# Shaping the Future of Developers

**I. ABSTRACT**

There is a profound AI-driven disruption happening in the world of software development. *28 million developers* across the globe are facing unprecedented changes in their profession, and most of them are unsure of how to navigate this evolving landscape (1). This is a once-in-a-generation opportunity to propel these developers into the future and save *hundreds of thousands of developer-years* globally. Yet, our current efforts are fragmented and lack a cohesive (2), unified vision. Our initiatives are siloed, focused narrowly on product features, with no strategic framework to guide the developers to be future ready and AI-first. To seize this opportunity, this paper grounds itself in an exhaustive analysis of current SDLC and Agile practices to unearth all the whitespaces (critical gaps, frictions, and inefficiencies) that are inherent in them. With a point of view on how AI can fix those whitespaces, the paper evolves a novel, AI-native software development method, code-named as **AI-DLC (AI-Driven Development Lifecycle)***.*

AI-DLC consolidates all the fragmented ways of using AI in development into a unified *Developer-AI Symbiosis*. Across all developer journeys, AI-DLC recommends a common approach of starting with AI to decompose the tasks into fine-grained hierarchical subtasks. Each level of decomposition progresses from conceptual to increasingly technical depths (conceptual to logical to physical). At each stage, AI provides solution options and outline the trade-offs. The developers analyze the suggestions and make strategic implementation decisions with minimal or zero blindside at both macro and micro levels. This paper further outlines the requirements for **a reimagined AI-first developer interface** designed to intuitively support this developer-AI collaboration. With AI-DLC, the future persona of developers will evolve into an 'AI Native Trailblazer' who achieves unprecedented productivity, iterates rapidly at the speed of judgment, translates business ambitions directly into code rather than merely features into code, and transcends today’s specialization silos effortlessly. As next steps, the paper proposes strategies to create hyper-scaling field mechanisms that drives a self-sustaining flywheel effect to rapidly reshape the global developer workforce using this vision.

**II. METHODOLOGY**

This paper assumes that we are currently in the **AI-Assisted** era, where AI enhances specific, fine-grained tasks in development, such as code completion, bug detection, and test generation. In the near future, we are stepping into the **AI-Drive**n era, where AI takes broader responsibility in planning, task decomposition, and moving closer to translating business ambitions into code. The **AI-Managed** is the next era where AI autonomously develops and maintains software without humans in the loop. This paper restricts its scope to the near-future **AI-Driven era**. AI-DLC, the proposed new interface and the perspectives on the evolution of developer persona will serve as end-state visions to move the developer population from the AI-Assisted to the AI-Driven era.

a) We will start with a comprehensive analysis of the Software Development Lifecycle (SDLC) and Agile methods. Our examination will focus on core developer journeys, including new development initiation, feature addition, library upgrades, testing, and refactoring. Through this analysis, we aim to identify critical "whitespaces" - the gaps, friction points, and inefficiencies within these established methods. These whitespaces often cause average developers to unintentionally create sub-optimal systems and compromise engineering outcomes in areas such as productivity, quality, and innovation.

b) For each identified whitespace, we will evaluate which ones can be addressed using current AI capabilities. For example, our existing capabilities can support AI-assisted functional decomposition and AI-driven unit testing. For remaining gaps, we will explore how future advanced AI techniques could be applied, such as developing domain-specific agents for DAST and automated observability emitter injection.

c) Using the mapping of AI capabilities to the whitespaces, we will design a transformative new method - the AI-Driven Development Lifecycle (AI-DLC). This method aims to fundamentally shift the development paradigm toward future-ready, AI-native engineering that fully leverages AI's potential. We will also reimagine the developer tooling that will go hand-in-hand with AI-DLC and develop a vision of the future developer persona who will employ AI-DLC in their daily work, contrasting with traditional development methods.

**III. ANALYSIS AND MAPPING**

This section dives into the steps a and b from the methodology above.

A. BACKGROUND

Every software development method aims to transform business requirements into functional code systematically. Traditional SDLC methods prioritize quality over agility, recommending a waterfall approach where each stage, particularly design, undergoes rigorous verification before progression—known as the *Big Design Upfront* approach. In contrast, Agile methodologies emphasize speed and flexibility by breaking tasks into smaller, iterative cycles. This has often resulted in the unintended *No Design Upfront* approach. Modern Agile techniques, such as Domain-Driven Design (DDD) and Behaviour-Driven Development (BDD), balance these extremes. They incorporate sufficient design rigor to maintain clarity without impeding development speed or iterative capabilities. These modern approaches, utilizing techniques like Event Storming (DDD) and User-Centric Scenarios (BDD), enable incremental design that ensures thorough consideration of essential system aspects without over-planning. But despite these improvements, several significant whitespaces remain in current software development practices as below.

B. REQUIREMENTS TO DOMAIN DESIGN

This journey involves developers translating business stakeholders' mental models into domain components while maintaining integrity, completeness, and accuracy. Despite robust methodologies like DDD and BDD designed for effective domain translation into software design, practical implementation remains challenging. Our DevTx engagements over the past two years revealed several cognitive challenges:

Methodology Adoption Barriers: Event Storming (DDD) and user-centric scenarios (BDD) require substantial learning investment. Developers, especially those lacking domain modelling experience, struggle to master best practices and facilitation skills to carry out the design process independently. This often results in suboptimal design decisions regarding high-cohesion and loose-coupling principles. For example, heavily interdependent business logic gets dispersed across multiple components, creating inconsistent and maintenance-heavy systems. These brittle implementations lead to significant downstream issues, with developers spending over 70% of their time addressing production maintenance rather than developing new features or improving systems(3).

Current State of AI: While AI can generate domain components for established sectors such as eCommerce, Supply Chain Management, and Travel Booking, it lacks consistency. Generated models frequently exhibit variation across interactions, with inconsistent terminologies and relationship definitions (4). Experimental results indicate that these models lack comprehensiveness (4). Furthermore, the limitations of current AI interfaces (ex. Q chat in VSCode), which rely primarily on linear, chat-based interactions, restrict effective domain exploration and iterative refinement. After just three degrees of interactions, the content in the chat window becomes overwhelming, making it extremely difficult to manually scroll and navigate back and forth to locate and refine the previous steps in the plan.

Potential Future AI Disruption: Domain modelling is largely redundant, as common business domains had been repeatedly designed across organizations worldwide, countless times. These models primarily consist of standard patterns and boilerplate components, with variations only in organization-specific business policies, workflow routing and rules. A hotel inventory subdomain, when architected using established design best practices, will naturally evolve toward structurally similar bounded contexts and tactical components, irrespective of the designer. This repetitive nature presents a transformative opportunity. We need purpose-built "Domain-Specific AI Agents" that efficiently generate consistent, comprehensive domain models with documentation while allowing customization for organization-specific rules and changes in the workflow. This automation will save hundreds of thousands of developer-months globally by shifting from a manual, expertise-dependent process to an automated, consistent approach.

C. DOMAIN DESIGN TO LOGICAL TO PHYSICAL ARCHITECTURES

This journey transforms a Domain Model into implementation-ready low-level design by applying design patterns and well-architected principles to meet functional and non-functional requirements.

Lack of Methods: The software industry lacks a mature method for converting bounded contexts from Domain Models or BDD scenarios into adaptable Cloud Native Architectures. An effective method should enable team autonomy for independent innovation within established guardrails while simplifying security, governance, and cost management. By aligning cloud resources with bounded contexts, organizations should be able to track usage and costs transparently for both IT and business stakeholders. While DevTx has pioneered an approach (Day-1 Systems Design Method) to address this challenge, it remains in early stages. This market gap presents a significant opportunity for innovative leadership in developing a transformative method.

Potential Future AI Disruption: Building on established domain models, Domain-Specific AI Agents will guide the architecture process by applying non-functional requirements to bounded contexts. These agents will recommend logical architectural options, including relevant cloud design patterns to meet the SLAs. Using well-architected principles, the AI Agent will further assist developers in aligning these logical components with appropriate cloud services, such as managed Kafka or Redis. For organizations with platform engineering constructs and guardrails, the AI Agent will also suggest suitable customer-owned building blocks.

D. ARCHITECTURE TO DEPLOYABLE CODE

While code generation using AI works effectively at the fine-grained unit level, developers face the following challenges:

Inconsistency and Completeness: Current AI can generate code based on cloud design patterns (ex. CQRS) and architectural principles, such as creating service stubs, data models, and API endpoints. But the generated code shows inconsistency across repeated requests and often requires manual adjustments for accurate integration with existing components.

Customization for Organizational Standards: Organizations maintain specific coding standards, naming conventions, preferred libraries, and internal reusable building blocks as part of their platform engineering. While AI can leverage internal coding practices/artefacts (e.g. Q Customizations), expansion of language support is urgently needed ([marshal link](https://marshal.corp.amazon.com/insight/375739)).

Adaptive Refinement through Iteration: Current AI assists with iterative refinements as requirements evolve, but struggles with changes spanning multiple components and files, often producing incomplete or inaccurate updates. This frequently leads to misalignment between planning and code generation.

Potential Future AI Disruption: We hypothesize that current limitations stem from AI's tendency to jump directly from top-level planning to code generation(5). Breaking down planning into more granular steps—each precisely mapped to specific code components—could enable AI to generate more consistent and accurate code while maintaining synchronization between planning and generation. Our experiments(5) show improved outcomes when manually prompting AI to generate fine-grained plans and guiding it through step-by-step execution

E. REFACTORING EXISTING APPLICATIONS

Almost every customer has one or more technical-debt-ridden workloads that impede business progress by resisting changes and additions. While we address this through our Build-with-DevTx offering currently, it typically takes six months to strangle and modernize a significant portion of such workloads. Current AI capabilities are insufficient in this space, and partner tools like CAST or vFunction fall short of [expectations](https://quip-amazon.com/0GybAZH91Zej/CAST-Workshop). In the future, Domain Specific AI Agents (section 3a) shall enable developers to design ideal modernized versions of their applications (as if they are designed from scratch) while also identifying and reusing applicable existing code fitting into the new design.

F. SOFTWARE QUALITY MANAGEMENT

Due to the method adoption barriers noted above, we observe that customers rarely apply practices like TDD. Most customers typically have insufficient test automation (scripting) capabilities within their QA teams. Comprehensive test data generation remains a significant challenge. The resulting heavy reliance on manual testing leads to poor coverage, making it difficult to prevent defects from reaching production. Despite dedicated SRE teams, organizations struggle to maintain production stability and meet business SLAs. Current AI capabilities show considerable advancement in generating unit and functional test cases from source code. AI can also create Gherkin scenarios and step definitions (BDD) by analysing natural language requirements today. In the future, specialized AI Test Agents shall generate unit, domain specific functional, and regression tests by correlating source code with DDD or BDD-styled domain models. This advancement would eliminate manual effort, improve coverage, and enable rapid iteration regardless of the chosen methodology.

G. APPLICATION SECURITY TESTING

Static and Dynamic Application Security Testing (SAST & DAST) are typically performed manually and irregularly, resulting in inconsistent coverage and delayed issue detection. Many customers lack proper infrastructure to conduct effective DAST in lower environments. Additionally, customers often fail to integrate SAST & DAST into their CI/CD pipelines. Developers face challenges in efficiently triaging and addressing identified security issues, creating friction between development, product, and security teams.Current AI capabilities are comprehensive for SAST but limited in DAST. Future AI development shall focus on automating DAST specific to business domains, environment creation, and test execution. Additionally, AI should provide vulnerability remediation solutions and, with developer approval, implement fixes and continue testing until DAST requirements are fully met.

H. OBSERVABILITY & OPERATIONS

Contemporary practices like Risk Storming are being refined to systematically identify, prioritize, and implement observability mechanisms aimed at mitigating reliability and performance risks. Current AI capabilities offer limited support in the areas of risk modeling and observability automation and predominantly focus on reactive monitoring functions, such as anomaly detection. Looking ahead, Domain AI Agents are envisioned to address these gaps by generating context-specific risk models, recommending tailored observability metrics, and embedding data collection mechanisms directly within the codebase. AI should further analyze metrics and logs to identify anomalies and deliver predictive insights, thus facilitating a shift towards the preventative paradigm in SRE. Any other existing operational utilities like drift detection are envisaged to be available as ‘external functions’ to be invoked by AI.

I. THE COST OF RESEARCH & TRADE-OFFS ANALYSIS

At every stage, developers make numerous macro and micro decisions. Currently, they spend significant time researching (e.g., SQL vs NoSQL, developing/debugging step-functions locally) and analyzing trade-offs. They consult various sources based on decision complexity—including books, websites, Stack Overflow, and GitHub. These constant, expensive detours result in incomplete consideration of options, delayed timelines, reduced productivity, cognitive overload, and disruption of their 'flow' state. The speed of iteration is limited by manual efforts in hypothesis creation, coding, and testing, along with coordination overhead. Feedback loops are slow, causing significant delays before new ideas can be implemented. Rapid iteration is further hindered by the need for specialists (ex. security, databases) and fragmented tools. Today, AI can be effectively used for research and experimentation, but developer experience is hindered by linear chat-styled user interfaces that lack tight mapping to code blocks. In the future, developers should be able to leverage AI contextually to research, make decisions, rapidly create and execute experiments, and iterate at the speed of their judgments, even when dealing with large, complex contexts.

J. DISTANCE FROM BUSINESS AMBITIONS

In any organization, business strategies are created with the primary aim of delivering value to customers. Each strategy is supported by various IT initiatives, which are subsequently divided into projects and further into individual tasks. Developers are often positioned at the end of this extended value chain, at the task level. They struggle to perceive the direct impact of their work on customer value creation and this disconnection diminishes the developer's sense of purpose. The AI-Driven Development Life Cycle (AI-DLC), in its fully realized form, will significantly reduce these degrees of separation, bridging the gap between customer value creation and code.

K. SUMMARY

From the analysis, three key requirements emerge as future AI capabilities:

1. Ability to decompose top-level ambitions into fine-grained, actionable steps (tree structured) with direct mapping to code blocks. This hierarchical breakdown enables developers to navigate efficiently and iteratively across the steps. This pattern supports continuous adaptation, allowing incremental modifications at any level while maintaining synchronization throughout the development flow as requirements evolve.
2. A reimagined AI interface (Fig.1) that replaces current linear chat-style interactions with a hierarchical structure. This interface enables developers to review top-level plans, evaluate intermediate steps, assess AI-suggested options, understand trade-offs, make decisions, and provide direction. The tree-like structure allows nodes to expand for detailed examination or collapse for streamlined viewing, helping developers manage complexity effectively while maintaining strategic engagement in critical decision-making.
3. A transition from generic Coding Assistants to Fine-tuned Domain Specific AI Agents that generate consistent, comprehensive components for each business domain. These agents would create technical components (as defined in DDD), code, tests, and documentation that comply with best practices, cloud design patterns, and well-architected principles.

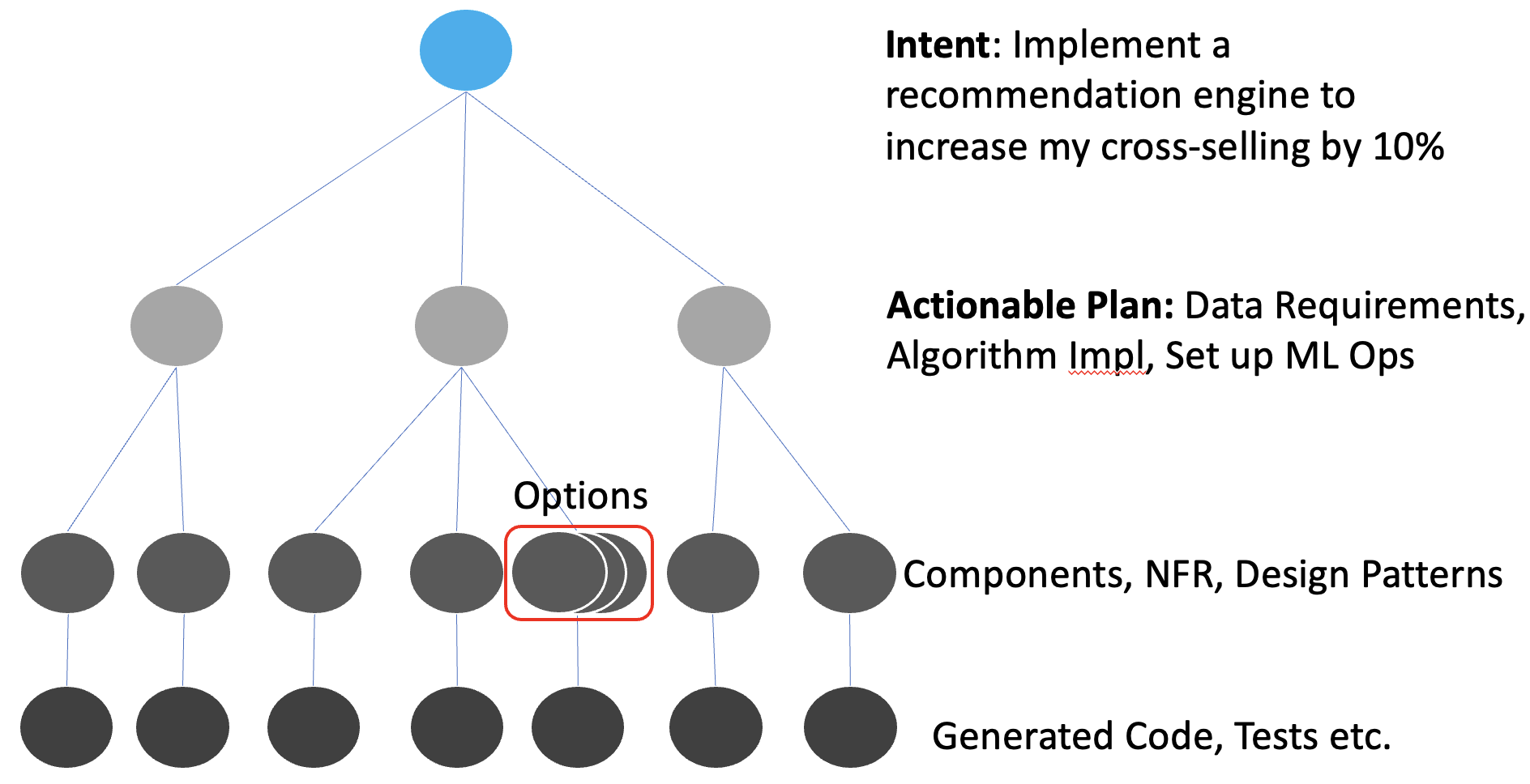


Fig 1. Example AI-DLC Flow

**IV. DEFINING AI-DLC**

In AI-DLC, the process begins with a business ambition, such as: *'I want to expand my XXX business that is running on a monolith application, into a new market in 2 months. It used to take 6 months to customize before*.' Developers will collaborate with AI to translate the ambition into an actionable plan. For example, AI may suggest a sequence of tactics like refactoring the application, externalizing configurations, adopting cloud-native runtimes etc. This plan unfolds like a tree structure, where each branch represents a step toward more detailed technical components. Developers will use Depth First or Breadth First approaches to break down each tactic into increasingly granular actions until they reach the leaf nodes—executable code that can be tested and deployed. At every stage of this decomposition, developers will engage in decision-making. They will assess AI-generated options at each node, weighing trade-offs and making key architectural decisions (ex. choice of design patterns) to direct the AI. Finally, AI takes care of the heavy-lifting in generating the code, allowing developers to focus on strategic choices and engineering best practices. AI-DLC represents a fundamental shift in how software is built, moving from manual analysis, research, design and coding tasks to AI-assisted decision-making and automated code generation.Following is an instance of a division of labour between developers and AI in AI-DLC:

1. Business Ambitions to Actionable Plans: [AI's Role] Analyse high-level business objectives and convert them into structured, actionable development tasks, suggesting strategies and breaking down initiatives (like refactoring and cloud services adoption) into tactical steps. [Developer's Role] Validate and refine AI-generated strategies, ensure business goal alignment, and provide domain-specific insights that AI may miss.
2. Translate Requirements to Domain Models: [AI's Role] Generate initial domain models from requirements, suggest tactical patterns (DDD), and refine entities, relationships, and interactions. [Developer's Role] Review and adjust domain models, incorporate deep domain knowledge, and ensure compliance with specific business rules. Guide AI to account for unique business nuances and complex relationships.
3. Architecture Design and Infrastructure Mapping: [AI's Role] Propose cloud-native architecture options, apply design patterns, and recommend appropriate cloud services for each architectural component. [Developer's Role] Select from suggested architectures based on organizational standards, scalability needs, and project goals. Make critical decisions regarding security, cost, and governance.
4. Code Generation and Implementation: [AI's Role] Generate boilerplate code, automate repetitive tasks, implement and integrate routine logic based on established patterns. [Developer's Role] Refine, optimize, and customize AI-generated code, especially for complex business logic or unique performance requirements. Drive innovation and creativity.
5. Testing and Quality Assurance: [AI's Role] Create various test types from domain models, automate repetitive testing scenarios, and run continuous test cycles. [Developer's Role] Evaluate AI-generated test cases, design complex scenarios for edge cases, and guide AI in improving test coverage and precision.
6. Deployment and Monitoring: [AI's Role] Automate deployment processes, manage observability emitters, analyse logs, track performance metrics, and identify anomalies. [Developer's Role] Set monitoring parameters, evaluate system performance, adjust observability thresholds, and plan performance improvements.
7. Iteration and Continuous Feedback: [AI's Role] Gather and analyse production feedback, identify improvement areas, and suggest refinements. [Developer's Role] Interpret AI insights for business impact, prioritize feedback for future iterations, and make strategic course corrections.

AI handles routine tasks for which it can be trained and fine-tuned, while developers focus on higher-order analysis, insights, creativity, and decision-making. These interactions are facilitated through a reimagined UI that enables developers to efficiently navigate from big-picture goals to specific tasks while maintaining complete control and context:

1. Hierarchical Flow: High-level ambitions occupy the top level, branching downward through AI-suggested tactical steps and design decisions to final code implementations. Each node can expand for detailed examination or collapse for a streamlined overview.
2. Interactive Nodes: Each node presents AI recommendations, allowing developers to select from options, edit directly, or make strategic decisions at any step. Changes at any level cascade downward, maintaining tree synchronization.
3. Seamless IDE Integration: The tree interface integrates directly with the developer's IDE, enabling fluid transitions between coordination and code without tool switching.

This presents an opportunity to develop next-generation tooling that complements the AI-DLC method, potentially through a VSCode fork similar to the [Cursor](https://www.cursor.com/) team's approach.

**V. DEFINING THE FUTURE PERSONA OF DEVELOPERS**

While this will evolve with the fast changing AI capability, the following is an initial point-of-view of developer persona that uses AI-DLC:

1. The Future Developer will be an **AI Native Trailblazer**, comfortable and naturally adept at AI, achieving an order of magnitude more effectiveness compared to the previous developers.
2. Unlike their past selves who simply moved **features into code**, the Future Developer will move **business ambitions into code**. They will collaborate with AI to fully understand business strategies, break them down into IT tactics, and seamlessly handle all the planning, decision-making, design and development within their IDE through continuous, context-rich interactions with AI
3. Using a combination of AI and creativity, they will **iterate at the speed of their judgment**, even while dealing with unprecedented ambiguity and complexity.
4. They will **effortlessly transcend the rigid specialization silos** of today — such as IaC, front-end, back-end development, databases, DevOps and application security—and seamlessly navigate across these domains and integrate with ease.
5. They will thrive in a state of **continuous flow**, never needing to leave their IDE for research, decision support, architecting, debugging, and any other step in the build-release workflow. This deep fulfilment of creating without cognitive interruption will enable them to reach **maximum developer satisfaction**, realising their full potential in the process.

**VI. NEXT STEPS**

The current state of AI is not a blocker to start adopting AI-DLC. We just have to develop prescriptive guidance based on where AI works well and where it needs manual handholding.

1. Develop deep material for AI-DLC with current state of AI to be used in developer engagements by Nov 2024. Start the pilot execution of the Future Developer Alliance (FDA) Program. FDA ([link to PRFAQ](https://amazon.awsapps.com/workdocs-amazon/index.html#/document/d07ba0e86157bb1a31dd0edb0980eb60511dcba26fb6f3e80c3fc0aec370e334)) is a hyper-scaling field program that drives a self-sustaining flywheel effect inside large organizations with more than 1000 developers.
2. Develop the [method definition whitepaper](https://amazon.awsapps.com/workdocs-amazon/index.html#/document/ecb7f8b3e28e6f4f2c9fa5249cc6d88cc68b9a8afde2b482c54eb847d5a6a452) for AI-DLC with the future state of AI. Gather feedbacks from developers in the customer engagements as inputs for evolving our AI services.
3. Evolve tooling (code named as Amazon Q Flow) and the Domain Specific AI Agents requirements using the AI-DLC principles as input to Q and NGDE service teams by Jan 2025.

**Appendix 1: FUDs of Developers**

A [survey conducted by SlashData](https://www.slashdata.co/post/how-generative-ai-will-affect-developers-work) reveals that 32% of developers strongly agree that AI could surpass their skills and render their jobs obsolete. Additionally, 40% are either unsure or disagree with this notion, indicating a substantial level of uncertainty about the future of their roles in light of AI advancements.

[Research from Evans Data Corporation](https://brainhub.eu/library/software-developer-age-of-ai) found that nearly 30% of software developers believe their development efforts will be replaced by AI in the foreseeable future. This statistic underscores a pervasive concern within the developer community about job security and the potential for AI to disrupt traditional programming roles.

[A study](https://theconversation.com/how-software-developers-feel-about-ai-reshaping-their-work-and-what-this-tells-the-rest-of-us-231211) analyzing emotional responses to AI tools, such as GitHub Copilot, indicated that while many developers initially felt fear and skepticism about AI's impact on their work, there was also a notable shift toward excitement as they recognized AI's potential to automate tedious tasks. However, this emotional journey reflects an ongoing struggle with uncertainty about how AI will reshape their responsibilities and workflows.

Furthermore, the [State of the Developer Nation report](https://www.slashdata.co/post/how-generative-ai-will-affect-developers-work) highlights that while many developers acknowledge the benefits of AI in enhancing productivity and efficiency, a significant portion remains apprehensive about its implications for their job roles. The report notes that 61% of developers express concern over ethical issues related to AI, including its potential to displace workers

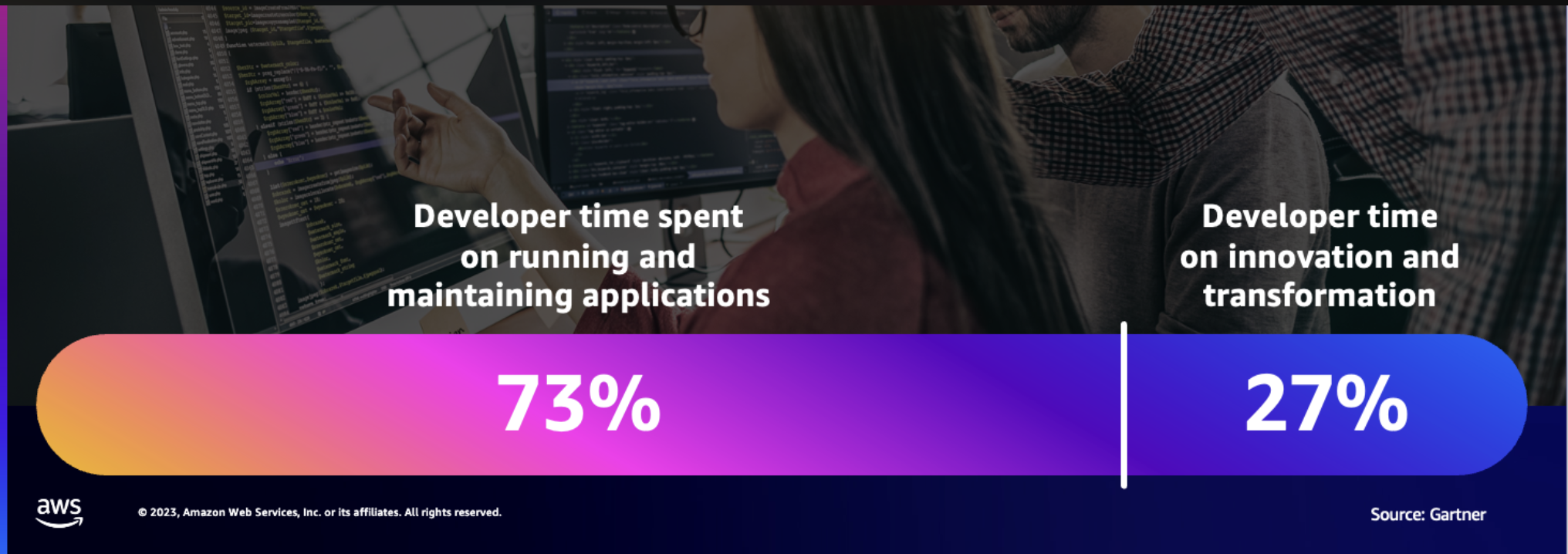
**Appendix 2: Detailed Problem Statements**

CxOs’ Imperatives: CxOs leading organizations with 1000+ developers (DBS, SPH, Verizon, Accenture, Infosys, Samsung etc.) had always been challenged with not only accelerating developer productivity but also fostering a cohesive environment that drives talent retention, engagement, and commitment to best practices. A lack of shared collective tenets, fragmented communication, and insufficient knowledge-sharing across teams had stifled internal cross-pollination and clogged overall developer throughput. Meanwhile, GenAI offers a previously unattainable ability to unify disparate teams under a shared AI-driven development model. CxOs recognize the importance of strong leadership in leveraging the promise of AI and they are focussed on creating structured frameworks for its implementation.

Developers’ Aspirations: Developers are constantly seeking a golden pathway to stay highly sought-after in the job market, develop the craftsmanship to deliver on their organization’s business ambitions, and evolve to be tomorrow's strategic innovators. They understand that GenAI is not just about building code and debugging; it embodies deep expertise in computer science, software engineering principles and development best practices. However, they need new skills to partner with AI and master this emerging specialization. To this end, there is an absence of targeted, future-ready programs with a clear vision of the AI-powered evolution of development that holistically addresses the critical challenges in all the developer journeys across the SDLC. AWS field engagements predominantly focus on selling the features of Q without addressing the higher-order developer challenges such as domain refactoring, accelerating the flow of ideas to production, enabling rapid feedback from production, fostering continuous quality improvements etc. This approach falls short of empowering developers to fully harness AI’s potential for their career growth

AWS Imperatives: From a customer-obsession standpoint, if our customers don't quickly adopt AI-native practices and empower a new generation of AI-driven developers, they risk falling behind their competition. Internally, we must continuously evolve our GenAI offering by crafting a bold, forward-looking hypothesis, rapidly testing and iterating with real-time developer feedback to stay ahead. In the field, we need more than just initial success with large customers—we need a flywheel mechanism that scales our best practices across their entire organization, driving exponential, hockey-stick growth in AWS AI services adoption. For our north-star goal of accelerating and sustaining the adoption of Amazon Q [150K Pro by xxx] in India, we must go beyond traditional engagement and implement a deeper, scalable 1:\* model with large GSIs and enterprises to achieve that breakthrough growth.

**Appendix 3: Maintenance vs value creation**

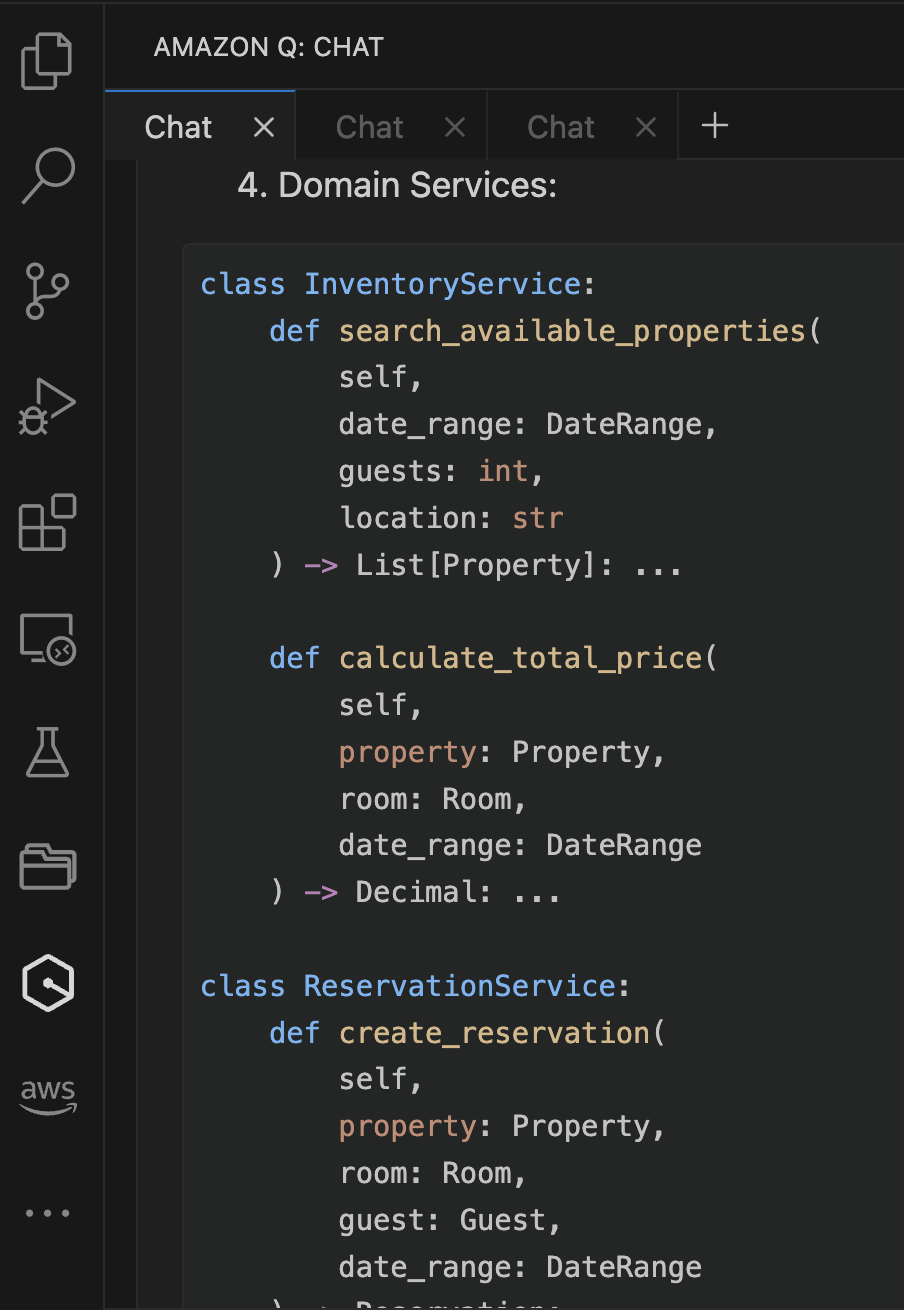
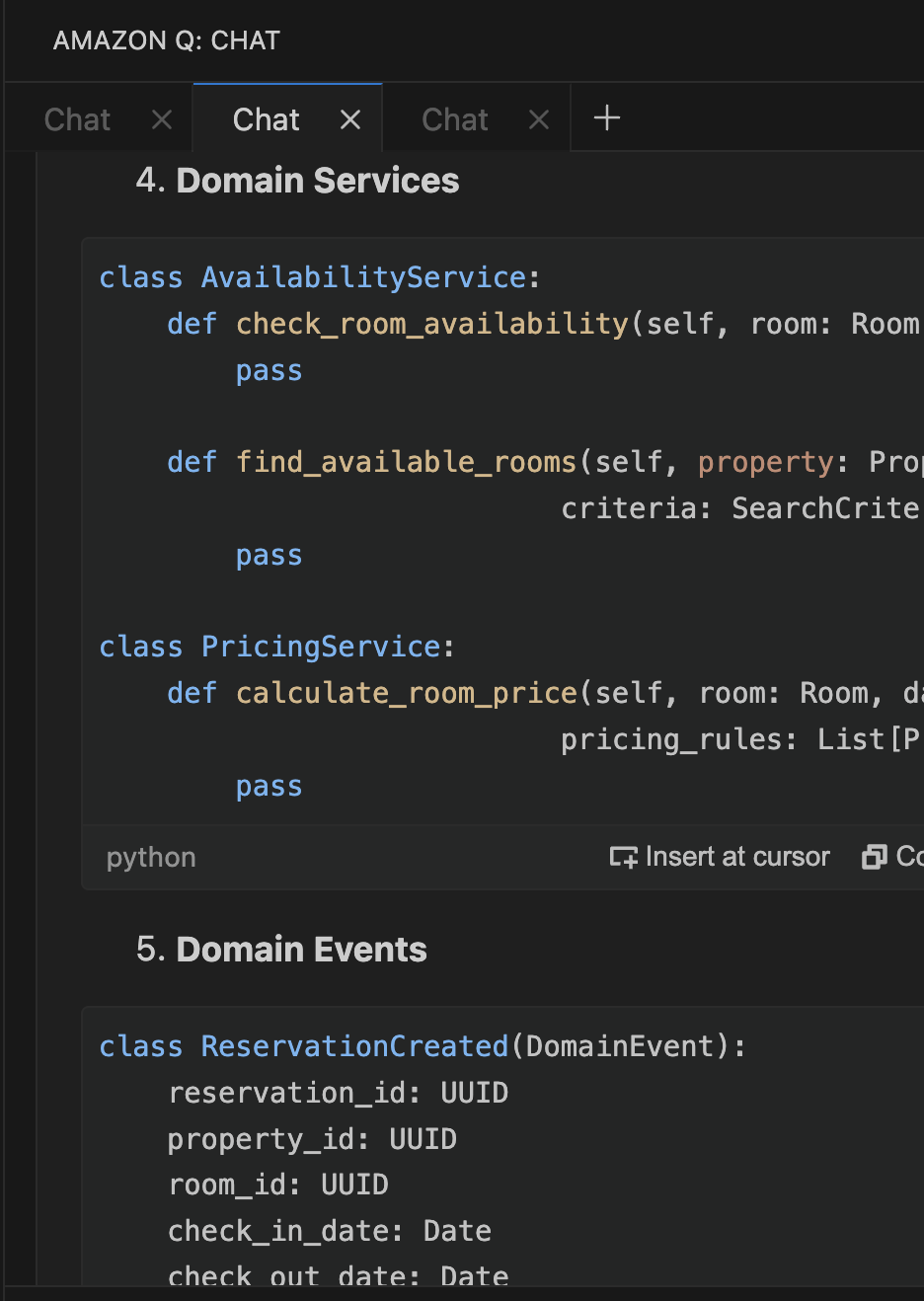


**Appendix 4: Inconsistency & lack of comprehensiveness**

For the same intent given, n instances of invoking Q generates n different set of domain components and all them were incomplete also, missing the important components.

Intent: I am designing the inventory subdomain of a hotel booking system in the scale of airbnb. Help me to design the domain model using Domain Driven Design principles.

Screenshots attached below:

**Appendix 5: Issues because of rushing to code from top level plan**

[This conversation](https://amazon.awsapps.com/workdocs-amazon/index.html#/document/3cf71c07007b2271af5dc0aaa6f0007a997aa882a5c90e426ae2dd74a9947718) captures the AI (claud-3.5-Sonnet) generating code directly from the developer intent. After generation it gets into a non-stop loop of fixing a defect but introducing new.

The conversations in the files with the name “incremental-steps” in [this folder](https://amazon.awsapps.com/workdocs-amazon/index.html#/folder/fdd119570286a1e804fcb26c9d3af86339b7a0e21a0646a4af2931efa7af9a87) is the alternative approach were we incrementally guided the AI and achieved the results reliably.