ChatGPT

育てるほど楽になるAI開発体制の技術インフラ解説

背景と課題

DeNAのスポーツ新規サービス開発プロジェクトでは、**AIを活用した開発生産性向上**に挑戦しています [1] [2]。しかし、当初のチーム状況では次のような課題がありました [3]:

- **コンテキスト（文脈）の共有不足**: README以外に設計情報や設定がまとまっておらず、AI（生成AIツール）の出力品質が安定しない [4]。
- **コード規約違反の発生**: プロジェクト固有のコーディング規約や暗黙知をAIが把握していないため、生成コードがガイドライン違反となるケースがあった [4]。
- **コードレビュー精度の低下**: 全社導入されていた自動レビューBOT（PR-Agent）は存在したものの、プロジェクト固有の文脈を与えていないため汎用的な指摘しかできず、有用性が限定的だった [5]。

このような課題に対し、「**人間のレビュー量を減らす**」ことと「**成果物の品質を標準化する**」という二本柱でワークフローを再設計しました [6]。つまり、機械的なチェックはAIによる一次レビューに任せ、人間は本質的なビジネスロジックやアーキテクチャの検証に集中できるようにする一方、AIが参照するガイドラインやドメイン知識を整備して**メンバーの経験に依存せず一定品質のコードを高速に生み出せる**環境を目指しました [7]。

コア技術と全体アーキテクチャ

解決策として、チームは**AIをプロジェクト文脈を熟知したチームメンバーの一員**とみなし、知識共有基盤をリポジトリ内に構築しました [8]。具体的には、**Anthropic社のLLM「Claude」を用いたAIコーディング支援ツール（Claude　Code）**や、AI統合開発環境の**Cursor**を活用しつつ、AIが参照すべき情報をリポジトリで一元管理しています [9] [10]。

リポジトリ内の**ディレクトリ構成**は以下のようになっており、ここにAI用コンテキストや設定を集約しています [11] [12]（内容を更新すればAIの出力も向上します [13]）:

- **プロンプトメモリ (`CLAUDE.md`)** – プロジェクト概要や手順を記載し、Claudeの長期メモリとして機能 [14]。
- **共通コンテキスト (`AGENTS.md`)** – プロジェクトで使う技術スタックや用語など、全エージェント共通の知識ベース [15]。
- **サブエージェント定義 (`.claude/agents/`)** – 特定タスク専用のAIエージェント役割定義。例えば `api-spec-reviewer.md`（API仕様レビュー担当）、`code-reviewer.md`（Goコードレビュー担当）、`e2e-test-implementer.md`（E2Eテスト実装担当）など [16]。各ファイルにエージェントの人格や任務（プロンプトテンプレート）を記述します。
- **カスタムコマンド定義 (`.claude/commands/`)** – AIに実行させる作業シナリオを定義。例: `pr-create.md`（AIがプルリクエストを作成する手順）, `code-review.md`（コードレビュー実行手順）等 [17]。
- **スキル定義 (`.claude/skills/`)** – AIエージェントが利用できる汎用機能モジュール群。例: `go-code-generation`（Goコード生成）, `github-issue-analysis`（Issue分析）等 [18]。エージェントからこれらのスキルを呼び出すことで共通処理を再利用します。

- **ドキュメント (`docs/`)** – コーディング規約やAPI設計ガイドライン、Firestore利用指針、プロジェクト固有知識などをMarkdown文書で格納 [12]。AIはこれらを参照してコード生成・レビューを行います。

こうした構成により、AIが汎用的知識だけでなく**プロジェクト固有のコンテキスト**を常に参照できるようになりました [10]。特にドキュメント類やガイドラインを充実させて**コンテキストを一元管理**することで、AI出力のブレを抑え、コード品質の均一化を図っています [10] [19]。

AI活用ワークフローの自動化

チームでは**実装・レビュー・ナレッジ蓄積**の3フェーズにAIを組み込んだワークフローを運用しています [20] [21]。各フェーズで適切なAIエージェントやツールが動作し、時間経過とともに**「育てるほど楽になる」開発サイクル**を実現しています。

1. **実装フェーズ** – 開発者はローカル開発環境でCursorやClaude Codeを活用してコーディングします [22]。AIはリポジトリ内のドキュメントや設定ファイルをコンテキストとして参照し、必要に応じてコードベースの情報を取得して開発者を支援します [23]。

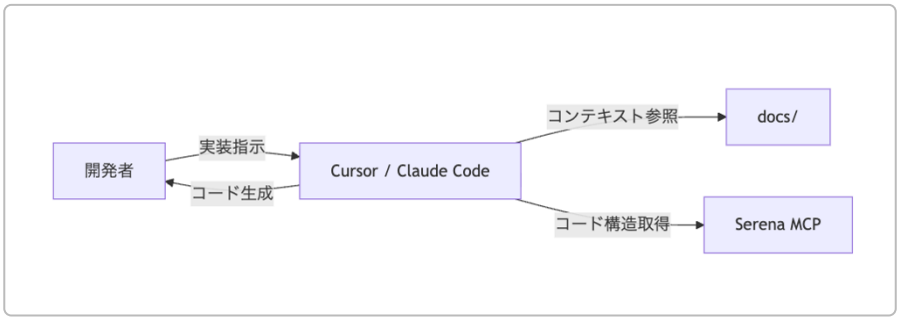

図: 実装フェーズにおけるAI支援の概念図。開発者の指示に対し、AI（Cursor/Claude Code）がリポジトリ内の文脈資料（docs配下）やコード構造（Serena MCP経由）を参照しながらコードを提案・生成する。

2. **レビューフェーズ** – 開発者がプルリクエスト(PR)を作成すると、GitHub Actions経由で**Claude Code Actions**（Anthropic Claudeを用いた自動レビューエージェント）が起動し、変更内容の自動レビューを行います [24]。変更ファイルの種類（OpenAPI仕様書や`.go`ソースなど）に応じて、事前に定義した該当分野の**サブエージェント**が選択され、コード規約違反やベストプラクティスからの逸脱を指摘します [25]。これにより機械的な指摘はAIが代行し、レビュアーは本質的なロジックや設計の議論に集中可能です [26]。
3. **ナレッジ蓄積フェーズ** – 人間のコードレビューで出た指摘事項は、**自動ドキュメント更新フロー**によって即座に知識ベースに反映されます [27]。具体的には、レビューコメントに対し所定のメンションを付けて返信すると、それをトリガーにClaude Code Actionsが議論内容を分析し、該当するガイドラインドキュメント（例：`docs/coding_guidelines.md`）を更新する新たなPRを自動生成します [28]。このPRを人間が確認・マージすることで、**「レビュー指摘 → ドキュメント更新 → AI出力精度向上」**というフィードバックループが回り続ける仕組みです [28]。
   図: ナレッジ循環フェーズの自動ドキュメント更新フロー。レビューコメントへの特定メンションをトリガーに、Claude Code Actionsが規約ドキュメントへの追記内容を提案し、新規PRとして作成する。開発者がそれを確認・マージすることでガイドラインが進化し、次回以降のAI支援品質が向上する。

上記のサイクルにより、AIシステムはプロジェクトが進むほど賢くなり、開発が**「育てるほど楽」**になっていきます。

特化型サブエージェントとプロンプト管理

汎用のAIアシスタントに何もかも任せるのではなく、**タスクごとに最適化されたサブエージェント**を定義した点が技術的ポイントです。すべての知識を一つの巨大なプロンプトに詰め込むと文脈が肥大・汚染してしまうため、「**小さく単一責任のエージェント**」とする原則に従ってエージェントを分割しました [29]。例えば以下のような役割別エージェントを用意しています [16]：

- **API仕様レビュアー (`api-spec-reviewer`)** – OpenAPI仕様書の構造や命名規則が社内ガイドラインに厳密に準拠しているかチェック [16]。
- **コードレビュアー (`code-reviewer`)** – Go言語のベストプラクティス違反やプロジェクト固有の設計パターン逸脱を検出 [16]。
- **E2Eテスト実装支援 (`e2e-test-implementer`)** – 複雑なエンドツーエンドテストの実装を専門に行うエージェント [16]。
- （他にもバグ修正提案などのエージェント定義あり）

各サブエージェントは、それぞれMarkdown形式の定義ファイルに**役割指示（プロンプト）**が記述されています。例えばAPIレビューエージェントの定義では、「あなたはOpenAPI仕様とRPC　API設計のエキスパートレビュアーです。主な責務はOpenAPI仕様書とAPI設計をレビューし、`docs/api_development_guideline.md` 記載のガイドラインに**厳密に準拠**していることを確認することです…」というように、**参照すべきガイドラインやレビュー観点が詳細に記載**されています [30] [31]。さらにレビュー手順や出力フォーマットまで定義し、重要度ラベル付きで問題点・該当箇所・ガイドライン参照・修正提案を含むフィードバックを出すよう指示しています [32] [33]。このように**プロンプトテンプレートを明示的に管理**することで、AIの挙動をチームのコーディング規約や品質基準に合致させています。

プロンプトやエージェント定義をリポジトリでバージョン管理しているため、チーム全員が同じAI設定を共有できます。新メンバーでもこれら定義に基づいたAI支援を受けられるため、**経験やスキルに依存せず一定水準の成果物を素早く出力**できる環境になっています [19]。いわば**プロンプトエンジニアリングの成果物を資産化**し、再利用可能にしている点が特徴です。

自動コードレビュー基盤（Claude Code Actions）

人間のレビュー負担を減らすために、GitHub　Actions上に**自動コードレビューのパイプライン**を構築しました。PR作成時にワークフローがトリガーされ、変更ファイルに応じて前述のサブエージェントを呼び出しAIレビューを実施します [25]。これはAnthropic ClaudeのAPIを活用した社内ツール「Claude Code Actions」によって実現されており、あたかもレビュアーのようにPRにコメントを投稿する仕組みです [25]。

自動レビューコメントの例として、あるPRではClaude Code Actionsが以下のような指摘を返しました [34]：

> **重要 (Critical)**: XXX関数でビジネスロジックが重複しています。コードの重複は保守性を下げるため削除を推奨します。【docs/coding_guidelines.md　L34-L36】同一ロジックは既にYYY関数で実装されています。…（以下略）

このように指摘内容には**重要度ラベル**が付き、問題の具体的箇所や違反している社内ルールの参照、修正の提案まで含まれます [26]。機械的なコーディング規約違反や単純ミスの洗い出しはすべてAIが担ってくれるため、**人間のレビュアーはガイドラインチェックから解放され、本質的な設計・アーキテクチャのレビューに専念可能**となりました [26]。自動レビューエージェントにより**レビュー回数そのものも大幅削減**されており、導入前は平均7.2回/PRあった人間のレビューが2.7回/PRまで減少しています [35]。コメント数も平均6.0件から1.9件に減り、レビューのノイズが大きく低減しました [35]。

大規模コードベースでのコンテキスト取得（Serena MCP）

モノレポ規模の巨大なコードベースでは、AIに必要な情報を適切に与えることが新たな課題でした。従来のキーワード検索や単純なファイル走査では目的の関数や型定義を見つけ出せず、AIの回答精度が下がる恐れがあります [36]。そこでチームは**Language Server Protocol (LSP)**ベースのコード構造サーバー「**Serena MCP**」を導入しました [37]。Serena MCPはコード上の**シンボル（関数・クラス名など）や依存関係をインデックス化**し、AIのクエリに対して即座に該当箇所を提示できるツールです [38]。例えば、AIがある関数の実装箇所や参照箇所を必要とした場合、Serenaを通じて該当コードを素早く取得できます。これによりAIは**コードベースの正確なコンテキストを把握**できるようになり、回答や生成コードの質・速度が大きく向上しました [39]。

なお、本プロジェクトでは特にコードコンテキスト取得にSerenaを活用していますが、一般的なアプローチとしては**ベクトルデータベース**を使ったドキュメント検索なども考えられます。今回のケースではコードシンボルを扱うためLSPを選択しましたが、いずれにせよAIが高速に必要情報へアクセスできる基盤を整備することが重要と言えます。

ナレッジ循環による継続的改善

AI開発体制を**育てていく**うえで鍵となるのが、前述した**ナレッジ循環フィードバックループ**です。人間の指摘事項を都度ドキュメントに反映しAIに学習（厳密には明示的な知識追加）させることで、時間の経過とともにAIの出力品質が向上し開発が楽になります [28] [40]。この仕組みにより**チームの知見が蓄積・共有**され、新人メンバーであっても過去の知見が反映されたAIサポートを享受できます。結果として、「使い捨てのAI」ではなく**「育てるAI」**として運用できている点が特徴です [41]。

例えば、コードレビューで「特定のケースでAPI仕様に漏れがある」という指摘があった場合、その内容が自動でAPI開発ガイドラインに追記されます [28]。次回以降、AIは更新後のガイドラインを参照するため、同様の漏れを事前に防ぐ提案が可能になります。**レビューを経るごとにAIが賢くなる**このループにより、時間の経過とともに開発効率も品質も右肩上がりで向上していきます [40] [42]。

課題への対策まとめと効果

以上の技術基盤により、当初の課題は次のように解決されました。

- **コンテキスト共有の徹底**: ドキュメントやエージェント定義をリポジトリで一元管理することで、AIが常に最新のプロジェクト文脈を参照可能になりました [10] [43]。これによりAI出力のばらつきが減り、チーム全員が統一された知識基盤の下で開発できます。
- **コード品質の標準化**: コーディング規約や設計パターンをAIが参照するようにし、さらに各種エージェントで専門チェックを行うことで、経験の浅いメンバーでも一定水準以上のコードを生成可能になりました [19] [32]。AI自身もガイドライン違反を検知・修正提案するため、コードの品質が自ずと組織標準に揃います [44]。
- **レビュー工数の削減**: 自動レビューが導入されたことで、人間のレビュー回数・コメント数は約**60〜70%削減**されました [35]。レビューの**ノイズ（機械的指摘）が除去**され、本質的な議論に集中できています。
- **ナレッジ蓄積と継続的な改善**: レビュー知見をドキュメント化するプロセスにより、AIを含むチーム全体の知識レベルが継続的に向上しています [28] [42]。開発が進むほどルールや知識ベースが充実し、AIの有用性も増していきます。

こうした効果により、実際に**「レビュー負荷の軽減」「コード品質の均一化」「ナレッジ共有の促進」**といった成果が得られました [35]。特にAIによる機械的チェック完結により、人間はよりクリエイティブな部分に注力できるようになった点は大きなメリットです。

まとめ

DeNAの事例では、生成AI（LLM）の力を個人の補助に留めず**チーム全体で育成・活用する開発体制**を築きました。その技術的鍵となったポイントは以下の3点です [45]:

- **コンテキストの一元管理**: プロジェクト文脈となるドキュメントやエージェント設定をリポジトリで管理し、AIに常に正確な背景知識を与える [43]。
- **サブエージェント & スキルによる分業**: タスクごとに専門エージェントを定義し、複数のAIエージェントが協調して開発を支援する [43]。
- **ナレッジ循環の仕組み化**: 開発プロセス内にフィードバックループを組み込み、AIのアウトプット精度を継続的に改善していく [43]。

経験豊富なエンジニアにとっても、このようなAI開発インフラは**MLOps的な観点**で示唆に富むものです。モデルそのものを学習・更新する代わりに、**プロンプトやコンテキストを管理・改善する**ことで、AIを組織の知的資産として育成できることを示しています。今後はプロダクトマネージャーなど非エンジニア職種にもAI支援が広がりつつあり、職種横断でビジネス価値を高める協働体制への発展も期待されています [46]。今回の取り組みは、AI時代におけるチーム開発の一つのモデルケースと言えるでしょう。

---

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [42] [43] [44] [45] [46] 育てるほど楽になる AI 開発体制を作っている話 | BLOG - DeNA Engineering
https://engineering.dena.com/blog/2026/01/ai-driven-develop/

[41] 育てるほど楽になる AI 開発体制を作っている話 - DeNA Engineering
https://jp.linkedin.com/posts/dena_%E8%82%B2%E3%81%A6%E3%82%8B%E3%81%BB%E3%81%A9%E6%A5%BD%E3%81%AB%E3%81%AA%E3%82%8B-ai-%E9%96%8B%E7%99%BA%E4%BD%93%E5%88%B6%E3%82%92%E4%BD%9C%E3%81%A3%E3%81%A6%E3%81%84%E3%82%8B%E8%A9%B1-blog-dena-engineering-activity-7415227423660621824-5KhT