● 第5章 開発リソースは足りないのに増え続ける重圧

5-1 AIを「コードを早く書く道具」以上のパートナーに

新たな課題の顕在化

改善活動を始めて1ヶ月が経過した。湊のチームは確実に変化していた。

要件定義の事前レビュー会が導入され、仕様変更は週に3-5回から週に1-2回に減った。週1回のリファクタリング時間も確保され、技術的負債の返済が少しずつ進んでいる。朝会も定着し、チーム内のコミュニケーションは改善した。

でも、新たな課題が浮上していた。

金曜日の午後、田中部長から呼び出しがあった。

「湊君、改善活動は良いけど、開発スピードは大丈夫？」

田中部長の表情には、経営陣からのプレッシャーが滲んでいた。

「確かに、仕様変更は減りましたし、品質も向上しています。でも、丁寧にやると時間がかかるんです」

湊は正直に答えた。改善活動によって品質は向上したが、開発速度は以前と変わらないか、むしろ少し遅くなった気がする。

「経営陣からは、もっと早く機能を出してほしいと言われている。改善は良いけど、スピードも大事なんだ」

田中部長の言葉に、湊は返答に困った。品質と速度、どちらも大事なのに、どう両立すればいいのか。

部長からの新たなプレッシャー

チームミーティングで、湊は状況を共有した。

「部長から、開発スピードについて指摘がありました。改善活動で品質は向上したけど、速度は以前と変わらない。どうすればいいか、みんなで考えたいんです」

山田が腕を組みながら言った。

「人手を増やすしかないのかな？でも、採用には時間がかかるし、教育にも時間がかかる」

「予算的に厳しいです。経営陣からは、現有リソースでどうにかしてほしいと言われています」

飛鳥PMが困った様子で答えた。

湊は深く考え込んだ。リソースは増やせない。でも、開発速度を上げる必要がある。どうすればいいのか。

湊の疑問

その時、同僚のエンジニアが湊に声をかけた。

「湊さん、GitHub Copilot使ってる？コード書くの速くなるよ」

「GitHub Copilot...ですか」

「うん。AIがコードを自動生成してくれるから、開発速度が上がる。僕も最近使い始めたけど、確かに速くなった気がする」

湊は興味を持った。もしAIツールで開発速度が上がるなら、試してみる価値があるかもしれない。

GitHub Copilotとの初対面

その日の夜、湊はGitHub　Copilotを試してみた。確かに、コードを書くスピードは速くなった。AIが提案するコードをそのまま使えば、開発時間は短縮できる。

でも、湊は違和感を覚えた。

生成コードへの違和感

生成されたコードを見ると、確かに動く。でも、コードの品質はどうなのか。保守性は？可読性は？技術的負債を増やすことにならないのか。

湊は生成されたコードをじっくり見直した。確かに動くが、無駄な処理があったり、命名が不適切だったり、コメントが不足していたりする。そのまま使うと、技術的負債が増える可能性がある。

「これって本当に生産性向上なのか？」

湊は疑問に思った。速く書けても、後で直す時間が増えたら意味がない。品質を犠牲にして速度を上げるのは、第1章の失敗と同じではないか。

翌日のチームミーティングで、湊は疑問を共有した。

「GitHub　Copilotを試してみたんですが、確かにコードを書くスピードは速くなりました。でも、生成されたコードの品質に不安があります。そのまま使うと、技術的負債が増える可能性があると思うんです」

山田が頷いた。

「確かに、AIが生成したコードをそのまま使うのは危険だね。でも、AIをうまく使えば、もっと効果的な活用ができるかもしれない」

「もっと効果的な活用...ですか」

「うん。コード生成だけじゃなく、AIを技術的負債の検出に使ったり、テストケースの生成に使ったり、ドキュメント作成に使ったり。もっと戦略的に使えないだろうか？」

山田の言葉に、湊は興味を持った。AIを「コードを早く書く道具」として使うのではなく、「開発全体を効率化するパートナー」として使う。それは新しい視点だった。

解説：AIを「コードを早く書く道具」以上のパートナーに

湊が気づいたように、AIツールを「コードを早く書く道具」として使うだけでは、本質的な生産性向上にはつながりません。

なぜコード生成だけでは不十分なのか

コード生成ツールを使うと、確かにコードを書くスピードは上がります。しかし、以下の問題が発生する可能性があります：

- **品質の低下**: 生成されたコードが無駄な処理を含んでいたり、命名が不適切だったりする
- **技術的負債の増加**: そのまま使うと、コードの複雑度が増し、将来的な開発速度が低下する
- **理解不足**: 生成されたコードを理解せずに使うと、バグの原因がわからなくなる

AIの戦略的活用とは

AIを戦略的に活用するとは、以下のような使い方をすることです：

- **技術的負債の検出:** AIにコードベースを分析させ、問題箇所を特定する
- **テストケースの生成:** AIに仕様書を読ませ、網羅的なテストケースを提案させる
- **ドキュメント作成支援:** AIにコードを読ませ、ドキュメントを自動生成する
- **コードレビュー支援:** AIにコードを分析させ、潜在的な問題を指摘させる

これらを組み合わせることで、開発全体の効率化が可能になります。

AIは「増幅器（Amplifier）」であり「鏡（Mirror）」である

2025年版DORAレポート『State of AI-Assisted Software Development』が発する最も強烈なメッセージは、「**AIは増幅器（Amplifier）である**」という概念です。データは、AIが組織に与える影響の二面性を明確に示しています。

強みの増幅

高パフォーマンスな組織がAIを導入すると、その強み（効率的なワークフロー、健全な文化）はさらに拡大され、圧倒的な成果を生み出します。例えば、既にCI/CDが整備されている組織では、AIによる自動テスト生成が効果的に機能し、開発速度が大幅に向上します。

弱みの増幅

課題を抱える組織がAIを導入すると、その機能不全（非効率なプロセス、コミュニケーションの断絶）が同様に拡大され、問題はより深刻化します。例えば、コードレビューのプロセスが整備されていない組織では、AIが生成したコードの品質問題がそのまま本番環境に流れ込み、バグが増加します。

AIは組織の「鏡」として機能する

この分析は、AIを万能な生産性向上ツールと見なす安易な見方を否定します。むしろAIは、組織の真の能力を映し出す「**鏡（Mirror）」として機能します。AI活用の成否はツール自体ではなく、その組織がすでに持つ仕組みや文化、すなわち**

「**システム**」に依存します。AIは、組織がDevOpsの成熟度をどの程度積み上げてきたかを容赦なく映し出すリトマス試験紙なのです。

詳細な手法については、章末の「手法5：AIとの効果的協働ガイド」を参照してください。

> ***出典**: DORA（DevOps Research and Assessment）レポート2025年版『State of AI-Assisted Software Development』、https://youtu.be/Dvo5Hhay-t0?list=TLGGO2hCbXy_mxozMTEyMjAyNQ*

5-2 AIによる技術的負債検出と品質向上の実践

AI活用方法のブレインストーミング

チームミーティングで、湊はAI活用について提案した。

「AIをもっと戦略的に使えないだろうか？コード生成以外の用途を考えてみませんか？」

5名のチーム全員が集まり、AI活用のアイデアを出し合った。

「技術的負債の検出はどう？AIにコードベースを分析させて、問題箇所を特定してもらう」

山田が最初に提案した。

「テストケースの自動生成もできるかもしれない。仕様書を読ませて、網羅的なテストケースを提案してもらう」

高橋が続けた。

「ドキュメント作成支援も。コードを読ませて、自動的にドキュメントを生成する」

佐藤が提案した。

「コードレビュー支援も。AIにコードを分析させて、潜在的な問題を指摘してもらう」

別のメンバーが加わった。

ホワイトボードには、AI活用のアイデアが並んでいた。

- 技術的負債の検出
- テストケース自動生成
- ドキュメント作成支援
- コードレビュー支援
- バグの予測
- パフォーマンス分析

湊はそれらを見つめながら言った。

「じゃあ、まずは技術的負債の検出から始めてみませんか？山田さんと一緒に実験してみたいんです」

「面白い、実験してみよう」

山田が頷いた。

リーダーとして湊が実験の方向性を整理した。まずは技術的負債の検出から始め、効果が確認できたら他の用途にも展開する。段階的に進めることで、リスクを最小限に抑えられる。

アイデアが形になる

実験の方向性が明確になった。

実験への期待と不安

翌週、湊と山田は技術的負債検出の実験を始めた。

技術的負債検出の実験

湊はAIツールにコードベースを分析させた。勤怠管理モジュール、経費精算モジュール、プロジェクト管理モジュール。それぞれのコードをAIに読み込ませ、問題箇所を特定してもらう。

数時間後、AIは分析結果を返してきた。

「この関数、複雑度が高すぎますね。循環的複雑度が15を超えています。リファクタリングを検討してください」

「ここ、テストカバレッジ不足です。この関数にはテストがありません」

「このコード、重複が3箇所で見つかりました。共通化を検討してください」

湊は驚いた。AIが指摘した箇所は、確かに問題のあるコードだった。人間だけでは気づかない視点もある。

「これは使える！人間だけでは気づかない視点もある」

湊は興奮しながら山田に報告した。

「予想以上の精度で問題箇所を特定できました。特に、コードの複雑度や重複の検出は、人間よりも正確かもしれません」

山田も画面を見つめながら頷いた。

「確かに、これは有用だね。でも、AIが指摘した全てが問題とは限らない。人間の判断も必要だよ」

「そうですね。AIの指摘を参考にしつつ、人間が最終判断をする。それが重要ですね」

湊は結果をチームで共有した。全員が驚いた。

「AIって、こんなことができるんですね」

佐藤が感心した様子で言った。

「でも、AIが指摘した全てが正しいわけじゃない。人間の判断も必要だよ」

山田が補足した。

解説：AIによる技術的負債検出の効果

湊と山田が実験したように、AIを技術的負債の検出に使うことで、以下の効果が得られます：

AIによる技術的負債検出のメリット

- **網羅的な分析**: 人間では見落としがちな問題箇所を、AIが自動的に発見する
- **客観的な評価**: 人間の主観に左右されず、客観的な基準でコードを評価する
- **時間の短縮**: 人間が手動でコードをレビューする時間を大幅に短縮できる
- **継続的な監視**: CI/CDパイプラインに組み込むことで、継続的にコード品質を監視できる

AIと人間の協働の重要性

ただし、AIが指摘した全てが問題とは限りません。以下の点に注意が必要です：

- **false positiveの存在**: AIが誤検出することもある。人間の判断が必要
- **コンテキストの理解**: AIはコードの意図を完全に理解できない。人間の判断が必要

- **ビジネス価値とのバランス**: 技術的負債の返済と新機能開発のバランスを取るのは人間の判断

AIと人間が協働することで、より効果的な技術的負債管理が可能になります。

詳細な手法については、章末の「手法5：AIとの効果的協働ガイド」を参照してください。

5-3 運用業務も効率化する

テスト自動化でのAI活用

技術的負債検出の実験が成功した後、湊は次のステップを提案した。

「次は、テスト自動化でのAI活用を試してみませんか？高橋さんと一緒に実験してみたいんです」

QA部の高橋が興味深そうに湊を見た。

「テストケースの自動生成ですか？それは面白そうですね」

翌週、高橋と湊はテストケース生成の実験を始めた。

高橋は仕様書をAIに読み込ませた。ユーザー登録機能、ログイン機能、データ分析機能。それぞれの仕様をAIに理解させ、テストケースを提案してもらう。

AIは数分でテストケースを返してきた。

「正常系のテストケース：ユーザー名とパスワードを正しく入力した場合、ログインに成功する」

「異常系のテストケース：パスワードが間違っている場合、エラーメッセージが表示される」

「境界値テスト：ユーザー名が255文字を超える場合、エラーメッセージが表示される」

高橋は驚いた。

「これまで見落としていたエッジケースも含まれています。人間が考えると、正常系と基本的な異常系しか思いつかないことが多いんです。でも、AIは境界値テストやエッジケースも提案してくれる」

高橋は興奮しながら続けた。

「それから、テストケースの作成時間も大幅に短縮できました。以前は1つの機能につき、テストケースを作成するのに半日かかっていました。でも、AIを使えば、1時間で完了します」

「それは大きな効果ですね」

湊は感心した。

「でも、AIが提案したテストケースをそのまま使うのではなく、人間がレビューして、必要に応じて修正する必要があります。AIは完璧ではないから」

高橋が補足した。

「そうですね。AIと人間の協働が重要ですね」

自動テスト作成の効率も大幅に向上した。AIが提案したテストケースをベースに、人間がレビューして修正する。そのプロセスで、テストケースの品質も向上した。

高橋は満足そうに言った。

「AIって、QAの仕事を奪うんじゃなくて、強化してくれるんですね。これまで手動でやっていた作業を自動化できるようになって、もっと創造的な仕事に集中できるようになりました」

開発チームとQA部の連携も強化された。AIが生成したテストケースを共有することで、開発チームとQA部の認識のずれが減った。

解説：AIによるテスト自動化の効果

高橋と湊が実験したように、AIをテスト自動化に使うことで、以下の効果が得られます：

AIによるテストケース生成のメリット

- **網羅的なテストケース**: 人間では見落としがちなエッジケースや境界値テストも提案される
- **時間の短縮**: テストケース作成時間を大幅に短縮できる（半日→1時間）
- **品質の向上**: AIが提案したテストケースをベースに、人間がレビューして修正することで、品質が向上する
- **認識の統一**: 開発チームとQA部でテストケースを共有することで、認識のずれが減る

AIと人間の協働の重要性

ただし、AIが提案したテストケースをそのまま使うのではなく、人間がレビューして修正する必要があります：

- **false positiveの存在**: AIが誤ったテストケースを提案することもある
- **コンテキストの理解**: AIは仕様の意図を完全に理解できない。人間の判断が必要
- **ビジネス価値とのバランス**: 全てのテストケースを実装するのではなく、優先順位をつけるのは人間の判断

AIと人間が協働することで、より効果的なテスト自動化が可能になります。

5-4 新規開発とリファクタリングの両立を支えるAI活用法

失敗と学び

すべてがうまくいくわけではない。AI活用にも失敗はあった。

ドキュメント自動生成の実験では、期待した結果が得られなかった。

湊は、コードベース全体をAIに読み込ませ、APIドキュメントを自動生成しようとした。AIは30分で100ページのドキュメントを生成した。でも、湊が読み返してみると、内容が薄い。

「内容が薄い。AIが生成したドキュメントは、表面的な説明しかしていない」

湊が作成したドキュメントを見て、山田が指摘した。

「例えば、この関数の説明。『ユーザー情報を取得する関数です』としか書いてない。でも、実際には、この関数は認証チェックも行っているし、キャッシュも使っている。そういう重要な情報が抜けている」

山田は画面を指差しながら続けた。

「コンテキストが不足している。AIはコードの意図を完全に理解できないから、詳細な説明ができない。コードだけを見ても、なぜその実装になったのか、どんな制約があるのかはわからない」

湊は深く頷いた。確かに、AIが生成したドキュメントは、表面的な説明しかしていなかった。

コードレビュー支援の実験でも、問題があった。

湊は、AIにコードレビューを依頼した。AIは100行のコードに対して、50箇所の問題を指摘した。でも、佐藤が確認してみると、実際には問題があるのは10箇所だけだった。

「false　positiveが多い。AIが指摘した問題の半分以上は、実際には問題じゃない」

佐藤が報告した。

「例えば、AIが『この変数名は不適切です』と指摘したけど、実際にはプロジェクトの命名規則に従っている。`user_id`という変数名は、プロジェクトでは標準的な命名規則なんです。でも、AIは一般的な命名規則と比較して、不適切だと判断した」

佐藤は別の例も示した。

「それから、AIが『この関数は長すぎます』と指摘したけど、実際にはこの関数は複雑なビジネスロジックを実装していて、分割すると可読性が下がる。AIはコードの長さだけを見て判断しているけど、コンテキストを理解していない」

湊は深く考え込んだ。

「AIも万能じゃないんですね。コンテキストを理解できないから、誤検出が多い」

「でも、使い方次第で強力なパートナーになる」

山田が続けた。

「ドキュメント自動生成は、完全に自動化するのではなく、AIが下書きを作って、人間が修正する。AIが生成したドキュメントをベースに、人間がコンテキストを追加する。そうすれば、効率的にドキュメントを作成できる」

「コードレビュー支援も、AIが指摘した箇所を人間が確認する。AIが指摘した箇所を、人間がフィルタリングして、本当に問題がある箇所だけを修正する。そうすれば、効果的に使えるよ」

湊は失敗から学んだ。AIを完全に自動化するのではなく、AIと人間が協働する。AIが得意なこと（データ分析、パターン検出）をAIに任せ、人間が得意なこと（コンテキスト理解、意思決定）を人間が担う。それが重要だった。

効果的な協働パターンの発見

3週間の実験を経て、効果的な使い方が見えてきた。

湊は実験の結果をまとめた。技術的負債検出は成功した。テストケース生成も成功した。でも、ドキュメント自動生成とコードレビュー支援は、期待した結果が得られなかった。

湊は、何が違ったのかを考えた。技術的負債検出とテストケース生成は、AIが得意な「パターン検出」に適していた。でも、ドキュメント自動生成とコードレビュー支援は、AIが苦手な「コンテキスト理解」が必要だった。

AIが得意なこと：

- データ分析、パターン検出、網羅的チェック
- 大量のコードを短時間で分析する
- 人間では見落としがちな問題を発見する

人間が得意なこと：

- コンテキスト理解、創意工夫、意思決定
- ビジネス価値とのバランスを取る
- チームの状況を考慮した判断をする

「AI+人間」のペアプログラミングが最も効果的だった。

湊はAIとペアプログラミングを試してみた。まず、AIに「ユーザー登録機能を実装したい」と伝えた。AIは、認証チェック、バリデーション、データベースへの保存を含むコードを提案した。湊は、そのコードをレビューして、プロジェクト

の規約に合わせて修正した。例えば、AIが提案したエラーハンドリングを、プロジェクトの標準的な方法に変更した。また、AIが提案した変数名を、プロジェクトの命名規則に合わせて修正した。

そのプロセスで、開発速度が30%向上した。AIがコードの骨組みを作り、人間が細部を調整する。その役割分担が、効率的だった。

「AIは競争相手じゃなく、協働パートナーなんですね」

湊は実感した。AIがコードを提案することで、開発の初期段階が速くなる。でも、そのコードをそのまま使うのではなく、人間がレビューして修正することで、品質も保てる。

山田も同じように感じていた。

「AIがコードを提案して、人間がレビューして修正する。そのプロセスで、開発速度が上がるし、品質も保てる。これが理想的な使い方だと思う」

山田は続けた。

「でも、AIをうまく使うには、プロンプトエンジニアリングのスキルも必要だ。AIに何を伝えれば、良いコードが提案されるのか。それを学ぶ必要がある」

湊は深く頷いた。AIを戦略的に活用するには、AIの特性を理解し、適切な使い方を学ぶ必要がある。それは、新しいスキルだった。

解説：AIと人間の効果的な協働パターン

湊と山田が発見したように、AIと人間が協働することで、より効果的な開発が可能になります。

AIと人間の役割分担

AIが得意なこと：

- データ分析、パターン検出、網羅的チェック

- 大量のコードを短時間で分析する
- 人間では見落としがちな問題を発見する

人間が得意なこと：

- コンテキスト理解、創意工夫、意思決定
- ビジネス価値とのバランスを取る
- チームの状況を考慮した判断をする

効果的な協働パターン

- **AI+人間のペアプログラミング:** AIがコードを提案し、人間がレビューして修正する。開発速度が30%向上
- **AIによる技術的負債検出 + 人間の判断:** AIが問題箇所を特定し、人間が優先順位をつけて対応する
- **AIによるテストケース生成 + 人間のレビュー:** AIがテストケースを提案し、人間がレビューして修正する

これらのパターンを組み合わせることで、開発全体の効率化が可能になります。

AI時代に成果を出し続ける組織の7つのケーパビリティ

2025年版DORAレポートは、AI時代に成果を出し続ける組織が共通して持つ、7つの不可欠な組織能力、すなわち「AIケーパビリティモデル」を定義しています。重要なのは、これらの能力の多くが全く新しいものではなく、DevOpsの基本原則がAIによってその重要性を増幅されたものであるという点です。

1. 明確に周知されたAIスタンス（Clearly communicated AI stance）

組織が公式にAIの活用を支持し、従業員が業務で利用することを積極的に支援する姿勢を明確に示すこと。経営層からの明確なメッセージは、組織的な学習と実践の土台となります。これがなければ、現場はAI利用に躊躇し、組織的な能力獲

得には繋がりません。

2. 健全なデータエコシステム（Healthy data ecosystem）

AIが学習し価値を生み出すために不可欠な、高品質でアクセスしやすい社内データ基盤。AIの能力はデータの品質に直結します。サイロ化されたデータを統合し、健全なエコシステムを構築する能力は、データの民主化を推進してきたDevOps文化の直接的な延長線上にあります。

3. 小さなバッチでの作業（Working in small batches）

変更を小さく分割し、迅速なフィードバックサイクルを回すこと。これは新しい概念ではなく、CI/CDの核となるDevOpsの基本原則です。AIが大規模なコード変更を生成できるようになった今、この古典的な原則の遵守が、スループット向上とレビューのボトルネック化を分ける決定的な境界線となっています。その重要性はAIによって根本的に増幅されました。

4. ユーザー中心の重点（User-centric focus）

常にユーザーの課題解決を目的とし、技術ありきではなく、ユーザー価値を起点に物事を考える文化。レポートは、この思考様式の有無がAI導入によるパフォーマンス向上と低下を分ける要因だと指摘します。これはアジャイル開発の核心であり、技術の進化がいかにユーザー価値から乖離してはならないかを改めて示しています。

5. AIの内部データへのアクセス（AI access to internal data）

AIツールが、Slackでの会話や社内ドキュメントといった組織内部のコンテキストを理解できる環境。組織固有の文脈を学習することで、AIは一般的な回答を超え、実用的なパートナーへと進化します。これは、情報共有と透明性を重んじるDevOps文化の技術的現れと言えます。

6. 高品質な内部プラットフォーム（High-quality internal platform）

プラットフォームエンジニアリングの成果として、開発者が共通で利用できる高品質なツールやインフラ基盤。これは、CI/CDの仕組みやインフラ自動化といった、DORAの初期から重要視されてきた技術的プラクティスの直接的な進化形です。高品質なプラットフォームは開発者体験（DevEx）を向上させ、組織全体の生産性を底上げします。

7. 強力なバージョン管理プラクティス（Strong version control practices）

全ての変更が追跡され、問題発生時に迅速に以前の状態へ復元できる堅牢な運用。AIが予測不能な変更を生み出す可能性に対し、これは不可欠なセーフティネットとなります。これもまた、DevOpsにおけるトレーサビリティと再現性の原則そのものです。

「飛び箱」の比喩：DevOpsの成熟度がAI時代の成否を分ける

これらの7つのケーパビリティは、一朝一夕に獲得できるものではありません。むしろ、その多くはDevOpsの歴史的積み重ねの上に成り立っています。「飛び箱」の比喩が、この歴史的文脈の重要性を力強く示しています。

- まず、継続的インテグレーション（CI）という土台を築く
- その上に、アジャイルな働き方という次の段を置く
- さらにその上に、DevOpsの文化とプラクティスという段を積み上げる

このように能力を段階的に積み上げてきた組織だけが、AI駆動開発という最も高い段を軽々と飛び越えることができます。基礎ができていない組織が、いきなりAIという高い段に挑むことは無謀な挑戦であり、AIが「増幅器」として機能不全を拡大させる結果を招くだけです。

詳細な手法については、章末の「手法5：AIとの効果的協働ガイド」を参照してください。

> ***出典:*** *DORA（DevOps Research and Assessment）レポート2025年版『State of AI-Assisted Software Development』、https://youtu.be/Dvo5Hhay-t0?list=TLGGO2hCbXy_mxozMTEyMjAyNQ、https://youtu.be/vdiBB0_UX-g?list=TLGGkcFCvb-0FbszMTEyMjAyNQ*

5-5 心理的安全性が支える開発生産性

持続可能な開発の実現

AI活用から1ヶ月後、目に見える成果が出てきた。

開発速度：平均30%向上。AI+人間のペアプログラミングの効果が表れていた。

バグ検出率：40%向上。AIによる技術的負債検出の成果だった。

チーム全体の疲労度も軽減された。AIが単純な作業を担うことで、人間はより創造的な仕事に集中できるようになった。

田中部長が湊を呼び出した。

「湊君、AI活用の成果報告を聞いたよ。開発速度が30%向上、バグ検出率が40%向上。これは素晴らしい成果だね」

「ありがとうございます。でも、まだ課題はありそうです」

湊は正直に答えた。

「AIをうまく使うには、チーム全体のスキルアップが必要です。それから、AIが提案したコードをそのまま使うのではなく、人間がレビューして修正する必要があります。そのプロセスを定着させるのが課題です」

「それは重要な課題だね。でも、湊君のチームは着実に改善している。この調子で続けてほしい」

田中部長は満足そうに言った。

テックリードの山田も湊を評価していた。

「湊さん、君が種を蒔いた木が、大きく育ってきてるね。AI活用も、チーム全体で考えて、段階的に進めたのが良かった。一人で抱え込まず、チーム全体で取り組む。それが成功の鍵だったんだよ」

湊は深く頷いた。確かに、一人で抱え込んでいた時は、何も変えられなかった。でも、チーム全体で考えれば、きっと道は開ける。

解説：AI活用による持続可能な開発の実現

湊のチームが実現したように、AIを戦略的に活用することで、持続可能な開発が可能になります。

AI活用による成果

- **開発速度の向上**: AI+人間のペアプログラミングにより、開発速度が30%向上
- **バグ検出率の向上**: AIによる技術的負債検出により、バグ検出率が40%向上
- **チームの疲労度軽減**: AIが単純な作業を担うことで、人間はより創造的な仕事に集中できる

持続可能な開発への道

AI活用を成功させるには、以下のポイントが重要です：

- **段階的な導入**: 一度に全てを変えるのではなく、小さな実験から始める
- **チーム全体での取り組み**: 一人で抱え込まず、チーム全体で考えて進める
- **AIと人間の協働**: AIを完全に自動化するのではなく、AIと人間が協働する

- **継続的な改善**: 一度の成功で終わらず、継続的に改善を続ける

これらのポイントを意識することで、AI活用による持続可能な開発が実現できます。

5-7 AI前提のプロセス設計（BPR）への挑戦

既存プロセスへの無理な当てはめからの学び

AI活用から3ヶ月が経過した頃、湊は新たな課題に直面していた。

「AIツールを導入したのに、かえって手間が増えてしまった。これは何かが間違っている」

湊はテックリードの山田に相談した。

「山田さん、AIツールを導入したのですが、期待した効果が得られません。かえって手間が増えてしまった気がします」

山田は深く頷いた。

「湊さん、それは典型的な失敗パターンだ。既存のやり方を変えずに、無理やりAIツールを当てはめようとすると、かえって手間が増えてしまう。これは失敗パターンだ」

「では、どうすれば良いのでしょうか？」

「プロセスをAI前提に作り変える必要がある。『もしAIの助けを最初から借りられるとしたら、仕事の進め方はどうあるべきか？』という視点から業務プロセスを再設計する必要がある」

湊は目を輝かせた。

「なるほど。プロセスそのものをゼロから見直す必要があるということですね」

「その通りだ。プロセスをAI前提に作り変えることで、真の効果が得られる。これはBPR（ビジネスプロセス・リエンジニアリング）と呼ばれる手法だ」

AI前提のプロセス設計への挑戦

湊はチーム全体でAI前提のプロセス設計に取り組み始めた。

まず、開発の流れ（vibe-coding）と管理の流れ（vibe-management）の2つの軸でAI前提のプロセスを設計した。

開発の流れでは、企画のアイデア出し（PRD）、システムの設計（DD）、実際のプログラミング（Dev）、テスト（Test）の各フェーズでAIを効果的に活用する方法を検討した。

管理の流れでは、プロジェクトの進捗管理（PM Ops）、メンバーの評価・育成（Human Ops）でAIを活用する方法を検討した。

次に、docs as code化を進めた。システムの仕様書や設計に関する情報を、AIがいつでも正確に読み取れる形式で一元管理する仕組みを構築した。AIが参照できる「教科書」のような仕組みを作ることで、AIの提案精度が大幅に向上した。

さらに、ガードレールを構築した。AIが誤って重要なデータを消去するような危険なコマンドを実行しようとしたり、アクセスしてはいけない機密情報に触れようとするのを防ぐ仕組みを整備した。

プロセス再設計の効果

3ヶ月でプロセス再設計を完了し、6ヶ月で効果を実感できるようになった。

生産性が1.3-1.5倍向上し、開発速度が大幅に改善した。リードタイムの短縮により、開発の各フェーズでの待ち時間が削減された。

湊は満足そうに言った。

「AIを前提としたプロセス設計により、真の効果が得られることがわかった。既存プロセスに無理やりAIを当てはめるのではなく、AI前提のプロセスに作り変えることが重要だ」

解説：AI前提のプロセス設計（BPR）の重要性

湊のチームが実現したように、AIを効果的に活用するには、プロセスをAI前提に作り変える必要があります。

既存プロセスへの無理な当てはめの失敗

既存のやり方を変えずにAIツールを当てはめようとすると、かえって手間が増えてしまいます。これは典型的な失敗パターンです。

AI前提のプロセス設計とは

「もしAIの助けを最初から借りられるとしたら、仕事の進め方はどうあるべきか？」という視点から業務プロセスを再設計することが重要です。

開発の流れ（vibe-coding）と管理の流れ（vibe-management）の2つの軸：

- **vibe-coding（開発の流れ）：** 企画のアイデア出し（PRD）、システムの設計（DD）、実際のプログラミング（Dev）、テスト（Test）
- **vibe-management（管理の流れ）：** プロジェクトの進捗管理（PM Ops）、メンバーの評価・育成（Human Ops）

docs as code化の重要性：
システムの仕様書や設計に関する情報を、AIがいつでも正確に読み取れる形式で一元管理することで、AIの提案精度が大幅に向上します。

ガードレールの構築：
AIが誤って重要なデータを消去するような危険なコマンドを実行しようとしたり、アクセスしてはいけない機密情報に触れようとするのを防ぐ仕組みを整備することが重要です。

AI活用の進化段階：
AI活用は段階的に進化します。prompt-engineering (vibe-coding) → context-engineering → Agent to Agent (A2A) の順に進化させることが重要です。

AI agentの活用レベル（監視レベル）の段階的移行：
エージェント支援 → ヒューマンインザループ → ヒューマンオンザループ → ヒューマンアウトオブザループの順に、適切な監視レベルを設定し、段階的に移行することが重要です。

プロセス再設計の効果

プロセス再設計により、以下の効果が得られます：

- **生産性1.3-1.5倍向上:** 成果が出ているチームでは生産量が1.3〜1.5倍に向上
- **リードタイムの短縮:** 開発の各フェーズでの待ち時間が削減される
- **開発速度の大幅な改善:** コードを書く時間は全体のほんの一部に過ぎず、開発のリードタイムは企画会議、仕様の再確認、レビュー待ちなどの「待ち時間」に支配されています。AI活用により、これらの待ち時間を削減できます

5-8 AI活用の組織的課題への対応

AI疲れの課題

AI活用から2ヶ月が経過した頃、チーム内で新たな課題が浮上していた。

佐藤が湊に相談してきた。

「湊さん、AIの提案を逐一確認するのが大変で、かえって時間がかかってしまいます」

湊は山田に相談した。

「山田さん、AIの提案を逐一確認するのが大変だという声が出ています。これはどう対処すべきでしょうか？」

山田は深く頷いた。

「湊さん、それは『AI疲れ』と呼ばれる現象だ。AIから次々と出てくる提案を常に判断し、検証し、修正し続ける作業は、人間の脳に大きな負荷をかける。この精神的な負荷が蓄積し、かえって生産性が低下してしまう現象は『AI疲れ』と呼ばれている」

「では、どう対処すべきでしょうか？」

「AIの提案を全て確認するのではなく、重要な部分だけに絞るべきだ。AI提案の判断プロセスを最適化することが重要だ」

AIが作ったコードへの説明責任の問題

さらに、別の課題も浮上していた。

経験の浅いエンジニアが、AIが生成したコードの意味を理解しないまま提出し、レビューに余計な時間がかかるという問題だった。

佐藤が困った表情で湊に報告した。

「湊さん、AIが生成したコードをそのまま提出したら、レビューで『なぜこのコードにしたの？』と聞かれて答えられませんでした」

山田は説明した。

「AIが作ったコードへの説明責任は重要だ。経験の浅いエンジニアが、AIが生成したコードの意味や設計思想を十分に理解しないまま提出してしまうと、レビュー担当者からの質問に答えることができず、結果としてレビューに余計な時間がかかったり、本人の成長を妨げたりする危険性がある」

湊は理解した。

「AIを使うときは、生成されたコードの意味を理解してから提出する必要があるということですね」

組織の鏡としてのAI

山田はさらに重要な指摘をした。

「湊さん、世界的な調査レポートである『DORAレポート』でも指摘されているように、AIは魔法の杖ではない。AIは組織の文化や実力を映し出す『組織の鏡』であり、良いチームでは生産性をさらに加速させる一方で、課題を抱えるチームでは混乱を広げてしまう『増幅器』として機能する」

湊は深く頷いた。

「つまり、AIを使いこなすためには、ツールの操作技術だけでは不十分で、エンジニア自身のスキルがより重要になるということですね」

「その通りだ。AIの提案の良し悪しを判断し、その意図を理解し、自らの言葉で説明できるだけの深い知識と経験が、これまで以上に求められる」

AIスキルの標準化・育成

湊はチーム全体でAIスキルの標準化・育成に取り組み始めた。

一部の詳しい人だけでなく、チーム全員がAIを効果的に使いこなせるよう、標準化と育成を実施した。AI活用のベストプラクティスを共有し、組織全体に浸透させた。

その結果、AI活用の浸透率が大幅に向上した。全体として、AIを「全く使っていない（0%）」と回答した人の割合が大きく減少し、「半分ぐらいは使っている（50%）」と回答した人の割合が大きく増加した。

解説：AI活用の組織的課題への対応

湊のチームが直面したように、AI導入により組織的な課題が発生します。これらの課題に対処することが重要です。

AI疲れへの対策

AIから次々と出てくる提案を常に判断・検証し続けることによる精神的な負荷が蓄積し、かえって生産性が低下してしまう現象が「AI疲れ」です。

対策方法：

- AIの提案を全て確認するのではなく、重要な部分だけに絞る
- AI提案の判断プロセスを最適化する
- AIとの協働における精神的な負荷を軽減する方法を確立する

説明責任の明確化

AIが生成したコードの意味や設計思想を理解する必要性があります。経験の浅いエンジニアが理解しないまま提出すると、レビューに余計な時間がかかり、成長を妨げます。

対策方法：

- AIが生成したコードの意味や設計思想を理解してから提出する文化の醸成
- レビュー時に「なぜこのコードにしたの？」という質問に答えられるようにする
- AIに依存しすぎず、自分で考える力を維持する

AIスキルの標準化・育成

一部の詳しい人だけでなく、チーム全員がAIを効果的に使いこなせるようになる必要があります。

対策方法：

- チーム全員がAIを効果的に使いこなせるよう、標準化と育成を実施
- AI活用のベストプラクティスを共有し、組織全体に浸透させる
- 組織全体でAI活用を推進する

組織の鏡としてのAI

AIは組織の強みも弱みも増幅する特性があります。良いチームでは生産性をさらに加速させる一方で、課題を抱えるチームでは混乱を広げてしまう「増幅器」として機能します。

重要なポイント：

- AIの提案の良し悪しを判断し、その意図を理解し、自らの言葉で説明できるだけの深い知識と経験が、これまで以上に求められる
- AIを使いこなすためには、ツールの操作技術だけでは不十分で、エンジニア自身のスキルがより重要になる

組織全体でのAI活用の浸透

AIスキルの標準化・育成により、組織全体でAI活用が進み、生産性が向上します。

効果：

- AI活用の浸透率が大幅に向上
- 全体として、AIを「全く使っていない（0%）」と回答した人の割合が大きく減少し、「半分ぐらいは使っている（50%）」と回答した人の割合が大きく

増加

- 組織全体での生産性向上

手法5：AIとの効果的協働ガイド

AI活用の基本原則

AIを効果的に活用するためには、以下の原則を守ることが重要です：

1. AIは補助ツールであり、代替ではない

- AIが提案したコードをそのまま使うのではなく、人間がレビューして修正する
- AIが指摘した問題をそのまま信じるのではなく、人間が確認する
- AIの判断を盲信せず、人間の判断を優先する

2. AIと人間の役割分担を明確にする

AIが得意なこと：

- 大量のデータ分析
- パターン検出
- 網羅的なチェック

人間が得意なこと：

- コンテキスト理解
- 創意工夫
- 意思決定

3. 段階的に導入する

- 小さな実験から始める
- 効果が確認できたら、段階的に拡大する
- 失敗から学び、改善を続ける

AI活用の具体的な手法

技術的負債検出でのAI活用

手順：

1. AIツールにコードベースを分析させる
2. AIが指摘した問題箇所を人間が確認する
3. 優先順位をつけて、段階的に改善する

効果：

- 網羅的な分析が可能
- 人間では見落としがちな問題を発見
- 時間の短縮

テストケース生成でのAI活用

手順：

1. AIに仕様書を読み込ませる
2. AIがテストケースを提案する
3. 人間がレビューして、必要に応じて修正する

効果：

- 網羅的なテストケースの生成
- テストケース作成時間の短縮
- エッジケースの発見

コードレビュー支援でのAI活用

手順：

1. AIにコードを分析させる
2. AIが潜在的な問題を指摘する
3. 人間が確認して、実際の問題かどうかを判断する

効果：

- レビュー時間の短縮
- 見落としがちな問題の発見
- コード品質の向上

AI活用のチェックリスト

導入前：

- [ ] AI活用の目的を明確にした
- [ ] チーム全体でAI活用の方針を合意した
- [ ] 小さな実験から始める計画を立てた

導入中：

- [ ] AIが提案したコードを人間がレビューしている
- [ ] AIの判断を盲信せず、人間の判断を優先している
- [ ] 失敗から学び、改善を続けている

導入後：

- [ ] AI活用の効果を測定している
- [ ] チーム全体でAI活用のスキルを向上させている
- [ ] 継続的に改善を続けている

AIケーパビリティチェックリスト

AI時代に成果を出し続ける組織が持つ7つのケーパビリティを、あなたの組織で確認してください。

1. 明確に周知されたAIスタンス

- [ ] 経営層からAI活用を支持する明確なメッセージが出ている
- [ ] 従業員が業務でAIを利用することを積極的に支援する方針がある
- [ ] AI利用に関するガイドラインやポリシーが整備されている

2. 健全なデータエコシステム

- [ ] 社内データがサイロ化されず、統合されている
- [ ] データへのアクセスが容易で、高品質なデータ基盤がある
- [ ] データの民主化が推進されている

3. 小さなバッチでの作業

- [ ] 変更を小さく分割してリリースしている
- [ ] 迅速なフィードバックサイクルが回っている
- [ ] CI/CDパイプラインが整備されている

4. ユーザー中心の重点

- [ ] ユーザーの課題解決を目的とした開発が行われている
- [ ] 技術ありきではなく、ユーザー価値を起点に考えている
- [ ] ユーザーフィードバックを開発に反映する仕組みがある

5. AIの内部データへのアクセス

- [ ] AIツールが社内ドキュメントやSlackの会話を理解できる環境がある
- [ ] 組織固有のコンテキストをAIが学習できる仕組みがある
- [ ] 情報共有と透明性が重視されている

6. 高品質な内部プラットフォーム

- [ ] 開発者が共通で利用できる高品質なツールやインフラ基盤がある
- [ ] プラットフォームエンジニアリングが推進されている
- [ ] 開発者体験（DevEx）が向上している

7. 強力なバージョン管理プラクティス

- [ ] 全ての変更が追跡されている
- [ ] 問題発生時に迅速に以前の状態へ復元できる
- [ ] トレーサビリティと再現性が確保されている

評価：

- 7つ全て該当：AI時代に成果を出し続ける組織の基盤が整っている
- 5-6個該当：良好な状態。残りのケーパビリティを強化する
- 3-4個該当：注意が必要。DevOpsの基本原則から見直す
- 0-2個該当：AI導入前にDevOpsの成熟度を高める必要がある

> ***出典:*** *DORA（DevOps Research and Assessment）レポート2025年版『State of AI-Assisted Software Development』*

手法6：プロンプトエンジニアリング事例集

効果的なプロンプトの書き方

AIを効果的に活用するためには、適切なプロンプトを書くことが重要です。

技術的負債検出のプロンプト例

```
以下のコードベースを分析して、技術的負債の可能性がある箇所を特定してください。

- 循環的複雑度が10を超える関数
- テストカバレッジが70%未満の関数
- コードの重複が3箇所以上ある箇所
- 命名が不適切な変数や関数

各問題箇所について、以下の情報を提供してください：
1. 問題の種類
2. 問題の深刻度（高/中/低）
3. 改善の提案
```

テストケース生成のプロンプト例

```
以下の仕様書を読んで、テストケースを生成してください。

機能：ユーザー登録
- ユーザー名：3文字以上20文字以下、英数字のみ
- パスワード：8文字以上、英数字と記号を含む
- メールアドレス：有効な形式であること

以下の種類のテストケースを生成してください：
1. 正常系のテストケース
2. 異常系のテストケース（各入力項目のバリデーションエラー）
3. 境界値テスト（最小値、最大値、境界値-1、境界値+1）
4. エッジケース（特殊文字、空文字、nullなど）
```

コードレビュー支援のプロンプト例

```
以下のコードをレビューして、潜在的な問題を指摘してください。

レビューの観点：
- コードの品質（可読性、保守性、パフォーマンス）
- セキュリティの問題
- バグの可能性
- ベストプラクティスへの準拠

各問題について、以下の情報を提供してください：
1. 問題の種類
2. 問題の深刻度（高/中/低）
3. 問題の説明
4. 改善の提案
```

プロンプトの改善方法

プロンプトを改善するには、以下のポイントを意識してください：

- **具体的な指示を出す**: 曖昧な指示ではなく、具体的な指示を出す
- **出力形式を指定する**: AIがどのような形式で出力すべきかを指定する
- **コンテキストを提供する**: AIが判断するために必要な情報を提供する
- **例を示す**: 期待する出力の例を示すことで、AIの理解を助ける

章のまとめ

第5章では、湊のチームがAIを戦略的に活用し、持続可能な開発を実現する過程を描きました。

主な学び

1. **AIは「コードを早く書く道具」以上のパートナー:** AIをコード生成だけに使うのではなく、開発全体を効率化するパートナーとして使う
2. **AIと人間の協働の重要性:** AIを完全に自動化するのではなく、AIと人間が協働することで、より効果的な開発が可能になる
3. **段階的な導入:** 一度に全てを変えるのではなく、小さな実験から始めて、段階的に拡大する
4. **失敗から学ぶ:** すべてがうまくいくわけではない。失敗から学び、改善を続けることが重要
5. **AI前提のプロセス設計（BPR）の重要性:** 既存プロセスに無理やりAIを当てはめるのではなく、AI前提のプロセスに作り変えることで、真の効果が得られる
6. **AI活用の組織的課題への対応:** AI疲れ、説明責任、AIスキルの標準化・育成など、組織的な課題に対処することが重要
7. **AIによる開発生産性向上戦略の実践:** 開発生産性ツリーの観点から、戦略1（ValueStreamの高速化）と戦略2（Ops工数のAI置き換え）を実践することで、開発生産性を150%に向上させる

5-9 AIによる開発生産性向上戦略の実践

プロセス再設計の成功とAI戦略の整理

AI前提のプロセス設計から3ヶ月が経過した頃、湊は開発生産性ツリーの観点からAI戦略を整理する機会を得た。

田中部長から報告会の依頼があった。

「湊君、AI前提のプロセス設計の成果を報告してほしい。経営層にも説明できる形で整理してくれないか」

湊は、開発生産性ツリーの観点からAI戦略を2つの軸で整理することにした。

田中部長との報告会

会議室で、湊はホワイトボードに開発生産性ツリーを描きながら説明を始めた。

「開発生産性ツリーの観点から見ると、AI戦略は2つに分けられます」

戦略1: ValueStreamの高速化（Incrementを1.5倍にする）

湊はホワイトボードに「戦略1」と書きながら説明した。

「まず、ValueStreamの各工程をAIネイティブなプロセスで高速化することで、同じ投入工数でより多くの成果物を生み出す戦略です」

「問題を定義するところ」

湊は「問題を定義するところ」と書きながら続けた。

「要求定義をもとに要件定義を詰める部分で、AIを使って意思決定を加速しています。人がPilot、AIがCopilotという役割分担です」

飛鳥PMが頷いた。

「確かに、要件定義の時間が短縮されて、意思決定が早くなりましたね。AIが情報を整理してくれるので、私たちは意思決定に集中できます」

「問題を解くところ」

湊は「問題を解くところ」と書きながら続けた。

「実装からリリースまでは、作業はAIが主導し、品質は人が担保します。人がCopilot、AIがPilotという役割分担です」

テックリードの山田さんが補足した。

「AIがタスクを分解し実装まで進めてくれるので、私たちは品質チェックに集中できます。これにより、実装からリリースまでの工程が大幅に高速化されました」

湊はまとめた。

「ValueStream全体のリードタイム短縮により、Increment（出口）を1.5倍にすることができます」

戦略2: Ops工数のAI置き換え（Capacityを1.5倍にする）

湊は「戦略2」と書きながら説明を続けた。

「次に、Ops工数をAIに置き換えることで、新規開発に回せる工数を増やす戦略です」

工数分析のプロセス

湊は工数分析の結果を示した。

「開発区分における保守開発・運用・管理業務の工数を分析しました。全体の約40%を占めていました」

山田が補足した。

「その中で、AIによる改善インパクトがある部分をプロセスから探しました。例えば、ポストモーテムの作成や障害分析、目標設定の管理などです」

AI Agentの導入と効果

湊は具体的なAI Agentの導入例を説明した。

「まず、ポストモーテムを書くAI　Agentを導入しました。障害対応後のポストモーテム作成が自動化され、工数が大幅に削減されました」

山田が続けた。

「次に、障害分析AIを導入しました。過去の障害パターンを分析し、事前に対策を提案してくれるAIです。これにより、障害対応の工数が削減されました」

湊は管理業務でのAI活用も説明した。

「メンバーの目標設定から評価までをAIがサポートし、管理業務の工数が削減されました。これまで手動で行っていた作業が自動化され、余剰工数が生まれました」

余剰工数の新規開発への転換

湊は最終的な効果を説明した。

「Ops工数の削減により、余剰工数25人月分を新規開発に回すことができました」

田中部長が確認した。

「つまり、Capacity（入口）を1.5倍にできたということですね」

湊は深く頷いた。

「はい。戦略1によりIncrement（出口）を1.5倍にし、戦略2によりCapacity（入口）を1.5倍にすることで、全体として開発生産性を150%に向上させることができます」

開発生産性ツリーの観点からの説明

湊は開発生産性ツリーの図を指しながら説明を続けた。

「開発生産性ツリーの観点から見ると、予算（財務）起点で考えると、AIへの投資が開発生産性の向上にどうつながるかが明確になります」

田中部長が興味深そうに聞いていた。

「なるほど。AIへの投資が、Capacity（投入工数）とIncrement（成果物）の両方を改善することで、開発生産性を150%に向上させるということですね」

湊は続けた。

「はい。戦略1は、同じ投入工数でより多くの成果物を生み出すことで、Increment（出口）を1.5倍にします。戦略2は、Ops工数を削減して新規開発に回す工数を増やすことで、Capacity（入口）を1.5倍にします」

チーム全体の成果実感

報告会の後、5名チーム全員がAI戦略の効果を実感していた。

開発速度の向上とOps工数の削減により、新規開発に集中できる環境が整った。

湊はチームメンバーに語りかけた。

「AIを戦略的に活用することで、持続可能な開発生産性の向上が実現できました。開発生産性ツリーの観点から整理することで、AI戦略の効果を明確に説明できるようになりました」

テックリードの山田さんが頷いた。

「確かに、AI戦略を開発生産性ツリーの観点から整理することで、経営層にも理解してもらいやすくなりましたね」

解説：AIによる開発生産性向上戦略の実践

湊のチームが実現したように、AIを戦略的に活用するには、開発生産性ツリーの観点から整理することが重要です。

開発生産性ツリーの観点からのAI戦略

開発生産性ツリーのセクション8では、AIによる開発生産性向上の戦略を定義しています。

目的:

- Capacity（投入工数）に対して、Increment（成果物）の量を現状100%から150%にする

2つの戦略:

1. **戦略1:** ValueStream（工数・工期）をAIネイティブなプロセスで高速化し、Increment（出口）を1.5倍にする
2. **戦略2:** Opsの工数をAIに置き換え(or協働)し余剰工数を25人月分新規に回して、Capacity（入口）を1.5倍にする

戦略1: ValueStreamの高速化（Incrementを1.5倍にする）

ValueStreamの各工程をAIネイティブなプロセスで高速化し、同じ投入工数でより多くの成果物を生み出す戦略です。

「問題を定義するところ」:

- 要求定義をもとに要件定義を詰める部分を「問題を定義するところ」として位置づける

- 人同士が泥臭く合意するところ
- 意思決定を加速するためにAIを使う
- **役割**: 人がPilot、AIがCopilot
    - 人間が意思決定の主導権を持ち、AIが情報整理や選択肢の提示を支援
    - 要件定義の効率化により、リードタイムを短縮

「問題を解くところ」:

- 要件定義をもとにタスクを分解し実装からリリースまでを行う部分を「問題を解くところ」として位置づける
- 作業はAI。品質は人が担保する
- **役割**: 人がCopilot、AIがPilot
    - AIがタスク分解、実装、テストケース生成など、実行作業をAIが主導
    - 人間が品質チェックや承認を行う
    - 実装からリリースまでの工程を高速化

効果: ValueStream全体のリードタイム短縮により、Increment（出口）を1.5倍にする。

戦略2: Ops工数のAI置き換え（Capacityを1.5倍にする）

開発区分における（保守開発〜管理業務）までの部分の工数をAIに置き換えることで、新規開発に回せる工数を増やす戦略です。

工数分析とAI置き換えプロセス:

1. **工数分析**: 開発区分における（保守開発・運用・管理業務）までの部分の工数を分析する
    - 例：全体の40%の工数をかけていた場合、AIによる改善インパクトがある部分をプロセスから探す

2. **AI置き換えの実施:** AIに置き換え可能な業務を特定し、AI Agentを導入
3. **効果測定:** 置き換えた工数を定量化し、新規開発への転換効果を測定

AI Agentの具体例:

保守開発におけるAI活用:

- ポストモーテムを書くAI Agent
- 障害分析をして事前に対策をしてくれるAI
- コードレビュー支援AI

運用におけるAI活用:

- 障害対応の自動化AI
- ログ分析・異常検知AI
- 問い合わせ対応AI

管理業務におけるAI活用:

- メンバーの目標設定から目標トラッキング、評価をサポートしてくれるAI Agent
- 会議議事録作成AI
- 予算策定支援AI

効果:　Ops工数の削減により、余剰工数25人月分を新規開発に回し、Capacity（入口）を1.5倍にする。

Capacity（投入工数）とIncrement（成果物）の関係

開発生産性ツリーの観点から見ると、AI戦略は以下の関係で整理できます。

- **Capacity（投入工数）**：開発に投入される工数（人月）
- **Increment（成果物）**：開発によって生み出される成果物の量

目標:　Capacity（投入工数）に対して、Increment（成果物）の量を現状100%から150%にする。

戦略の組み合わせ:

- **戦略1:** Increment（出口）を1.5倍にする → 同じ投入工数でより多くの成果物を生み出す
- **戦略2:** Capacity（入口）を1.5倍にする → Ops工数を削減して新規開発に回す工数を増やす
- **両戦略の組み合わせ:** 開発生産性を150%に向上させる

実践のヒント

戦略1の実践:

- 要件定義フェーズと実装フェーズでAIの役割を明確にする
- 「問題を定義するところ」では、人がPilot、AIがCopilotの役割分担を確立する
- 「問題を解くところ」では、人がCopilot、AIがPilotの役割分担を確立する

戦略2の���践:

- 工数分析から始め、AIによる改善インパクトがある部分をプロセスから探す
- 段階的にAI Agentを導入し、効果を測定しながら進める
- 余剰工数を新規開発に回すことで、Capacity（入口）を1.5倍にする

開発生産性ツリーの観点から:

- AI戦略の効果を開発生産性ツリーの観点から測定する
- Capacity（投入工数）とIncrement（成果物）の関係を明確にする
- 経営層にも理解してもらいやすい形で説明する

詳細な手法については章末の「手法5：AIとの効果的協働ガイド」を参照してください。

次章への展望

湊のチームはAI活用により、開発速度を30%向上させ、バグ検出率を40%向上させました。さらに、開発生産性ツリーの観点からAI戦略を整理することで、AIへの投資が開発生産性の向上にどうつながるかを明確に説明できるようになりました。

第6章では、湊がエンジニアの視点だけでなく、PM、PdM、事業責任者の視点も取り入れた新しい生産性評価の枠組みを提案する過程を描きます。

AI活用による技術的な改善が、ビジネス価値につながることを示すことで、エンジニアだけが生産性を求められる重圧から解放される道を探ります。