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。

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。

AIツールへの疑問

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。

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。

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。

5-2 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。

技術的負債検出の実験

翌週、湊と山田は技術的負債検出の実験を始めた。

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。

「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。

「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」

　コードレビュー支援の実験でも、問題があった。

「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。AIはコンテキストを理解できないから、誤検出が多い」

　湊は深く考え込んだ。

「AIも万能じゃないんですね」

「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。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。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。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。

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。AIがコードを提案し、湊がレビューして修正する。そのプロセスで、開発速度が30%向上した。

「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。

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」

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。

　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。

5-5 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。

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

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

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。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。