For organizations building complex, AI-enabled systems in healthcare and life sciences, this combination of engineering discipline and regulatory awareness is critical to achieving sustainable outcomes.

Core Principles

Embedded Security and Compliance

Security and compliance are integrated into development processes from the beginning. According to NIST guidance, secure development practices should be embedded throughout the software lifecycle rather than treated as separate activities.

Architecture and Lifecycle Design

Software systems are designed to evolve over time while maintaining traceability, documentation, and regulatory alignment.

Outcome

By combining engineering expertise with regulatory awareness, Estenda helps organizations build platforms that are:

- **secure**
- **compliant**
- **scalable**
- **maintainable over time**

This enables organizations to adopt AI and other advanced technologies while maintaining operational discipline.

Conclusion

Artificial intelligence is changing how software is developed, but it does not reduce the complexity of building systems that are safe, compliant, and reliable in real-world healthcare environments.

For organizations in **medtech, life sciences, and digital health,** software is no longer a supporting function. It is a core component of clinical value, product differentiation, and long-term competitive advantage. These systems must operate within strict regulatory frameworks, integrate with complex ecosystems, and evolve continuously as standards, data, and clinical needs change.

In this context, purchasing software may offer speed, but it rarely delivers the level of control, adaptability, and alignment required for regulated, innovation-driven environments. It can introduce constraints that limit differentiation, complicate compliance, and create long-term dependencies on vendor roadmaps.

As a result, leading organizations are moving toward a **strategic build approach**. They are investing in platforms that are purpose-built around their workflows, data models, and regulatory obligations, and designed to evolve over time.

This does not mean building in isolation. The most effective organizations combine internal product leadership with experienced development partners who bring deep expertise in regulated systems, secure architecture, and lifecycle management.

AI plays an important role in accelerating this process, but it is not a substitute for disciplined engineering. Long-term success depends on building systems that are governed, validated, and continuously managed throughout their lifecycle.

For healthcare technology leaders, the implication is clear. The question is no longer whether to build or buy. It is whether your organization is prepared to **own and shape the software platforms that define your future.**

Artificial intelligence is changing how software is built, but it does not reduce the responsibility of owning systems that are safe, compliant, and reliable. For healthcare technology leaders, the question is no longer simply whether software can be developed faster. It is whether the organization is prepared to own and govern the platforms that define its future.

AI makes building easier. It does not make ownership optional.