William Blair

AI INSIGHTS

Equity Research
Technology, Media, Communications | Infrastructure Software

January 7, 2026
Industry Report

Jason Ader, CFA +1 617 235 7519
jader@williamblair.com

Arjun Bhatia +1 312 364 5696
abhatia@williamblair.com

Jake Roberge +1 312 364 8056
jroberge@williamblair.com

Ralph Schackart, CFA +1 312 364 8753
rschackart@williamblair.com

Frederick Gooding III +1 312 551 4588
fgooding@williamblair.com

Arjan Kochar +1 312 364 8864
akochar@williamblair.com

Cracking the Code:

How AI Is Transforming Software Development

Contents

Introduction

Apart from general-purpose chatbots, customer support AI agents, and third-party AI applications targeted at vertical markets like legal and content creation, adoption of AI in the enterprise has been slower than many expected. The main gating factors have been scarce AI skillsets, model output accuracy, data readiness, high inferencing costs for large language models (LLMs), and immature governance and security frameworks.

At the same time, we have seen widespread adoption of AI coding tools, which we believe is by far the most prevalent use-case today for enterprise AI. Since the release of GitHub Copilot in 2022, AI coding has evolved from the partial integration of AI within the coding process (e.g., code suggestions, autocomplete) to full compilations/blocks of code being generated from natural language prompts (i.e., vibe coding), and, increasingly, autonomous software agents that can perform discrete developer tasks and work asynchronously with human developers.

While software development is unlikely to be fully automated by AI anytime soon—and the actual productivity gains thus far for the average developer have been hotly debated—the implications of this paradigm shift in software development are vast and still underrecognized, with uncertain impact on incumbent tool vendors, business models, and engineering headcount. What is clear is that AI coding is the tip of the spear, with the massive influx of capital and start-ups into AI coding reflecting industry recognition that coding automation fundamentally accelerates software creation and hence, business velocity.

With this report, we aim to provide investors with an overview of the rapid evolution of AI coding tools, the competitive landscape for these tools, the true benefits to developer productivity from AI coding, and the downstream impacts of AI coding on the traditional software development lifecycle (SDLC) and the software ecosystem in general.

Our overarching conclusions are:

1. ***AI coding is fundamentally disrupting the SDLC*** – AI coding is materially accelerating software development cycles, automating workflows, and reducing time, cost, and resource constraints historically associated with the SDLC. LLM vendors and innovative start-ups are key disruptors in this new era.

2. ***AI coding massively expands the dev tools TAM*** – AI coding has introduced an entirely new monetizable layer in the developer toolchain. We expect the average developer will ultimately pay roughly $2,000 per year for a set of these tools, which would amount to an annual TAM of ~$100 billion just for AI coding.

3. ***Systems of record will be more critical than ever*** – with the rise of AI coding and software agents leading to orders of magnitude more code creation and pipeline complexity, systems of record across the toolchain will be even more necessary to ensure code quality, auditability, compliance, security, and observability.

4. ***AI coding unlikely to reduce developer headcount*** – while developer roles (and the definition of who is a developer) will change, human skills and specialization will still be needed to architect, supervise, troubleshoot, secure, and provide context to an ever-expanding number of software projects (and agents).

5. ***AI coding creates secular tailwinds for infrastructure software*** – the expected flood of new AI applications and agents will result in vastly greater data volumes that will ripple across the infrastructure software ecosystem, driving demand for databases, data security/governance, data observability, data protection, and storage.

6. ***AI coding will grow the software pie*** – instead of "AI eating software," Jevon's Paradox suggests that AI will grow the software pie as it becomes easier and cheaper to create new apps, enabling code to eat into more of the labor economy. That said, because AI reduces technical barriers to entry, incumbent vendors must reexamine their competitive moats and focus on nontechnical structural advantages like distribution, platform gravity, and ecosystem depth and breadth.

7. ***JFrog is our top pick in the dev tools market*** – we believe the company will benefit as a foundational system of record for software releases, with JFrog Artifactory serving as an immutable source of truth (and versioning mechanism) for the onslaught of built binaries and containers that will emerge from the AI coding revolution. In addition, the company's expansion into security and governance only increases its relevance as more code created by AI means more potential vulnerabilities to manage and more compliance elements to maintain.

Key Findings

Coding Is No. 1 Use-Case for Enterprise AI Today

In our enterprise GenAI report published in February 2024, we called out software development as one of the most promising early use-cases for GenAI. Our rationale at the time was that the specific rules and syntax of programming code make it especially well-suited to how LLMs learn, reason, and generate output, while the abundance of publicly available code on the web enhances model training. Today, developers are leveraging AI coding assistants like Cursor, Claude Code, GitHub Copilot, Lovable, Replit, Cognition, Tabnine, and Codex across a wide range of tasks, including code completion/suggestions, code generation, unit testing, debugging/refactoring, and code translation/migration. Adoption has been swift, with 84% of developers either using or planning to use AI coding tools—up from 76% in 2024—according to Stack Overflow's 2025 Developer Survey. Among professional developers, 51% are using AI tools daily, with another 17% using them weekly—though this data is backward-looking and likely understates current usage. Importantly, AI coding tools have introduced an entirely new, monetizable layer in the developer toolchain—with our expectations that the average developer will ultimately pay roughly $2,000 per year for a set of these tools, which would amount to an *annual TAM of about $100 billion just for AI coding.*

AI Coding Benefits Are Real ... and Just Getting Started

The extent of the productivity benefits that developers are gaining from AI coding tools is a hotly debated topic, with many contradictory data points and surveys. On the whole, engineers that have embraced AI are seeing meaningful efficiency gains, though gains are uneven depending on complexity of the code or project and whether the tasks are "greenfield" or "brownfield". According to an October 2025 JetBrains survey of almost 25,000 developers across 19 countries, nearly 9 out of 10 developers save at least one hour every week using AI tools, and 1 in 5 saves eight hours a week or more. Our research indicates that senior engineers are seeing the most productivity gains (in some cases 10-100x), because the more knowledge a developer has on the nuances of programming languages, product architecture, and project context, the more they can maximize the value of the new tools. On the flip side, multiple developer studies question the actual productivity benefits to programmers from AI: 46% of respondents stated they do not trust the accuracy of AI-generated code (Stack Overflow); 95% of respondents said they spend extra time fixing AI-generated code (Fastly); productivity benefits shrink dramatically when applying AI to mature, existing codebases (Stanford Software Engineering Productivity Study); and experienced

open-source developers completed tasks more slowly with AI tools than without (METR). While the quality of code generated by AI tools is still an open question—no-code methods such as vibe coding have borne the brunt of criticism for their shortcomings in generating production-quality code—it is also fair to say that we are still in the early innings of product innovation and the tools will only get better from here.

Context Is King in AI Coding

A key challenge with AI coding agents is that they often lack the proper context to execute tasks reliably and securely. More specifically, organizations today lack a centralized, policy-based oversight mechanism that governs what agents should see/know to accomplish assigned tasks as well as an experienced human developer. This has led start-ups like Cursor, Tabnine, Sourcegraph (Cody), and Augment to introduce the concept of a "context engine," which is a centralized system that pulls relevant code, docs, symbols, and project structure into the AI's working context so coding decisions/agentic tasks are accurate and auditable. Until agents can match human developers' understanding of domain "vocabulary", code structure, dependencies, and company-specific best practices and security & compliance requirements, these tools will be limited in their ability to truly automate software development workflows.

LLM Vendors in Pole Position for AI Coding, but Plenty of Room for Pure Plays

The dizzying array of AI coding tools raises several questions: how many of these tools can the market support, how profitable (and sticky) are these tools, and which vendors are best positioned? At this early juncture, we believe that LLM vendors like Anthropic, OpenAI, and Google have a structural advantage. This is because they not only offer their own established coding assistants (Claude Code, Codex, Gemini) but, more importantly, also control the foundation models that are the back-end processing engines for virtually all of the pure-play tools, which gives the LLM vendors unique pricing power and user telemetry to continually improve their models. That said, we expect product quality, user experience, workflow integration, and ease of use to remain relevant, which should benefit pure-play, first-mover AI coding vendors such as Cursor, Cognition, Replit, and Lovable. These vendors are also not blind to the risk of overdependence on the third-party frontier models (especially from a gross margin perspective), which helps explain why Cursor recently introduced its own coding-specific LLM (which reportedly is a fine-tuned version of the Chinese open-source model Qwen). Going forward, a key question governing the success (and margin profile) of pure-play coding start-ups is whether open source LLMs are able to cross the chasm on post-training and reinforcement learning, which is currently holding back their usability and performance compared to the closed-source frontier models referenced above. Ultimately, the barriers to long-term success for pure-play AI coding vendors are intimidating in view of low switching costs among notoriously fickle developers and the market/pricing power of LLM vendors and incumbents like Microsoft/GitHub. That said, specialization and innovation still matter, and the TAM here looks massive (see page 23)—which should support multiple pure-play winners in this space.

Adoption of AI in Software Development Varies Across Organizations

Not surprisingly, cutting-edge start-ups and sophisticated technology and financial services enterprises are leading the charge in the embrace of AI coding tools—developers in these organizations are not just using AI for coding but also for tasks like prototyping, testing, and quality assurance (QA). At the top of the adoption pyramid, Anthropic CEO Dario Amodei recently stated that Claude AI is now writing 90% of code for most teams at the company, while Microsoft CTO Kevin Scott recently predicted that 95% of all code will be generated by AI in 2030. Lower down the pyramid, traditionally risk-averse customers (like those in regulated industries with extensive amounts of brownfield, legacy code) have been slower to adopt the new AI tools beyond the experimental stage, preferring to wait until critical technical and cultural hurdles are overcome before infusing AI into their coding workflows and pipelines. According to JetBrains 2025 State of Developer Ecosystem Survey, 9% of respondents report no AI adoption in development workflows, 21% are in the exploratory stage (researching but have yet to implement), and 27% are in the pilot stage

(experimenting with AI in limited workflows). Key adoption hurdles include: 1) operational challenges (messy outputs from AI-generated code, coding assistants struggling to execute complex tasks); 2) technical challenges (model hallucinations, training data biases); 3) security/compliance challenges (larger attack surface areas with more machine-generated code, sensitive data exposure); 4) legal/regulatory challenges (potential copyright violations around licensed code); and 5) process change challenges (AI requires developers to learn new tools and workflows).

AI Is Massively Disrupting the SDLC—Not in the Future but Today

Driven by broad adoption of coding assistants and agents, AI is in the process of gradually and systematically dismantling many of the historical constraints associated with the software development lifecycle (SDLC), namely time, cost, and resource intensity. The main result has been a meaningful acceleration in development cycles, with projects that normally would have taken months to plan, staff, and ship now being completed in a matter of days or weeks at the most sophisticated organizations. While coding is getting all the attention, the AI-driven SDLC transformation extends beyond raw code generation into areas like writing unit tests, automating QA, generating documentation, fixing bugs, code reviews, and even orchestrating deployment pipelines. Consequently, the historical linear sequencing of the DevOps toolchain and agile methodologies—i.e., plan→code→build→test→secure→release→monitor—is melting away as AI blurs boundaries between traditional development phases and reduces the need for discrete tools and teams. This "melting" of the traditional SDLC has major implications for both the developer tool ecosystem (with incumbent vendor positioning increasingly dependent on their ability to innovate and adapt business models to the new reality) and the broader software application ecosystem (where the minimal cost to develop new software will cause app vendors to focus on the durability of their competitive moats). As in prior tech platform shifts, we expect significant vendor consolidation to be fueled by AI disruption, with bigger companies likely to use their balance sheets to add the latest-and-greatest tech while smaller companies will look to become part of broader distribution networks.

AI Makes SDLC Systems of Record More Important Than Ever

While AI is upending the traditional SDLC, it is also creating new post-code authoring challenges around security, compliance, and code quality—the classic trade-off between speed and trust in software development has never been more glaring. More code created by AI means more potential vulnerabilities to manage, more compliance elements to maintain, more integration points to validate, and ultimately more operational risk. As a result, we believe the criticality of foundational systems of record (SORs) in the SDLC will only increase in the AI era—these include planning (issue and project tracking), source code management (Git-based version control systems), software releases (artifact and dependency management), security and identity (vulnerability data and access control), and runtime production (monitoring and observability of applications in production). At the same time, systems of interaction are being completely disrupted as AI increasingly allows users to describe intent in natural language and AI agents to perform multistep tasks autonomously across multiple SoRs. Industry estimates suggest that 80% of software development work occurs following code authoring, which means that management, orchestration, security, verification, and monitoring will be more vital than ever before.

Developer Headcount Likely to Rise Despite Role Evolution and Convergence

While industry pundits debate the impact of AI coding on future developer headcount—and whether learning to code will even be necessary in this new era—our educated guess at this juncture is that developer jobs will grow, albeit at a slower rate than software output. Our research suggests that AI's multiplier effect on developer productivity could drive companies to hire more developers rather the fewer developers. The thinking here is that if software "rules the world" (i.e., there is a virtually infinite amount of software work that needs to be done) and the ROI of an AI-fluent developer is orders of magnitude higher than a legacy developer, why would an organization not want to bring on more developers? More specifically, we believe that human skills and specialization will still be needed to architect, supervise, troubleshoot, secure, and provide context to an ever-expanding

number of software projects and agents. A good analogy to AI coding assistants is spreadsheets, which increased rather than reduced finance jobs by making financial analysis more powerful and accessible. Looking ahead, AI is amplifying differences in roles, responsibilities, and expertise levels among professional developers—we are increasingly seeing signs of role convergence, where developers are becoming not just editors of AI-generated code but also project managers and QA reviewers. For example, designers who once handed off static mockups to prototype engineers are now able to generate semi-functional prototypes themselves using vibe coding tools. Moreover, the definition of a developer is likely to evolve as AI democratizes software development through technologies like vibe coding. Ultimately, AI is collapsing functional boundaries in the traditional developer toolchain, concentrating value with those engineers that have embraced AI technology and who can think holistically about product design and implementation and not just coding.

Seat-Based Pricing Models Are Evolving

As AI enables smaller teams to produce far more software, the traditional relationship between headcount and software output is increasingly being questioned. While still early, this dynamic poses long-term challenges to seat-based pricing models, which have historically provided predictable, sticky revenue streams for DevOps and collaboration vendors. If the number of developers grows more slowly—because each developer can produce exponentially more output—total seat counts may stagnate or shrink, even as overall software production accelerates. In response, vendors like Cursor and GitLab are augmenting seat-based models with usage-based pricing that factors in some measure of consumption units like tokens (computing/storage resources), API calls, and/or agents—in effect, passing through LLM costs to users. This should mitigate potential risks associated with seat-based pricing, even as it is not yet a fait accompli that AI will negatively impact developer headcount.

AI's Impact on Incumbent DevSecOps Vendors Will Vary

Our research suggests that incumbent DevSecOps vendor positioning will depend heavily on their ability to execute and adapt business models to the new reality. With the wave of innovation unleashed by AI coding, we believe that the pendulum in software development is swinging back from consolidated platforms toward best-of-breed tools as developers will want to choose the latest and greatest technologies (and may view platforms as constraining their freedom of choice). This raises strategic questions for platform proponents like GitLab that face increasing competition from new AI-native tools and are vulnerable to the potentially eroding value of seat-based pricing models. The same holds true for GitHub, though these risks are somewhat mitigated by deep integrations with the Microsoft ecosystem and broad enterprise reach and distribution. Elsewhere, while Atlassian benefits from significant developer inertia and organizational lock-in—and as noted above, its SOR for project planning/issue tracking should continue to be a grounding element within the SDLC—AI threatens to make traditional project management less relevant over time, diminishing the need for manual planning, coordination, and orchestration. Meanwhile, we believe downstream vendors like JFrog (focused on release management and security) and Datadog (focused on run-time monitoring and observability) are more insulated from disruption as they are tied to growth in overall software volumes and related infrastructure needs. As in past platform shifts, speed, adaptability, and the ability to compound network and data advantages will determine which vendors remain relevant and which risk being left behind.

AI Coding Creates Secular Tailwinds for Infrastructure Software

With AI accelerating software release cycles, we expect to see an explosion in the volume of code and applications created over the next decade. Higher application starts and usage will lead to more data volumes, which will ultimately ripple across the broader infrastructure software ecosystem, driving demand for more databases (every app needs a database), more data security/governance technology (more data to manage and secure), more data observability tools (more apps to monitor in production), more data protection tools (more data to back up), and more storage (need to store all the new data generated from higher application usage). This bodes well for leading-edge vendors in these subsegments, though the impact timeline is difficult to predict.

AI Forcing All Software Companies to Reexamine Competitive Moats
With AI rapidly accelerating development cycles, every software company must reassess the durability of its competitive moat. Software companies with limited technical defensibility and point solutions are most vulnerable to AI disruption, as these offerings can be easily replicated at very low cost using AI. We believe that technical moats in software have already been eroding over the past 10 to 15 years as software has become easier to build with innovations like cloud infrastructure, microservices, APIs, containerization, low-code tools, etc. In response, the most successful software companies have expanded their moats beyond the technical to include platform breadth, distribution, domain expertise and verticalization, and network effects/third-party integrations. While we expect a vibrant start-up community to make waves, incumbents with large installed bases, proprietary datasets, and embedded workflows are uniquely positioned to deploy AI in ways that are difficult for new entrants to replicate. Looking ahead, we believe data scale, tight human-to-AI workflow integration, and security/governance will also be defensible moats for software companies in the AI era.

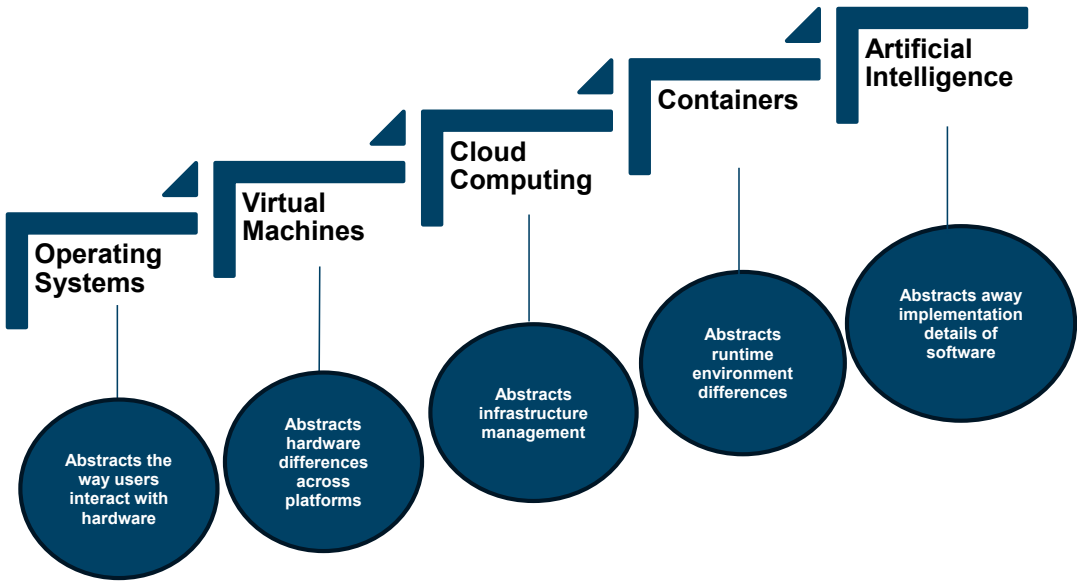
How Do AI Coding Assistants Work and Who Are the Main Players?

AI coding assistants are queryable software development tools that leverage machine learning and natural language (NL) prompting to execute various tasks, including code completion, code explanation, code generation, and code suggestions. AI coding assistants help accelerate, automate, and orchestrate rote software development tasks so developers can spend more of their time on higher-value tasks. These tools also enhance the relevance of developers across more coding languages. For instance, developers that have expertise in a particular programming language such as JavaScript, Python, or C++ can now leverage LLMs to "translate" from one programming language to another. This is just the beginning though—AI coding assistants offer tremendous potential to enhance and speed up the overall SDLC through the use of agents (see page 21).

At a high level, AI coding can be viewed as the next logical step in software's evolution, which we would characterize as a continual process of abstraction and simplification for users. Each generation of tools, languages, and paradigms hides more of the underlying complexity in software, allowing developers to focus on intent rather than implementation. Each successive advance—from operating systems to virtualization to cloud computing to containers to AI—abstracts away operational burdens from users, accelerating the time from ideation to production at lower marginal cost. Below is our view of the major phases in this historical progression of software abstraction:

- **Operating Systems**: Abstracts how users interact with hardware (servers, storage, memory, and networking)—users don't need to talk to the hardware itself or understand how it works
- **Virtual Machines (VMs)**: Abstracts hardware differences across platforms; allowing developers to write once and run anywhere.
- **Cloud Computing**: Abstracts infrastructure management—infrastructure becomes invisible to end-users; all they need is a computer and an internet connection
- **Containers:** Abstracts runtime environment differences, allowing an application plus all its dependencies to be packaged as a single portable, immutable, and reproducible unit—regardless of the underlying host OS or machine
- **AI:** Abstracts implementation details of software, letting developers define what they want instead of how to build it

Exhibit 1
Cracking the Code: How AI Is Transforming Software Development
AI as the Next Level of Software Abstraction



Source: William Blair Equity Research

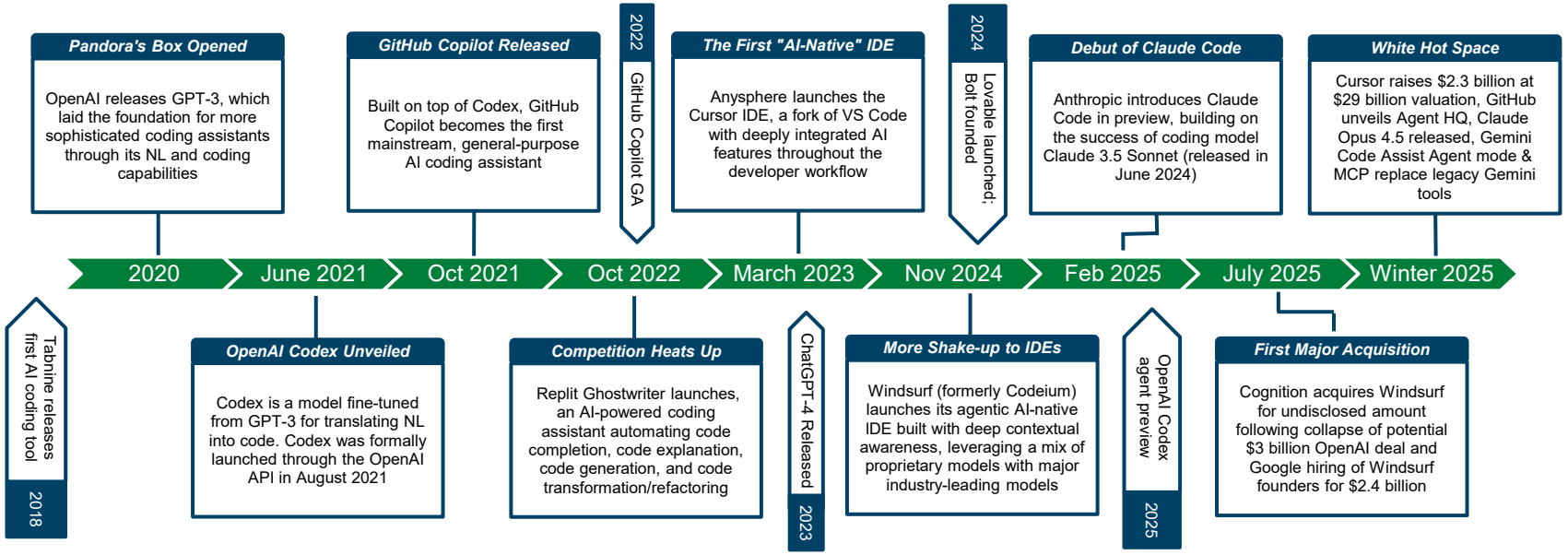
Coding LLMs

The backbone of AI coding assistants are LLMs (either a standard flagship model like GPT or a fine-tuned coding model specifically trained on various programming languages). Coding has turned into a prime use-case for LLMs due to the rigid syntax and semantics of programming languages (they are rule based and adhere to specific logic—a great fit for LLMs), which enables the model to analyze code, make predictions, and generate new code. Put simply, code is just another language that LLMs can learn, reason, and generate output from, with the abundance of publicly available code on the web facilitating training and intelligence.

LLMs supporting AI coding assistants are trained on billions of lines of code, gathered from the vast amount of open-source code repositories or proprietary code from vendors. For example, OpenAI's original Codex model was trained on 159 gigabytes of Python code from 54 million GitHub repositories. Popular coding LLMs include Claude Sonnet 4.5 and Claude Opus 4.5 (Anthropic), GPT-5-Codex (OpenAI), Gemini 3 (Google), Code Llama (Meta), Mellum (JetBrains), Mistral 7B (Mistral AI), DeepSeek Coder (DeepSeek), and Qwen3 (Alibaba).

Model training can be broken down into two different stages: 1) model understanding, where the model sifts through the vast amount of code training data to acquire a deeper understanding of the complexity, patterns, and syntax of the different languages in the training data; and 2) development of internal representations (also known as the pattern recognition systems), which serve as the foundation for how the model analyzes code and utilizes reasoning to answer a user's query. These internal representations contain information around the code's semantics, syntax, and relationships between different code elements, which helps the model with code comprehension and empowers the model to manipulate code concepts to answer a user query when provided with the right context.

Exhibit 2
Cracking the Code: How AI Is Transforming Software Development
Timeline for Introduction of AI Coding Assistants



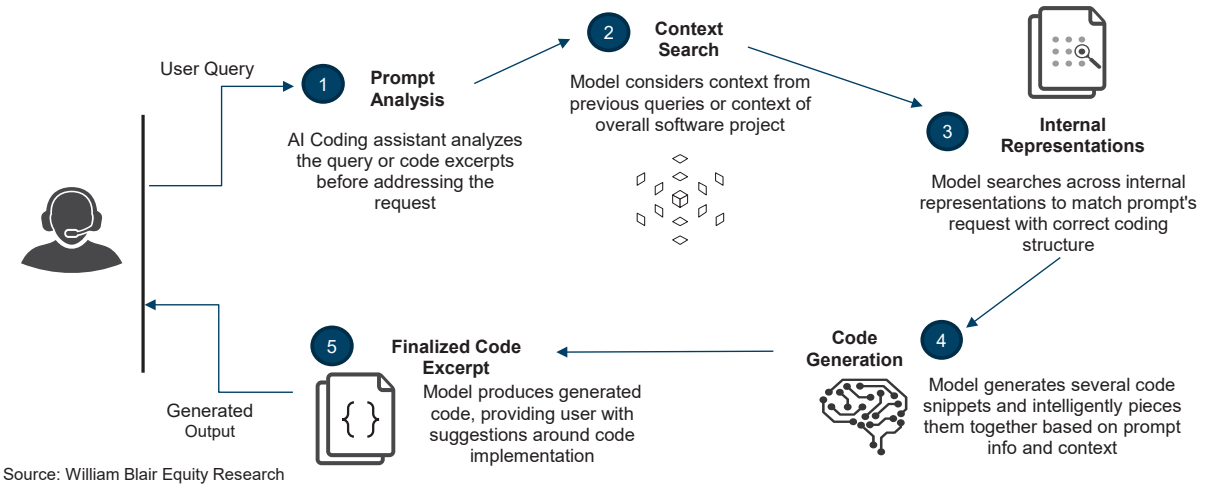
Source: William Blair Equity Research

How Coding Assistants Work

When an AI coding assistant receives a user query, the model will analyze the prompt (which may consist of a question in NL or contain multiple code excerpts) before addressing the request. More advanced AI coding assistants (referred to as AI coding agents) will consider additional context from previous queries or context surrounding the overall software project. The model will then search across its internal representations using the information gathered from its contextual analysis to match the prompt's request with the correct coding structure.

Once finished with its internal search, the model then generates various code snippets and intelligently pieces them together based on the information and context received from the user query. This forms the final output of generated code, where the model often provides suggestions around specific implementation or asks the user whether the model should continue to generate code/refine the output.

Exhibit 3
Cracking the Code: How AI Is Transforming Software Development
How AI Coding Assistants Work



Source: William Blair Equity Research

Importance of IDEs

Akin to Microsoft Word for developers, an integrated development environment (IDE) is a software application that provides a complete workspace for writing, editing, testing, and debugging code. Historically, free or low-cost IDEs like Visual Studio Code (VS Code) and IntelliJ (JetBrains) dominated the IDE landscape, with the vendor community not focused on monetizing this layer due to technical commoditization, developer expectations that the IDE/code editor is a basic utility that should be free, and rich monetization opportunities elsewhere in the DevOps toolchain.

With the advent of AI, this has all changed, with dozens of coding tools generating billions of dollars in subscription revenue that is incremental to the historical developer tools TAM (Cursor and Claude Code each have already reached $1 billion in ARR, see page 23). Indeed, we believe that the IDE is more strategic than it has ever been, as it is the developer's primary workspace, where programming happens, and where AI has the most near-term impact. IDEs are also the logical hub for intelligent software agents, which can suggest tests, flag security risks, and automate deployments across the SDLC.

AI-native IDEs

AI-native IDEs are built from the ground up around AI as the core of the workflow, not as an add-on or plug-in to an existing IDE. Typically, built upon existing IDEs (e.g., Cursor is a VS Code fork), AI-native IDEs retain all the basic functionality of code editors (including code debugging, editing, testing, etc.) but also allow developers to offload complex coding tasks to the IDE, which can be executed autonomously or semi-autonomously. This is possible because these tools have deep context awareness—entire repos, dependencies, and build environments are visible to the AI. As such, they go well beyond code completion or code suggestions and are aimed at orchestrating and redefining the entire SDLC through a single AI interface. Examples of AI-native IDEs include Cursor and Replit.

AI coding plug-ins

As the name suggests, AI coding plug-ins are software extensions that embed an LLM directly inside an existing IDE or code editor like VS Code, JetBrains IntelliJ, or Visual Studio. These plug-ins do not replace the IDE itself but effectively serve as a developer's assistant, helping them with tasks like autocompleting or generating code snippets, explaining unfamiliar code or APIs, detecting bugs and suggesting fixes, writing documentation or tests, and refactoring existing code for clarity or efficiency. Compared to AI-native IDEs, the main benefit of plug-ins is that they allow developers to adopt AI coding without changing their workflow or the IDEs they are accustomed to, while paving the way toward true agentic development environments.

CLI-based plug-ins

A further distinction within AI coding plug-ins is those that are CLI based (run in the terminal as a command line interface alongside normal dev workflow, e.g., Claude Code, OpenAI Codex CLI, Gemini CLI) and those that are GUI based (run in the IDE as a graphical user interface, e.g., GitHub Copilot). CLIs serve as a channel of interaction between developers and AI coding assistants, with developers able to leverage features such as code generation, debugging, and task automation natively in terminals without having to download an extension to other code editors/IDEs or needing to purchase one of the new IDEs entirely.

Mapping the AI Coding Competitive Landscape

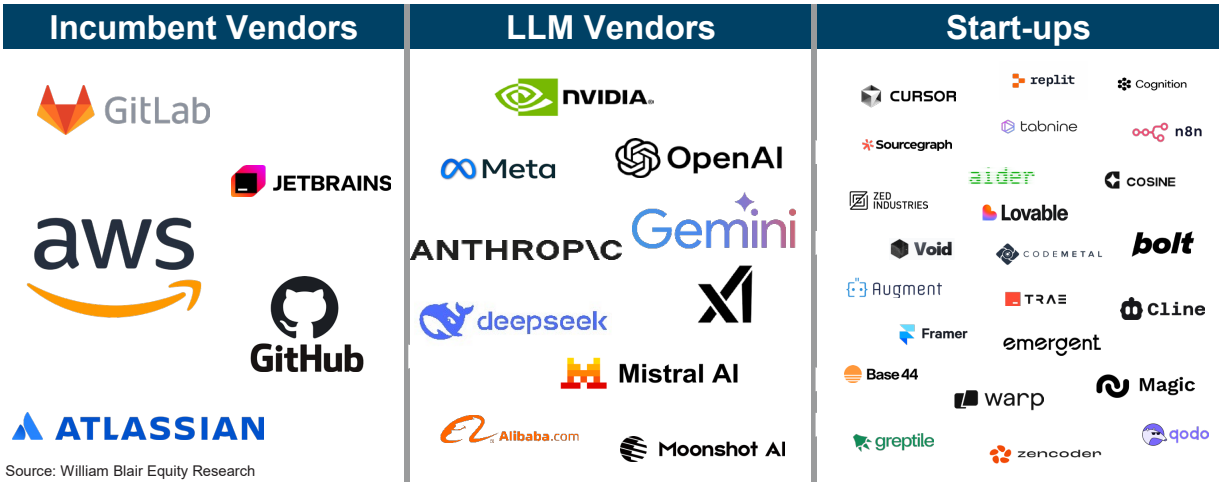
While GitHub Copilot (launched in mid-2021) gets the most credit for sparking the AI coding revolution, the AI coding space now contains a myriad of tools from established and new vendors. At a high level, we segment the market into three main buckets of competition:

1. ***Incumbent vendor tools*** – include tools like GitHub Copilot (Microsoft), GitLab Duo, JetBrains AI Assistant, and Amazon Q Developer

2. ***LLM vendor tools –*** include Claude Code (Anthropic), Codex (OpenAI), Gemini Code Assist (Google), and Grok Code Fast Assistant (xAI); these vendors are moving up the stack—instead of staying focused on base models and low-level infrastructure, they are increasingly building higher-level, end-user-facing products that sit closer to where business value and revenue are captured

3. ***Start-up vendor tools*** – include dozens of coding assistants from vendors like Cursor (Anysphere), Windsurf (Cognition), Lovable, Replit, Bolt, Warp, Tabnine, and Base44 (Wix)—with distinctions around whether the tool is targeted at professional or amateur developers (i.e., vibe coding) and how deeply the tool is embedded in agentic workflows

Exhibit 4
Cracking the Code: How AI Is Transforming Software Development
AI Coding Competitive Landscape



Source: William Blair Equity Research

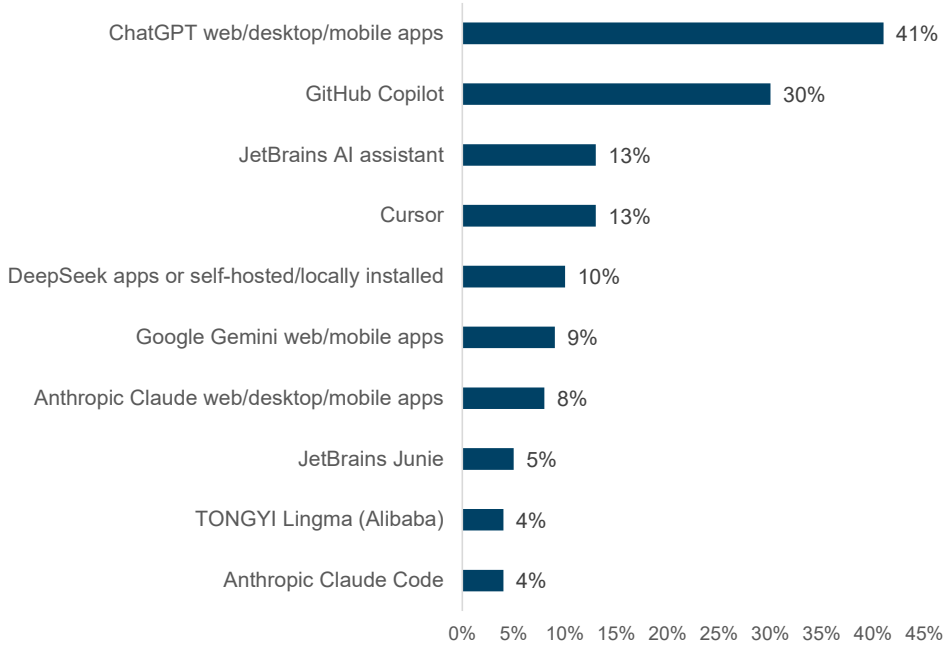
LLM Vendors Will Be Tough to Beat, but Room Still Exists for Best-of-Breeds

At this juncture, we believe LLM vendors like Anthropic, OpenAI, and Google have a structural advantage in AI coding. This is because they not only offer their own established coding assistants (Claude Code, Codex, Gemini) but, more importantly, also control the foundation models that are the back-end processing engines for virtually all of the pure-play tools, which gives the LLM vendors unique pricing power and user telemetry to continually improve their models. *Looking ahead, one major wildcard is whether open source LLMs (from the likes of Nvidia, Mistral, Meta, DeepSeek, and Qwen) are able to cross the chasm on post-training and reinforcement learning, which is currently holding back their usability and performance compared to the closed-source frontier models.*

Pure-play AI coding vendors like Cursor, Cognition, Replit, and Lovable are not blind to the risk of overdependence on the frontier models. It is no secret that gross margins for many of these players are very low at the moment, as they need to pay for every API call back to a closed-source LLM (the biggest component of their COGS). This helps explain why Cursor recently introduced its own coding-specific LLM called Composer (which reportedly is a fine-tuned version of the open-source Chinese model Qwen), though Cursor will also continue to partner with Anthropic and OpenAI. In addition, Cursor recently adjusted its flat per-seat pricing model to a usage-based model that will protect margins by effectively passing through LLM costs to heavy users (see page 23 for more details).

Beyond the cost element, we believe Cursor can remain competitive with the big LLM providers by using reinforcement learning (post-training) to refine its Composer model. Models depend on a feedback loop from users to continually improve, and Cursor already has millions of users providing them with massive amounts of product-specific data (traces, diffs, accept/reject signals). Lastly, user experience and workflow integration will continue to be relevant in this space (and may ultimately prove stickier than expected), which supports the viability of at least some of the pure-play start-ups (assuming they address the margin question).

Exhibit 5
Cracking the Code: How AI Is Transforming Software Development
JetBrains Survey: AI Tools Regularly Used for Coding and Development



Source: JetBrains Developer Ecosystem 2025 Survey and William Blair Equity Research

How Developers Are Using AI Today

Based on multiple surveys, most developers are gaining significant productivity benefits from using AI coding tools. According to an October 2025 JetBrains survey of almost 25,000 developers across 19 countries, nearly 9 out of 10 developers save at least one hour every week using AI tools, and 1 in 5 saves eight hours a week or more. AI is particularly useful in saving developers' time on tedious, undifferentiated tasks such as writing boilerplate, repetitive code, writing and reviewing documentation, converting code from one programming language to another, and summarizing code changes. At the same time, programmers are wary of using AI for writing more complex business logic, debugging, understanding code, and performing actions in their terminals.

Exhibit 6
Cracking the Code: How AI Is Transforming Software Development
AI Use-Cases: Most Likely and Least Likely

Most Likely Delegated to AI	Top 5	Least Likely Delegated to AI
Writing boilerplate, repetitive code	1	Communicating through email and messaging
Searching for development-related information on the internet	2	Writing application logic code
Converting code to other languages	3	Performing actions in the terminal / CLI
Writing code comments or code documentation	4	Debugging
Summarizing recent code changes	5	Understanding code

Source: JetBrains Developer Ecosystem 2025 Survey and William Blair Equity Research

Moreover, according to an ongoing Stanford Software Engineering Productivity Study (run by SWEPR), developer productivity benefits shrink dramatically when applying AI to mature, existing codebases. A study of 136 teams across 27 companies revealed the following results across four key scenarios:

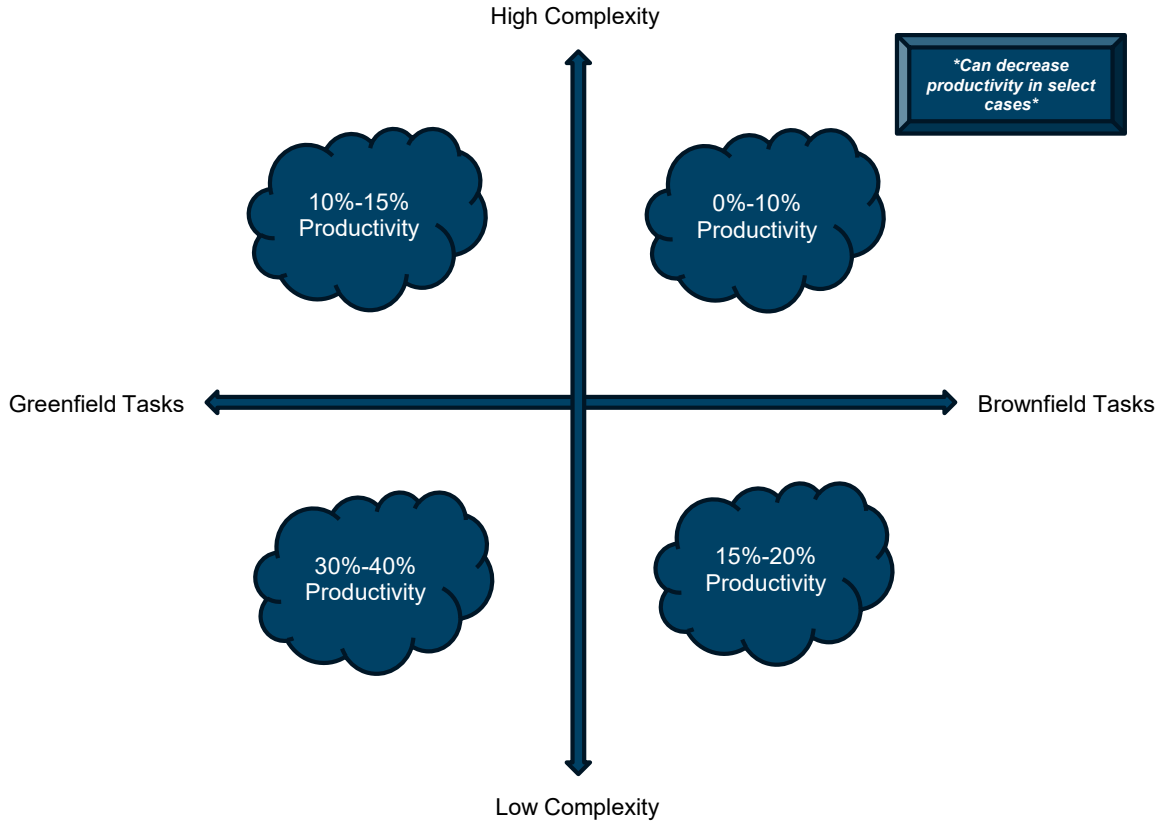
1. ***Low-complexity, greenfield tasks (AI's sweet spot):*** AI boosts productivity by 30%–40% for simple, new projects.

2. ***Low-complexity, brownfield tasks:*** AI boosts productivity by 15%–20% for simple tasks within an existing codebase.

3. ***High-complexity, greenfield tasks:*** AI boosts productivity by around 10%–15% for complex, new projects.

4. ***High-complexity, brownfield tasks:*** AI boosts productivity by only 0%–10% for the most difficult tasks in a mature, existing codebase. In some cases, AI can even decrease productivity.

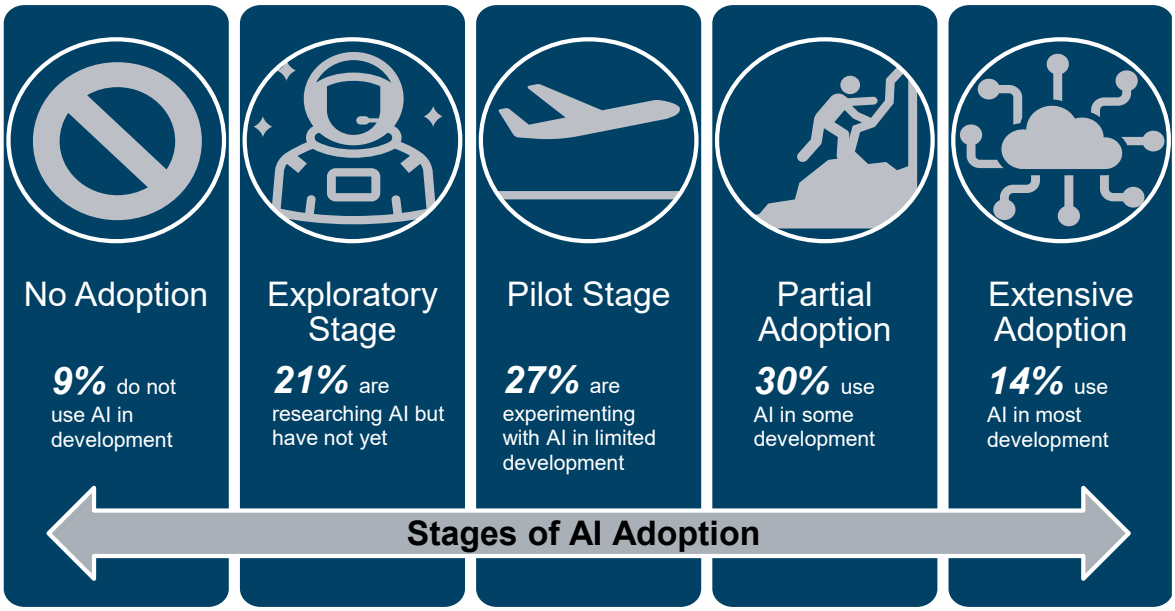
Exhibit 7
Cracking the Code: How AI is Transforming Software Development
Stanford Survey: Where Is AI the Most Helpful?



Source: "Does AI Actually Boost Developer Productivity?" by Yegor Denisov-Blanch of Stanford University, William Blair Equity Research

In terms of overall adoption of AI in software development, the JetBrains survey referenced above (exhibit 6) reveals that most users are in the middle stages of adoption, with 14% of those surveyed using AI in most development workflows and only 9% not using AI whatsoever. As noted earlier, our research indicates that senior engineers are seeing the most productivity gains (in some cases 10-100x), because the more knowledge a developer has on the nuances of programming languages, product architecture, and project context, the more they can maximize the value of the new tools.

Exhibit 8
Cracking the Code: How AI Is Transforming Software Development
JetBrains Survey: If Not Now, When?



Source: JetBrains Developer Ecosystem 2025 Survey and William Blair Equity Research

Agent HQ From GitHub: Creating a Control Plane for AI Coding Tools

At its GitHub Universe 2025 conference in October, Microsoft announced Agent HQ, which aims to create a unified control plane for managing multiple AI coding agents from competitors including Anthropic, OpenAI, Google, Cognition, and xAI—all within a customer's existing paid GitHub Copilot subscription. Rather than forcing developers into using GitHub Copilot—it is clear that the average developer likes different tools for different jobs—Microsoft is presenting itself as an orchestration/governance layer beneath the wide and growing array of popular coding assistants. This positions GitHub not just as a source code repository but as a central hub for AI-driven software development—a move that could pressure incumbents to respond with similar strategies.

This is a wise move by Microsoft, in our view, especially given the growing momentum of newer coding assistants (GitHub Copilot is losing market share to tools like Cursor, Claude Code, and OpenAI Codex). Agent HQ holds the potential to expand GitHub's monetization opportunities while reinforcing its DevOps dominance by embedding multi-agent orchestration and governance directly into GitHub's developer workflow.

Within the Agent HQ architecture, developers will still rely on Git-based source code repositories for pull requests (PRs) and use their preferred CI/CD engines (GitHub Actions, Jenkins, CircleCI, or other third-party offerings). What differentiates Agent HQ is the ability of coding assistants/agents from multiple vendors to operate within GitHub's Mission Control security and governance layer, leveraging the same identity controls, branch permissions, and audit logging that enterprises already trust for human developers. This centralized but granular permissioning approach differs fundamentally from standalone tools like Cursor and Claude Code where users today must grant broad permissions across entire repositories.

The Cursor Story

Founded in 2022 by four MIT students and based in San Francisco, California, Anysphere is the company behind Cursor, an AI-native IDE forked from VS Code. Cursor allows users to toggle between different AI models on the back end (from OpenAI to Anthropic to Google) and claims greater functionality than traditional code editors/IDEs by offering not just generic code suggestions, completion, and debugging but also the ability to analyze the entire code base of a software project so that its suggestions mesh better with the adjacent code (what industry analysts call "context engineering").

With the release of Cursor 2.0 in October 2025, the company has launched its own AI model, called Composer, which the company claims is 4 times faster than frontier models and reportedly is a fine-tuned version of the open-source Chinese model Qwen. This new version of Cursor also offers users the ability to create and run eight custom agents in parallel—the company argues that having multiple agents attempting the same problem and picking the best result will significantly improve the final output, especially for harder tasks. In December 2025, Cursor announced the acquisition of Graphite, a start-up specializing in code review and debugging. By combining code writing and code reviewing tools, Cursor aims to accelerate its vision of an end-to-end, AI-powered software development experience.

Today, Cursor is being used by over half of the *Fortune* 500, and the company recently surpassed $1 billion in ARR, up from $100 million in January 2025. In November 2025, the company announced a series D financing round, raising $2.3 billion at a $29.3 billion post-money valuation (nearly 12 times the valuation in January 2025). New investors in the round included Coatue, Nvidia, and Google, adding to the existing investor roster including Accel, Thrive, Andreessen Horowitz, and DST.

Exhibit 9
Cracking the Code: How AI Is Transforming Software Development
Notable AI Coding Venture Rounds, 2023-2025

Company Name	Amount Raised	Valuation	Date	Series	Lead Investors
Lovable	$330	$6,600	12/18/2025	B	CapitalG, Menlo Ventures
Anysphere	$2,300	$29,300	11/13/2025	D	Accel and Coatue
Code Metal	$37	$250	11/12/2025	A	Accel
n8n	$180	$2,500	10/9/2025	C	Accel
Vercel	$300	$9,300	10/1/2025	F	Accel and GIC
Emergent	$23	NA	9/24/2025	A	Lightspeed Venture Partners
Replit	$250	$3,000	9/10/2025	NA	Prysm Capital
Cognition AI	$400	$10,200	9/8/2025	NA	Founders Fund
Anthropic	$13,000	$183,000	9/2/2025	F	ICONIQ
Framer	$100	$2,000	8/28/2025	D	Meritech and Automico
Zed	$32	NA	8/20/2025	B	Sequoia Capital
OpenAI	$8,300	$300,000	8/1/2025	NA	Dragoneer Investment Group
Lovable	$200	$1,800	7/15/2025	A	Accel
xAI	$10,000	NA	7/1/2025	NA	Morgan Stanley
Anysphere	$900	$9,900	6/9/2025	C	Thrive Capital
Anthropic	$3,500	$61,500	5/3/2025	E	Lightspeed Venture Partners
n8n	$60	$350	3/24/2025	B	Highland Europe
Anysphere	$105	$2,600	1/14/2025	B	Thrive Capital
xAI	$6,000	$50,000	12/23/2024	C	A16Z, Blackrock, Fidelity, Lightspeed Ventures, Morgan Stanley, Sequioa
Qodo (formerly CodiumAI)	$40	NA	9/30/2024	A	Susa Ventures and Square Peg
Codeium	$150	$1,250	8/29/2024	C	General Catalyst
Anysphere	$60	$400	8/9/2024	A	Andreesen Horowitz, Thrive Capital
xAI	$6,000	$24,000	5/26/2024	B	Valor Equity Partners, Vy Capital, Andreessen Horowitz, and Sequoia Capital
Cognition	$175	$2,000	4/25/2024	NA	Founders Fund
Augment Inc.	$227	$977	4/24/2024	B	Eric Schmidt, Index Ventures, Sutter Hill Ventures, Lightspeed Partner Ventures
Tabnine	$25	NA	11/8/2023	B	Telstra Ventures
Framer	$27	NA	9/28/2023	C	Meritech
Anthropic	$450	$4,000	5/23/2023	C	Spark Capital
Replit	$97	$1,160	4/25/2023	B	Andreesen Horowitz (a16z)

Note: All numbers in millions. NA = funding round was not a series round
Source: Pitchbook Data, Inc., William Blair Equity Research

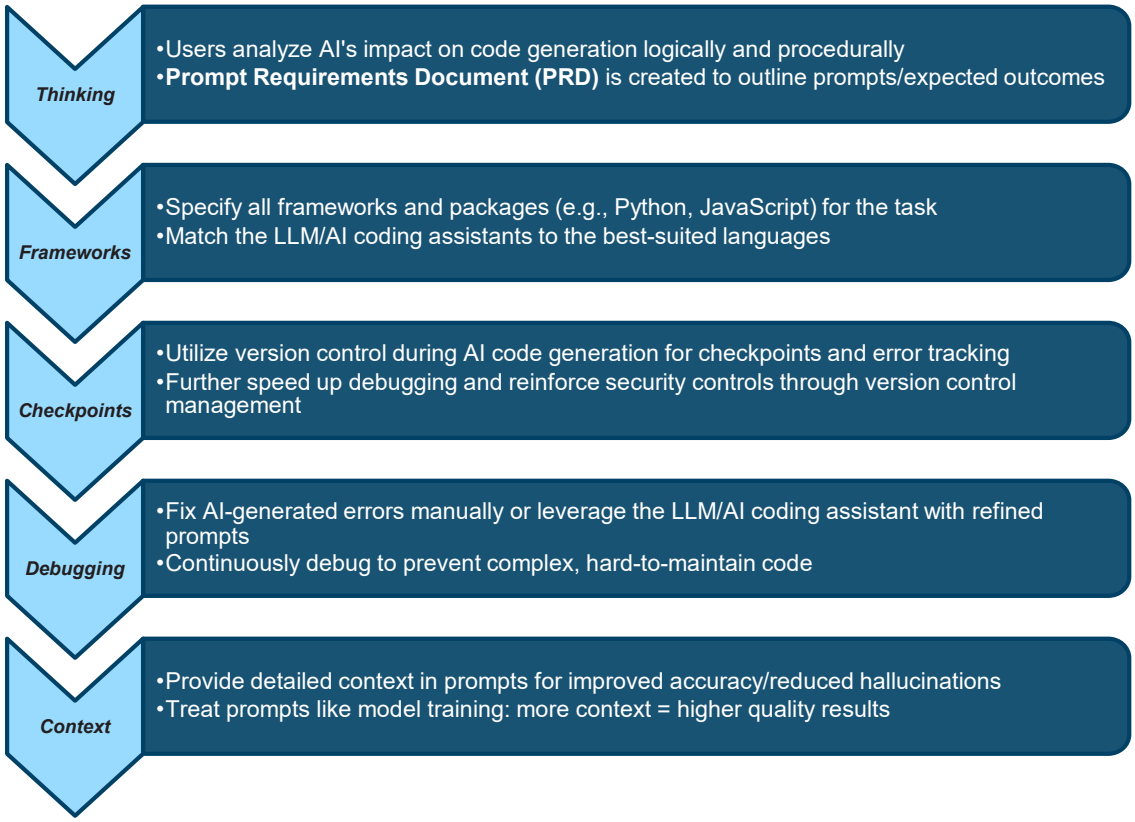
Vibe Coding May Not Be Ready for Prime Time (but Is Also Not a Passing Fad)

Vibe coding is a new approach to AI-assisted coding where software developers can direct the AI to write code based on natural language prompts rather than manually writing the code themselves with assistance from AI. The concept is as simple as it sounds, with users communicating with AI agents or LLMs through plain speech or text to create code tailored to their end-goal (whether it be creating a new environment, application, or line of code).

The vibe coding workflow is a three-step process: 1) users describe the task/code they want to create to the LLM; 2) the AI generates the task/code output, with many AI-coding assistants utilizing an "auto-run" mode that enables the tools to run without asking users for confirmation between each step; and 3) the code/task output undergoes testing and refinement before users

integrate the production code into the pipeline, with the cycle repeating itself if a user wants to add on to their existing code. Ultimately, vibe coding aims to allow the developer to work at a higher abstraction level with the AI filling in much of the lower-level implementation (boilerplate, rote code creation, etc.).

Exhibit 10
Cracking the Code: How AI is Transforming Software Development
Understanding the Vibe Coding Process/Framework



Source: Tina Huang, William Blair Equity Research

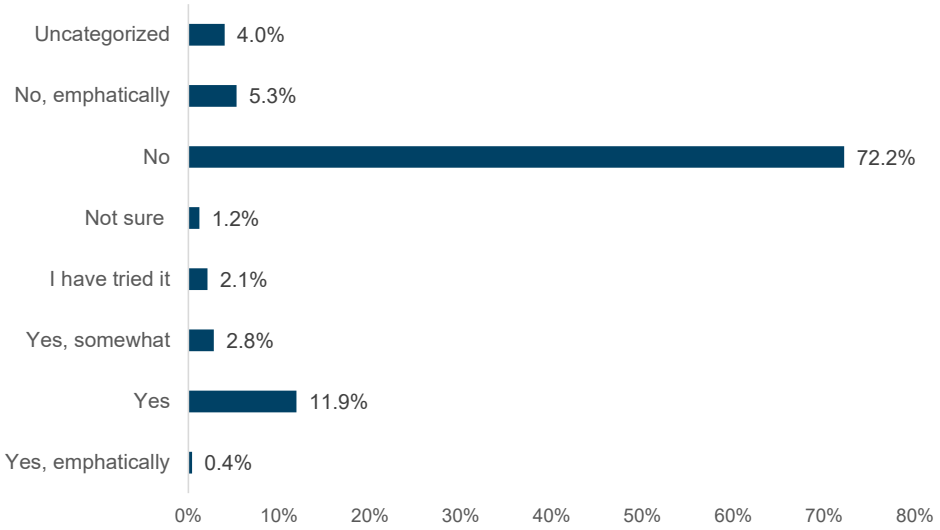
While vibe coding tools like Lovable and Replit are seeing strong adoption, professional developers have warned of the myriad shortcomings of vibe coding, viewing these tools as more appropriate for rapid prototyping than generating production-quality code. Key criticisms of vibe coding include the lack of structure, maintainability, and efficiency in the code generated by these tools, with developers complaining that debugging turns into a time sink. In addition, because the code generated from these tools emerges through intuition or auto-completion, it often lacks the proper context or architectural consistency, forcing developers to spend extra time on addressing edge cases, optimization, or production-level concerns. Then again, vibe coding democratizes code development, dramatically lowering the barrier of entry to coding for nontechnical users, who can now build basic applications through natural language prompts that previously would have required professional software developers and service tickets to accomplish.

If we think of software development as on a spectrum, with binaries (zeros and ones) being at one end and natural language at the other, vibe coding dramatically pushes the SDLC toward the natural language side of the spectrum, superseding low-code/no-code platforms that have emerged over the past decade. While vibe coding today is often conflated with slop and bad code, we do

not see it as a passing fad, with this technology likely to evolve and mature very quickly in coming years. Indeed, it is not too far-fetched to think of a future where a user can describe an application and its parameters and subsequently have a good framework for it to be built semi-autonomously.

At the same time, we are skeptical that professional developers will trust vibe coding any time soon for production-grade code, especially for the foundational pillars of an application, including the business logic and data layers. This could partly be for reasons of self-interest—because at best the human developer does not want to lose control of the development process and at worst, they do not want to lose their job. But more realistically, this is because software is finicky and complex, and code quality is therefore paramount—and developers despise bad code. As evidence, according to GitLab's 2025 Global DevSecOps report, 73% of professionals surveyed have experienced problems with code created by vibe coding tools (using natural language prompts without understanding how code works).

Exhibit 11
Cracking the Code: How AI Is Transforming Software Development
Developer Vibe Coding Adoption*



***The question asked: In your own words, is "vibe coding" part of your professional development work?**
Source: Stack Overflow and William Blair Equity Research

How Coding Agents Work

AI coding assistants have gone through substantial technical transformation over the past few years, evolving from simple code suggestion/auto-completion tools into collaborative, AI-powered agents that support a wide range of software development tasks (including code testing, code refactoring, security scanning, code reviews, etc.). The shift from AI-assisted coding to AI-driven development, where AI coding assistants are part of a tightly integrated vertical stack, is ultimately reshaping how developers think and approach the software development process. This has significant implications for the DevOps toolchain as a whole, and traditional DevOps vendors may be at risk if they do not adapt and evolve with AI-powered features that either generate efficiencies or reduce complexity for users.

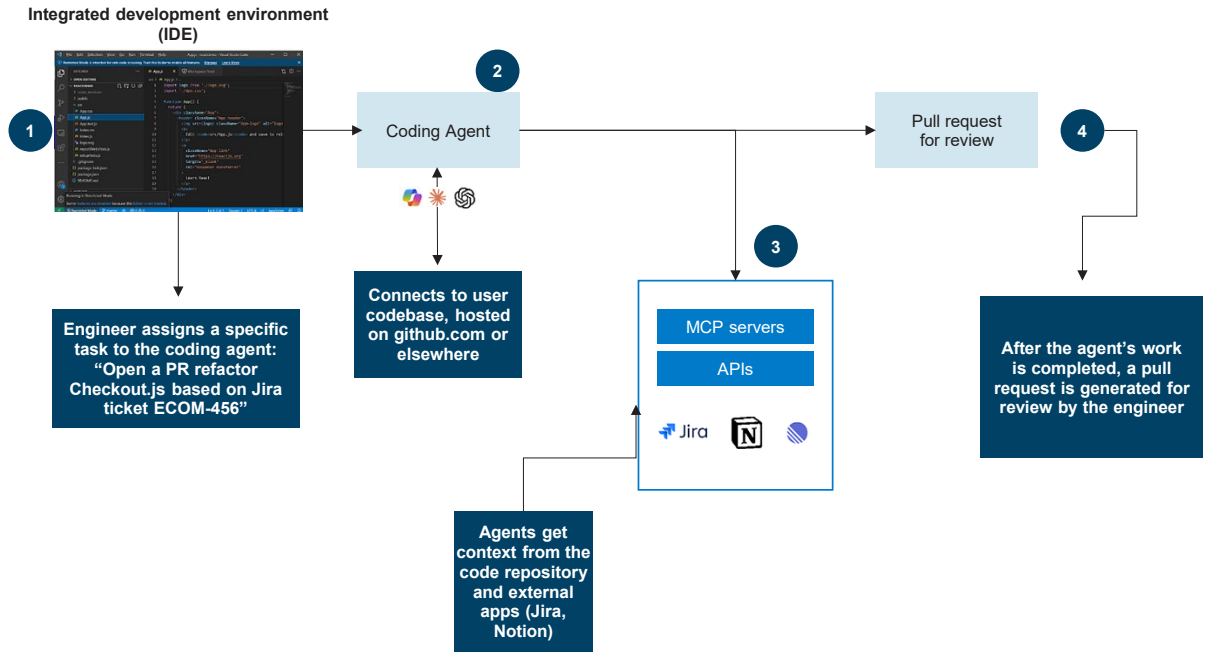
At the core, an AI coding agent can take a goal, plan the steps required, and execute those steps across the toolchain—including code changes, branches, PRs, tests, and deployments—with minimal human intervention. AI agents strive to mimic human interactions, with their sophisticated ML algorithms focused on mirroring developer coding patterns while optimizing various coding

tasks across the SDLC (see exhibit 12 below). Specifically, AI coding agents integrate with other tools within the SDLC and leverage memory from past tasks/coding history for richer context, allowing them to perform more complex tasks (e.g., ability to make decisions/create multistep plans, capacity to execute multiple tasks in parallel).

Context is critical for today's AI coding agents to match the capabilities of an experienced human developer—this encompasses domain "vocabulary", code structure, dependencies, and company-specific best practices and security and compliance requirements. At the same time, organizations today lack a centralized, policy-based oversight mechanism that governs what agents should see/know to accomplish tasks reliably and securely. This has led start-ups like Cursor, Tabnine, Sourcegraph (Cody), and Augment to introduce the concept of a "context engine," which is a centralized system that pulls relevant code, docs, symbols, and project structure into the AI's working context so coding decisions/agentic tasks are accurate and auditable.

While software development may not be truly automated anytime soon, the technology is moving at a lightning-fast pace with AI coding agents eventually expected to address issues from a developer's backlog, implement features, run tests, generate code reviews, and even fix security vulnerabilities. For example, OpenAI's new Aardvark agent (in private beta) can scan source code repositories on an ongoing basis to flag vulnerabilities, test the exploitability of code, prioritize bugs by severity, and propose fixes. Other examples of AI coding agents include GitHub Copilot Coding Agent, GitLab Duo Agent Platform, Google Antigravity, Replit Agent, and Cursor Agent. In its latest 2.0 release, Cursor introduced a multi-agent interface that lets users run up to eight AI agents in parallel, each working in its own isolated environment.

Exhibit 12
Cracking the Code: How AI Is Transforming Software Development
How AI Coding Agents Work



Source: Department of Product and William Blair Equity Research

AI Coding Massively Expands the Dev Tools TAM

AI has introduced an entirely new layer in the developer toolchain—encompassing code generation, reasoning, and autonomous/agentic task execution—for which developers and companies are willing to pay. Before AI, IDEs/code editors like VS Code were effectively free. With the advent of AI, the IDE has moved from a commodity into a monetizable SaaS category, materially expanding the developer tools TAM (which IDC estimates was $27.4 billion in 2024, including minimal revenue from AI coding tools as the technology was still nascent).

Aggregating publicly available data, we estimate that the AI coding market will surpass $5 billion in sales in 2025 alone (Cursor and Claude Code each surpassed $1 billion in annual run-rate revenue late last year, with GitHub Copilot reaching over 26 million users). This includes revenue from incumbents like Microsoft/GitHub, GitLab, and JetBrains; LLM providers like Anthropic, OpenAI, and Google (includes commercial products like Claude Code and OpenAI Codex, and some portion of subscription revenue from generic chatbots like ChatGPT, Claude, and Gemini used for coding tasks); and new-gen players like Cursor, Cognition, Lovable, and Replit.

Shift to Usage-Based Pricing

Pricing for AI coding solutions is evolving rapidly, with the original flat per-seat pricing model (around $20 per month) no longer viable (as heavy users were taking advantage of flat pricing, making them uneconomical for the vendor). As an example, Cursor is no longer giving Pro users unlimited access for a flat fee. The company's new Pro plan, introduced in June 2025, provides a monthly allowance (credit pool) of roughly $20 for "frontier-model usage." Once a user consumes that allowance (via API calls to LLMs), further usage is either: a) automatically routed to Cursor's Auto mode (leveraging cheaper/slower models), or b) charged on a pay-as-you-go basis (i.e., overage billing) for consuming premium models beyond the allowance. Cursor also introduced a higher-tier plan—a $200/month Ultra plan—intended for heavy/power users who consume more compute (e.g., large-context tasks, frequent AI-assisted coding, heavy refactoring) and thus need a much bigger usage allowance (around 20x compared to the Pro tier).

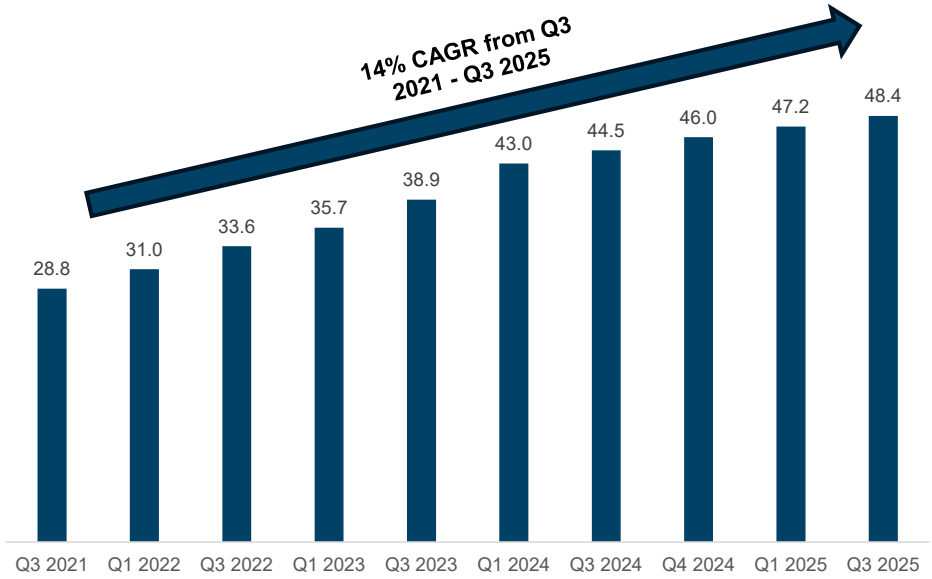
Estimating the TAM

Pricing from other AI coding vendors generally follows this consumption-based (usage credits) approach, which will significantly raise the average monthly price the average developer pays beyond the roughly $20 per month per user that vendors like GitHub and Cursor were initially charging. Based on our anecdotal research, we believe a) the average developer will buy more than one AI coding tool, potentially three or four (different tools for different jobs); and b) we are still in the early days of consumption as developers get comfortable with the tools—once they lean in, we expect to see consumption ramp up by an order of magnitude.

As a result, we estimate that the average professional developer will ultimately spend at least $2,000 per year on a set of AI coding tools, with power users likely spending well into the thousands of dollars. Multiplying this by the global population of professional developers (estimated at 48 million in 2025) would amount to an annual TAM approaching $100 billion, just for AI coding tools/agents. We note that this high-level analysis likely understates the full TAM for AI coding as it excludes nonprofessional users that are rapidly adopting vibe coding tools like Lovable and Replit.

In terms of impact on IT budgets, we see this spending as more incremental than substitutive—eating into the labor budget versus the IT budget. This is because of the enormous productivity benefits that AI coding brings to the table and the relatively low cost of these tools ("a drop in the bucket", according to a tech company CEO we spoke with) compared to average annual developer compensation that ranges from $150,000 at the low end to seven figures at the high end (when factoring in stock grants/options).

Exhibit 13
Cracking the Code: How AI Is Transforming Software Development
Global Developer Population Over Time



Note: All numbers shown in millions unless otherwise specified
Source: SlashData Developer Nation Surveys and William Blair Equity Research

Risks of AI Coding

AI-generated code can lead to finished applications/products that are more vulnerable to cyberattacks (vulnerabilities in the code), legal issues (models leveraging copyrighted data/code), sensitive data exposure (models accessing proprietary code/data), incorrect code (derived from model hallucinations), and elongated debugging processes (users may struggle to fix code errors if they do not understand how to code or how the code works). We detail key risks and challenges below.

Operational Challenges
One of the main operational hurdles users encounter with AI coding assistants is the risk of dealing with overly complex code for the queried task. This can become a significant issue for both technical and nontechnical users, where users may receive an answer from their query that contains an abundance of code or information unnecessary to the prompt's objective. This overcomplexity of code can lead to additional delays with time to deployment, with users needing to spend time finding errors or debugging code.

Models can also struggle with solving complex tasks, which can become frustrating for users and contribute to further bottlenecks in the software development process (users must become more active managers of AI agents). This phenomenon is commonly referred to as "the 70% problem," where AI coding assistants can complete 70% of the task but the remaining 30% remains a constant battle between the user and the model as new bugs and issues arise from AI-generated suggestions to fix errors in the code. As an increasing number of users from all technical backgrounds are looking to offload coding projects onto AI agents or leverage AI to experiment with complex coding ideas, this can lead to significant setbacks in time to deployment and future code maintenance issues as the project evolves/becomes more complex.

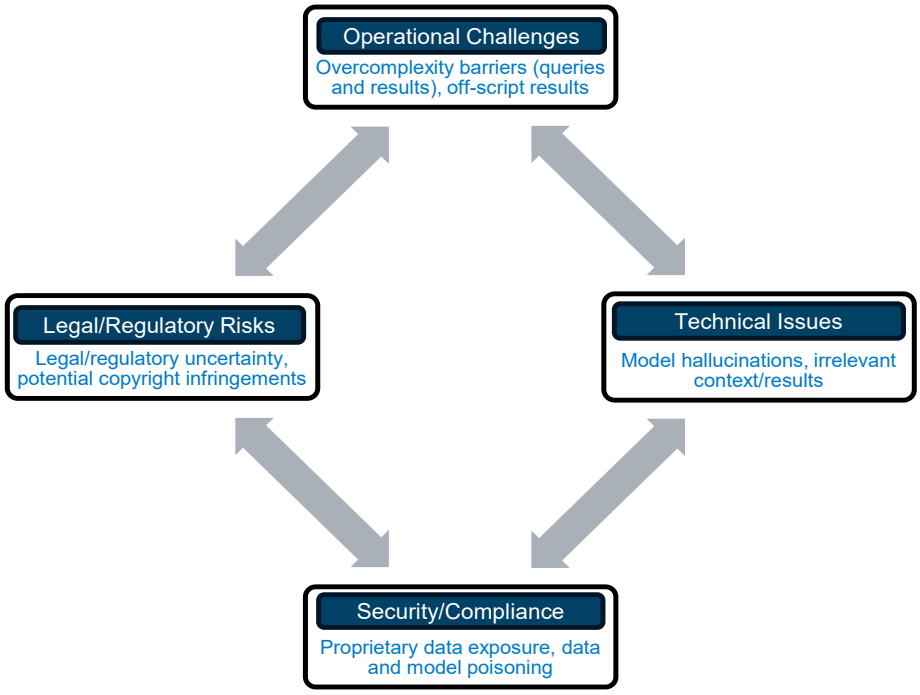
To help mitigate the above problems, many AI coding assistants build in a step-by-step reasoning approach to generating code/software engineering tasks to facilitate better human oversight of the AI coding process. Developers can quickly review each step of the code generation process before directing the AI to continue onto the next step (as the developer becomes more confident in the technology, they can activate "auto-run" mode for the AI coding assistant to autonomously generate code).

In addition to the step-by-step approach, users should supply as much context as possible in their queries to avoid potential shortcomings of AI coding assistant output. As mentioned earlier in the vibe coding section, users should employ a planning step before inputting their query (users can either create the query plan themselves or ask the AI coding assistant to "think" and brainstorm a plan for the desired output).

However, the need for context is a two-way street as users yearn for the utmost transparency with AI coding assistants to ensure efficient oversight and management (more specifically when AI coding assistants hallucinate). The disclosure of the back-end model's reasoning process is a crucial element to debugging and fixing issues with AI-generated code as users can better diagnose errors or incorrect outputs through access to the model's chain of thought (CoT). As a result, context is paramount to driving operational efficiencies with AI coding assistants (and the flagship models powering them under the hood).

Lastly, a more minor challenge we have seen is AI coding assistants explicitly not following user directions. While largely uncommon, two main examples of this occurrence include models either refusing to generate the requested code or generating code and completing various software engineering tasks that were not assigned by the user. This operational challenge can be controlled and monitored by the inclusion of humans in the feedback loop, who can often identify and manually fix the operational hiccup of the model.

Exhibit 14
Cracking the Code: How AI Is Transforming Software Development
Risks of AI Coding Assistants



Source: William Blair Equity Research

Technical Issues
A common worry surrounding AI coding assistants is the likelihood of model hallucinations and incorrect output. By nature, LLMs are probabilistic and not deterministic (i.e., they generate plausible code, not necessarily provably correct code), which creates fundamental accuracy hurdles to overcome. That said, the top-performing AI coding assistants thus far have performed in the 60%-70% range against the SWE-bench (a Princeton-developed software engineering benchmark to evaluate the performance of LLMs). Although the greater-than-50% accuracy is impressive for these models in their early innings of development, there is still ample room for model improvement as AI coding providers look to eliminate all hallucinations in creating a more autonomous and reliable technology.

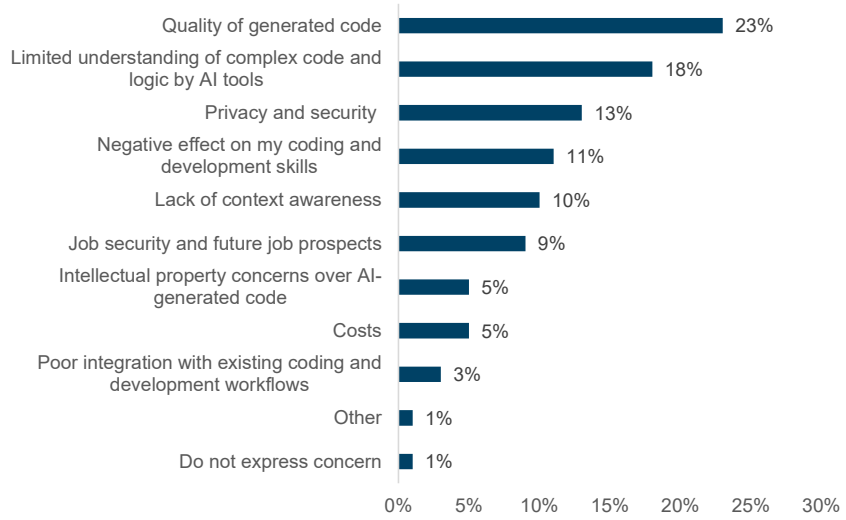
However, as stated earlier in the report, users can replug AI-generated code back into models to further debug and fix issues with the AI-generated code output. Models can learn from additional context combined with the initial hallucinations, creating a more powerful AI coding assistant that should generate increasingly accurate output. In addition, AI coding assistants may produce results with overcomplexity or irrelevant context due to biases in the training data. If models were trained to generate code for specific tasks, this may skew results when users look to leverage AI coding outside the scope of its trained data.

Security and Compliance
From a security and compliance perspective, one of the biggest risks that may arise is AI coding assistants training on proprietary customer data/code. To mitigate this risk, most AI coding vendors retain a strict default data policy (specifically for business and commercial users) explicitly stating that public models are not trained on customer data. If customers would like their data to be leveraged for future model training, they have the option to do so through an opt-in consent agreement.

In addition, while less of a threat due to the heavily fortified security protocols of vendors, AI coding assistants are also susceptible to data and model poisoning. Bad actors could potentially corrupt models with malicious code inputs, which can lead to the production of vulnerable code or compromised systems.

Legal/Regulatory Risks
Against the backdrop of continued uncertainty about the domestic and international legal and regulatory landscape for AI, coding vendors (and in turn, their customers) may face legal action from the reproduction of licensed code snippets without proper attribution (illustrated by the class action lawsuit filed against Microsoft, GitHub, and OpenAI in November 2022). This risk is magnified by the sheer volume of AI-generated code and the difficulty in identifying copyrighted code within model output data.

Exhibit 15
Cracking the Code: How AI Is Transforming Software Development
JetBrains Survey: Biggest Concerns About AI



Source: JetBrains Developer Ecosystem 2025 Survey and William Blair Equity Research

AI Is Fundamentally Reshaping Software Development

AI is dismantling many of the historical constraints associated with software development, namely time, cost, and resource intensity. The main impact has been a meaningful acceleration of software development cycles—projects that once took months to plan, staff, and ship can now be completed in a matter of weeks or days, with far fewer engineers needed, though the rate of adoption varies widely depending on the organization.

For example, at a sophisticated enterprise, a typical user interface improvement project might have historically required six weeks of planning; a team of 15 designers, engineers, and managers, a month of feature building; and another three to four months of development, testing, and QA. With a full embrace of AI-assisted development, that same project can now be accomplished in a few weeks, with only a handful of engineers.

While most of the disruption has been occurring at the coding layer, AI tools are now infiltrating additional stages of the SDLC, including writing unit tests, automating QA processes, generating documentation and code reviews, scanning for security vulnerabilities, debugging and refactoring code, monitoring production systems, and even orchestrating deployment pipelines. Testing and QA functions, in particular, which were once major sources of manual effort, are now becoming more automated, with the traditional waterfall model between developers, testers, and release teams breaking down.

This means that the traditional linear sequencing of the DevOps toolchain—from plan to code to build to test to secure to release to monitor—is melting away as AI blurs boundaries between traditional phases and reduces the need for discrete tools. Real-world examples here include the integration of coding and testing (the AI can generate tests as the code is being written) and the convergence of the design and prototyping phases for new software projects (used to be done by separate teams but now a single engineer equipped with AI can build this rapidly).

That said, not every traditional constraint in the SDLC is being overcome. As AI-generated code volumes grow, the attack surface for security vulnerabilities will only expand, and compliance frameworks still typically require human signoffs. While firms are experimenting with AI code reviewers, regulators have yet to bless these as acceptable for compliance purposes. In the near term, this creates a bottleneck at the governance layer, even as the coding layer becomes increasingly frictionless.

Systems of Interactions Versus Systems of Record

Our belief is that AI's main transformational impact on the SDLC at this point is in systems of interaction (SOIs) and not systems of record. SOIs serve as the primary interfaces through which developers carry out their day-to-day tasks, including coding, builds, collaboration, code reviews, and QA. These routine workflows sit on top of SORs, which serve as the "ledgers" of the SDLC, providing single-source-of-truth grounding to the various systems of interactions.

Foundational SORs in the SDLC include:

- ***Planning:*** issue and project tracking (top products: Atlassian Jira, GitHub Projects)
- ***Source code management:*** Git-based version control systems (top products: GitHub, GitLab, Atlassian BitBucket)
- ***Software releases:*** artifact and dependency management (top products: JFrog Artifactory, GitHub Packages)
- ***Security and identity:*** vulnerability databases and access control (top products: Snyk, Veracode, Black Duck, SonarQube, Checkmarx, JFrog, GitLab, GitHub, Harness)
- ***Runtime production:*** monitoring and observability of applications in production (top products: Datadog, Grafana)

While we are seeing rapid innovation and fragmentation in systems of interaction (agents, copilots, new UIs, etc.), they all need to write back to a small set of stable SORs. Put simply, AI coding tools and agents need to rely structurally on foundational SORs to be secure, grounded, explainable, and auditable.

Exhibit 16
Cracking the Code: How AI Is Transforming Software Development
Foundational Systems of Record in the SDLC

Source: William Blair Equity Research

Revenge of the IDE

As noted earlier, an integrated development environment (IDE) is a software application that provides a complete workspace for writing, editing, testing, and debugging code. Today's IDEs are undergoing a major disruption driven by the embedding of AI coding tools (e.g., GitHub Copilot) within existing IDEs or the full-scale replacement with AI-native IDEs (e.g., Cursor). Given the potential productivity gains developers can reap from these next-generation IDEs—and the fact that IDEs are increasingly being perceived as control points in the SDLC stack and not just simple code editors—vendors are finding ripe monetization opportunities (e.g., both Cursor and Claude Code have surpassed $1 billion in annual run-rate revenue in late 2025). Generating code with AI is just the start—the ultimate goal of these next-generation IDEs is to orchestrate development tasks through an end-to-end agentic platform (which is progressing rapidly).

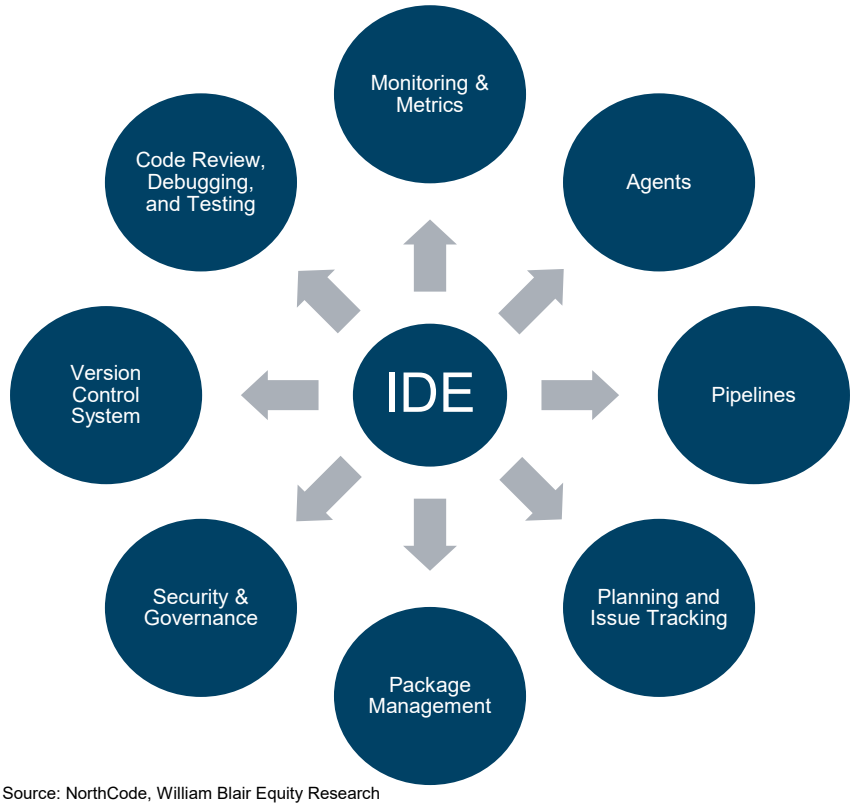
Because it is home to coding, the IDE (or terminal in the case of CLI-based tools) is on the road to displacing the source code repository/version control system (VCS) as the fulcrum or "center of gravity" in the DevOps toolchain, in our view. Historically, we posited that the VCS (e.g., GitHub/GitLab) was the DevOps center of gravity (see our DevOps white paper here), because this is where both source code and infrastructure configurations were stored and managed and where developer PRs were made (i.e., requests to make changes to existing code).

Our viewpoint shift is based on three main points: 1) the IDE/terminal is the developer's primary workspace, where programming happens and where AI has the most near-term impact; 2) AI accelerates the shift left trend in software development as "Ops" functions like testing, QA, security, and deployment can now be integrated to some extent inside the IDE/terminal—this shift reflects a desire to move these functions closer to the point where developers code, with the goal of driving greater automation and speeding up development cycles; and 3) embedding more DevOps functions in the IDE saves developers time and reduces context switching between IDE, VCS, CI/CD

dashboards, etc. Because AI-native IDEs see the full codebase and understand context for a given application, they are also the logical hub for housing and orchestrating intelligent software agents, which can suggest tests, flag security risks, and automate deployments.

Looking ahead, we believe that modern IDEs will increasingly absorb developer-facing interactions with traditional VCS, CI/CD, and deployment tools. In the traditional model, developers wrote code in the IDE, pushed it to the VCS, triggered CI/CD pipelines, and deployed the application to the cloud once testing and security checks were completed. In an AI-first DevOps model (which, to be clear, is still immature), the IDE/terminal becomes the hub of developer activity, connecting directly to VCS, CI/CD, cloud, and AI agents. Waiting for the code to be pushed to the VCS simply takes too long and will slow things down. With this approach, the developer never has to leave the IDE/terminal for most workflow tasks, which enables version control and CI/CD pipelines to be abstracted away to a large extent. For example, the IDE can launch agents that will handle many of the tasks that historically needed to be executed manually within the VCS or CI/CD tool.

Exhibit 17
Cracking the Code: How AI Is Transforming Software Development
Center of Gravity Shifting to the IDE



Source: NorthCode, William Blair Equity Research

VCS Staying Power

To be clear, as stated earlier, the IDE being a system of interaction does not eliminate the need for core DevSecOps SORs like the VCS. This is because the VCS serves as a centralized, single source of truth for all code across an organization, performing critical tasks such as branching/merging of code, managing PRs, history tracking, collaboration across distributed teams, and compliance and audit trails. We have already seen early indicators of the multiplier effects of AI coding on the growth of source code repositories, with Microsoft noting on its first quarter fiscal 2026 earnings call that AI coding is contributing to significant growth in the amount of PRs and git repos being

generated on GitHub (with GitHub user growth hitting record levels—a new developer is added every second to the platform). Similarly, the artifact repository (e.g., JFrog Artifactory, GitHub Packages) is necessary as an immutable source of truth (and versioning mechanism) for the built binaries and containers that comprise a production application.

That said, IDE layer disruption does raise critical questions on the various ripple effects that we might see downstream in the DevOps toolchain, and which vendors are most vulnerable to such disruption. For example, we think standalone CI/CD tools are more vulnerable to disruption/replacement as CI/CD logic is scriptable and can therefore be integrated directly into the IDE. In fact, modern cloud development platforms such as Replit are increasingly integrating CI/CD capabilities directly inside the code editor. While this improves developer productivity, it also could commoditize CI/CD by shifting it from a visible, differentiated product into a functionally invisible infrastructure layer.

Over time, this may make it more challenging for vendors to monetize CI/CD directly, pushing them instead to capture value through broader platform offerings or adjacent services. This is why we see a vendor like CircleCI pivoting to become an "autonomous validation platform." While AI speeds up code generation, it also risks producing more errors, regressions, or unstable code, which requires validating AI-assisted commits in real time and detecting risky patterns, flaky tests, and breaking changes before they merge into the main branch (see next bullet).

Governance Functions Still Needed (More Than Ever)
While AI coding has great potential to dramatically improve developer productivity, it will also create new post-authoring code challenges around security, compliance, and code quality. AI-generated code can introduce subtle bugs, security vulnerabilities, and inefficiencies, with a small error liable to create outsized costs in production systems. More code created by AI means more potential vulnerabilities to manage, more compliance elements to maintain, more integration points to validate, and ultimately more operational risk. This should drive demand for code management, security, and governance tools that can scale with accelerated code creation and ensure consistent quality and compliance across large, distributed teams. Industry estimates suggest that 80% of software development work is done following code authoring, which suggests that management, orchestration, security, verification, and monitoring technology will be more vital than ever.

SDLC Pendulum Swinging From Platforms to Best of Breed
Software markets generally swing back and forth between best-of-breed tools and platforms, depending on the maturity of a given technology, budget pressure, and tool sprawl. For the past several years, we have argued that the DevOps market was trending toward tool consolidation/standardized platforms as customers were frustrated with fragmented toolsets, inefficient integrations across vendor products, and high costs. However, with the wave of innovation unleashed by AI coding, we believe the pendulum is swinging back toward best-of-breed tools as developers will want to choose the latest and greatest technologies and view platforms as constraining their freedom of choice and movement (locking them in). If we are right here, this trend has significant implications for the DevOps market, especially for vendors like GitHub and GitLab whose main pitch to customers has been the value of their consolidated platforms. The counterpoint argument—which vendors like Cursor and GitLab would likely posit—is that AI increases the value of shared context, not isolated tools, and that only platforms can aggregate context consistently, maintain long-lived state, and enforce policies uniformly across the SDLC. This may be where we are headed longer term, but we still expect a period of fragmentation and disruption in the near term as customers react to the onslaught of new tools and changing workflows.

Exhibit 18
Cracking the Code: How AI Is Transforming Software Development
Major Quotes on AI Coding

Source: William Blair Equity Research

Are Developer Jobs at Risk? Tackling the Million-Dollar Question

For several decades, computer science has been one of the most sought-after degrees, with graduating college students virtually guaranteed a job given the imbalance between software engineering supply and demand. However, the swelling productivity benefits that developers are gaining from AI tools have raised significant questions about the number of developers that might be needed going forward, with doomsayers believing that AI coding assistants will ultimately make many developer jobs obsolete and learning to code a fool's errand.

These fears have been exacerbated by a recent Stanford Digital Economy Lab study, which found that entry-level software engineer jobs have declined due to AI, with a 13% drop in employment for workers aged 22 to 25 in highly AI-exposed fields like software development since 2022, even after controlling for firm-level shocks. In contrast, employment for workers in less exposed fields and more experienced workers in the same occupations has remained stable or continued to grow. According to the study, this trend is primarily driven by AI's ability to automate tasks that were once entry level, such as basic coding, impacting the traditional career ladder and potentially requiring new pathways for junior talent. This study corroborates our research indicating that senior engineers are seeing the most productivity gains (in some cases 10-100x), while the value of junior engineers is increasingly being questioned.

The counterpoint here is that AI's multiplier effect on developer productivity could drive companies to hire more developers rather the fewer developers. Put simply, if we assume software "rules the world"—and there is a virtually infinite amount of software work that needs to be done—and

the ROI of an AI-fluent developer is orders of magnitude higher than a legacy developer, why would an organization not want to bring on more developers? Indeed, at the recent AWS re:Invent user conference, our conversations with industry executives revealed continued developer hiring across the companies we spoke with but with demand skewing heavily toward senior engineers and AI-native undergrads (personas that can drive the highest productivity improvements from emerging tools). Meanwhile, midlevel engineers that have been slow to upskill face a larger disintermediation risk, which may temper overall developer headcount growth in the near term.

Exhibit 19
Cracking the Code: How AI Is Transforming Software Development
AI in the Development Workflow*

***The survey question asked: Which parts of your development workflow are you currently integrating into AI or using AI tools to accomplish or plan to use AI to accomplish over the next 3 - 5 years? Please select one for each scenario.**

Source: Stack Overflow, William Blair Equity Research

Senior Versus Junior Engineers

Although AI coding could initially slow developer hiring, especially for junior roles, as companies strive to become more resource-efficient and take a step back to assess AI's productivity gains, our best guess at this juncture is that developer jobs will ultimately grow along with total software output (though seat growth will likely lag growth in software output). Our reasoning is that human skills and specialization—and fluency in programming languages and how to build software—will still be essential to architect, supervise, troubleshoot, secure, and provide context to an ever-expanding number of software projects and agents. A good analogy to AI coding assistants is spreadsheets, which increased rather than reduced finance jobs by making financial analysis more powerful and accessible to workers.

Put simply, as software creation accelerates, AI is more likely to augment than replace human developers as contextual understanding of software projects and products (and how to fix what might be broken) remains a critical input. For example, IBM CEO Arvind Krishna stated in May 2025 that the company was increasing hiring for "critical thinking" job functions including programming, sales, and marketing as IBM deploys and integrates AI within its internal business processes.

With respect to professional developers, our research suggests that AI is amplifying differences in roles, responsibilities, and expertise levels. As noted earlier, senior engineers are seeing the most productivity gains because they understand product context, domain, and system design; the sheer amount of code being produced will need to be managed, evaluated, and secured by experienced developers. In addition, if organizations want to maintain a pipeline of senior engineers, it would be foolish to stop hiring junior developers (especially those with AI fluency). Ultimately, in the age of AI, a developer's value will increasingly be tied to their ability to compose and decompose a problem—otherwise software builds and agentic workflows will get stuck at some point in the process.

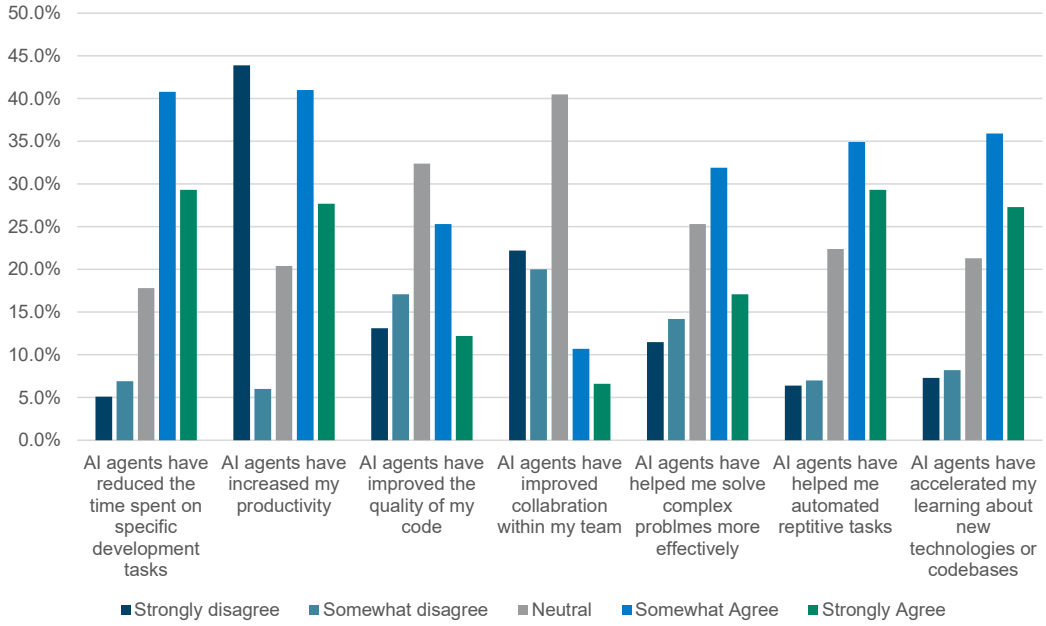
Exhibit 20
Cracking the Code: How AI Is Transforming Software Development
AI and humans in the future*

***The survey question asked: In the future, if AI can do most coding tasks, in which situations would you still want to ask another person for help? Select all that apply.**
Source: Stack Overflow, William Blair Equity Research

Looking ahead, we are increasingly seeing signs of role convergence, where developers are becoming not just editors of AI-generated code but also project managers and QA reviewers. This convergence also extends to other roles in the SDLC. For example, designers who once handed off static mockups to prototype engineers are now able to generate semi-functional prototypes themselves using vibe coding tools.

Instead of spending their days manually writing code—which was only around 20% of their jobs anyway—developers will become orchestrators of code (i.e., managers of coding bots and agents) given that their specialized knowledge and expertise will be crucial for verifying, debugging, and maintaining the output of AI coding tools. The combination of extensive developer experience and expertise with AI coding assistants can enable further automation of the coding process, lowering downtime and improving code quality and security—highlighting the synergies from human developers working alongside AI coding agents.

Exhibit 21
Cracking the Code: How AI Is Transforming Software Development
Impact of AI agents in Developer Workflow*



*** The question asked: To what extent do you agree with the following statements regarding the impact of AI agents on your work as a developer?**

Source: Stack Overflow, William Blair Equity Research

Seat-Based Pricing Models

Fears about developer headcount have left investors questioning the long-term viability of traditional seat-based pricing models, which have historically provided predictable, sticky revenue streams for DevOps and collaboration vendors. If the number of developers grows more slowly—because each developer can produce exponentially more output—total seat counts may stagnate, even as overall software production accelerates.

Notwithstanding the debate about future developer headcount, vendors are beginning to augment seat-based pricing models with usage-based pricing that factors in some measure of consumption units like tokens (computing/storage resources), API calls, and/or agents. We have seen this in the AI coding space with vendors like Cursor, which adjusted its flat per-seat pricing model in mid-2025 to a usage-based/compute-credit model.

Meanwhile, incumbent DevOps vendors like GitLab are introducing hybrid pricing, combining traditional seat-based pricing with usage-based billing. The company's recently launched Duo Agent Platform will allow GitLab to scale its pricing (in the form of on-demand or precommitted credits for an enterprise account) along with the number of agents in the platform that the engineer will spin up to complete various tasks across the SDLC. We expect other vendors to follow suit with a similar hybrid approach that factors in some element of consumption on top of seat-based pricing.

AI's Impact on Incumbent DevSecOps Vendors

As in past major technology platform shifts, speed, adaptability, and the ability to compound network and data advantages will determine which incumbent vendors remain relevant and which risk being left behind. This is no different in the DevSecOps market, with the AI freight train causing every vendor to assess its strategy, product roadmap, and competitive position.

Below, we provide our high-level views on how AI is likely to impact the major public DevSecOps vendors, with a rank ordering based on the level of potential disruption risk (from most risk to least risk).

1. **GitLab** (disruption risk: medium)
   Our view that the software development pendulum is swinging back toward best-of-breed tools from platforms (as developers want to choose the latest and greatest AI technologies) raises fundamental questions on GitLab's strategy, which has been predicated on the value of its consolidated DevSecOps platform. On the coding front, GitLab has been a laggard with its Duo offering, which has enabled competitor tools to increasingly become the primary interaction layer in the development workflow (superseding the Git repository). Moreover, GitHub's release of Agent HQ could increase competitive pressure on GitLab as customers coalesce around the tight coupling of VS Code with Agent HQ. On the CI/CD front, if agents within the IDE can write tests, diagnose pipeline failures, and even restructure CI configurations, the value of a sophisticated CI tool like GitLab CI may be diminished.

   On the flip side, if SORs are becoming more crucial than ever in this new era of software development, GitLab as a strong No. 2 player in the VCS/Git repo duopoly should benefit from higher volumes of code generation and PRs. In addition, with GitHub becoming more tightly integrated with Microsoft Azure, other hyperscalers may prefer to recommend GitLab to customers as a more cloud-neutral option. Lastly, as mentioned above, GitLab's traditional seat-based pricing model is already evolving to include usage-based pricing, which eases to some extent the risk of AI reducing developer headcount or causing it to grow more slowly than what we have seen historically.

2. **Atlassian** (disruption risk: low to medium)
   Atlassian as a SOR for project planning/issue tracking should continue to be a grounding element within the SDLC. Developer surveys consistently show that Jira is one of the stickiest tools within the DevSecOps market, based on user familiarity, high switching costs, and cross-team dependence. That said, while Atlassian benefits from significant developer inertia and organizational lock-in, AI threatens to shake up the agile software development process and make traditional (manual) project management less relevant over time, especially if an increasing amount of software development is done autonomously. In addition, despite our long-term positive view on developer headcount, Atlassian's seat-based pricing model likely needs to evolve to include some measure of usage or outcome-based pricing.

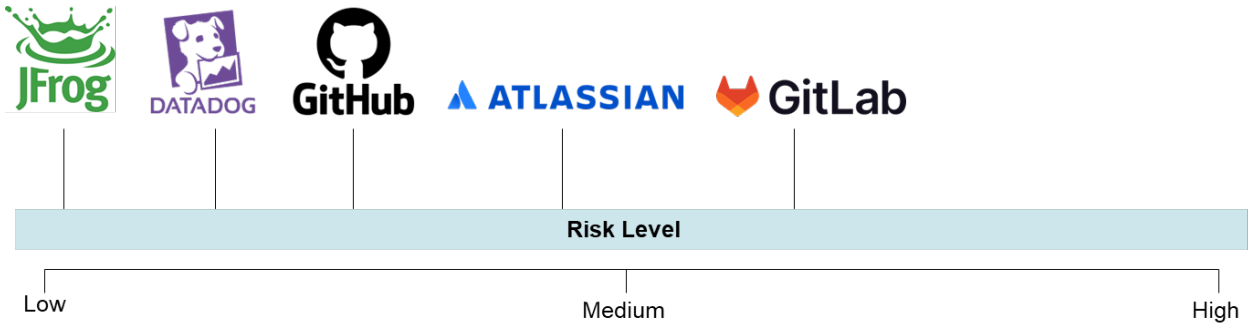
3. **GitHub/Microsoft** (disruption risk: low to medium)
   While AI also threatens GitHub's seat-based pricing model and core historical value proposition of DevSecOps integration and workflow consolidation—as the market pendulum swings to best-of-breed tooling—these risks are mitigated by GitHub's deep integrations with the Microsoft ecosystem and broad enterprise reach and distribution. In addition, while Agent HQ could be seen as a defensive move to counteract the waning popularity of GitHub Copilot, we believe it will resonate with enterprises looking to centrally manage (and secure) the plethora of AI coding assistants that have emerged in recent years.

Elsewhere, we have already seen early indicators of the multiplier effects of AI coding on the growth of source code repositories, with Microsoft noting on its first quarter fiscal 2026 earnings call that AI coding is contributing to significant growth in the amount of PRs and git repos being generated on GitHub (with GitHub user growth hitting record levels—a new developer is added every *second t*o the platform).

4. **Datadog** (disruption risk: low)
   Our thesis that SORs will be more crucial than ever in this new era of software development should bode well for Datadog, which is the clear leader in monitoring and observability of applications in production (runtime behavior). Enterprises need an authoritative source of truth for how an application is behaving in production, along with a feedback loop to development teams to fix issues as they arise. In addition, Datadog should be more insulated from AI-related disruption as its business is tied to growth in overall software volumes and related infrastructure needs rather than developer seats/headcount. At the same time, the observability space is seeing increasing competition from innovative start-ups such as Grafana, Chronosphere (to be acquired by Palo Alto Networks for $3.35 billion), and up-and-coming SRE (site reliability engineering) agents.

5. **JFrog** (disruption risk: low)
   As a foundational SOR for software releases—and the clear category leader in this space—JFrog Artifactory will continue to be critical as an immutable source of truth (and versioning mechanism) for the built binaries and containers that comprise a production application (and the explosion in the creation of these binaries fueled by AI). In addition, the company's expansion into security and governance only increases its relevance as a core SOR for securing the software supply chain: more code created by AI means more potential vulnerabilities to manage, more compliance elements to maintain, more integration points to validate, and ultimately more operational risk. Lastly, we believe JFrog is more insulated from AI-related disruption as its business is tied to growth in overall software volumes (more source code created=more binaries) and related infrastructure needs rather than developer seats/headcount.

Exhibit 22
Cracking the Code: How AI Is Transforming Software Development
Risk of Disruption for Covered Vendors from AI



Source: William Blair Equity Research

AI's Impact on Infrastructure Software Ecosystem

With AI dramatically accelerating software development cycles, we expect to see an explosion in the volume of code, applications, and data created over the next decade. Higher application/data volumes will ultimately ripple across the infrastructure software ecosystem, creating secular demand tailwinds for numerous subsegments (though the impact timeline is difficult to predict). Infrastructure subsegments we expect to see medium- to long-term benefits from the infusion of AI in software development include:

- **Databases** – Every application needs a database, so it follows that a rise in application creation will boost the database market, despite its hypercompetitive nature. In addition, more software creation will lead to more data creation, which will benefit consumption-based database-as-a-service (DBaaS) offerings. Key beneficiaries here include MongoDB, hyperscaler databases, and start-ups like Neon (now owned by Databricks), Supabase (private), and Neo4j (private).

- **Data Security/Governance** – More software being created means more data to manage and secure. Key beneficiaries here include vendors like Varonis, Cyera (private), Microsoft (Purview), Rubrik, Commvault, Veeam/Securiti AI (private), and BigID (private).

- **Data Protection/Cyber-resilience** – Higher software volumes will lead to more data creation, which in turn will drive demand for data protection infrastructure. Key beneficiaries here include Rubrik, Commvault, Veeam (private), and Cohesity (private).

- **Storage** – Higher application volumes and usage will lead to more data generation, which needs to be stored somewhere. Key beneficiaries include hyperscalers (for cloud storage); scale-out software/system vendors like VAST Data (private), Weka (private), and DDN (private); and enterprise-focused players like Pure Storage, Nutanix, NetApp, Dell, IBM, and HPE.

- **Data Observability** – Higher volumes of application creation will drive demand for monitoring and observability tools that provide essential health checks on applications in production. Key beneficiaries here include Datadog, Dynatrace, Palo Alto (Chronosphere), Grafana (private), Cribl (private), New Relic (private), and Sentry (private).

AI Forcing All Software Companies to Reexamine Competitive Moats—but Not in the Way Many Fear

With AI rapidly accelerating development cycles, software businesses must reassess the durability of their competitive moats and ensure that these moats will remain relevant in this AI era. Companies whose differentiation rests primarily on narrow technical functionality or single-feature point solutions are most vulnerable, as these capabilities can increasingly be replicated at very low marginal cost using AI-assisted development.

However, we would argue that pure technical moats in software have been eroding for over 10-15 years, well before generative AI and AI coding assistants became mainstream. Cloud infrastructure, APIs, microservices, containerization, open source, and low-code/no-code tooling have already democratized the technical building of software. While AI meaningfully accelerates this trend, it does not fundamentally change it, especially when software companies are competing at the highest levels. If the only moat software companies had was technical, then tech giants like Microsoft, Oracle, or SAP would have long ago just copied all software apps with their seemingly endless developer resources, and we would not have the breadth of software companies that we have today.

The most successful software companies have over time expanded their moats beyond the technical. At one point or another, every software company has been asked the question "why can't someone raise money, hire developers, and build this to compete with you". The answer from successful companies has never been that fundraising will be difficult or replicating what they have built is impossible. Instead, defensibility for most software companies has evolved to focus on platform breadth, distribution advantages, deep domain expertise, vertical specialization, unique data that informs product roadmap, and ecosystem/network effects. In addition, for scaled software companies especially, we believe determining *what to build next* is fundamentally more important—and more challenging—than the technical act of building it.

Salesforce, for example, long ago moved beyond basic CRM functionality to become a horizontal platform spanning sales, service, marketing, data, and now AI. This was all supported by a massive marketplace of third-party integrations, a broad partner ecosystem, and broad customer data, all deeply embedded in customer workflows. This allowed Salesforce to build a business with $40 billion in revenue and a $240 billion market cap.

Today (and indeed for many years), any good team of engineers can effectively "build" a CRM system leveraging an off-the-shelf database. But this will not be the same as buying one from Salesforce, which has learned how to build and iterate a best-of-breed CRM system based on input from tens of thousands of scaled customers. This makes it unlikely that an airline, an insurance company, or a manufacturing business would stray from its core competency to build a CRM system and continue to maintain and evolve it over time.

ServiceNow followed a similar path, evolving from a point solution focused on IT ticketing to one of the broadest platforms in software. ServiceNow's moat is less about any single workflow and more about owning the enterprise service layer across IT, HR, customer service, operations, and many more functional areas. This breadth creates high switching costs for ServiceNow customers, with stickiness based on process standardization, governance, and scale rather than UI or code complexity.

To partially illustrate this point, we go back to a framework we first wrote about in May 2023, highlighting the opportunity for incumbent software vendors (see exhibit 23 below). We believe this largely holds true today as the fundamental advantage for incumbents in the AI era is based on:

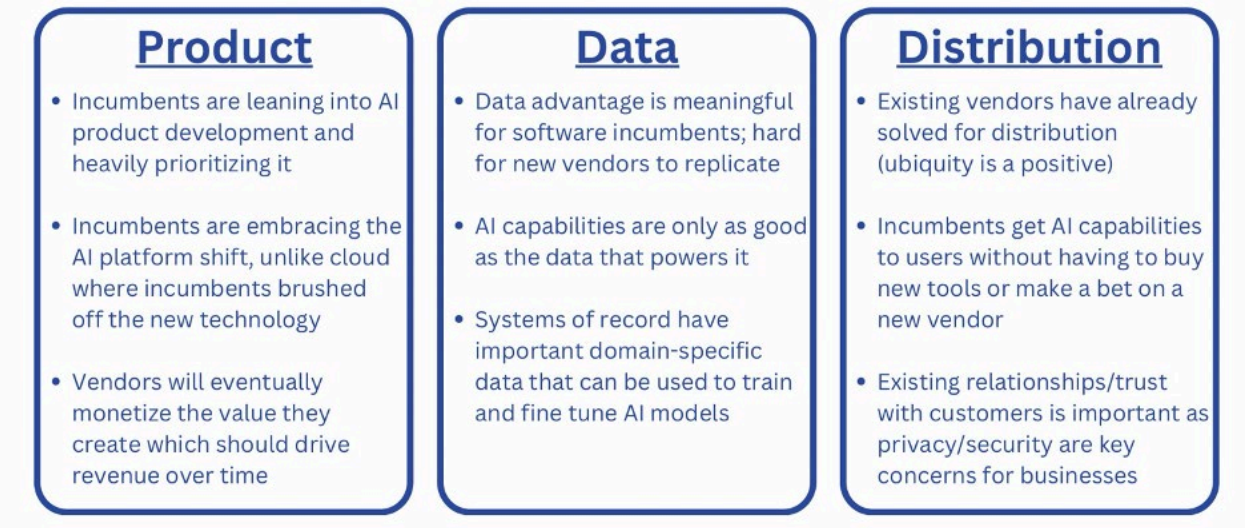
1. Their ability to innovate and launch new products, especially those with AI capabilities;
2. The data they have collected over the years, which new entrants don't have access to; and
3. Their distribution advantage, especially in cases where enterprises view them as trusted vendors

To be clear, we are not saying that there are zero opportunities for new AI startups in the market—there clearly are plenty. But for upstarts that are simply creating an "AI version" of a CRM or ITSM system, we think the path ahead will be difficult and they will find it challenging to create an edge over fast-moving incumbents. We think the better opportunity for AI startups will be to address new workflows that didn't exist in the pre-AI software era.

In many cases, we believe AI strengthens existing moats rather than undermines them. Incumbents with large installed bases, proprietary datasets, and embedded workflows are uniquely positioned to deploy AI in ways that are difficult for new entrants to replicate. Intuit is a useful example: its competitive advantage is not that it can build tax or accounting software faster than others, but that it has decades of financial data, deep regulatory expertise, and a trusted brand/relationships with SMBs. With AI, Intuit layers new use-cases and AI agents that automate some of the human workflow on top of an already sticky platform.

To sum up, our assertion is that while AI does not eliminate software moats, it raises the bar for what constitutes a durable one. Long-term winners will be defined not by who can write code fastest, but by who can stay relevant as the underlying tech landscape changes—with platform breadth, data scale and quality, human-AI workflow integration, distribution, trust, and the right to expand into new use-cases as arbiters of future success.

Exhibit 23
Cracking the Code: How AI Is Transforming Software Development
Why Incumbents are Well Positioned to Benefit from GenAI



Source: William Blair Equity Research

Conclusion

AI is fundamentally reshaping the software development landscape, accelerating release cycles and lowering barriers to entry while introducing new challenges around security, compliance, and code quality. The rise of AI coding tools is driving significant productivity gains, especially for experienced developers—and substantially expanding the TAM for developer tools—but also raising important questions about the future roles and skills required in the industry.

As the competitive landscape evolves, incumbent DevSecOps vendors must adapt quickly, reassessing their business models and focusing on defensible moats. While AI will not eliminate the need for human developers, it will redefine their responsibilities, emphasizing the importance of oversight, critical thinking, and holistic product design. Ultimately, those vendors who embrace AI's potential while navigating its risks will be best positioned to thrive in this next era of software innovation.

The prices of the common stock of other public companies mentioned in this report follow:

Alibaba Group Holding Limited	$26,720.00
Alphabet Inc. (Outperform)	$317.32
Atlassian Corporation Plc. (Outperform)	$153.44
Commvault Systems, Inc. (Outperform)	$125.58
Datadog, Inc. (Outperform)	$133.64
Dell Technologies Inc.	$124.01
Dynatrace, Inc. (Outperform)	$42.64
GitLab, Inc. (Outperform)	$37.51
Hewlett Packard Enterprise Company	$24.13
International Business Machines Corporation	$294.97
Intuit Inc. (Outperform)	$633.84
JFrog, Ltd. (Outperform)	$58.87
Meta Platforms, Inc. (Outperform)	$658.79
Microsoft Corporation (Outperform)	$472.85
MongoDB, Inc. (Outperform)	$420.82
NetApp, Inc. (Market Perform)	$105.08
Nutanix, Inc. (Outperform)	$50.56
Nvidia Corporation (Outperform)	$188.12
Oracle Corporation (Outperform)	$192.64
Palo Alto Networks, Inc. (Outperform)	$182.12
Pure Storage, Inc. (Outperform)	$69.66
Rubrik, Inc. (Outperform)	$73.50
Salesforce, Inc. (Outperform)	$256.26
SAP SE	$241.33
ServiceNow, Inc. (Outperform)	$147.60
Varonis Systems, Inc. (Outperform)	$32.86

IMPORTANT DISCLOSURES

One or more of the research analysts responsible for covering Alphabet, Inc. mentioned in this report owns shares of the security.

William Blair or an affiliate was a manager or co-manager of a public offering of equity securities for Rubrik, Inc. within the prior 12 months.

William Blair or an affiliate beneficially own or control (either directly or through its managed accounts) 1% or more of the equity securities of Confluent, Inc. and Pure Storage, Inc. as of the end of the month ending not more than 40 days from the date herein.

William Blair or an affiliate is a market maker in the security of JFrog, Ltd., Microsoft Corporation, GitLab, Inc., Arista Networks, Inc., AvePoint, Inc., Backblaze, Inc., Box, Inc., Cisco Systems, Inc., Commvault Systems, Inc., Confluent, Inc., DigitalOcean Holdings, Inc., Dropbox, Inc., F5, Inc., MongoDB, Inc., N-able, Inc., NetApp, Inc., Nutanix, Inc., Oracle Corporation, Pure Storage, Inc., Rubrik, Inc., Snowflake Inc., Varonis Systems, Inc., Datadog, Inc., Atlassian Corporation Plc and Alphabet, Inc.

William Blair or an affiliate expects to receive or intends to seek compensation for investment banking services from JFrog, Ltd., Microsoft Corporation, GitLab, Inc., Arista Networks, Inc., AvePoint, Inc., Backblaze, Inc., Box, Inc., Cisco Systems, Inc., Commvault Systems, Inc., Confluent, Inc., DigitalOcean Holdings, Inc., Dropbox, Inc., F5, Inc., MongoDB, Inc., N-able, Inc., NetApp, Inc., Nutanix, Inc., Oracle Corporation, Pure Storage, Inc., Rubrik, Inc., Snowflake Inc., Varonis Systems, Inc., Datadog, Inc., Atlassian Corporation Plc and Alphabet, Inc. or an affiliate within the next three months.

William Blair or an affiliate received compensation for investment banking services or non-investment-banking services from Rubrik, Inc. within the last 12 months. Rubrik, Inc. is or was, within the last 12 months, an investment banking client of William Blair & Company and/or one or more of its affiliates.

Officers and employees of William Blair or its affiliates (other than research analysts) may have a financial interest in the securities of JFrog, Ltd., Microsoft Corporation, GitLab, Inc., Arista Networks, Inc., AvePoint, Inc., Backblaze, Inc., Box, Inc., Cisco Systems, Inc., Commvault Systems, Inc., Confluent, Inc., DigitalOcean Holdings, Inc., Dropbox, Inc., F5, Inc., MongoDB, Inc., N-able, Inc., NetApp, Inc., Nutanix, Inc., Oracle Corporation, Pure Storage, Inc., Rubrik, Inc., Snowflake Inc., Varonis Systems, Inc., Datadog, Inc., Atlassian Corporation Plc and Alphabet, Inc.

This report is available in electronic form to registered users via R*Docs™ at https://williamblairlibrary.bluematrix.com or www.williamblair.com.

Please contact us at +1 312 236 1600 or consult https://www.williamblair.com/equity-research/coverage for all disclosures.

Jason Ader, Jake Roberge, Ralph Schackart and Arjun Bhatia attests that 1) all of the views expressed in this research report accurately reflect his/her personal views about any and all of the securities and companies covered by this report, and 2) no part of his/her compensation was, is, or will be related, directly or indirectly, to the specific recommendations or views expressed by him/her in this report. We seek to update our research as appropriate. Other than certain periodical industry reports, the majority of reports are published at irregular intervals as deemed appropriate by the research analyst.

DOW JONES: 48996.10
S&P 500: 6920.93
NASDAQ: 23584.30

Additional information is available upon request.

Current Rating Distribution (as of January 9, 2026):

Coverage Universe	Percent	Inv. Banking Relationships *	Percent
Outperform (Buy)	71	Outperform (Buy)	11
Market Perform (Hold)	28	Market Perform (Hold)	3
Underperform (Sell)	1	Underperform (Sell)	0

*Percentage of companies in each rating category that are investment banking clients, defined as companies for which William Blair has received compensation for investment banking services within the past 12 months.

The compensation of the research analyst is based on a variety of factors, including performance of his or her stock recommendations; contributions to all of the firm's departments, including asset management, corporate finance, institutional sales, and retail brokerage; firm profitability; and competitive factors.

OTHER IMPORTANT DISCLOSURES

Stock ratings and valuation methodologies: William Blair & Company, L.L.C. uses a three-point system to rate stocks. Individual ratings reflect the expected performance of the stock relative to the broader market (generally the S&P 500, unless otherwise indicated) over the next 12 months. The assessment of expected performance is a function of near-, intermediate-, and long-term company fundamentals, industry outlook, confidence in earnings estimates, valuation (and our valuation methodology), and other factors. Outperform (O) - stock expected to outperform the broader market over the next 12 months; Market Perform (M) - stock expected to perform approximately in line with the broader market over the next 12 months; Underperform (U) - stock expected to underperform the broader market over the next 12 months; not rated (NR) - the stock is not currently rated. The valuation methodologies include (but are not limited to) price-to-earnings multiple (P/E), relative P/E (compared with the relevant market), P/E-to-growth-rate (PEG) ratio, market capitalization/revenue multiple, enterprise value/EBITDA ratio, discounted cash flow, and others. Stock ratings and valuation methodologies should not be used or relied upon as investment advice. Past performance is not necessarily a guide to future performance.

The ratings and valuation methodologies reflect the opinion of the individual analyst and are subject to change at any time.

Our salespeople, traders, and other professionals may provide oral or written market commentary, short-term trade ideas, or trading strategies-to our clients, prospective clients, and our trading desks-that are contrary to opinions expressed in this research report. Certain outstanding research reports may contain discussions or investment opinions relating to securities, financial instruments and/or issuers that are no longer current. Investing in securities involves risks. This report does not contain all the material information necessary for an investment decision. Always refer to the most recent report on a company or issuer. Our asset management and trading desks may make investment decisions that are inconsistent with recommendations or views expressed in this report. We will from time to time have long or short positions in, act as principal in, and buy or sell the securities referred to in this report. Our research is disseminated primarily electronically, and in some instances in printed form. Research is simultaneously available to all clients. This research report is for our clients only. No part of this material may be copied or duplicated in any form by any means or redistributed without the prior written consent of William Blair & Company, L.L.C.

This is not in any sense an offer or solicitation for the purchase or sale of a security or financial instrument.

The factual statements herein have been taken from sources we believe to be reliable, but such statements are made without any representation as to accuracy or completeness or otherwise, except with respect to any disclosures relative to William Blair or its research analysts. Opinions expressed are our own unless otherwise stated and are subject to change without notice. Prices shown are approximate.

This report or any portion hereof may not be copied, reprinted, sold, or redistributed or disclosed by the recipient to any third party, by content scraping or extraction, automated processing, or any other form or means, without the prior written consent of William Blair. Any unauthorized use is prohibited.

If the recipient received this research report pursuant to terms of service for, or a contract with William Blair for, the provision of research services for a separate fee, and in connection with the delivery of such research services we may be deemed to be acting as an investment adviser, then such investment adviser status relates, if at all, only to the recipient with whom we have contracted directly and does not extend beyond the delivery of this report (unless otherwise agreed specifically in writing). If such recipient uses these research services in connection with the sale or purchase of a security referred to herein, William Blair may act as principal for our own account or as riskless principal or agent for another party. William Blair is and continues to act solely as a broker-dealer in connection with the execution of any transactions, including transactions in any securities referred to herein.

For important disclosures, please visit our website at williamblair.com.

This material is distributed in the United Kingdom and the European Economic Area (EEA) by William Blair International, Ltd., authorised and regulated by the Financial Conduct Authority (FCA). William Blair International, Limited is a limited liability company registered in England and Wales with company number 03619027. This material is only directed and issued to persons regarded as Professional investors or equivalent in their home jurisdiction, or persons falling within articles 19 (5), 38, 47, and 49 of the Financial Services and Markets Act of 2000 (Financial Promotion) Order 2005 (all such persons being referred to as "relevant persons"). This document must not be acted on or relied on by persons who are not "relevant persons."

This report is being furnished in Brazil on a confidential basis and is addressed to the addressee personally, and for its sole benefit. This does not constitute an offer or solicitation for the purchase or sale of a security by any means that would constitute a public offering in Brazil under the regulations of the Brazilian Securities and Exchange Commission (*Comissão de Valores Mobiliários*) or an unauthorized distribution under Brazilian laws and regulations. The securities are authorized for trading on non-Brazilian securities markets, and this report and all the information herein is intended solely for professional investors (as defined by the applicable Brazilian regulation) who may only acquire these securities through a non-Brazilian account, with settlement outside Brazil in a non-Brazilian currency.

Any statements in this report that are attributable to IDC Research, Inc. ("IDC") represent William Blair's interpretation of data, research opinion or viewpoints published as part of a syndicated subscription service by IDC and have not been reviewed by IDC. IDC's research is current as of the date IDC published it, not the date that William Blair's reports are published. Further, IDC's research contains IDC's opinion, not representations of fact, and are subject to change without notice.

William Blair & Company, L.L.C. licenses and applies the SASB Materiality Map® and SICSTM in our work.

William Blair

Equity Research Directory

John Kreger, Partner Director of Research +1 312 364 8612
Kyle Harris, CFA, Partner Operations Manager +1 312 364 8230

Scott Hansen, Partner Associate Director of Research +1 212 245 6526

CONSUMER

Sharon Zackfia, CFA, Partner +1 312 364 5386
Group Head–Consumer
Lifestyle and Leisure Brands, Restaurants, Automotive/E-commerce

Jon Andersen, CFA, Partner +1 312 364 8697
Consumer Products

Phillip Blee, CPA +1 312 801 7874
Home and Outdoor, Automotive Parts and Services, Discount and Convenience

Dylan Carden +1 312 801 7857
E-commerce, Specialty Retail

ECONOMICS

Richard de Chazal, CFA +44 20 7868 4489

ENERGY AND POWER TECHNOLOGIES

Jed Dorsheimer +1 617 235 7555
Group Head–Energy and Power Technologies
Generation, Efficiency, Storage

Neal Dingmann +1 312 801 7835
Oil and Gas, Rare Earth

Tim Mulrooney, Partner +1 312 364 8123
Energy and Environmental Services

FINANCIAL SERVICES AND TECHNOLOGY

Adam Klauber, CFA, Partner +1 312 364 8232
Group Head–Financial Services and Technology
Financial Analytic Service Providers, Insurance Brokers, Property & Casualty Insurance

Andrew W. Jeffrey, CFA +1 415 796 6896
Fintech

Cristopher Kennedy, CFA +1 312 364 8596
Fintech, Specialty Finance

Jeff Schmitt +1 312 364 8106
Wealthtech, Wealth Management, Capital Markets Technology

GLOBAL SERVICES

Tim Mulrooney, Partner +1 312 364 8123
Group Head–Global Services
Commercial and Residential Services

Andrew Nicholas, CPA +1 312 364 8689
Consulting, HR Technology, Information Services

Trevor Romeo, CFA +1 312 801 7854
Staffing, Waste and Recycling

HEALTHCARE

Biotechnology

Matt Phipps, Ph.D., Partner +1 312 364 8602
Group Head–Biotechnology

Sami Corwin, Ph.D. +1 312 801 7783

Lachlan Hanbury-Brown +1 312 364 8125

Andy T. Hsieh, Ph.D., Partner +1 312 364 5051

Myles R. Minter, Ph.D., Partner +1 617 235 7534

Healthcare Technology and Services

Ryan S. Daniels, CFA, Partner +1 312 364 8418
Group Head–Healthcare Technology and Services
Healthcare Technology, Healthcare Services

Brandon Vazquez, CFA +1 212 237 2776
Dental, Animal Health, Medical Technology

Life Sciences

Matt Larew, Partner +1 312 801 7795
Life Science Tools, Bioprocessing, Healthcare Delivery

Andrew F. Brackmann, CFA +1 312 364 8776
Diagnostics

Max Smock, CFA +1 312 364 8336
Pharmaceutical Outsourcing and Services

INDUSTRIALS

Brian Drab, CFA, Partner +1 312 364 8280
Co-Group Head–Industrials
Advanced Manufacturing, Industrial Technology

Ryan Merkel, CFA , Partner +1 312 364 8603
Co-Group Head–Industrials
Building Products, Specialty Distribution

Louie DiPalma, CFA +1 312 364 5437
Aerospace and Defense, Smart Infrastructure

Ross Sparenblek +1 312 364 8361
Diversified Industrials, Robotics, and Automation

TECHNOLOGY, MEDIA, AND COMMUNICATIONS

Jason Ader, CFA, Partner +1 617 235 7519
Co-Group Head–Technology, Media, and Communications
Infrastructure Software

Arjun Bhatia, Partner +1 312 364 5696
Co-Group Head–Technology, Media, and Communications
Software

Dylan Becker, CFA +1 312 364 8938
Software

Louie DiPalma, CFA +1 312 364 5437
Government Technology

Jonathan Ho, Partner +1 312 364 8276
Cybersecurity, Security Technology

Sebastien Naji +1 212 245 6508
Infrastructure Software, Semiconductor and Infrastructure Systems

Maggie Nolan, CPA, Partner +1 312 364 5090
IT Services

Jake Roberge +1 312 364 8056
Software

Ralph Schackart III, CFA, Partner +1 312 364 8753
Internet and Digital Media

Stephen Sheldon, CFA, CPA, Partner +1 312 364 5167
Vertical Technology – Real Estate, Education, Restaurant/Hospitality

EDITORIAL AND SUPERVISORY ANALYSTS

Steve Goldsmith, Head Editor and SA +1 312 364 8540
Katie Anderson, Editor and SA +44 20 7868 4451
Audrey Majors, Editor and SA +1 312 364 8992
Beth Pekol Porto, Editor and SA +1 312 364 8924
Lisa Zurcher, Editor and SA +44 20 7868 4549