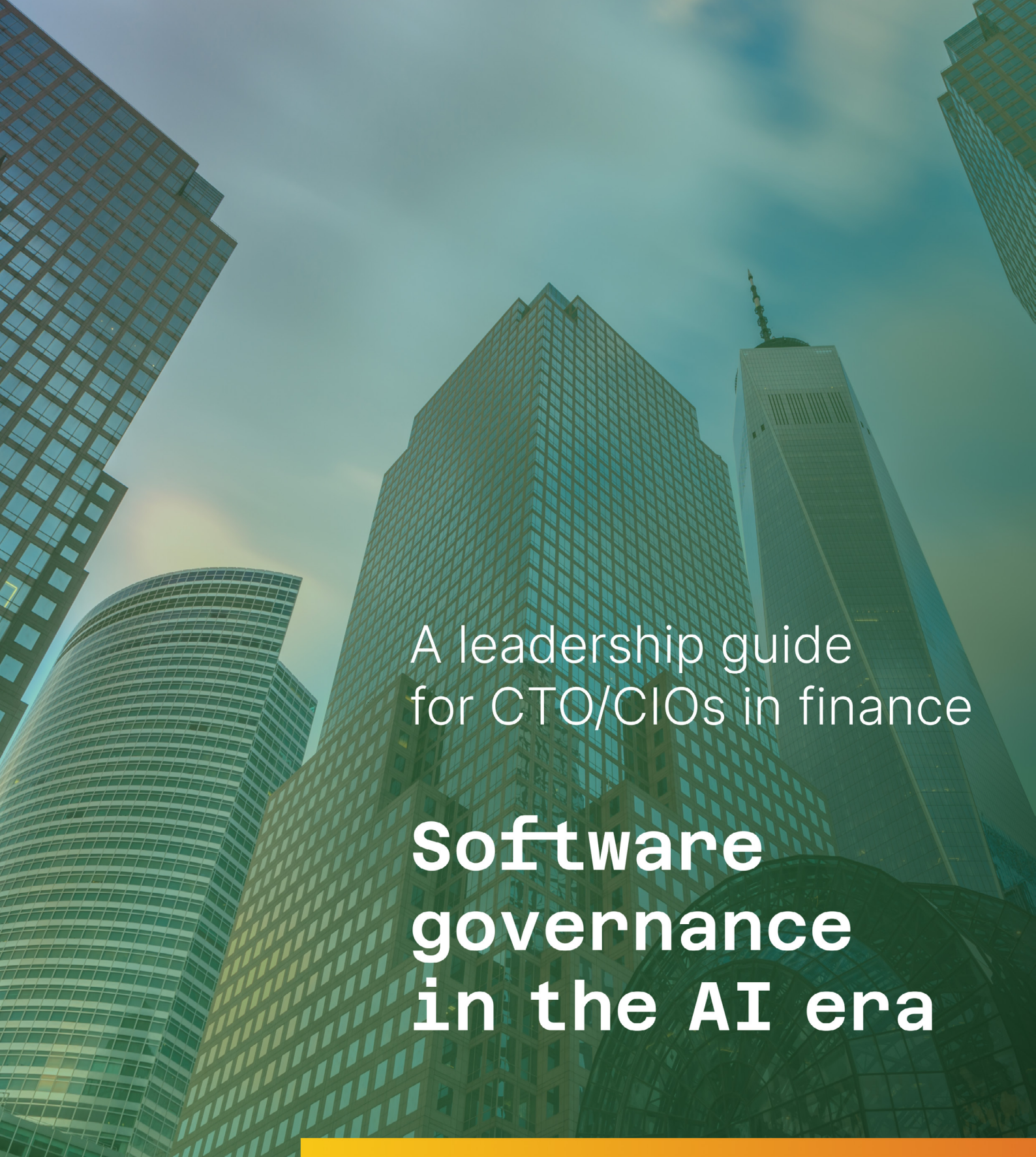




A leadership guide  
for CTO/CIOs in finance

# Software governance in the AI era



# Executive summary

Software governance in financial services is entering a new era. As AI<sup>1</sup> reshapes how we build and run systems—from AI-assisted coding to AI systems—CTOs<sup>2</sup> face a dual challenge: moving faster, while staying in control.

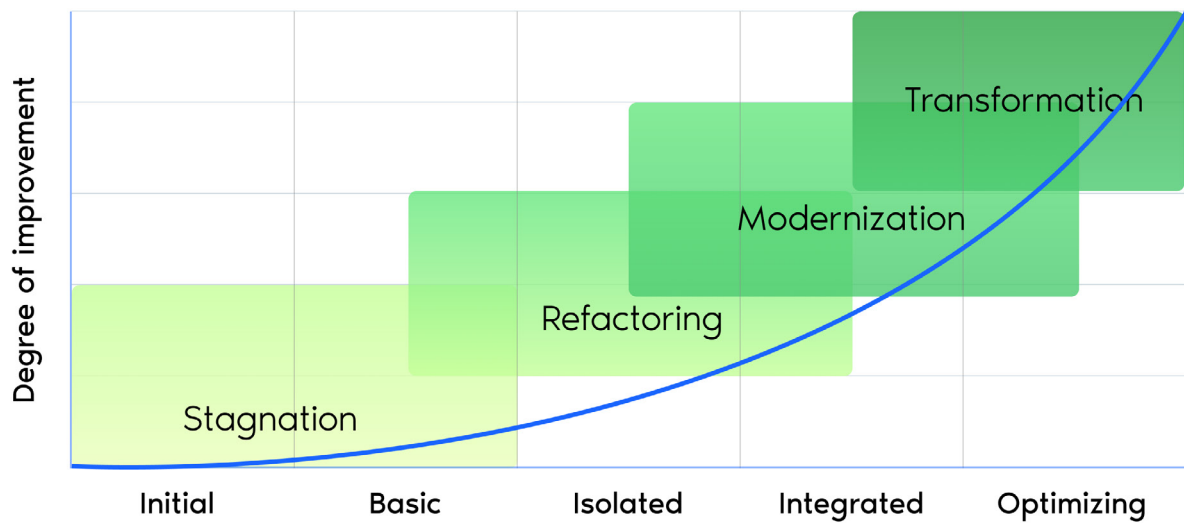
- AI-assisted coding promises faster delivery. But without clear governance, it risks accelerating complexity and cost instead of creating value. In financial services, where system stability, security, and regulatory compliance are non-negotiable, this risk is amplified.
- AI systems are software systems and should be treated as such. That said, AI systems are unlike any software we've worked with before. They learn, adapt, and operate with a level of autonomy that traditional systems don't. This creates opportunities to improve efficiency, decision-making, and customer experiences, but also complexity, and with it, real risks.

Software maturity and AI adoption are parallel journeys that intersect at multiple points. Financial institutions don't have to wait for one to complete before starting the other, but they do need a clear understanding of where they stand on both, to make informed decisions that drive progress.

This guide by Software Improvement Group (SIG) helps CTOs in financial services:

- Understand and mitigate common risks involved in AI coding practices
- Treat AI systems as software systems with unique characteristics and optimize the engineering process
- Diagnose their current software maturity from a holistic perspective
- Map their maturity journey across a four-stage framework—**from Stagnation to Transformation**

The outcome of this guide is being able to create a **practical roadmap** that builds trust and advances the software maturity of your organization.



### Stage 1 - Stagnation

Most organizations we assess are in this stage – operating with limited insight, driven by engineering priorities rather than business outcomes.

### Stage 2 - Refactoring

Organizations here have moved beyond firefighting, but structural improvements are still gradual and localized.

### Stage 3 - Modernization

At this stage, teams can tackle coordinated system-level change, but without full business alignment, opportunities are still missed.

### Stage 4 - Transformation

Few organizations are at this point: IT is a strategic partner to the business, driving innovation as much as enabling it. Getting here should be a north star for CTOs.

<sup>1</sup> In this guide, we distinguish between:

- **AI-assisted code development** ("AI code"): for example, using AI coding assistants and Large Language Models to generate code used to develop or refactor code.
- **Building and managing AI systems** ("AI systems"): for example, creating or implementing AI systems.

<sup>2</sup> For readability, this guide refers to the role of CTO. However, responsibilities often overlap with those of the CIO—depending on the organizational structure—it is equally applicable to CIOs.

# Maturity matters in the age of AI

In finance, CTOs oversee some of the most complex technology ecosystems in business, where regulation, security, and legacy systems amplify both opportunity and risk. You're responsible for managing a complex, evolving ecosystem including code, architecture, teams, and processes – often without complete visibility into how these elements connect.

That lack of visibility is not just operationally inefficient; it's a governance risk.

In IT, basic questions like *"How many applications do we have? How much has been invested in them? Who's working on what, and what are they delivering?"* often get only approximate answers. In other parts of an organization, that level of uncertainty would be unthinkable. In finance, for example, a CFO is expected to know precisely how much cash is available, the level of debt, and the liabilities on the balance sheet.

Most organizations operate with limited insight into the state of their software landscape. Common characteristics include:

- Architecture is inherited and rarely questioned.
- Critical risks and inefficiencies remain hidden until they become costly problems.
- Digital transformation initiatives often launch without a grounded understanding of the foundations they depend on.

These gaps have always [carried risk](#). Today, an average system contains **19 security vulnerabilities**, and poor build quality can already drive annual maintenance costs into the millions.

And it's not just our research; Deloitte interviewed technology and software engineering leaders from various banks in 2024 and revealed that many banks' engineering projects [tend to be inefficient and lack scalability](#). This can lead to [expensive maintenance, integration challenges, and reduced performance speeds](#).

Today, AI can accelerate productivity and innovation. However, without mature governance, it can also scale hidden weaknesses faster.

# AI coding assistants are changing how we build software

The biggest challenge is no longer writing code; it's about understanding it at scale and governing it across your systems.

In FSI, adoption is already widespread, with [84% of developers in banking using LLM-driven coding assistants](#). This means the question isn't whether teams will adopt these tools, but how effectively organizations can govern their use.

So, let's first address the elephant in the room: If your software maturity is low, should you hold off on AI coding adoption?

That's like asking a group of children not to touch the candy you left out on the table...it's simply not going to work.

The key isn't prohibition. It's governance.

Rather than blocking adoption, [financial institutions are being urged to evolve governance frameworks](#). Measure results, build guardrails, and let teams experiment responsibly. This builds the confidence, control, and insight needed to scale AI use safely as maturity grows.

The risks discussed in this chapter aren't reasons to avoid AI. They're reminders that success depends on **intentional use, supported by visibility and oversight**. AI can – and should – enhance productivity. But it must not come at the expense of quality, security, or compliance.

# Four common AI coding risks

AI-assisted coding introduces powerful opportunities to improve developer productivity and accelerate delivery. But like any new capability, it also introduces risk.

In financial services, those risks are amplified by strict regulatory oversight, legacy environments, and the high operational cost of failure. Below are four common risks to consider, and the practical steps you can take to manage them from the start.



## Gen AI output depends on existing data

Gen AI makes suggestions based on patterns in its training data. LLMs are trained on market-average code and rely on associative, pattern-based reasoning, which makes them fast but not always accurate. Furthermore, AI is much better at generating standard code than it is at reviewing code and quality checks.

- [25% of developers estimate that 1 in 5 AI-generated suggestions contain factual errors or misleading code.](#)
- Without proper review, these errors can spread through the codebase and create hidden defects. Gartner predicts that by 2027, 25 percent of software defects escaping to production will result from a [lack of human oversight of AI-generated code.](#)

### Mitigation:

- Treat AI-generated code like any external dependency: require human review before merging.
- Educate developers on responsible use, and measure impact (acceptance rates, defects, rework) to guide standards.
- Consider adding [automated guardrails](#) in the IDE to uphold code quality and maintainability levels.



## Security vulnerabilities are common in AI code

AI-generated code introduces serious security risks because it's often trained on flawed or outdated public code. Next to this, it may include insecure coding patterns or even hallucinate components –like made-up package names–which attackers can exploit.

In addition, because AI lacks awareness of your architecture and policies, it can accidentally bypass security protocols or introduce unsafe dependencies.

In financial institutions, where critical systems process sensitive data and support high-value transactions, these vulnerabilities [can quickly escalate](#) into compliance breaches or incidents affecting operational resilience.

- Studies show that [around 40% of LLM-generated code suggestions](#) include vulnerabilities.

**Mitigation:** Developers should exercise caution in relying solely on LLMs for security-critical tasks. Specialized measures are imperative to address the security weaknesses.

- Treat AI-generated code like any external dependency: require human review before merging.
- Enforce stricter review gates for security-critical areas.
- Consider establishing [automated security guardrails](#) in the IDE so that security considerations can be factored into the design phase.



## Architectural complexity

AI tools lack context about your internal codebase, architecture, and business rules, which can result in hard-to-maintain systems, increased technical debt, and failed modernization projects.

This risk is particularly acute in finance, where many institutions still rely on legacy systems to support critical operations. 37% of FSI systems with legacy technologies have a below-average architecture rating, compared to just 11% for modern technologies. Poor architecture slows down change significantly, making updates 40% slower compared to higher quality systems.

In such complex environments, ungoverned AI-generated code can amplify these issues. On top of this, AI-generated code is difficult to distinguish from human-written code, making it challenging to track its usage across the software portfolio.

- [65% of developers](#) say AI coding assistants lack awareness of the overall codebase, internal architecture, or company standards.

**Mitigation:** Because LLMs don't have full system context or understanding of your architecture, they can't see system-wide architecture, dependencies, or duplication patterns.

- Don't rely on LLMs for system-level changes; complement with [deterministic analysis solutions](#).
- Get a holistic view of [AI coding assistant adoption and the volume of AI-generated code across your portfolio](#), so that you can track usage and adoption effectively.



## Legal complications

AI-generated code snippets can resemble open-source components covered by restrictive licenses, creating potential compliance risks.

- Roughly [35% of AI-generated samples](#) contain possible licensing conflicts or unverified provenance.

**Mitigation:** Consider implementing comprehensive safeguards against AI intellectual property risks. For example, combining human oversight with sophisticated [license detection tools](#) that scan AI-generated code for potential infringement issues before integration.

- Treat AI-generated code with the same scrutiny as third-party dependencies.
- Use license scanners and provenance tools before integration.
- Maintain Software Bill of Materials (SBOM), track usage, and require attribution where necessary.
- Establish a clear policy around acceptable use of AI-generated content in production environments.

As described earlier, and in essence, many of the risks are not unique to AI coding. Maintainability, security, architecture, and licensing issues have always existed. But generative AI can amplify and accelerate them at scale, especially when used without visibility or oversight.

For financial institutions, these risks underline a critical truth: **technology decisions are governance decisions**. With strong guardrails in place, AI can enhance delivery and resilience. Without them, it can expose institutions to operational, legal, and regulatory risk.

Now that we've zoomed in on AI coding specifically, let's zoom in on the other use case: AI systems.

# AI systems: A new engineering frontier

Next to staying on top of AI code practices across the SDLC, CTOs play a pivotal role in ensuring that organizations fully harness the value of AI systems as well.

Their leadership is crucial in steering AI-system initiatives toward achieving tangible business value. The right approach helps pinpoint where AI systems can truly enhance your business, while embedding AI within existing frameworks to minimize disruption and enable sustainable growth. In financial services, this can mean applying AI to enhance critical processes like customer service, fraud detection, credit risk modelling, payments, and regulatory reporting – where system reliability and explainability are essential.

While AI-assisted coding changes how we *build* software, AI systems change what software is.

## AI systems vs. traditional software

Unlike conventional software, which follows fixed, pre-programmed rules, AI learns, evolves, and makes autonomous decisions.

According to [ISO/IEC 5338](#), co-developed by Software Improvement Group (SIG), AI is classified as a software system with unique characteristics. These include the ability to think autonomously, learn from data, make decisions based on that data, and even potentially talk, see, listen, and move.

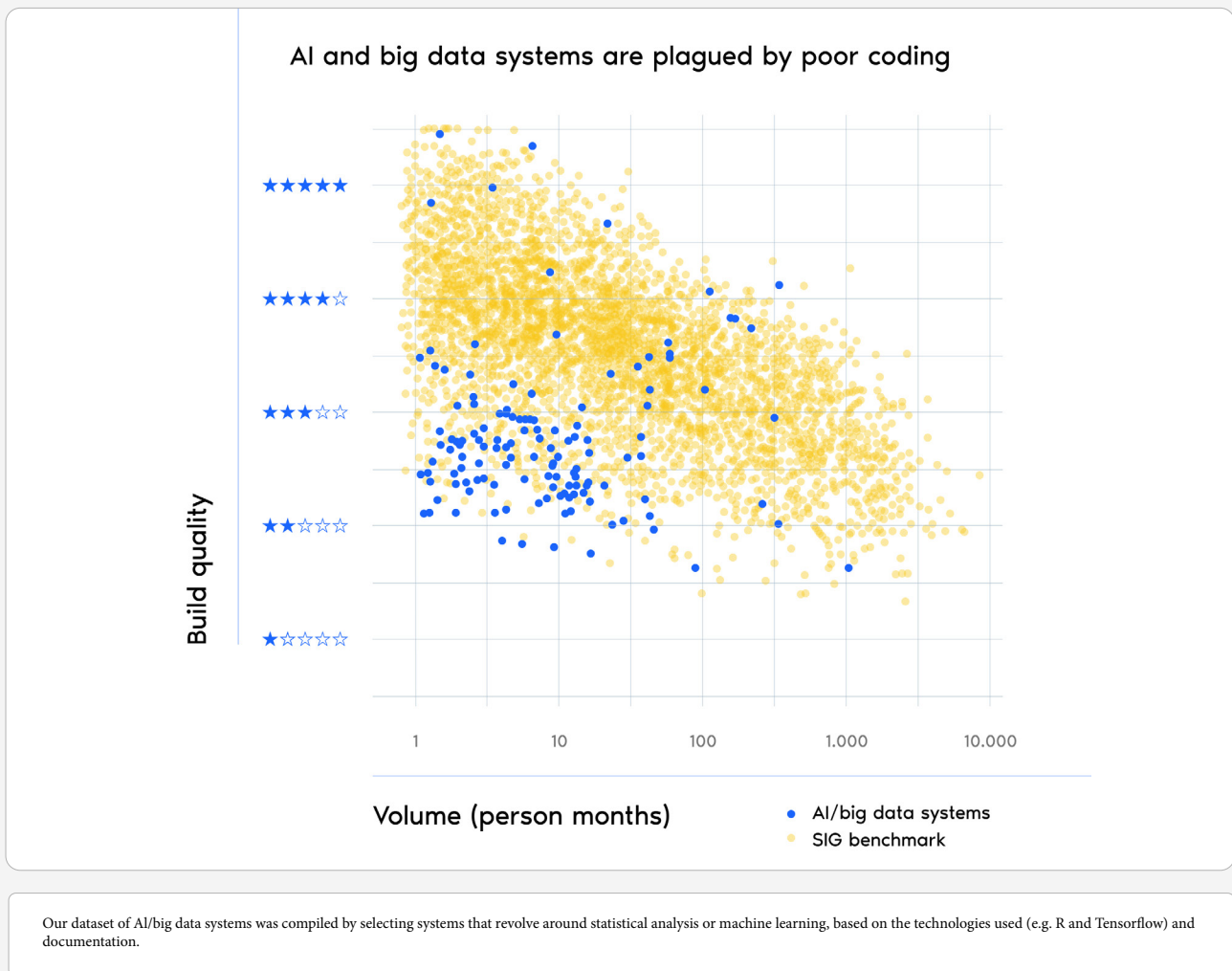
In addition, AI needs regular retraining to stay accurate.

What sets [AI software apart from traditional software](#) is that it doesn't just follow fixed rules. Instead, it learns by analyzing large sets of data, finding patterns, and using that knowledge to make educated guesses when making decisions, solving problems, or answering questions.

Because of its unique features, AI is often misunderstood and can carry risks, including security issues, mistrust, and potential harm. To manage these risks, AI systems need strong engineering practices and proper regulation within the organization.

## The AI system maintainability crisis

Our benchmark data exposes a serious issue: most AI and big data systems suffer from poor maintainability and testability. In fact, [73% of AI/big data systems score below the benchmark average in maintainability](#), with an average rating of just 2.7 stars, significantly lower than traditional software systems.



But why is this happening?

Our research highlights two major issues:

- **Complex, bloated code:** AI systems often have long, unfocused code blocks handling multiple responsibilities, making them difficult to modify, analyze, and reuse.
- **Lack of testability:** AI systems contain just 1.5% test code, compared to 43% in traditional systems, making errors harder to detect, and AI models riskier to update.

## The role of strong architecture

As AI adoption grows, the underlying architecture becomes just as important as the model.

According to [Dataversity](#), nearly 50% of organizations are expected to modernize their data architectures to support real-time analytics and enable AI/ML capabilities. Gartner also notes that CIOs need a tech stack capable of integrating a wide variety of AI and data inputs.

In other words: if your architecture isn't ready, your AI systems won't be either.

Modern architectures are key to enabling:

- Seamless data integration
- Continuous model training and retraining
- Scalable deployment of AI services
- Faster, safer experimentation across teams

Without these foundations, AI initiatives get stuck in prototypes and never make it to production.

## The art of AI engineering

Engineering robust, future-proof AI systems is still a relatively new field.

We see many organizations struggling to transition AI from experimental projects in the lab to scalable, secure, compliant, and maintainable real-world applications.

The engineering challenges stem from how AI engineers—such as data scientists—are traditionally managed and trained. Their focus is often on quickly creating insights and models, not on building systems that are secure, reliable, maintainable, reusable, easy to transfer, and testable.

## AI systems and Python

Most AI systems are built using Python, a go-to language for data science and machine learning.

While it is certainly possible to create and maintain high-quality AI systems in Python, we often see that Python also lends itself to building systems with high anticipated maintenance costs.

This issue is not an inherent flaw of the Python language itself. Rather, it arises from the way projects are developed.

Our recommendation is to incorporate software engineering skills into your AI projects.

This approach helps prevent the creation of legacy code and avoids developing AI codebases that are difficult to test, modify, and scale.

## Building reliable AI systems: best practices

To make AI stable, secure, and scalable, organizations must treat it as serious software. That means moving beyond experimental code and adopting robust engineering practices from the start.

Key steps include:

- **Bridging AI and software engineering teams**  
Integrating software engineers into AI projects ensures long-term stability and future-proofing.
- **Applying software engineering best practices**  
AI development must prioritize modular, maintainable, and well-documented code.
- **Strengthening AI governance**  
Organizations must define clear policies for AI transparency, ethics, and regulatory compliance.
- **Implementing continuous validation**  
AI models degrade over time and require regular retraining and rigorous testing.

## Making AI systems dependable, at scale

As this chapter has shown, AI systems are certainly software, but it's unlike any software we've worked with before. It learns, adapts, and operates with a level of autonomy that traditional systems don't.

That creates opportunity, but also complexity, and with it, real risks to security, reliability, and trust. To build AI systems that are safe, scalable, and aligned with business goals, CTOs must adopt disciplined engineering practices, including:

- Continuous validation
- Regular retraining
- Strong documentation
- And clear accountability throughout the lifecycle

Treating AI as a core capability is the only way to move beyond experimentation. For a more detailed approach, download our free [AI Readiness Guide](#).

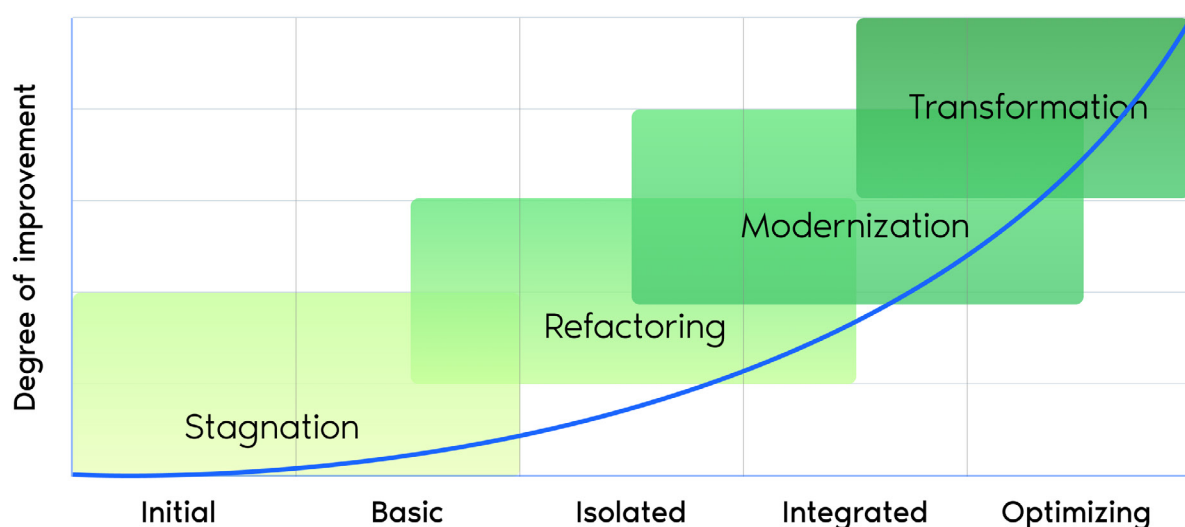
# A practical framework: Four stages of software maturity

Now that we have covered some of the new challenges and perspectives of AI code and AI systems, let's zoom back out and look at software maturity from a more holistic point of view.

As a CTO, you are responsible for your organization's technology strategy, ensuring it aligns with business goals. Making sound decisions starts with knowing where you're standing.

In FSI, this clarity is critical. Regulatory scrutiny, cyber risk, and legacy constraints mean that even small changes in software quality can have [disproportionate business impact](#). The difference between stagnation and transformation often comes down to governance: how well you can see, measure, and manage your software estate.

To support this, Software Improvement Group has developed a four-stage maturity model based on 25+ years of analyzing software systems and guiding major financial institutions to increase innovation, lower costs, and move faster.



This framework helps CTOs to quickly diagnose software risk and organizational readiness, and align tech investments with reality.

| Stage          | What you'll see                                                                                                                 | What to do                                                                                           |
|----------------|---------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| Stagnation     | Legacy systems, no inventory, hidden risk, unclear ownership.                                                                   | Establish visibility across code, risk, and governance. Clarify roles and responsibilities.          |
| Refactoring    | Known issues and patchwork fixes persist; security issues and technical debt are increasingly visible but remain unprioritized. | Align fixes to business value, elevate buried risk, build roadmap buy-in across teams.               |
| Modernization  | Strategic roadmap forming, architecture under rework.                                                                           | Strengthen architecture, scale metrics, connect modernization to outcomes with consistent processes. |
| Transformation | Mature architecture, trusted metrics, AI-ready processes and governance.                                                        | Operationalize monitoring, quantify impact, and align KPIs to business value.                        |

Each stage calls for different priorities. The following overview describes what you'll see, and what to do next.



# Stage 1: Stagnation

Most FSI organizations we assess are here – operating with limited insight, driven by engineering priorities rather than business outcomes.



## Typical signs

- IT is engineering-led, with budgets treated purely as cost-control tools.
- Focus on delivering features and fixing bugs, not addressing structural debt.
- Management-level insight into system quality, risk, and ownership is missing.
- Roles and responsibilities are unclear, with ownership of systems often fragmented across teams or individuals.



## CTO priorities

- Establish visibility: baseline code quality, architectural health, and dependencies.
- Introduce measurable quality metrics across teams and systems.
- Focus on understanding before implementing major changes.



## Stage 2: Refactoring

Organizations here have moved beyond firefighting, but structural improvements are still gradual and localized.



### Typical signs

- Some structural improvements at the code level.
- Technical debt is being reduced, but fixes are localized.
- Limited architectural insight; changes are often reactive.
- Cross-team communication is ad hoc; success depends on individual champions rather than standardized processes.



### CTO priorities

- Address root causes of recurring issues, not just symptoms.
- Build an improvement backlog tied to business outcomes.
- Enable cross-team conversations on architecture standards.



## Stage 3: Modernization

At this stage, teams can tackle coordinated system-level change, though a lack of full business alignment still limits the overall impact.



### Typical signs

- Coordinated modernization efforts at the system level.
- Ability to re-platform, re-architect, and replace outdated components.
- Process discipline improves; cross-functional groups ensure alignment at the system level.
- Executing major changes across multiple layers or systems simultaneously is still challenging.
- Business context is sometimes missing, so local optimizations don't always add up to the best enterprise-wide outcome.



### CTO priorities

- Scale architectural standards across teams.
- Embed business context and process mapping into modernization work to ensure changes benefit cross-functional workflows and multi-system outcomes.
- Introduce governance that balances speed with quality.
- Connect modernization efforts directly to revenue, cost savings, and resilience.



## Stage 4: Transformation

Few organizations are at this point: IT is a strategic partner to the business, driving innovation as much as enabling it. Getting here should be a north star for CTOs.



### Typical signs

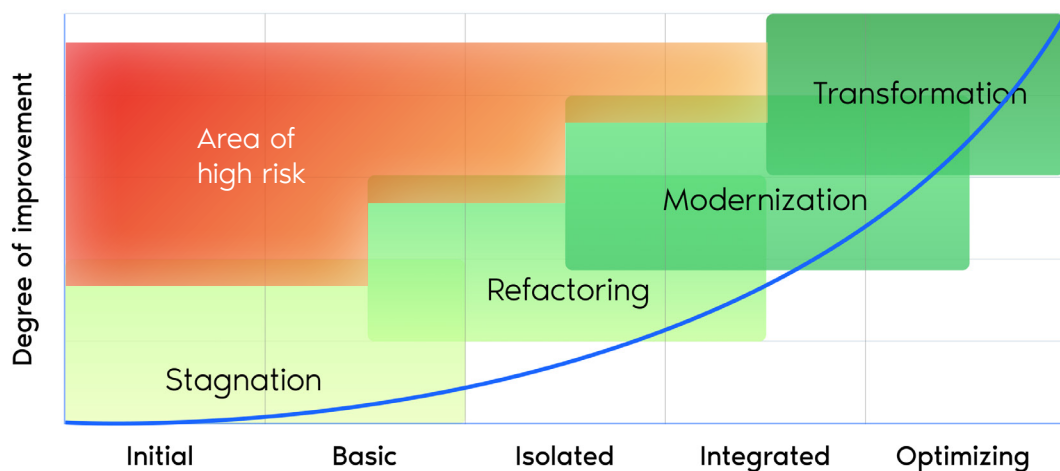
- IT and business have a shared understanding; strong governance ensures the organization knows where to invest, where impediments lie, what priorities to set, and which business growth opportunities to pursue through focused IT investments.
- Enterprise architecture actively drives innovation and market differentiation.
- Trusted, real-time insights available at the C-level, in decision-ready formats.



### CTO priorities

- Continuously monitor quality, risk, and performance across the entire landscape.
- Invest in long-term resilience alongside speed.
- Use architecture as a lever for business model innovation.

# Avoiding the “high-risk leap”



With your maturity level defined, the next challenge is pacing change effectively. After diagnosing the four stages, it's tempting to jump straight to the end.

For many financial institutions, this pressure is amplified by regulatory timelines and customer expectations. That's where many CTOs stumble into the 'area of high risk.'

The reality: skipping maturity stages rarely accelerates transformation. Instead, it creates fragile architecture, escalating costs, and initiatives that stall under their own weight.

Think of the four stages as a sequence, not a menu. Each stage builds the technical, organizational, and cultural capacity to sustain the next. Moving too fast risks amplifying unresolved weaknesses, especially with AI, where automation can spread problems at scale.

## To move at the right pace:

- **Anchor your strategy to your current stage.** Invest in the improvements that will unlock the next level of maturity
- **Make progress visible.** Track and communicate the improvements in quality, risk reduction, and delivery metrics so stakeholders back each step.

# Define, then decide

## Self-assessment: Where are you now?

Before deciding how to modernize or transform, it's essential to pinpoint your current maturity stage.

Tick the statements that are true for your organization – your highest “cluster” will likely indicate your maturity level.

|                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  <h3>Stagnation</h3> <ul style="list-style-type: none"> <li><input type="radio"/> IT budget is treated purely as a cost-control tool.</li> <li><input type="radio"/> Focus is on delivering features and fixing bugs, not structural improvement.</li> <li><input type="radio"/> No management insight into technical debt or architecture.</li> </ul> |  <h3>Refactoring</h3> <ul style="list-style-type: none"> <li><input type="radio"/> Some visibility into technical issues at the system level.</li> <li><input type="radio"/> Quality is mainly measured through testing, not proactive metrics.</li> <li><input type="radio"/> Limited architecture discussions happen, but without cross-team consistency.</li> </ul> |
|  <h3>Modernization</h3> <ul style="list-style-type: none"> <li><input type="radio"/> Certain systems have been modernized, but improvements are siloed.</li> <li><input type="radio"/> Some management insight into system-level architecture.</li> <li><input type="radio"/> Business objectives occasionally inform IT priorities.</li> </ul>       |  <h3>Transformation</h3> <ul style="list-style-type: none"> <li><input type="radio"/> IT and business strategies are mostly aligned at the system and enterprise level.</li> <li><input type="radio"/> There's active governance of architecture and technical debt.</li> <li><input type="radio"/> Management decisions use real-time technical insights.</li> </ul> |



# Conclusion: Governance as the cornerstone

Strong governance and clear visibility are not just about control. They are the foundations of maturity. At lower maturity levels, decisions are reactive and risks remain hidden. As governance strengthens, people, processes, and technology can better align so that every change is intentional and every investment deliberate.

At its core, governance is about embedding transparency, accountability, and alignment at every level.

In the AI era, software is created, changed, and deployed faster than ever. That velocity can unlock immense value or magnify hidden risk, depending on the strength of governance behind it.

Many [financial organizations report](#) that the technology improves software development, customer service, and other areas. However, issues around risk, regulatory compliance, data security, privacy, and governance have hindered the pace of adoption.

With the right governance in place, AI becomes a powerful multiplier of innovation for CTOs. Without it, it can become a fast track to fragility.

# Written by Software Improvement Group

Software Improvement Group (SIG) empowers organizations to govern the software their business runs on. Through complete portfolio analysis and tailored strategic advice, SIG helps companies embrace AI with control, improve software quality and security by focusing strategic efforts across people, process, and technology.

Sigrid®—SIG's software governance platform—analyzes over 400 billion lines of code across more than 30,000 systems and 300+ technologies, offering evidence-based insights to help organizations prioritize and manage their most critical IT initiatives.

Founded in 2000 and headquartered in Amsterdam, SIG has offices in New York, Copenhagen, Brussels, and Frankfurt. The company complies with leading ISO/IEC standards, including 27001 and 17025, and co-developed ISO/IEC 5338—the new global standard for AI lifecycle management.

Combining expert consulting with over 25 years of industry-leading research, SIG is the global authority on software portfolio governance.

Trusted by

 TELUS

  
KLM  
Royal Dutch Airlines

















*"Software Improvement Group helped us keep control of technical debt and identify where we needed to focus."*

**Harald Thoonen,**  
Solution Architect at Rabobank

*"Tooling like Sigrid provides transparency, allowing us to manage our software proactively and maintain high standards. This is crucial for securely sharing personal data in our digital processes and staying ahead of potential security risks."*

**Kelly Bonneure,**  
Program Coordinator – MAGDA, Digitaal Vlaanderen.

*"Thanks to Sigrid, we've significantly enhanced our expanding software portfolio's quality, slashing maintenance costs by 75%, effectively mitigating open-source security risks, and allowing us to make better-informed investment decisions."*

**Anthony Fitzpatrick,**  
VP Engineering at Kallidus

Find out how we help you  
reach your next stage of  
software maturity



