AI との構造化された協業：現代ソフトウェア開発のためのフレームワーク

Fumio SAGAWA
（原文著者）

Abstract

本稿では、AI がコードを記述するようになった現代における人間のソフトウェアエンジニアの役割を再定義し、AI を単なる自動化ツールとしてではなく、真の協業相手として捉えるための構造的なアプローチを提案します。実際のプロジェクトで使用されている、以下の 2 つの重要なモデルを探求します。

1. 責任を分割するための 3 層構造
2. 開発をシステムとして定義するための 6 つの要素フレームワーク

これらのモデルを通じて、人間と AI がどのように連携し、より効果的なソフトウェア開発を実現できるかを示します。

1　はじめに：自動化を超えて

今日、AI は驚異的なスピードでコードを生成しています。この新たな現実において、人間のソフトウェアエンジニアの役割は何でしょうか？

多くの開発者は未だに AI を単なるツールとして利用しています。コードを素早く書いたり、バグを修正したり、構文を説明させたり、といった使い方です。これらは確かに便利ですが、それは AI 活用のほんの始まりに過ぎません。

では、もし AI が私たちの設計プロセスに参加し、チーム内の役割を理解し、開発自体を構造化する手助けをしてくれるとしたらどうでしょう？

この「ツール」から「参加者」へのシフトこそが、AI との協業を真に面白くする転換点なのです。

2　開発の 3 層モデル

役割分担を明確にするために、私は開発を以下の 3 つの層に分ける構造を提案します。

- **意図（Intention）層**：ビジネスロジック、ユーザーニーズ、設計の意図など、「何を達成したいか」に関わる部分です。これは主に人間の責任領域となります。人間の深い理解、創造性、倫理的な判断が求められます。
- **実行（Execution）層**：コード生成、変換、自動化など、「それをどのように実現するか」に関わる部分です。AI が最も得意とする領域です。高速かつ正確なコード生成や繰り返し作業の自動化を担います。
- **評価（Evaluation）層**：テスト、イテレーション（繰り返し改善）、フィードバックループなど、「それがどれくらいうまくいっているか」を検証する部分です。これは人間と AI が共有することも、人間が主導することもあります。AI はテストコードの生成や結果の分析を手伝い、人間は最終的な品質判断や改善方向の決定を行います。

このモデルは、「何を求めているか (What we want)」、「それをどう実現するか (How to do it)」、「それがどれくらいうまくいっているか (How well it works)」を分離します。これは AI 対人間という対立構造ではなく、役割の構造化と分離による協業のアプローチです。

3　6 つの要素フレームワーク

さらに踏み込んで、私は開発タスクを 6 つの要素に抽象化して考えます。これを「6 つの要素フレームワーク」と呼んでいます。

- **シナリオ（Scenario）**：ビジネスの流れ。入力 → 処理 → 出力という一連のユーザー操作やシステム動作を表します。これはユーザー体験全体やビジネス要件の理解が必要なため、人間の得意とする領域です。
- **パラメータ（Parameter）**：技術的な前提条件。使用するプログラミング言語、ライブラリ、フレームワーク、コーディング規約などを定義します。これは人間が初期設定を行いますが、AI はこれらの規約に従ったコード生成で貢献できます。
- **コア（Core）**：データ構造、モデル、ドメイン設計。アプリケーションの核となるデータ表現やビジネスロジックの設計です。AI は定義されたモデルに基づいてコード（クラス定義やデータアクセスコードなど）を生成するのに優れています。
- **オペレーター（Operator）**：ユーザーの役割、権限、UI の振る舞い。誰が何にアクセスでき、UI がユーザーの操作にどう反応するかといったインタラクションに関わる部分です。ユーザー体験の設計やセキュリティの考慮が必要なため、人間の判断が重要です。
- **リンク（Links）**：API 連携、外部システムとの通信。他のサービスやデータベースとの接続部分です。AI は既存の API 仕様に基づいた連携コードの生成や、通信プロトコルに則った実装をサポートできます。
- **アウトプット（Output）**：フロントエンドのビュー、レスポンス、ドキュメント。ユーザーが目にする画面表示や、API からの応答データ、各種ドキュメント生成です。AI は定義されたデザインやデータ構造に基づいて UI コンポーネントやレスポンス形式のコードを生成するのに適しています。

これらの各要素は、それぞれの得意な領域に基づいて AI または人間に明確に割り当てることができます。例えば、AI は「コア」や「アウトプット」のコード生成は得意ですが、「シナリオ」や「オペレーター」のように人間の判断や深い理解が必要な部分には苦戦します。

このフレームワークを使うことで、開発は協力して実行可能なシステムへと分解されます。

これはプロンプトエンジニアリングの本質的な構造でもあります。もし、これらの 6 つの要素を整理せずにプロンプトを書いているなら、それは曖昧な指示を与えているに過ぎません。試してみれば、それがどれほど明らかかすぐにわかるでしょう。

4　チームメンバーとしての AI を用いたアジャイル開発

アジャイル開発は単にデリバリーの速さだけを追求するものではありません。フィードバック、イテレーション、そして協業がその核心にあります。

では、AI を単なるツールではなく、コードを書き、リファクタリングし、応答することができる共同開発者として扱ってみたらどうでしょうか？

実際の Agile チームにおいて、私は AI に特定の役割を割り当て始めました。UI コンポーネントの実装、ドメインロジックのコードへの変換、ボイラープレートコードの生成などです。これはうまくいっています。

そしてさらに重要なことに、それはチームダイナミクスそのものを変えます。私たちは、常に稼働していて

電光石火のように速いチームメイトのように、AI を念頭に置いて設計を始めるようになるのです。

5　アドバイス：ツールで止まらない

もしあなたが AI をコード補完やスニペット作成にだけ使っているとしても、それはそれで良いでしょう。

しかし、それは始まりに過ぎません。

AI の可能性を最大限に引き出すには、それを開発プロセスに組み込みましょう。チームメンバーとして扱いましょう。その役割を定義しましょう。その出力を評価しましょう。そして、それと共にイテレーション（改善を繰り返し）しましょう。

その時に、本当の変革が起こるのです。

6　結論

ソフトウェア開発は変化しています。

しかし、私たちは置き換えられるわけではありません。私たちは再配置されているのです。

層を分離し、要素を定義することで、AI が最も得意とする領域で AI を活用し、人間のスキルが最も重要となる領域に焦点を当てることができます。

未来は AI 対人間ではありません。

それは AI **と共に**人間が、共に（together）より良いものを構築する未来です。