コーディングエージェントの現状と課題解決

AI品質マネジメント夏合宿2026　コーディングエージェントグループ

コーディングエージェントとは

- 自然言語の指示を受けて、仕様整理、設計、実装、テスト、修正を支援・実行するAIエージェント。
- 単なるコード補完ではなく、ファイル操作、調査、実行、テスト、修正提案まで行う自律的な開発支援環境。
- 必ずしもコーディング以外の、資料整理などに使うことも多い。今後は、コーディング専用ツールと汎用AIエージェントの境界が曖昧になっていく。
- 利用者は、詳細な実装指示ではなく、目的や要求を言葉で伝えて開発を進める。
  - ベテランが使う場合は、壁打ち相手や高速な実装補助として機能する。
  - 素人が使う場合は、ブラックボックス的に成果物を受け取る形になりやすい。

課題

1. 出力の品質を利用者が判断できない
   - エージェントは仕様・設計・コード・テストまで生成できるが、出力はブラックボックスになりやすい。
     ベテランは途中で修正・レビューできるが、素人は結果を見ることしかできない。
     そのため、動くものが出ても品質・安全性・保守性を判断できない。
2. 大規模・長期的な開発にスケールしにくい
   - 大規模開発では、要求を機能やモジュールに分割し、仕様として落とす必要がある。
     OSやコンパイラのような大きな抽象化を必要とする設計は、現状のAIだけでは難しい。
     AIが生成した構造を人間が理解・検証できるかも課題になる。
3. ベテラン依存と人材育成の問題
   - 品質担保にはベテランのレビュー観点やコーナーケース確認が必要だが、常に配置するのは難しい。
     一方で、AIがジュニアのOJT機会を奪うと、将来のベテランを育てにくくなる。
     ベテランの知見をAI Agent環境に移植し、素人や新人が学びながら安全に使える仕組みが必要。

課題1：コード品質の担保

- コーディングエージェントの使われ方
  - プログラミングのベテランが使う
    - 生成されるコードを人が確認しながら効率的にプログラムを書く
    - 大域アーキテクチャ設計、品質ゴールや担保の**役割は人間の方**にある
    - コード書きの高速化や、資料調査からのコード書き起こしが主なAIの役割
    - Human-machine Teaming（Human Mentor or Coworking）
  - 必ずしもその分野のプログラミングに精通していない人が使う
    - ブラックボックスのエージェントとして使う
    - **最終的なコードの質もAIの出力に依存**
    - 目的や要求を言葉で伝えてコードを完全に任せる
    - 出てきたコードを必ずしも理解してレビューできないこともある
    - 適切なアーキテクチャ設計などは
    - 行われているか疑わしい
  - **こちらの課題が大きい**

課題1：コード品質の担保

- 当面の対策の方向性
  - ベテランが通常行う「要求確認・前提確認・設計レビュー・コーナーケース確認・危険操作の停止」を、初心者の標準プロセスとして**ガイド化する**。
  - AIにすぐにコードを書かせず、用途、利用者、扱うデータ、公開範囲、失敗時の影響を確認し、未確認事項とAIが置いた仮定を明示してから実装に進む。
  - 専門用語で質問せず、**「分からない」を正式な回答として扱う**。分からない場合は安全側の既定値を採用する。
  - コード生成前後に、設計意図、変更範囲、リスク、テスト観点、未検証事項を要約し、初心者でもレビュー依頼や判断ができる形にする。
  - ファイル削除、外部通信、認証情報アクセス、本番変更、DB一括更新などの危険操作は、AIが自動実行せず、理由・影響範囲・代替案を示して確認を求める。
  - コード生成前後に、設計意図、変更範囲、リスク、テスト観点、未検証事項を要約し、初心者でもレビュー依頼や判断ができる形にする。

- 将来課題：コーディングエージェント環境自体の改良
  - **ベテランのHMTノウハウを、AI Agentを介して素人向けコーディング環境に移植する。**
  - 玄人が持つレビュー観点、設計判断、失敗事例、コーナーケース確認を、質問テンプレート、ポリシー、ガードレールとしてエージェント環境に組み込む。
  - 初心者モード、慎重モード、教育モード、専門家モードなどを標準機能として実装し、**利用者の理解度と対象システムのリスクに応じて挙動を切り替える**。
  - エージェントが出す質問そのものについて、玄人が監修し、抜け漏れ、分かりにくさ、誘導的な質問、安全上重要な確認の欠落を継続的に改善する。
  - 高リスクな開発では、AIが自動的にレビュー依頼用サマリを作成し、玄人・セキュリティ担当・品質保証担当の確認に接続する仕組みを整備する。
  - 生成コードだけでなく、要求、前提、設計判断、却下した代替案、テスト観点、未解決事項を開発環境内に記録し、後から追跡できるようにする。
  - 組織やドメインごとの設計規約、セキュリティ基準、運用ルールをエージェントが参照できる形で管理し、玄人が継続的に更新・監査する。

課題2：大規模なシステム開発への適用

- 問題
  - 大規模システムでは、要求をそのままAIに渡しても扱いきれない。機能、モジュール、インターフェース、データ構造、権限、運用境界に分割し、段階的に仕様化する必要がある。
  - アーキテクチャや機能分割が不十分なまま大きな要求を与えると、**AIの処理能力やコンテキスト範囲を超過**し、整合したコード生成がうまくいかなくなる。
  - OSやコンパイラのような大きな抽象化を必要とするシステムでは、AIがコードを積み上げるだけでは十分でない。人間がアーキテクチャ上の方針を与え、AIは各モジュールの具体化や検証補助を担う形が現実的である。
  - 大規模開発では、AIが生成した構造を人間が理解・検証できること自体が品質条件になる。設計意図、依存関係、責任分界、未解決事項を記録し、後から追跡できる開発プロセスが必要である。

課題2：大規模なシステム開発への適用

- 現状の（玄人開発での）進め方
  - 最初に**人間がシステム全体の目的、品質ゴール、主要モジュール、境界条件を定め**、AIにはその範囲内で詳細化・実装させる。
  - アーキテクチャ設計、モジュール分割、インターフェース定義、データモデル設計を（時にはAIとの壁打ちで）作成し、各段階で人間が確認する。
  - AIに各モジュールの仕様、テスト観点、依存関係、未確認事項を出させ、モジュール単位でレビュー・統合できる形にする。
  - 大規模な変更では、AIが変更範囲、影響範囲、後方互換性、既存仕様への影響を必ず説明するようにする。
  - 小さなブロックの積み上げで済む領域と、大きな抽象化や設計判断が必要な領域を分け、後者には玄人の関与を必須にする。

- 将来課題：大規模開発向けエージェント環境の整備
  - **要求分割の方法論を整理**する。大規模な要求を、どの単位で機能・モジュール・タスクに分けるべきかを整理し、AIが扱える粒度と人間が理解・レビューできる粒度の両方を満たす基準を作る。
  - 人間とAIの役割分担を定義する。どこまでを人間がアーキテクチャとして決め、どこからAIに詳細化・実装させるかを明確にし、大きな抽象化や責任分界は人間が関与すべき領域として切り出す。
  - コード生成後の工程まで含めた分割基準を作る。分割単位は実装しやすさだけでなく、レビュー、統合、テスト、運用のしやすさまで考慮し、小さすぎず大きすぎない中間粒度を定める。
  - **これらをAgentic AIに実行させる方法、能力、ノウハウを開発**していく。AIが自身の得手不得手を把握して、人と協働してアーキテクチャ設計を進めていく環境を作っていく。
  - 複数のモジュールを並行・分担して開発する場合の責任分界、整合性確認、統合レビューの方法を整備する。
  - 従来のバグ数やテストカバレッジだけでなく、設計意図、変更履歴、依存関係、レビュー可能性もAI・人双方が見る品質指標に含める。

課題3：ベテラン消失・OJT・知識継承

- 問題
  - AIがジュニアエンジニアの仕事を代替すると、OJTとして失敗し、レビューを受け、設計判断を学ぶ機会が失われる。
  - 短期的には開発速度が上がっても、将来のベテランを育てる経路が細り、OS設計や大規模アーキテクチャのような暗黙知が継承されにくくなる。
  - 教科書化されていないアプリケーション設計やノウハウは、ベテランがいなくなると再構成が困難になる。
  - 一つの方向性として、完全にAIに依存する極端な方向も有り得るが、確実に将来を任せる方向性としては不安がある。
- 現状での対策：**OJT機会を完全にAIへ置き換えない**
  AIに丸投げせず、要求整理・設計理由の説明・レビュー対応など、人間が学ぶ工程を残す。
  - ベテランレビューを重点投入する
    全量レビューではなく、設計判断・責任分界・セキュリティ・コーナーケースなど重要箇所に絞る。
  - AIに説明させてから使わせる
    生成コードの前提・設計意図・リスクを説明させ、利用者が理解・要約してから使う。
  - 失敗・レビューコメントを学習資産として残す
    AIの誤りやレビュー指摘をチェックリストやプロンプトに蓄積し、知見を明文化する。
  - 教育用タスクと本番タスクを分ける
    本番ではAIを活用しつつ、教育目的では人間が設計・実装・レビューを経験する場面を意図的に残す。
- 将来課題：**知識継承型エージェント環境の整備**
  - ベテランの暗黙知をどう抽出し、レビュー観点、設計原則、教育コンテンツとして構造化するかが課題である。
  - AIが生成した高度な抽象化や設計を、人間が理解できる形で整理・教材化する仕組みが必要である。
  - 新人がAIに依存するだけでなく、徐々にレビュー観点を身につけているかを評価する方法が必要である。
  - 組織内で失敗事例、レビューコメント、設計判断を継続的に蓄積し、AI Agent環境を更新していく。