AI-Driven Development Lifecycle (AI-DLC) Method Definition

Raja SP, Amazon Web Services

I. CONTEXT

The evolution of software engineering has been a continuous quest to enable developers to focus on solving complex problems by abstracting away lower-level, undifferentiated tasks. From early machine code to high-level programming languages and the adoption of APIs and libraries, each step has significantly boosted developer productivity. Now, the integration of Large Language Models has revolutionized how software is created, introducing conversational natural language interactions for tasks like code generation, bug detection, and test generation. This marks the **AI-Assisted** era, where AI enhances such fine-grained, specific tasks.

As AI evolves, its applications are expanding beyond code generation to include requirements elaboration, planning, task decomposition, design, and real-time collaboration with developers. This shift is kick-starting the **AI-Driven** era, where AI actively orchestrates the development process. But, existing software development methods, designed for human-driven, long-running processes, are not fully aligned with AI's speed, flexibility, and advanced capabilities (ex. agentic). Their reliance on manual workflows and rigid role definitions limits the ability to fully leverage AI. Retrofitting AI into these methods not only limits its potential, but also reinforces outdated inefficiencies. To fully leverage AI's transformative power, SDLC methods need to be reimagined. This reimagination requires AI to be a central collaborator, aligning workflows, roles, and iterations to enable faster decision-making, seamless task execution, and continuous adaptability.

This paper introduces and defines the **AI-Driven Development Lifecycle (AI-DLC),** a reimagined, AI-native methodology designed to fully integrate the capabilities of AI, setting the foundations for the next evolution in software engineering.

II. KEY PRINCIPLES

The principles in this section form the foundation for defining AI-DLC, shaping its phases, roles, artifacts, and rituals. These assumptions are critical for validating the proposed method, as they provide the underpinning rationale behind its design.

1. REIMAGINE RATHER THAN RETROFIT

We choose to reimagine a development method rather than keeping the existing methods like SDLC or Agile (e.g., Scrum) and **retrofitting** AI into them. These traditional methods were built for longer iteration duration (months and weeks), which led to rituals like daily standups and retrospectives. In contrast, proper application of AI leads to rapid cycles, measured in hours or days. This needs continuous, real-time validation and feedback mechanisms, rendering many of the traditional rituals less relevant. Would effort estimation (ex. story points) be as critical if AI diminishes the boundaries between simple, medium and hard tasks? Would metrics like velocity be any relevant or should we start replacing it with Business Value, as an example? Also, AI is increasingly evolving to automate manual practices including planning, task decomposition, requirements analysis, application of design techniques (ex. domain modelling), thereby shortening the number of phases it takes from moving intentions to code. These new dynamics warrant a **reimagination based on first principles thinking**, rather than a retrofitting. We need automobiles and not the faster horse chariots.

2. REVERSE THE CONVERSATION DIRECTION

AI-DLC introduces a fundamental shift where **AI initiates & directs the conversations** with humans instead of humans initiating the conversation with AI to complete their tasks. AI drives workflows by breaking down high-level intents (ex. implementing a new business function) into actionable tasks, generating recommendations, and proposing trade-offs. Humans serve as approvers, validating, selecting options, and confirming decisions at critical junctures. This AI-driven approach allows developers to focus on high-value decision-making while AI handles planning, task decomposition, and automation. By reversing the traditional dynamic, AI-DLC ensures that human involvement is purposeful, concentrating on oversight, risk mitigation, and strategic alignment, thereby enhancing both velocity and quality. An analogy to illustrate this is Google Maps: humans set the destination (the intent), and the system provides step-by-step directions (AI's task decomposition and recommendations). Along the way, humans maintain oversight and moderate the journey as needed.

3. INTEGRATION OF DESIGN TECHNIQUES INTO THE CORE

Agile frameworks like Scrum or Kanban leaves the design techniques (ex. Domain Driven Design) out of scope and recommends the teams to choose their own. This has left critical whitespaces that led to poor software quality overall. Software quality issues in US alone was estimated to cost $2.41 Trillion in 2022 (study). Rather than decoupling design techniques out, AI-DLC will have them as its integral core. There will be different flavors of AI-DLC for teams following Domain Driven Design (DDD), Behavior Driven Development (BDD) or Test-Driven Development (TDD) respectively. This paper discusses the DDD

flavor of AI-DLC which will use DDD principles to break down systems into independent, right-sized bounded contexts that can be rapidly built in parallel. AI will inherently apply these techniques during planning and task decomposition, requiring developers only to validate and adjust. This integration is key to enabling hourly or daily iteration cycles while eliminating manual heavy-lifting and maintaining software quality (the mantra of "Build Better Systems Faster").

4. ALIGN WITH AI CAPABILITY

This paper is optimistic about the future potential of AI but fully realistic about its current state. AI-DLC recognises that current AI is advancing but not yet reliable in autonomously translating high-level intentions into executable code or independently operating without human oversight, while also **ensuring interpretability and safety.** At the same time, the **AI-Assisted** paradigm, where developers perform the majority of the intellectual heavy lifting with AI merely providing augmentation, fails to unlock the full potential of AI in development. AI-DLC adopts the **AI-Driven** paradigm, which balances human involvement with the capabilities and limitations of current AI. In this, the **developers retain ultimate responsibility for validation, decision-making, and oversight**. This balance ensures that the strengths of AI are leveraged effectively without compromising the critical safeguards provided by developer judgment.

5. CATER TO BUILDING COMPLEX SYSTEMS

AI-DLC caters to building systems that demand continuous functional adaptability, **high architectural complexity**, **numerous trade-offs management**, scalability, integration and customization requirements. These necessitate the application of advanced design techniques, patterns, and best practices, typically involving **multiple teams working cohesively** within large and/or regulated organizations. Simpler systems that can be developed by non-developer personas that needed few or no trade-off management are outside the scope of AI-DLC and are better suited for low-code/no-code approaches.

6. RETAIN WHAT ENHANCES HUMAN SYMBIOSIS

While reimagining the method, we will retain the artifacts and touchpoints from the existing methods that are critical for human validation and risk mitigation. For instance, user stories align humans' and AI's understanding of what needs to be built, acting as well-defined contracts. We will retain user stories as they are in the re-imagined method also. Another example is the Risk Register that ensures AI-generated plans and codes comply with organizational risk frameworks. These retained elements will be optimized for real-time use, allowing rapid iterations without compromising alignment or safety.

7. FACILITATE TRANSITION THROUGH FAMILIARITY

The new method shall not demand extensive trainings and any existing practitioner should be able to orient and start practicing it in a single day. To support easier adoption via associative learning, AI-DLC will preserve the underlying relationships between familiar terms in older methods while introducing modernized terminology. For example, Sprints in Scrum represent iterative cycles for building and validating. But Sprints are usually 4 to 6 weeks long in the pre-AI era. With AI-DLC, the iteration cycles will be continuous and in terms of hours or days. Therefore, we need to intentionally rename Sprints. AI-DLC rebrands Sprints as Bolts, emphasizing rapid, intense cycles that deliver unprecedented velocity.

8. STREAMLINE RESPONSIBILITIES FOR EFFICIENCY

By leveraging AI's ability to perform task decomposition, and decision-making, developers will be empowered to transcend traditional specialization silos such as infrastructure, front-end, back-end, DevOps, and security. This convergence of responsibilities reduces the need for multiple specialized roles, streamlining the development process. But Product Owners and developers remain integral to the framework, retaining critical responsibilities for oversight, validation, and strategic decision-making. These roles ensure alignment with business objectives, maintain design quality, and maintain compliance with risk management frameworks, preserving the balance between automation and human accountability. In the method definition, we will stick to first principles, keeping the roles minimum, with additional roles introduced only when critically necessary.

9. MINIMISE STAGES, MAXIMISE FLOW

Through automation and convergence of responsibiliti4es, AI-DLC aims to minimize the handoffs and transitions, enabling continuous iterative flow. But human validation and decision-making remain critical to ensure that AI-generated code does not become rigid ('quick-cement') but stays adaptable for future iterations. To address this, AI-DLC incorporates minimal but sufficient number of phases specifically designed for human oversight at critical decision junctures. These validations act as a form of 'loss function', by identifying and pruning wasteful downstream efforts before they occur.

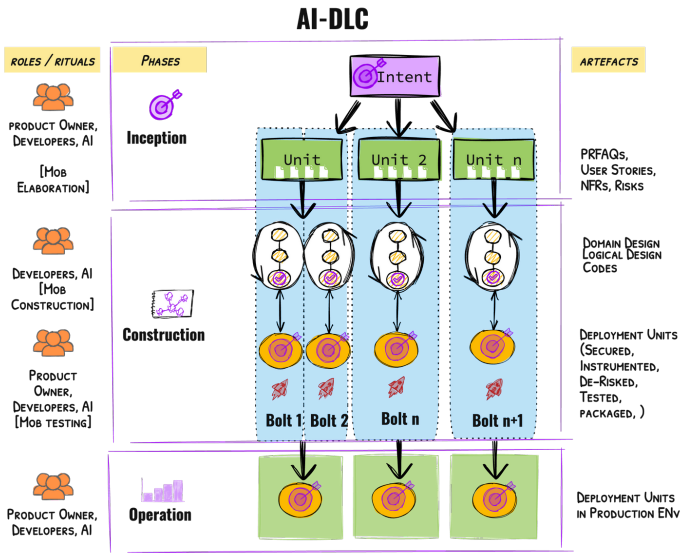
10. NO HARD-WIRED, OPINIONATED SDLC WORKFLOWS

AI-DLC avoids prescribing opinionated workflows for different development pathways (such as new system development, refactoring, defect fixes, or microservice scaling). Instead, it adopts a truly AI-First approach where AI recommends the Level 1 Plan based on the given pathway intention. Humans verify and moderate these AI-generated plans through interactive dialogue with AI, continuing this process through Level 2 (subtasks) and subsequent hierarchy levels. At the task execution level, AI implements the tasks while humans maintain oversight through verification and validation of outcomes. This flexible approach ensures the methodology is adaptable and can evolve alongside

AI capabilities while maintaining human control over critical decisions.

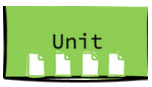
III. CORE FRAMEWORK

This section outlines the core framework of AI-DLC, detailing its phases, roles, workflows, and key artifacts.



1. ARTEFACTS

An **Intent** in AI-DLC is a high-level statement of purpose that encapsulates what needs to be achieved, whether a business goal, a feature, or a technical outcome (ex. performance scaling). It serves as the starting point for AI-driven decomposition into actionable tasks, aligning human objectives with AI-generated plans.



A **Unit** represents a cohesive, self-contained work element derived from an Intent, specifically designed to deliver measurable value. For instance, an Intent to implement a business idea may be decomposed into Units representing independent functional blocks, analogous to Subdomains in DDD or Epics in Scrum. Each Unit encompasses a set of tasks (user stories, in this case, that articulate its functional scope) In the context of AI-DLC, the process of decomposing Intents into Units is driven by AI, with developers and/or Product Owners validating and refining the resulting Units to ensure alignment with business and technical objectives. The units are **loosely coupled**, enabling autonomous development and independent deployment downstream.

A **Bolt** is the smallest iteration in AI-DLC, designed for the rapid implementation of a Unit or a set of tasks within a Unit. Bolts (analogous to Sprints in Scrum) emphasize intense focus and high-velocity delivery, with build-validation cycles measured in hours or days rather than weeks. Each Bolt encapsulates a well-defined scope of work (e.g., a collection of user stories within a Unit), enabling incremental progress while maintaining alignment with the overall objectives of the Unit it supports. A Unit can be executed through one or more Bolts, which may run in parallel or sequentially. AI will plan the Bolts with developers / Product Owners validating it.

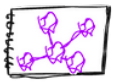
The **Domain Design** artefact models the core business logic of a Unit, independently of the infrastructure components. In the first version of AI-DLC, AI uses domain-driven design principles to create the strategic and tactical modelling elements including aggregates, value objects, entities, domain events, repositories and factories. The **Logical Design** translates Domain Designs by extending them for meeting the non-functional requirements using the right choice of architectural design patterns (ex. CQRS, Circuit Breakers etc.). AI creates the Architecture Decision Records (ADRs) for validation by the Developers. With the Logical Design specification, AI will generate the **Code** and **Unit Tests**, ensuring adherence to well-architected principles by selecting appropriate AWS services and constructs. At this stage, the AI agent will conduct the unit testing, analyse the results and provide recommendations on fixes to the Developer.

The **Deployment Units are** the operational artifacts encompassing the packaged executable code (ex. container images for Kubernetes environments, serverless functions such as AWS Lambda), configurations (ex. Helm Charts), and infrastructure components (ex. Terraform or CFN stacks) that are tested for functional acceptance, security, NFRs and other risks. AI generates all associated tests, including functional tests, static and dynamic security tests, and load testing scenarios. After the human validation and adjustments on the test scenarios and cases, the AI agent executes the test suits, analyses the results and correlate failure points with code changes, configurations or other dependencies. Thus, these units are rigorously tested for functional acceptance, security compliance, adherence to non-functional requirements (NFRs), and mitigation of operational risks, guaranteeing their readiness for seamless deployment.

2. PHASES & RITUALS

The **Inception Phase** focuses on capturing **Intents** and translating them into Units for development. This phase uses the "*Mob Elaboration*", a collaborative requirements elaboration and decomposition ritual. This happens in a single room with a shared screen led by a facilitator. During Mob Elaboration, AI plays a

central role in proposing an initial breakdown of the Intent into User Stories, Acceptance Criteria and Units, leveraging domain knowledge, and the principles of loose coupling and high cohesion for rapid parallel execution downstream. The Product Owner, Developers, QA and other relevant stakeholders (the mob) collaboratively review and refine these AI-generated artefacts by adjusting under-engineered or over-engineered parts and aligning them with real-world constraints. The outputs of this phase include well defined Units and their respective components containing a) PRFAQ, b) User Stories, c) Non-Functional Requirement (NFR) definitions, d) Description of Risks (matching with organization's Risk Register, if present), e) Measurement Criteria that traces to the business intent and the f) Suggested Bolts using which the Units can be constructed. Mob Elaboration condenses weeks or even months of sequential work into a few hours, while achieving deep alignment both within the mob and between the mob and the AI.

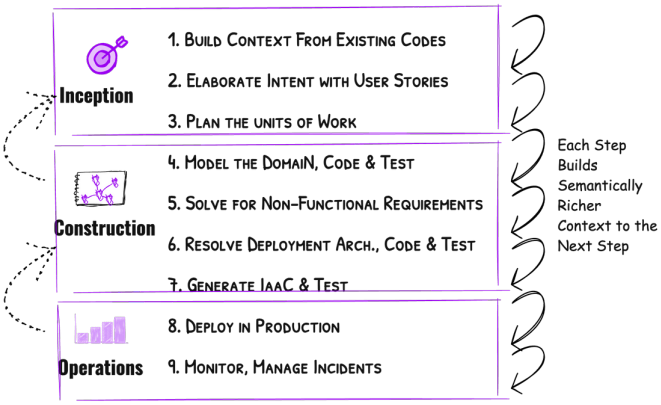


The **Construction Phase** The encompasses the iterative execution of tasks, transforming the Units defined during the Inception Phase into tested, operations-ready Deployment Units. This phase progresses through *Domain Design*, where AI models the business logic independently of technical considerations, to *Logical Design*, where non-functional requirements and appropriate cloud design patterns are applied. AI generates detailed code from Logical Designs by mapping components to appropriate AWS services while adhering to well-architected principles. The phase concludes with automated testing to ensure functionality, security, and operational readiness. Developers focus on validating AI-generated outputs at each step and making critical decisions, ensuring quality and adaptability in each iteration. In the brown-field (existing application) scenarios, the construction phase involves first elevating the codes into a semantic rich modelling representation so that the context to AI becomes concise and accurate. The suggested modelling representations are *static models* (just the domain components, responsibilities and their relationships) and *dynamic models* (how the components interact to realize the significant use cases)

AI plays a pivotal role throughout this phase, recommending tasks and providing options (design patterns, User Experience, Tests etc) at each task. AI-DLC recommends that this be done with all teams collocated in a single room, similar to Mob Elaboration. The teams exchange the integration specifications (from domain model stage), make decisions and deliver their bolts. AI-DLC calls this the mob-construction ritual.

The **Operations** Phase in AI-DLC centers on the deployment, observability, and maintenance of systems, leveraging AI for operational efficiency. AI actively analyzes telemetry data, including metrics, logs, and traces, to detect patterns, identify anomalies, and predict potential SLA violations, enabling proactive issue resolution. Additionally, AI integrates with predefined incident runbooks, proposing actionable recommendations such as resource scaling, performance tuning, or fault isolation and execute the resolutions when approved by the Developers. Developers serve as validators, ensuring AI-generated insights and proposed actions align with SLAs and compliance requirements.

3. THE WORKFLOW



Given a business intent (ex. Green-Field development, Brown-Field enhancement, modernization, or defect fixing), AI-DLC begins by prompting AI to generate a Level 1 Plan that outlines the workflow to implement the intent. This plan serves as an initial proposal, which is then transparently reviewed, validated, and refined by humans to ensure alignment with business goals and engineering constraints. At the heart of AI-DLC is the principle of applying human oversight to progressively enrich the artefacts of each step, transforming them into **semantically rich contex**t for the next. Each step serves as a strategic decision point where **human oversight functions like a *loss function* -** catching and correcting errors early before they snowball downstream. This repeats recursively. Each step in the Level 1 Plan is further decomposed into finer-grained, executable sub-tasks by AI, again under human oversight to ensure accuracy and contextual appropriateness.

All artefacts generated (intents, user stories, domain models, or test plans) are persisted and serve as a "context memory" that the AI references across the lifecycle. Like traditional SDLC methods, AI-DLC is inherently iterative, allowing for continuous refinement and adaptation. Additionally, all artefacts are linked, allowing for backward and forward traceability (ex. connecting domain model elements to specific user stories) ensuring that the AI retrieves the correct and most relevant context at every stage. Throughout the process, AI performs the strategic planning, task decomposition, generation etc. and humans provide the oversight and validation.

IV. AI-DLC IN ACTION: Green-Field Development

We will examine the scenario in which the Product Owner commences the process by articulating a high-level intent, such as "*Develop a recommendation engine for cross-selling products*." AI Recognizes this as an intent to build a new application and produces the Level 1 plan like the Workflow steps in the above section. The team validates, verifies and adds/fixes the stages in the level 1 plan. With the finalized Level 1 Plan, AI progresses to the Inception Phase. Refer to the prompts in *Appendix A* as a way to interact and provide oversights to AI.

1. INCEPTION PHASE

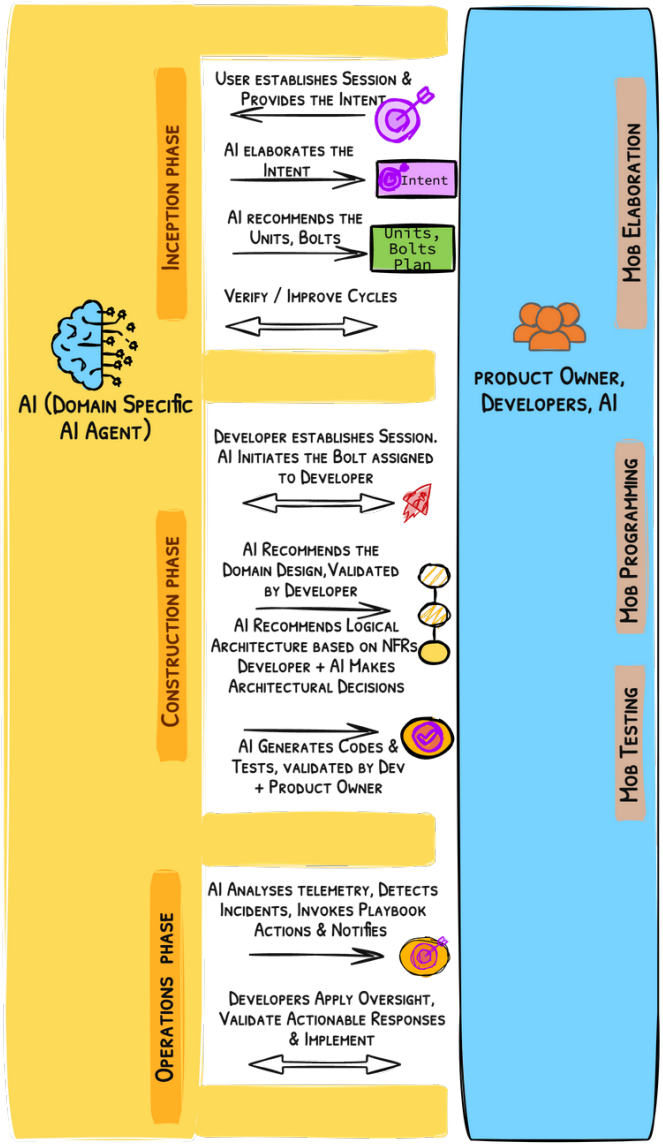
The following bullet points outline the key interactions in the Mob Elaboration ritual.

a. AI asks clarifying questions (e.g., *"Who are the primary users? What key business outcomes should this achieve?"*), ensuring a comprehensive understanding of the goal and minimize the ambiguity in the original intention
b. AI elaborates the clarified intention into **user stories**, **non-functional requirements (NFRs)**, and **risk descriptions**. The team validates these artefacts and provides oversight, and the corrections needed to AI.
c. AI composes the highly cohesive stories into **Units**, e.g., *"User Data Collection," "Recommendation Algorithm Selection,"* and *"API Integration."*
d. The Product Owner validates these outputs, adjusting wherever needed to refine the Units. Example: The Product Owner notices that *User Data Collection* lacks privacy compliance details and adjusts the requirements to include GDPR-specific considerations.
e. AI generates a **PRFAQ** for the module (optional), summarizing the business intent, functionality, and expected benefits.
f. Developers and Product Owners validate the PRFAQ and associated risks, ensuring alignment with the overall objectives.

2. CONSTRUCTION PHASE

The following bullet points outline the key activities involved in this phase, focusing on the Mob Programming and Mob Testing rituals.

a. The Developer establishes the session with AI. AI prompts the developer to begin with the Unit assigned to them.
b. AI models the core business logic for the assigned Unit using Domain-Driven Design principles. Example: For the *"Recommendation Algorithm"* Unit, AI identifies relevant entities like *Product, Customer, and Purchase History* and their relationships.
c. Developers review and validate the domain models, refining business logic and ensuring alignment with real-world scenarios (e.g., how to handle missing purchase history for new customers)
d. AI translates the domain models into logical designs, applying NFRs like scalability and fault tolerance. Example: AI recommends architectural patterns (e.g., event-driven design) and technologies (e.g., AWS Lambda for serverless computation).
e. Developers evaluate AI's recommendations, approve trade-offs, and suggest additional considerations if needed. (e.g., Accepts Lambda for scalability but overrides the storage to DynamoDB for faster query performance)
f. AI generates executable code for each Unit, mapping logical components to specific AWS services.



g. It also auto-generates functional, security, and performance tests (e.g., For the *"Recommendation Algorithm"* Unit, AI creates code to implement collaborative filtering and integrates it with a DynamoDB data source)

h. Developers review the generated code and test scenarios/cases, making adjustments where necessary to ensure quality and compliance.

Testing and Validation:

a. AI Executes all tests (functional, security, and performance), analyzes results, and highlights issues.
b. Proposes fixes for failed tests, e.g., optimizing query logic for better performance.
c. Developers validate AI's findings, approve fixes, and rerun tests as needed.

3. OPERATIONS PHASE

Deployment**:**

a. AI packages the module into **Deployment Units** (e.g., container images, serverless functions).
b. Developers approve the deployment configuration and initiate rollout to staging and production environments.

Observability and Monitoring**:**

a. AI analyzes metrics, logs, and traces to identify anomalies and predict potential SLA violations. Example: AI detects a latency spike during peak usage and proposes scaling the recommendation engine to handle increased traffic.
b. AI integrates with playbooks to suggest actions for operational issues. If API response times degrade, AI recommends increasing DynamoDB throughput or rebalancing API Gateway traffic.
c. Developers validate AI's recommendations, approve mitigations, and monitor resolution outcomes.

V. AI-DLC IN ACTION: Brown-Field Development

Brown-field refers to making changes to an existing system in terms of either adding new features, optimizing it for non-functional requirements or fixing technical debts including refactoring and fixing defects. In this context, we will examine a scenario where the product manager needs to add a new feature to an existing application.

1. INCEPTION PHASE

The inception phase activities in the brown-filed are same as that of the green-field

2. CONSTRUCTTION PHASE

a. AI elevates the codes into a higher-level modelling representation. The models comprise of static models (components, descriptions, responsibilities and relationships) and dynamic models (how the components interact to realize the most significant use cases)
b. Developers collaborate with product managers to review, validate and correct the static and dynamic models that are reverse engineered by AI.
c. With these extra steps, the rest of the construction phase is similar to that of the green-field scenario.

3. OPERATIONS PHASE

The operations phase activities in the brown-filed are same as that of the green-field

VI. ADOPTING AI-DLC

AI-DLC does not deviate much from the existing Agile methods and it is designed with easier adoption as a key outcome. Still organizations that are practicing the traditional methods for longer and those who are in the process of inventing their own variation of AI-Native methods need specific strategies for adopting AI-DLC. We believe the following 2 approaches will make this easier.

a. Learning by Practicing – AI-DLC is actually a set of rituals (Mob Elaboration, Mob Construction etc.) that can be practiced as a group. Instead of learning the method via documentation and traditional trainings, we will get the practitioners to practice the rituals with the AI-DLC guides in multiple real world scenarios that are currently being solved by the practitioners. The AWS Solution Architects have created a field offering called AI-DLC Unicorn Gym that packages this approach for hyper-scaling adoption in large organizations.

b. By embedding AI-DLC in the new Developer Experience Tooling – our customers are building their own orchestration tools that cut across SDLC, providing a unified experience for their developers. (ex. FlowSource from Cognizant, CodeSpell by Aspire, AIForce by HCL etc.) By embedding AI-DLC in these tools, the developers in large organizations will seamlessly practice AI-DLC without any need for significant adoption drives.

APPENDIX A

The following prompts can be used to interact with AI for practicing AI-DLC.

##Setup Prompt

We will work on building an application today. For every front end and backend component we will create a project folder. All documents will reside in the aidlc-docs folder. Throughout our session I'll ask you to plan your work ahead and create an md file for the plan. You may work only after I approve said plan. These plans will always be stored in aidlc-docs/plans folder. You will create many types of documents in the md format. Requirement, features changes documents will reside in aidlc-docs/requirements folder. User stories must be stored in the aidlc-docs/story-artifacts folder. Architecture and Design documents must be stored in the aidlc-docs/design-artifacts folder. All prompts in order must be stored in the aidlc-docs/prompts.md file. Confirm your understanding of this prompt. Create the necessary folders and files for storage, if they do not exist already.

##Inception

User stories

Your Role: You are an expert product manager and are tasked with creating well defined user stories that becomes the contract for developing the system as mentioned in the Task section below. Plan for the work ahead and write your steps in an md file (user_stories_plan.md) with checkboxes for each step in the plan. If any step needs my clarification, add a note in the step to get my confirmation. Do not make critical decisions on your own. Upon completing the plan, ask for my review and approval. After my approval, you can go ahead to execute the same plan one step at a time. Once you finish each step, mark the checkboxes as done in the plan.

Your Task: Build user stories for the high-level requirement as described here << describe product descrition>>

<<<After reviewing and changing the plan>>>>

Yes, I like your plan as in the <<md file>>. Now exactly follow the same plan. Interact with me as specified in the plan. Once you finish each step, mark the checkboxes in the plan.

Units

Your Role: You are an experienced software architect. Before you start the task as mentioned below, please do the planning and write your steps in the units_plan.md file with checkboxes against each step in the plan. If any step needs my clarification, please add it to the step to interact with me and get my confirmation. Do not make critical decisions on your own. Once you produce the plan, ask for my review and approval. After my approval, you can go ahead to execute the same plan one step at a time. Once you finish each step, mark the checkboxes as done in the plan.

Your Task: Refer to the user stories mvp_user_stories.md file. Group the user stories into multiple units that can be built independently. Each unit contains highly cohesive user stories that can be built by a single team. The units are loosely coupled with each other. For each unit, write their respective user stories and acceptance criteria in individual md files in the design/ folder.

<<<After reviewing and changing the plan>

I approve. Proceed.

###Construction

Domain (component) model creation

Your Role: You are an experienced software engineer. Before you start the task as mentioned below, please do the planning and write your steps in an design/component_model.md file with checkboxes against each step in the plan. If any step needs my clarification, please add it to the step to interact with me and get my confirmation. Do not make critical decisions on your own. Once you produce the plan, ask for my review and approval. After my approval, you can go ahead to execute the same plan one step at a time. Once you finish each step, mark the checkboxes as done in the plan.

Your Task: Refer to the user stories in the design/seo_optimization_unit.md file. Design the component model to implement all the user stories. This model shall contain all the components, the attributes, the behaviours and how the components interact to implement the user stories. Do not generate any codes yet. Write the component model into a separate md file in the /design folder.

<<<After reviewing and changing the plan>>>>

I approve the plan. Proceed. After completing each step, mark the checkbox in your plan file.

##: Code Generation

Your Role: You are an experienced software engineer. Before you start the task as mentioned below, please do the planning and write your steps in an md file with checkboxes against each step in the plan. If any step needs my clarification, please add it to the step to interact with me and get my confirmation. Do not make critical decisions on your own. Once you produce the plan, ask for my review and approval. After my approval, you can go ahead to

execute the same plan one step at a time. Once you finish each step, mark the checkboxes as done in the plan.

Task: Refer to component design in the search_discovery/nlp_component.md file. Generate a very simple and intuitive Python implementation for the Natural Language Processing (NLP) Component that is in the design. For the processQuery(queryText) method, use amazon bedrock APIs to extract the entities from the query text. Generate the classes in respective individual files but keep them in `vocabMapper` directory.
Refer to the generated codes in vocabMapper directory. I want the EntityExtractor component to make a call to GenAI. The current implementation uses the local vocabulary_repository. Can you analyse and give me a plan on how I can leverage GenAI for both the Entity Extraction and Intent Extraction.

##: Architecture

Your Role: You are an experienced Cloud Architect. Before you start the task as mentioned below, please do the planning and write your steps in a deployment_plan.md file with checkboxes against each step in the plan. If any step needs my clarification, please add it to the step to interact with me and get my confirmation. Do not make critical decisions on your own. Once you produce the plan, ask for my review and approval. After my approval, you can go ahead to execute the same plan one step at a time. Once you finish each step, mark the checkboxes as done in the plan.

Task: Refer component design model: design/core_component_model.md, units in the UNITS/ folder, cloud architecture in the ARCHITECTURE/ folder, and backend code in the BACKEND/ folder. Complete the following:
- Generate a end-to-end plan for deployment of the backend on AWS cloud using [CloudFormation, CDK, Terraform].
- Document all the pre-requisites for the deployment, if any.
Once I approve the plan:
- Follow the best practice of clean, simple, explainable coding.
- All output code goes in the DEPLOYMENT/ folder.
- Validate that the generated code works as intended, by creating a validation plan, generate a validation report.
- Review the validation report and fix all identified issues, update the validation report.

##: Build IaC/Rest APIs

Your Role: You are an experienced software engineer. Before you start the task as mentioned below, please do the planning and write your steps in an md file with checkboxes against each step in the plan. If any step needs my clarification, please add it to the step to interact with me and get my confirmation. Do not make critical decisions on your own. Once you produce the plan, ask for my review and approval. After my approval, you can go ahead to execute the same plan one step at a time. Once you finish each step, mark the checkboxes as done in the plan.

Task: Refer to the services.py under the construction/<>/ folder. Create python flask apis for each of the service there.