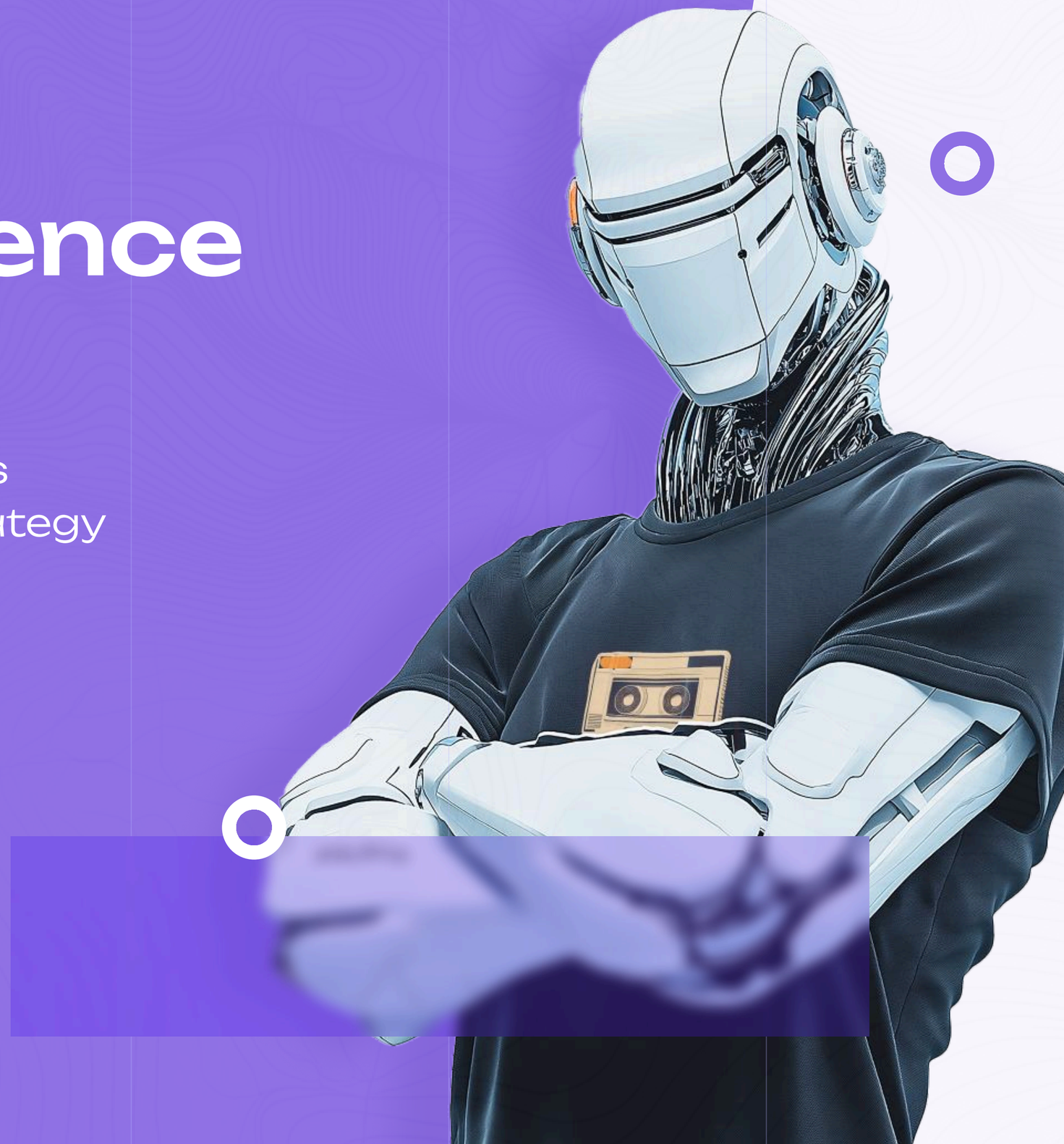
The Intelligence Advantage

How AI-Driven Development is Reshaping Software and Strategy



AMSTERDAM STANDARD

kyln AI

Executive Summary

AI is reshaping software development at breakneck speed. This white paper explores how programming practices and business outcomes are being transformed as a result.

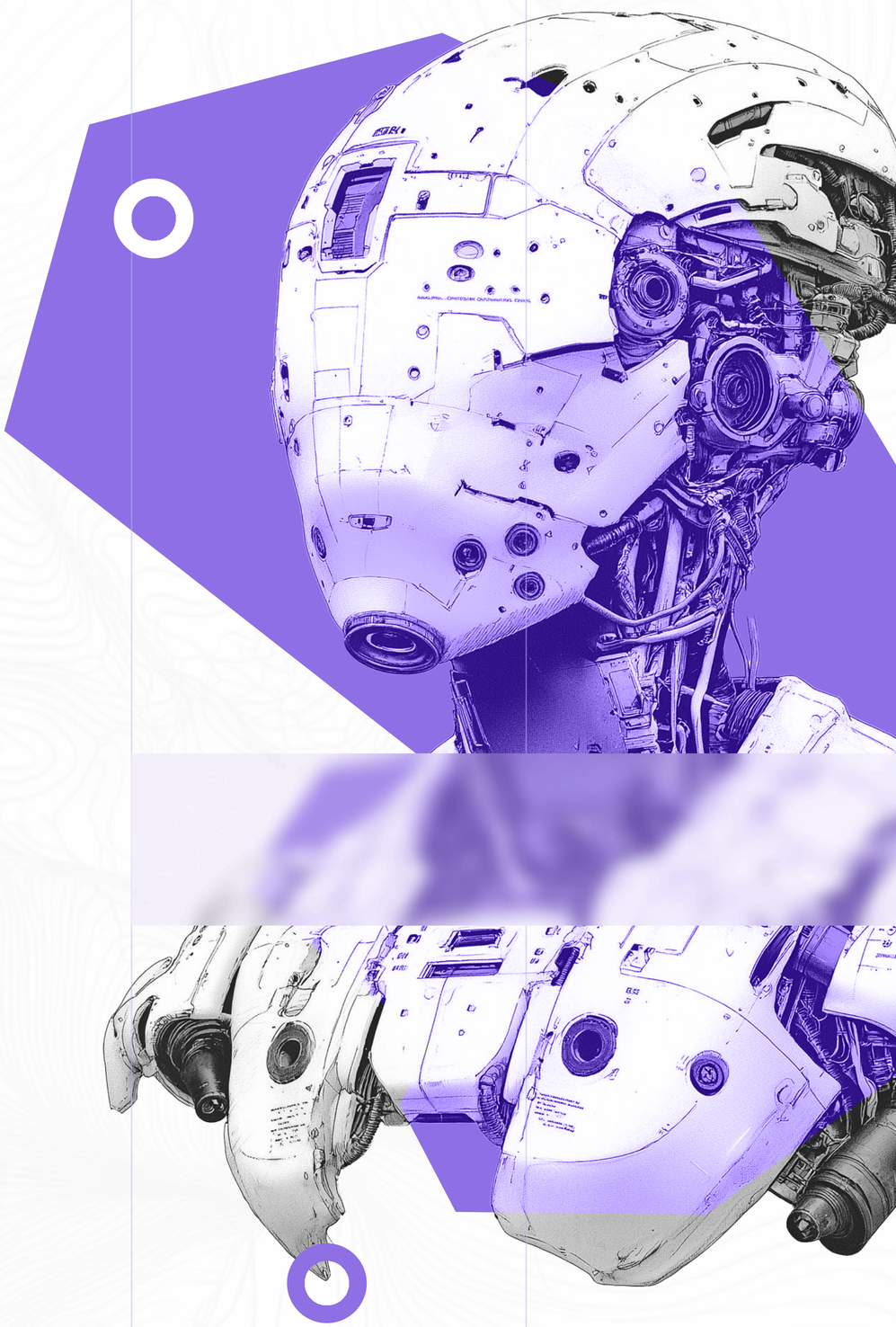
What started as simple code completion has evolved into AI-driven development - where developers act more as solution architects than traditional coders. Early adopters are seeing dramatic productivity gains, with experienced developers who embrace AI achieving the best results.

Contrary to fears about job displacement, **AI is elevating the role of skilled developers** while making coding more accessible to others. Organizations face both technical and cultural challenges, but those successfully integrating AI programming tools gain significant competitive advantages through faster delivery and reduced technical debt.

While AI tools may increase some development costs, the value delivered per dollar rises substantially. Meanwhile, AI is making programming available beyond professional developers, enabling personalized software creation that expands technology literacy across society.

This paper provides a practical framework for understanding these changes and offers guidance on effectively harnessing AI in your development process. To be transparent, 100% of the content was prepared by a human, refined and edited by Perplexity AI and Claude AI, as well as peer review. All robotic imagery was generated in Midjourney AI.

Mikołaj (Miki) Sitek
AI Ambassador



Introduction

Just a few years ago, developers dismissed AI-generated code as primitive and error-prone. Fast forward to today, and tools like GitHub Copilot, Cursor or Cline backed by AI models like Claude or GPT-4 have become indispensable programming assistants.

We've now entered the era of “vibe coding” - a term coined by former OpenAI researcher Andrej Karpathy. In this paradigm, programmers spend less time writing code and more time directing AI systems that generate, test, and refine code based on natural language instructions.

This isn't just hype - it's the clear trajectory we're on.

Note: Throughout this document, we will use the term "AI-driven development" to maintain professional terminology, though you may encounter "vibe coding" in industry discussions of this approach. Pure Vibe Coding entails not even checking the produced code, which is not recommended.

> "...in twelve months, we may be in a world where AI is writing essentially all of the code”
>
> ~Dario Amodei, Anthropic's CEO

About this whitepaper

Our goal is to provide a clear-eyed view of this transformation with practical insights you can apply today. This paper examines what this shift means for developers, businesses, and organizations. We cover:

- How AI is changing day-to-day development work
- The business value and return on investment
- How developer roles and skills are evolving
- Quality assurance and security considerations
- Organizational adaptation challenges
- What comes next and how to prepare

Table of Contents

TLDR - Key Takeways

The evolution of AI in programming has started. AI disrupted the market, changes need to be made. On the right, software development evolution over the decades.

PRODUCTIVITY

The developer that manually codes

Typing out every character to code.

IDEs provide templates and code snippets.

IDE plugins add code completion and smarter suggestions.

AI chat bots generate large code blocks.

2025

AI agents develop full functionalities with human guidance.

AI self-repairs code, multiple agents collaborate.

The solution architect that orchestrates AI

We are here. Heading towards AI management, replacing manual work with business driven problem solving.

The Evolution of Programming in the AI Era

Programming has always evolved with new tools and abstractions. We moved from assembly language to high-level languages, from text editors to IDEs, and from manual memory management to garbage collection. Each step made developers more productive by **handling low-level details automatically**.

AI represents the next major step in this evolution. But unlike previous advances that simply automated small tasks, AI is capable of understanding context, generating complete solutions, and learning from feedback.

The software development landscape has witnessed several shifts over decades:

Wave 1
Code Completion

[2018-2021]

Early AI coding tools like TabNine and the first version of GitHub Copilot offered smart autocomplete features. These tools were impressive but limited to suggesting a few lines at a time, usually based on what you had already typed.

Wave 2
Code Generation

[2021-2023]

As large language models improved, tools began generating entire functions, classes, and even small programs based on comments or natural language descriptions. This marked the transition from AI as a typing assistant to AI as a coding partner.

Wave 3
AI Driven Development

[2023-]

The latest evolution is where developers become more like solution architects directing AI to build entire features or applications. The developer provides the vision, requirements, and quality control while AI handles the implementation.

The AI-Driven Development Approach

Developers found the first generations of AI code as unreliable, incomplete and not particularly helpful.

Today, **a skilled developer can effectively delegate tasks to AI** while focusing on higher-level direction, quality assessment, and strategic decision-making.

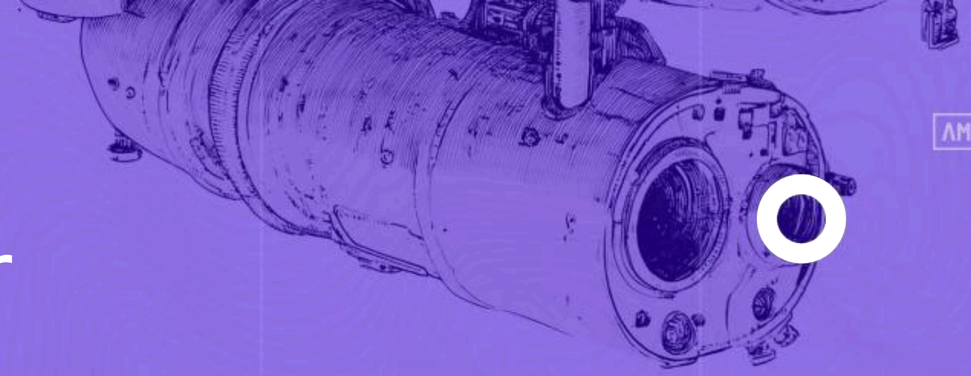
This shift is controversial.
Many experienced developers argue that AI-assisted coding "is no longer programming," meant as criticism. But this perspective misses a crucial point - eliminating routine tasks is a positive thing.

The less creative aspects of development (documentation, testing, refactoring, building CRUD operations, etc.) can now be delegated, freeing developers to focus on innovative problem-solving and architectural design.

AMSTERDAM STANDARD

kyln AI

> "AI assisted coding is no longer programming"
>
> ~One of our FullStack Developers @ Amsterdam Standard



AMSTERDAM STANDARD

kyIN AI

Knowledge will matter

A common misconception is that AI makes deep technical knowledge obsolete. The opposite is true.

Our own research and field experience consistently show that the more experienced the developer, the better results they achieve with AI. As one senior architect observed,

"AI excels at execution but struggles with novel solution design; humans bring the creative problem-solving that AI lacks."

A developer no longer needs to know every detail of a technology or framework, remember syntax or rules, but must understand core concepts and terminology.

Our simple experiment

In a controlled experiment we conducted, a senior frontend developer with no prior Laravel experience was tasked with building a complete backend API from scratch using only AI assistance. Despite having no familiarity with PHP or Laravel-specific patterns, the developer successfully:

- Set up a Laravel application environment
- Configured database connections and migrations
- Implemented authentication and authorization
- Created RESTful endpoints that retrieved data from a relational database
- Deployed the solution to a test environment

A Laravel expert later evaluated the solution and found it functionally sound, though noted several areas where deeper domain knowledge would have improved the implementation.

Knowing the right fundamentals is key

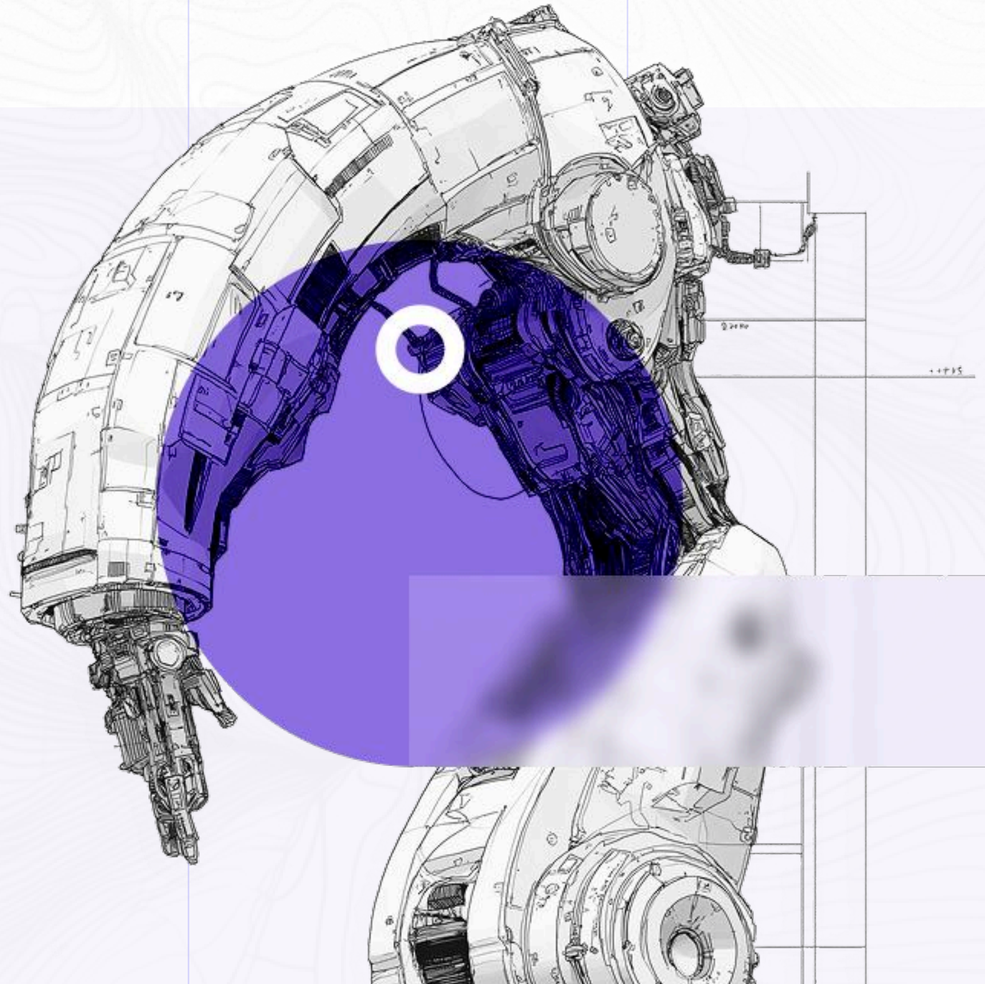
This experiment demonstrates that conceptual understanding combined with AI assistance can bridge significant knowledge gaps.

The frontend developer reported that **"knowing the right terminology to use would have been helpful"** and that "understanding basic backend concepts like migrations and MVC made the difference between success and failure." This suggests that while AI can compensate for gaps in specific technical knowledge, **fundamental architectural understanding remains essential.**

Unlocking new perspectives

Software engineers that focus on understanding the core fundamentals will significantly benefit from AI assistance. Knowledge based on nuances of programming languages will lose in value.

This shift will likely reduce the diversity of frameworks in use. Popular, well-documented frameworks that AI can easily work with will increase adoption, while niche or complex frameworks shall fade. This could be a consolidation point, standardising programming patterns, making it easier for AI to perform reliably.



The New Developer

The AI-powered development workflow differs dramatically from traditional approaches. A future developer will engage in:

Problem Definition
#1

Developers spend more time clearly articulating requirements and desired outcomes. Essentially going over the architecture in their heads - rubber duck method.

Solution Architecture
#2

High-level design decisions remain human-driven. Humans excel at placing the dots, AI fills in the blanks, connecting them.

AI Direction
#3

Developers prompt AI to generate code that implements the design.

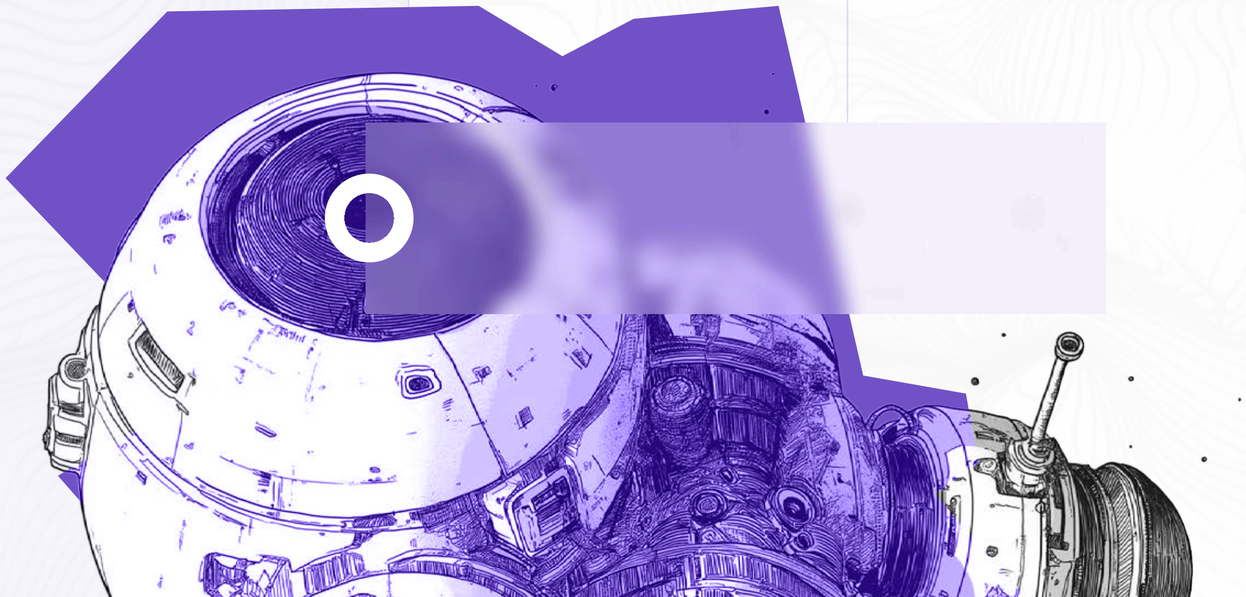
Review and Refinement
#4

Humans mostly review output, request changes, explain issues or bugs, and can even have AI test functionality.

Integration
#5

Developers ensure AI-generated code works within the larger codebase and adhere to safety regulation.

This workflow shifts focus **from writing code to defining problems** and evaluating solutions critically. Test Driven Development can also play a critical role in AI assisted development.
A developer works with AI until all test cases pass before implementing a new feature.



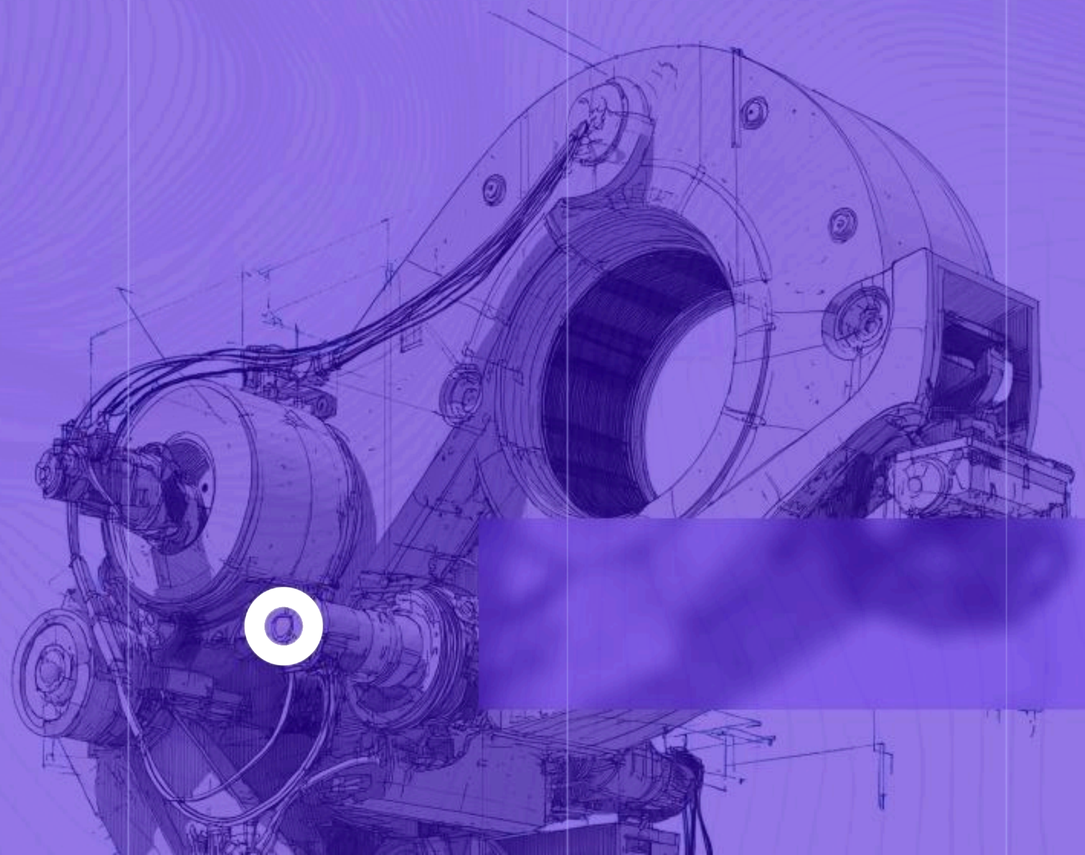
Will code quality even matter?

Many worry about the quality of AI-generated code. But what defines "good quality code"? **Code that's readable, efficient, and maintainable?**

Ironically, in many organizations, resistance to AI-generated code comes from teams already struggling with technical debt and quality issues in their human-written codebases.

We ourselves have witnessed hundreds of companies with aging legacy code and tech debt. This suggests that human-centered development processes have their own significant quality challenges.

AI-driven development, when properly implemented with appropriate human oversight, offers a potential path out of technical debt.



AMSTERDAM STANDARD

kyln AI

Engage in this thought experiment:

If all code is created, reviewed and maintained by AI, does it matter what's under the hood?

At the end of the day, the app runs smooth, fulfils business goals, does so optimally and without security threats.

GPS is an analogous example. The technology has evolved to near perfection over time. We trust the algorithms will guide us to our destination.

The underlying route calculations remain invisible to us, and we've abandoned the practice of verification through paper maps.

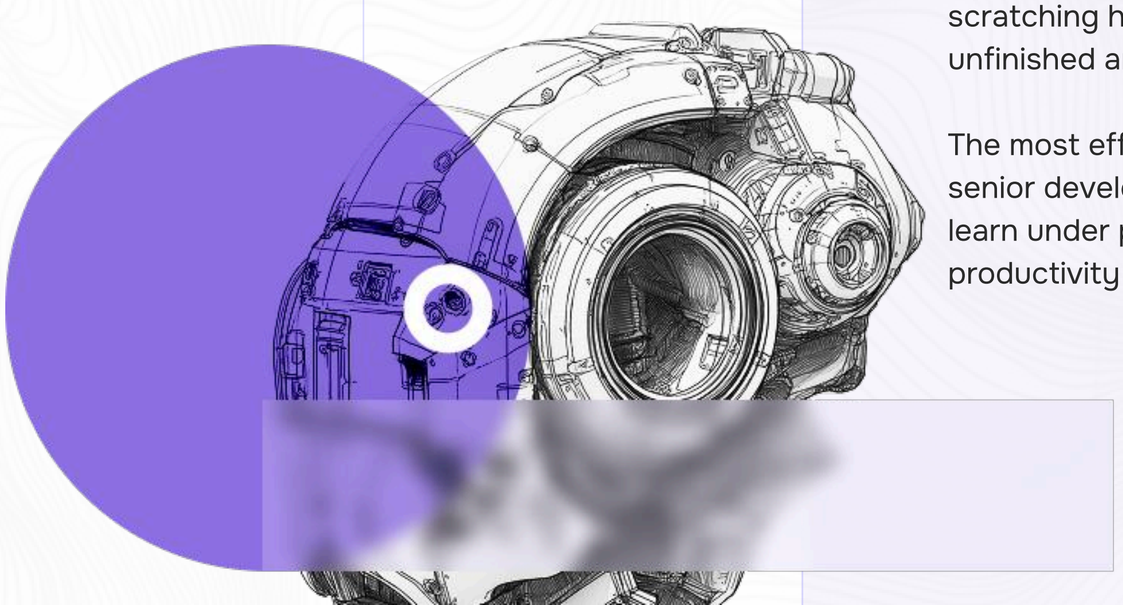
Security Considerations for AI-Generated Code

An important aspect often overlooked in discussions about AI-driven development is security. AI systems trained on public code repositories may inadvertently reproduce vulnerable patterns or outdated security practices.

Organizations must implement:

- Automated security scanning for all code (eg. Sentry, Snyk)
- Clear security requirements in AI prompts and specifications
- Regular security training for developers to recognize vulnerable patterns
- Comprehensive security testing before deployment

When properly managed, AI can actually enhance security by consistently implementing current best practices and identifying potential vulnerabilities that human developers might miss.



The Junior Developer Risk

Organizations should be cautious about strategies that rely heavily on junior developers paired with AI tools as a cost-saving measure.

While AI can bridge some knowledge gaps, **it creates a dangerous scenario where neither the junior developer nor the AI may recognize significant architectural flaws**, security vulnerabilities, or performance issues.

We have already observed (internally, in our own projects) a few situations where rushed and uncheck work of a less experience programmer, left the senior code reviewer scratching his head in dumbfound confusion. Witnessing unfinished and unused code being pushed to review.

The most effective approach pairs experienced architects and senior developers with AI tools, allowing junior developers to learn under proper guidance while still benefiting from AI productivity gains.

Reducing Technical Debt

The invisible tax your company pays daily. Have you calculated its burden on your business yet? One of the most significant long-term business benefits of AI-assisted development is the potential reduction in technical debt. AI systems can:

- Apply consistent patterns across codebases
- Comprehensively document code and design decisions
- Generate thorough test coverage
- Refactor legacy code to modern standards
- Discovering and limiting duplicate code

AI can help organizations modernize technical infrastructure without the large costs traditionally associated with major refactoring efforts. **Read more in our AI refactor case study!**

Today's AI systems face notable constraints: finite context windows and stronger performance with established technologies & frameworks.

However, these barriers are temporary—as infrastructure evolves, today's limitations will become tomorrow's obsolete memories.

The Knowledge Retention Challenge

When developers write code manually, they benefit from the **generation effect** - It's a psychological phenomenon where actively making or creating information (like writing it down) leads to better memory retention compared to passively consuming it (like just reading).

One effective approach is using AI to generate changelog summaries after each completed task.

These summaries document what files changed, how they impact the system, and provide a reference for developers. This practice helps trace when issues were introduced and maintains developer awareness, increasing memory retention. Also for the AI.

An emerging pattern is for AI to track which tasks have been finished. This also helps with retaining AI context.

Cultural Resistance and Adaptation

The developer

The transition to AI-assisted development faces significant cultural challenges. Many experienced developers **resist these changes** for various reasons:

- Concerns about skill relevance and job security
- Professional identity tied to manual coding expertise
- Skepticism about AI's ability to match (or exceed) human-quality code
- Preference for familiar workflows and practices

The company

Organizations that navigate this cultural transition effectively gain substantial competitive advantages over those that resist change. Successful organizations address these concerns through:

- Transparent communication about how roles will evolve
- Investment in reskilling and transition support
- Gradual implementation with clear demonstration of benefits
- Recognition and reward for effective AI utilization

Evolving Team Structures

Smaller, More Senior Teams

Smaller, more experienced teams augmented by AI capabilities. It's no longer about cheaper quantity but more experienced quality.

Embedded AI Specialists

New roles focusing on optimizing AI tools for development workflows.

Stronger Business-Technology Integration

With reduced implementation overhead, developers work more closely with business stakeholders

CASE STUDY

Refactoring with AI

Is AI a viable option for **real-world** development tasks **today**? Our evidence suggests a definitive yes.
We conducted an experiment with one of our clients that provides compelling data.

The Background
A 13-year-old legacy application built on a custom PHP framework with various Laravel workarounds added over time.

The Task
Extract a standalone module and refactor it to full Laravel specification while maintaining complete functional parity without regressions.

The Contenders
We established a comparative trial between our AI Hub team using our KYLN(ai) framework and one of the client's experienced developers who regularly worked with this module.

Size & Scope
The end result encompassed 80 files.
4 805 lines of code created.

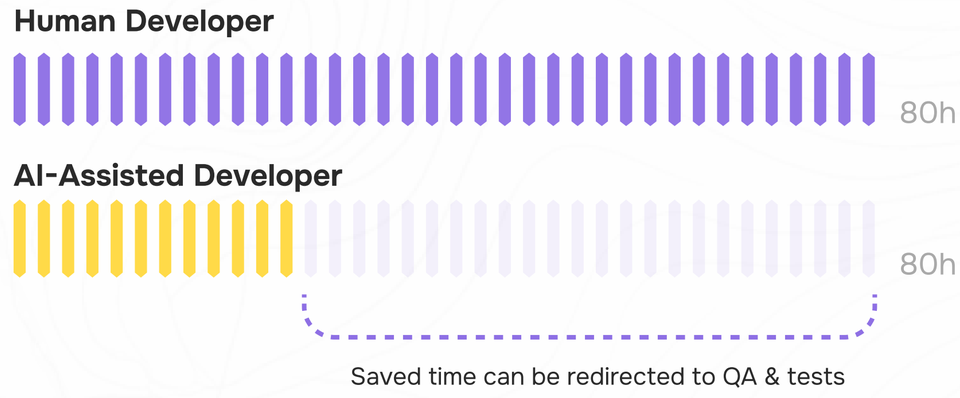
AMSTERDAM STANDARD

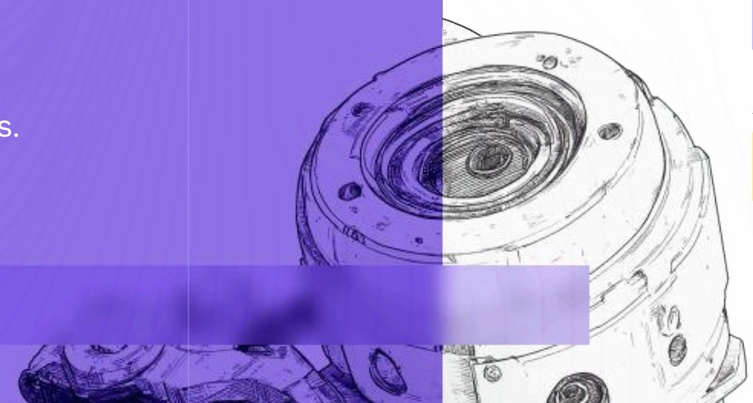
kyln AI

	Human Time **(hours)**	AI-Assisted Time **(hours)**	AI Usage Costs **(Estimated $)**
Rewriting of first adapter (including database models generation)	65	**15.5**	$ 17
Rewriting of next adapter	8*	**5.5**	$ 13
Code review	4	4	-
Implementing code review	**2.75**	4	$ 6
Total	**80***	**29**	**$ 36**

* This task was not completed within the agreed timeframe, therefore it is an estimation.

AI-Assisted development was 63.25% more efficient at refactoring

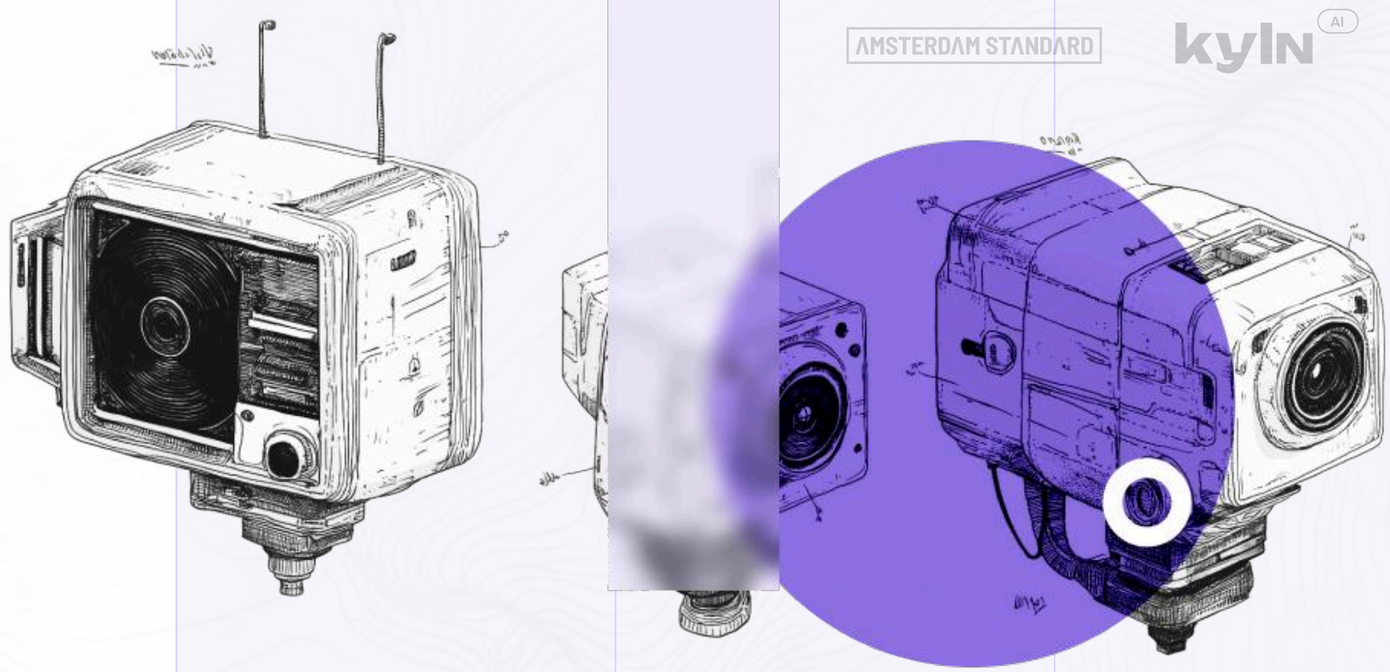




Caveats

Many decision makers will accept the **63% boost at face value** and hop on the AI bandwagon. However, there are several key observations from our experiment that deserve attention:

1. **Code Quality:** Though the AI-written code was working as expected, we found the human's code more clean and better aligned with Laravel standards.
   Our AI code on the other hand passed all tests, was fully documented and lacked technical debt.
2. **Significant Onboarding Time:** The team had to spend considerable time meandering through the complex codebase and preparing instructions for the AI.
3. **Variable results:** Our experiment shows that refactoring the next adapter was significantly faster, but that might not be the case with every module. Each section can bring its own unique obstacles that may challenge AI systems differently.
4. **AI Costs:** It's important to factor in the cost of AI usage. In our case, one developer added an additional $36 for ultimately a single task. If we assume a monthly cost of approximately $200 per developer, these expenses can add up significantly for larger teams.



Saved time doesn't mean more features.

The productivity gains from AI-driven development present organizations with a strategic choice. Rather than simply accelerating feature delivery, forward-thinking companies can redirect these efficiency dividends toward quality enhancement.

The time saved through AI assistance creates opportunities for more comprehensive testing, user experience refinement, and data-driven optimization or expanded A/B testing.

Adopting a quality-over-quantity mindset, allows development teams to perfect existing functionality rather than perpetually chasing feature expansion. This approach yields products with greater stability, usability, and market fit.

Business Value and Transformation

The most immediate business impact of AI-assisted development is dramatically improved productivity. Our field research reports an estimated:

- 30-50% reduction in development time for **standard features**
- 40-60% decrease in time spent on **routine coding tasks**
- 25-35% faster bug resolution and quality assurance cycles

These efficiency gains translate directly to competitive advantage through faster time-to-market. When development teams can deliver working software faster, businesses can respond more quickly to market opportunities, customer feedback, and competitive threats.

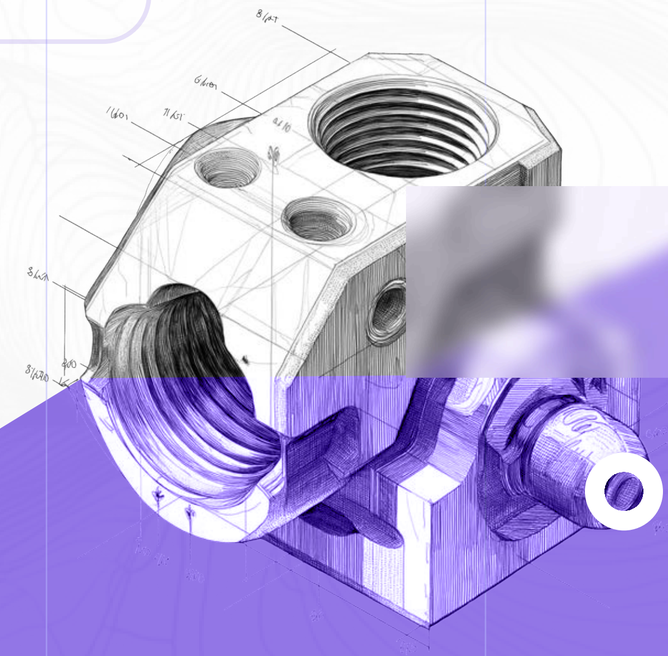
While these productivity gains represent real possibilities, it's crucial to understand that measurements in software development are inherently challenging—just as project estimations have always been difficult.

Not every task benefits equally from AI assistance, and results vary significantly based on codebase complexity, domain specificity, and technical constraints.

Some tasks may see only modest improvements or cases where AI isn't beneficial at all! This inconsistency highlights the continued importance of experienced developers who can strategically determine where AI will truly accelerate delivery.

Not every task benefits equally from AI

In some cases AI is an obstacle



A/B testing becomes economically viable

Traditionally, A/B testing required significant resource investment — creating multiple implementation versions meant doubling development effort while **ultimately discarding half the work.**

It is now economically viable to generate multiple implementation approaches within hours rather than days.

This capability allows for testing beyond simple A/B comparisons to A/B/C/D testing or even more variants, each exploring different user experiences, architectural approaches, or feature implementations.

By evaluating more alternatives before full implementation, teams can **make data-driven decisions that improve user retention and satisfaction.**

The realistic cost of AI

Contrary to expectations, AI-assisted development doesn't necessarily reduce overall development costs. In fact, software development costs may increase in the short to medium term.

- **New Subscription Models**: Development environments and tools increasingly shift to subscription-based pricing when they incorporate AI capabilities
- **API Access Costs**: AI-powered coding features rely on API calls to large language models, each with associated usage fees
- **Skill Premium**: Developers who effectively leverage AI often command higher salaries as they produce more value
- **Infrastructure Requirements**: building your own AI products face additional infrastructure costs for model hosting and inference

While some routine tasks become faster, these savings can be offset by the new costs introduced. The actual economics will improve as adoption grows and competition increases.

The Rise of Personal Software

Another emerging trend is the creation of temporary apps or custom software for personal use by “weekend coders”.

AI coding makes it economically viable to build specialized tools for individual needs rather than relying on generic solutions, which might not even exist.
Imagine you have a specific task that you do repetitively, there is no existing tool that handles your exact use case. Write it yourself with an AI Agent or using the many AI Automation workflows software that exists.

Consider these practical examples:

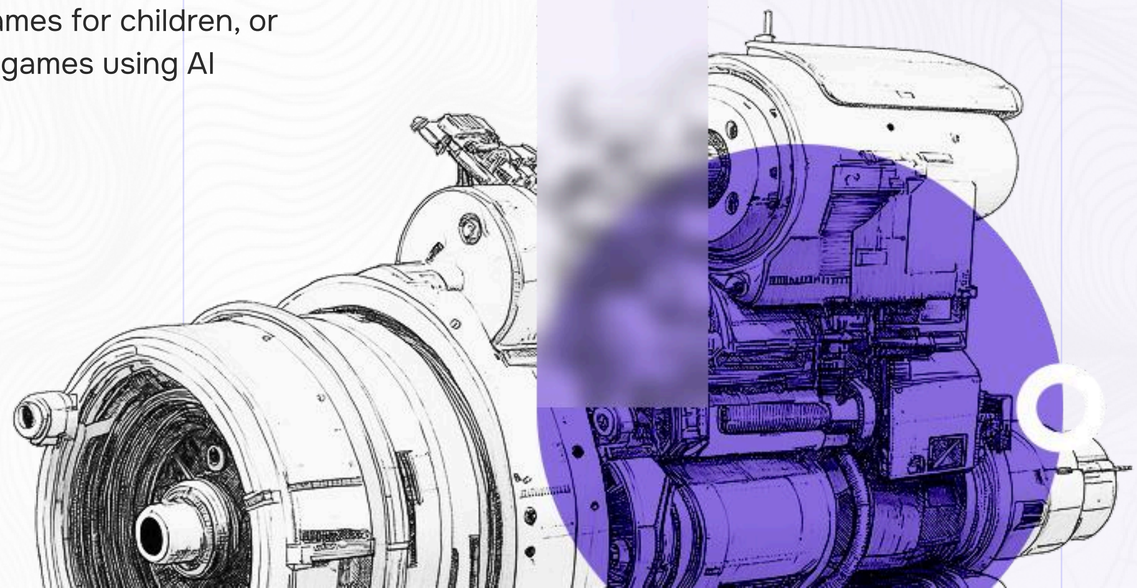
- Creating a custom filter that triggers specific actions when emails from certain people arrive
- Building a specialized calculator for complex, conditional calculations without resorting to complex spreadsheets
- Developing simple educational games for children, or teaching them to build their own games using AI

The concept of "disposable software" becomes practical - applications built for limited use that solve a specific need and can then be discarded.

Users can "spend an hour prompting with an Agentic AI, build and test the product. Once it does what you need, you can remove it."

Non-specialists can now create useful software while technical experts leverage the same AI tools to build more sophisticated solutions. By lowering the technical barrier to entry, AI will dramatically expand the population of people who engage with programming.

Many who would have dismissed coding as 'too complex' or 'not for me' will discover they can create meaningful software, leading to both increased IT literacy across society and a new wave of programmers from previously untapped talent pools.

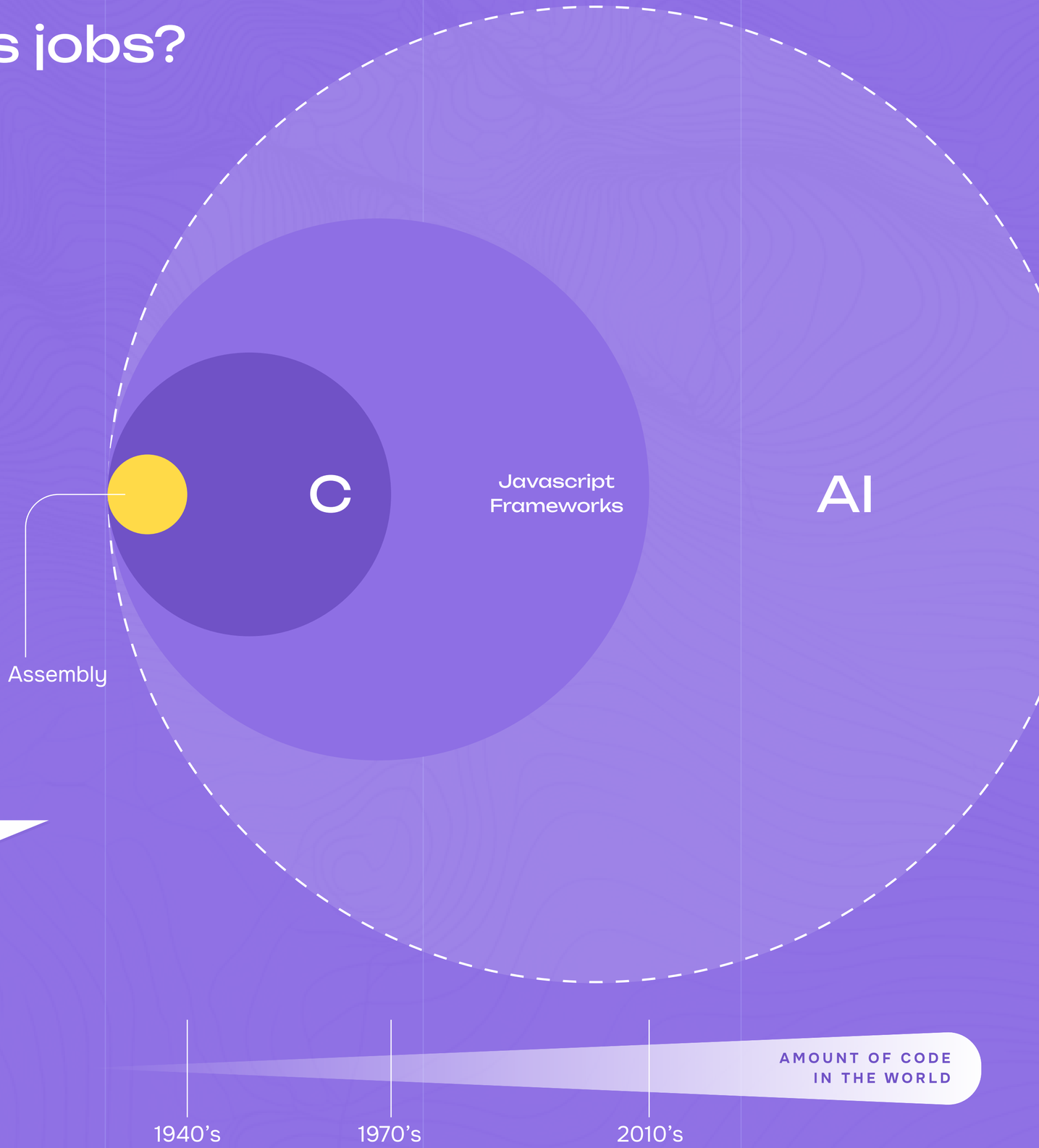


More programmers = less jobs?

This broader participation **won't lead to an oversaturation of programming jobs or wage depression**. Rather, it will expand the total universe of what gets programmed.

More people creating software means more innovations, more industries becoming programmable, and entirely new technological frontiers being explored. The pie grows substantially larger rather than being divided into smaller pieces.

The transition from Assembly to C offers a perfect historical parallel to this. When C emerged as a more accessible alternative to Assembly, it didn't create an oversupply of programmers. Instead, it dramatically expanded what could be programmed and who could program it. This abstraction unlocked new applications, new industries, and entirely new computing paradigms.



In the diagram, replace "C" with "**Photography**", and then "frameworks" with "**The iPhone**". Suddenly, millions gained access to high-quality cameras for everyday use. This availability created entirely new professions—social media influencers, content creators, and citizen journalists.

Yet for important events like weddings, most people still hire professional photographers, despite every guest having a smartphone capable of recording the entire celebration.

The path forward

Companies seeking to harness these benefits need a clear strategy:

1. **Invest in Expertise**: The most successful implementations pair AI with experienced developers who understand core concepts and can effectively direct AI systems.

2. **Transform Gradually**: But swiftly. Begin with targeted applications where AI can deliver immediate value, then expand as teams gain familiarity and confidence.

3. **Reimagine Processes**: Adapt workflows to emphasize problem definition, quality assessment, and integration rather than line-by-line coding.

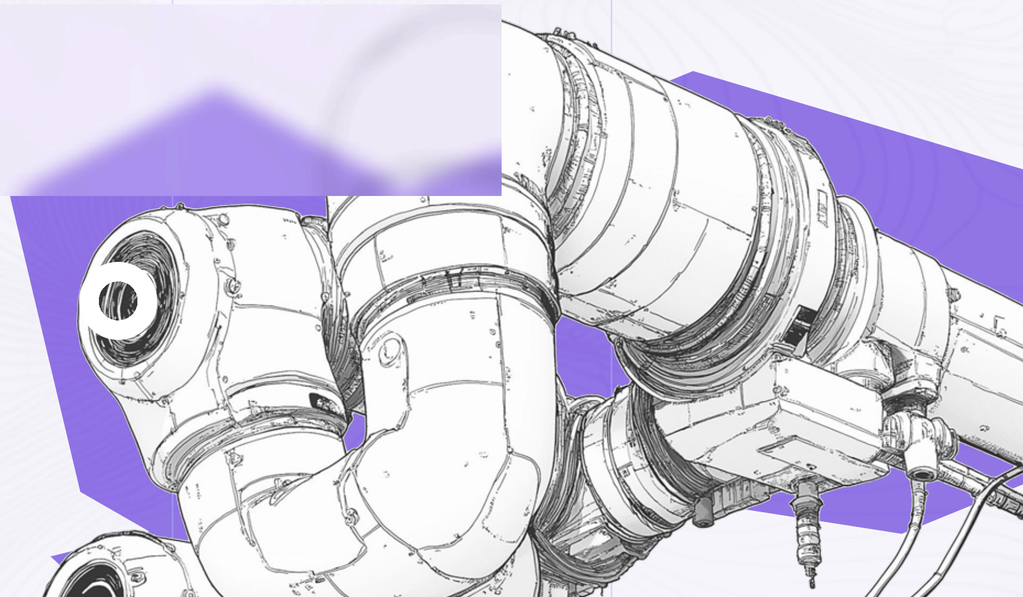
4. **Address Cultural Concerns:** Proactively engage with developer concerns about changing roles and skills, emphasizing how AI augments rather than replaces human creativity.

5. **Enhance Quality Assurance:** Implement rigorous verification systems to ensure AI-generated code meets production standards for security, performance, and reliability.

The gap between organizations effectively leveraging AI in development and those resisting the transition is widening daily.

Early adopters are already gaining significant competitive advantages through faster delivery, more iteration and refining solutions by easily exploring alternatives. It has become economically viable to experiment with multiple MVP's and POC's, finding the right path for the companies future.

The question facing technology leaders is **no longer whether to adopt AI-driven development, but how quickly and effectively they can implement it** while managing the associated cultural and technical challenges. Those who navigate this transition successfully will position themselves at the forefront of the next generation of software innovation and business value creation.



Whitepaper recap

AI is here to stay, and has created ripples in the foundations of software development.
Focus should shift towards:

- **Embracing AI** adoption and adaptation especially across development teams 06
- Developing **AI orchestration** skills to reduce manual work to increase productivity 09
- Implementing proper quality controls and prevent shortcuts that lead to technical debt 12
- Addressing **workforce concerns** through transparent communication and training initiatives 13
- Recognizing that we are in the early phases of AI transformation with **further evolution ahead** 08
- Leveraging AI not only for new development but also for modernizing legacy systems 14
- Utilizing productivity gains to **perfect value** rather than simply increasing output 17
- Implementing regular training programs
- Preparing for resistance from both employees and clients 13
- Prioritizing knowledge and expertise over volume of output 06
- Focusing on core fundamentals rather than language-specific details 07
- Budgeting appropriately for new AI costs through subscriptions and usage fees 17

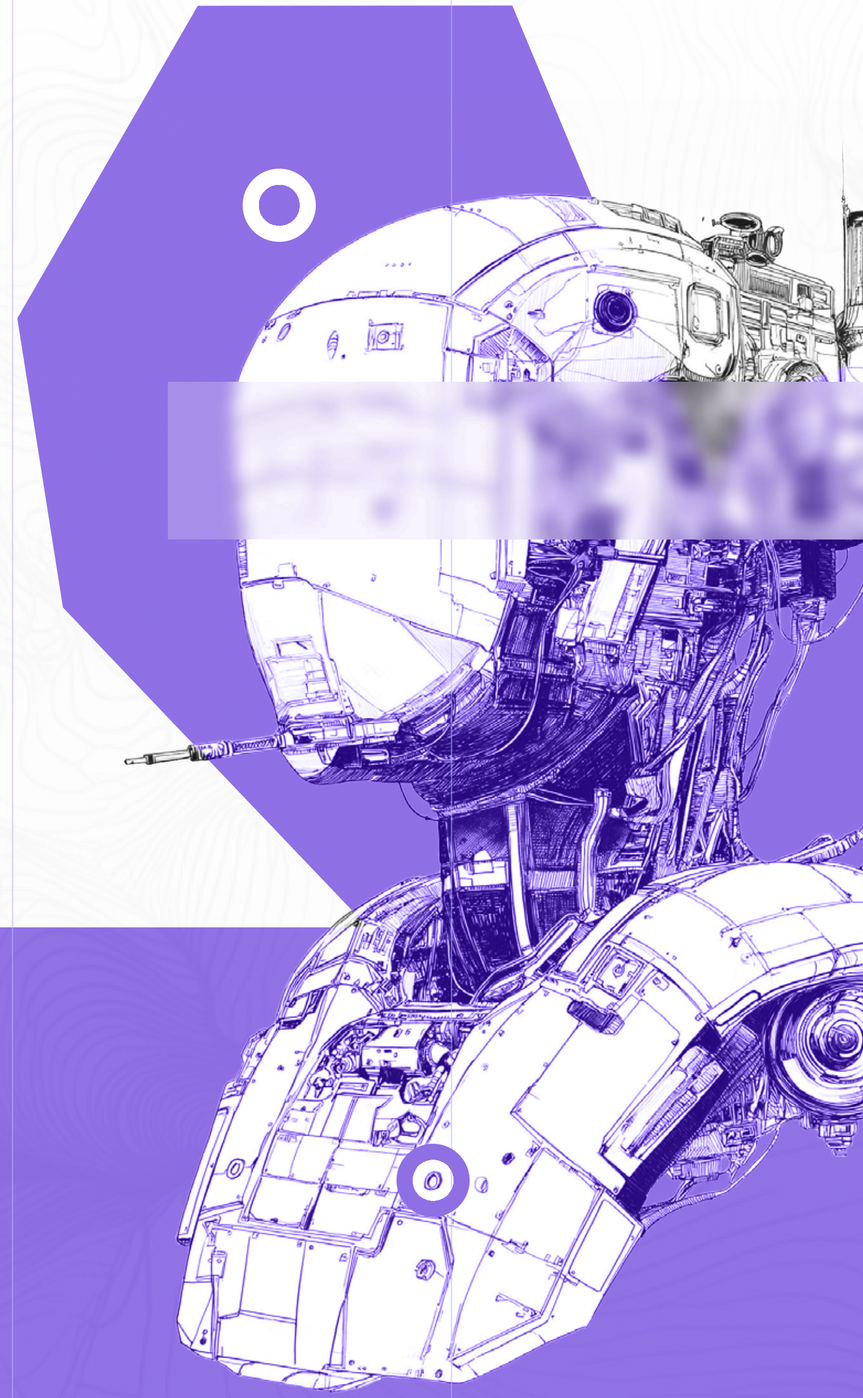
Conclusions

We stand at a historic inflection point in software development. The AI programming revolution isn't merely another tooling upgrade—it represents a fundamental reimagining of how software is created. Just as the transition from Assembly to high-level languages transformed programming decades ago, AI is now transforming the developer experience and business outcomes in equally profound ways.

The Dual Promise

The promise of AI-assisted development is twofold:
First, it offers unprecedented productivity gains in routine coding tasks, the efficiency improvements are too substantial to ignore. These aren't marginal gains but transformative changes that allow businesses to deliver more value.

Second, it enables a higher level of human creativity. By freeing developers from the mechanical aspects of coding, AI allows them to focus on the truly human elements of software creation: understanding user needs, designing elegant architectures, and solving novel problems. Rather than diminishing the developer's role, AI elevates it.



The time to act, is now

Don’t feel overwhelmed by the rapid pace od AI advancement. Invest in your companies future, get in touch, discover our exploratory consultations, knowledge sharing workshops or simply begin your AI build with us.
www.amsterdamstandard.com/ai

Sources

Most of the content in this document is developer accounts and our own observations, research and field experience. Amsterdam Standards AI Hub, is an internal R&D unit designed to investigate, examine and utilize AI. Other sources include:

https://fortune.com/2025/03/26/silicon-valley-ceo-says-vibe-coding-lets-10-engineers-do-the-work-of-100-heres-how-to-use-it/

https://devops.com/ai-in-software-development-productivity-at-the-cost-of-code-quality/

https://artsmart.ai/blog/ai-in-productivity-statistics/

https://www.statista.com/statistics/1440348/ai-benefits-in-development-workflow-globally/

https://simonwillison.net/2025/Mar/19/vibe-coding/

https://hypersense-software.com/blog/2025/01/29/key-statistics-driving-ai-adoption-in-2024/

https://en.wikipedia.org/wiki/Vibe_coding

https://en.wikipedia.org/wiki/Generation_effect

https://www.gitclear.com/ai_assistant_code_quality_2025_research

https://www.youtube.com/watch?v=2b_KlROMfp8&ab_channel=Syntax

https://www.linkedin.com/pulse/ai-assisted-development-andrej-karpathys-vibe-coding-future-moreno-su6cc/

https://arc.dev/talent-blog/impact-of-ai-on-code/

https://arxiv.org/pdf/2502.13199

https://jmsnew.iobmresearch.com/index.php/pjets/article/view/1210/719

100% of this document content was created by a human.
Perplexity AI and Claude AI was used to edit and refine the text.
100% of imagery was generated in Midjourney AI.

Main Contributors

Mikołaj (Miki) Sitek
AI Ambassador

Klaudia Lemiec
AI & ML Tech Lead

Piotr Piechura
AI Hub R&D

Tomasz Rżany
Chief AI Officer