投稿日 2025/01/28

# GitHub Copilotを用いたAIによるコードレビューの活用

TISでは、生成AIを活用した開発を推進しており、その一環としてGitHub Copilotの利用を進めています。
本記事は、開発現場でのGitHub Copilotの活用方法について、コードレビューに焦点をあててご紹介します。

## サマリ

・GitHub Copilotのコードレビュー活用により、開発者のセルフレビュー時間を大幅に短縮し、品質向上を実現。

・独自開発したVisual Studio Codeの拡張機能「Promptis」とレビュープロンプトの整備により、組織全体での効率的な活用を推進。

・人間の適切な関与を維持しながら、AIツールを補助的に活用することで、コードレビューのプロセス全体を最適化。

## GitHub Copilot Chatを用いたコードレビュー

GitHub Copilot Chatに自然言語で記述されたレビューチェックリストを入力することで、コードの改善点の有無を即座に得ることができます。ルールベースのチェックでは拾いづらい**抽象表現でのレビューが可能です。**

### 抽象表現でのコードレビュー例

#### コードレビュープロンプト

```
コードを評価し、結果をmarkdown表形式で示す。表以外は表示しないこと。
表に続けて、結果が×,△のものに対して改善案を示して。

## 表
観点、結果(○,×,△,-)、指摘

## チェック観点
・各モジュール・クラス・関数は単一責任の原則に従っているか
・大きすぎる関数やクラスは適切に分割されているか
・継承の深さは適切か、不必要な継承を避けているか
・コードの重複を避け、DRYの原則を遵守しているか
・命名規則はPEP8に準拠し、わかりやすいか
・ファイル構造は論理的で整理されているか
```

コードレビュー結果例

## AIコードレビュー結果

| 観点 | 結果 (○,×,△,-) | 指摘 |
| --- | --- | --- |
| 各モジュール・クラス・関数は単一責任の原則に従っているか | △ | `SaliencyApp` クラスが多くの責任を持っているため、機能ごとに分割することが望ましい。 |
| 大きすぎる関数やクラスは適切に分割されているか | △ | `process_image`, `process_video`, `process_pdf` 関数が大きいため、さらに分割することが望ましい。 |
| 継承の深さは適切か、不必要な継承を避けているか | ○ | 不必要な継承は使用されていない。 |
| コードの重複を避け、DRYの原則を遵守しているか | △ | 画像処理の部分で重複が見られるため、共通部分を関数化することが望ましい。 |
| 命名規則はPEP 8に準拠し、わかりやすいか | ○ | 命名規則はPEP 8に準拠している。 |
| ファイル構造は論理的で整理されているか | ○ | ファイル構造は論理的で整理されている。 |

**改善案**

**`SaliencyApp` クラスの分割**

`SaliencyApp` クラスの責任を分割し、以下のようにリファクタリングします。

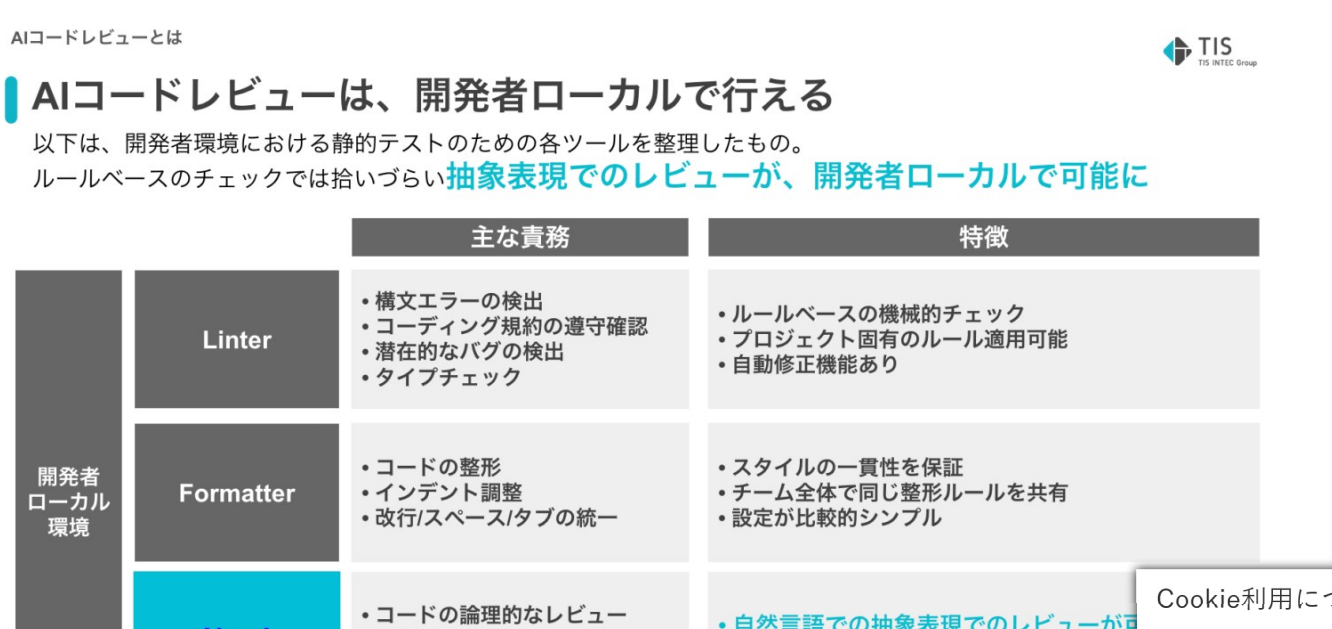
```
class SaliencyApp:
    def __init__(self, root):
        self.root = root
```

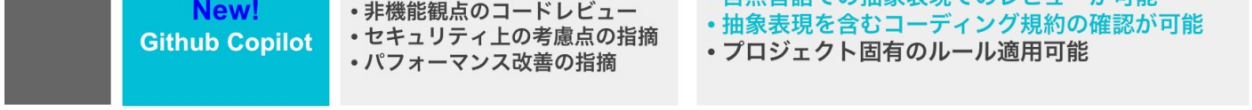
AIコードレビュー結果（サンプル）

## AIコードレビューの利点：開発者ローカルで行える

GitHub Copilot Chatを用いたAIコードレビューは開発者環境で行うことができます。

既存の静的解析ツールに加えて、**抽象表現での**開発者自身のセルフレビューにより、コード品質が向上します。






## AIコードレビュー 導入前後

AIコードレビュー 導入前後を比較して、導入メリットをみていきましょう。

### ①レビュー時間と労力を節約

第一に、開発者自身のセルフレビューの一部が代替され、作業負担が減ります。
AIコードレビューの導入前のセルフレビューが約60分とすると、導入後は約10分程度まで短縮可能です。

### ②開発者のセルフレビュー品質の向上

AIコードレビューは、24時間365日即座にフィードバック可能です。開発途中段階でのこまめにチェック可能で、迅速なフィードバックループが確立できます。また、一貫性のある評価基準での分析が可能で、見落としやすい細かい問題も検出可能です。特に、非機能観点のコード品質向上に著しい効果があります。

開発者の時間の21%はバグ修正に費やされており[1]、本番環境でのバグ発見時の修正は10倍のコストが発生すると仮定すると[2]、バグの早期発見による生産性効果は大きいです。

**バグの早期発見による生産性効果の試算**

| バグの発見工程 | バグ修正コスト（仮定値※） | 従来手法 | AI活用後（仮定値※） | 削減効果（仮定値※） |
|---|---|---|---|---|
| 開発中 | 1倍 | 10% (3.36h) | 40% (13.44h) | +10.08h |
| コードレビュー時 | 2倍 | 20% (13.44h) | 30% (20.16h) | +6.72h |
| テスト工程 | 5倍 | 30% (50.4h) | 20% (33.6h) | -16.8h |
| 本番環境 | 10倍 | 40% (134.4h) | 10% (33.6h) | -100.8h |
| 合計コスト（工数換算） | - | 201.6h | 100.8h | -100.8h (50%減) |

💡 試算の前提条件：
**※注：以下は仮定値であり、実際の効果はPJの開発プロセスや成熟度により変動する可能性があります。**

- 1人月の開発時間：160時間
  - バグ修正時間比率：21%
- 本番環境でのバグ発見時の修正コスト：開発時の10倍
  - 工程が後になるほどバグ修正コストが上昇するものとする
- 改善の仮定：
  1. 開発中の早期発見増加：10% → 40%
  2. レビュー時の発見向上：20% → 30%
  3. テスト工程での発見減少：30% → 20%（早期発見により）
  4. プロダクションでの発見減少：40% → 10%（早期発見により）

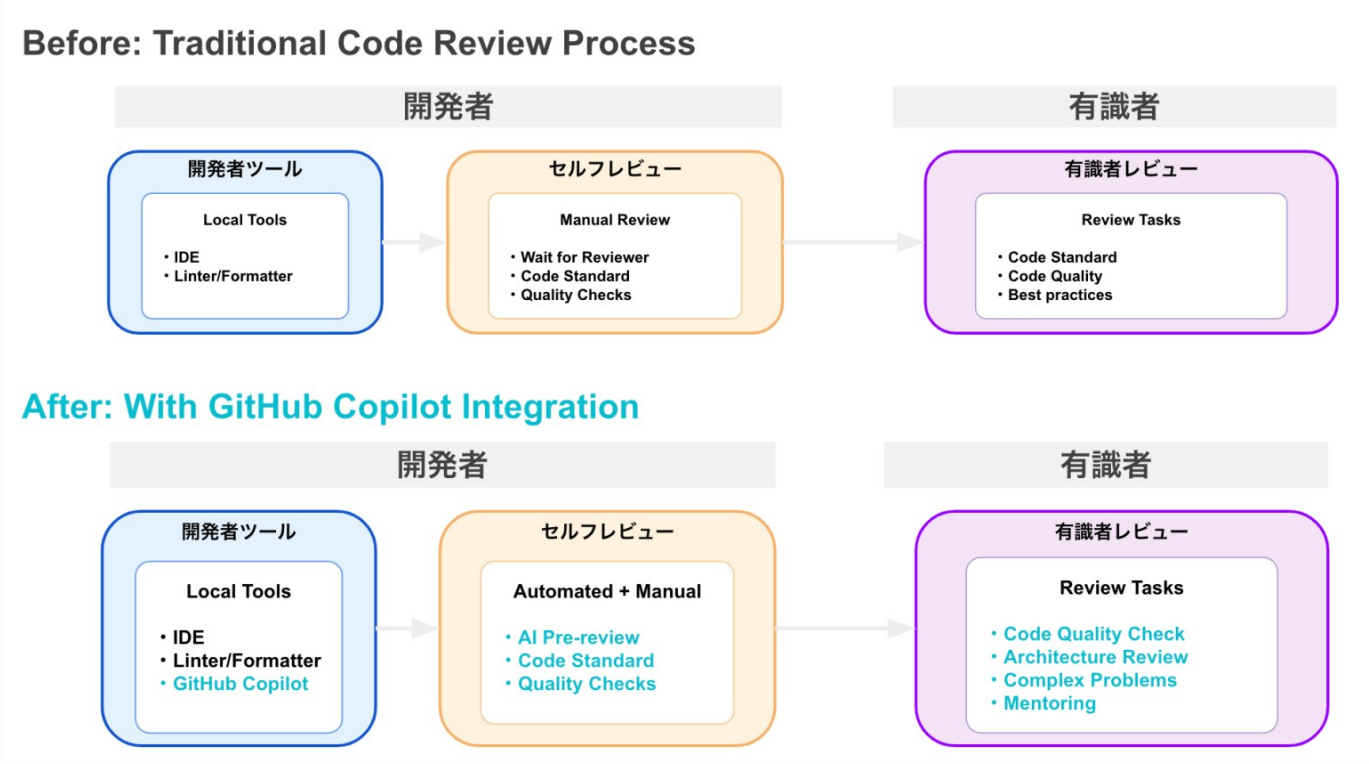
バグの早期発見による生産性効果の試算例

### ③有識者のレビュー作業負担の軽減・役割の変化

大規模開発では、新規着任者、ジュニアのエンジニアを含む複数名が並行して製造を行います。

AIコードレビューの導入前は、コード品質の均一化のために有識者レビュー時間の大半が費やされるのが実情でした。命名規則の統一やコードの可読性など、機能品質以外のレビュー指摘も多く検出されます。**有識者の稼働が集中し、レビュー待ち→レビュー→修正→再レビューのサイクルが長期化する傾向がありました。**

AIコードレビューの導入後は、開発者のセルフレビュー品質が向上したことにより有識者レビューの負担が軽減されました。




有識者への負担軽減は、有識者(人間)によるコードレビューの質も向上します。**コード品質の最終チェックに集中でき、アーキテクチャ評価や戦略的判断により多くの時間を確保可能に**なります。

## AIコードアシスタントとの付き合い方

新技術であるAIコードレビューを使いこなすにあたり、注意すべき点もあります。

### AIコードレビューの注意点

AIコードレビューの注意点を以下にまとめます。

**・AIはビジネスロジックの理解が限定的**

**・一般的でないチーム特有の個別方針や特殊な要件への対応は限定的**

**・出力の一貫性を保つには、プロンプトの工夫が必要**

**・組織全体で活用可能なプロンプトの共有と再利用がキーとなる**

これらの注意点に対しては、組織的な対策を講じることでAIコードレビューの効果を最大化することができると考えます。

## 人間の関与(Human in the Loop)の重要性

AIのコード提案を主体ではなく補助的なツールとして位置づけることは重要です[3]。特にクリーンで重複のない効率的なコードとなっているかを意識し、コードの提案を受け入れる場合は、後のコードレビューをセットで行います。

コード品質の判断はAI任せにするのでなく、人間（有識者）によるレビュー品質をいかに高めるかという観点でAIを扱うことが肝要です。その点で後述するコードレビュープロンプトの整備が有効となります。

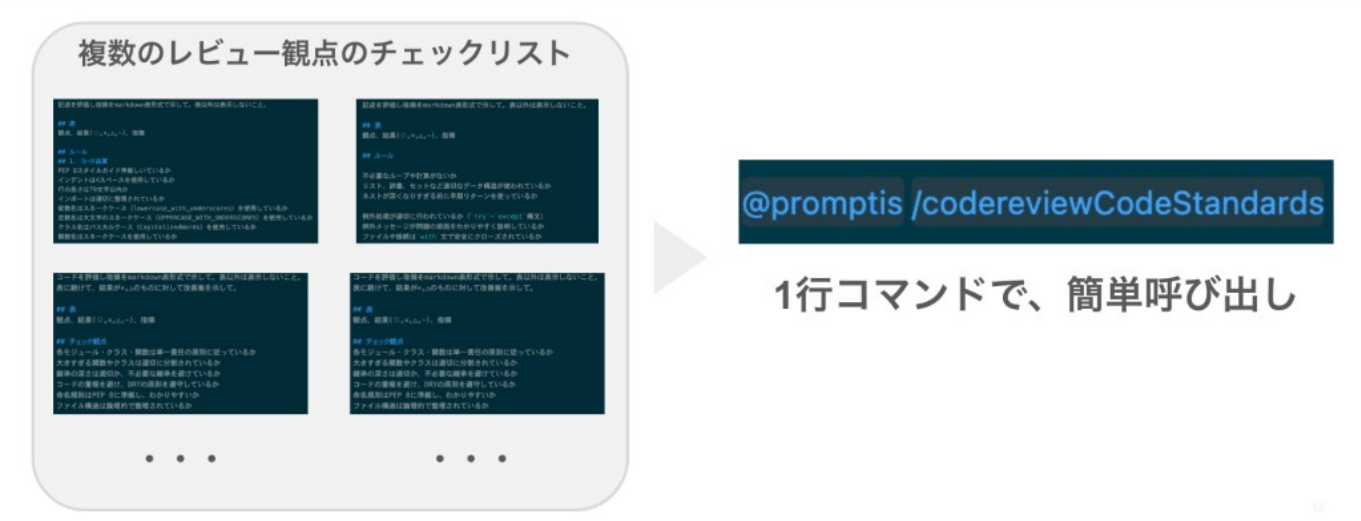
## 「組織全体の効率化」に向けた施策

TISでは、AIコードレビューの生産性効果の最大化に向けてふたつの施策を展開しています。

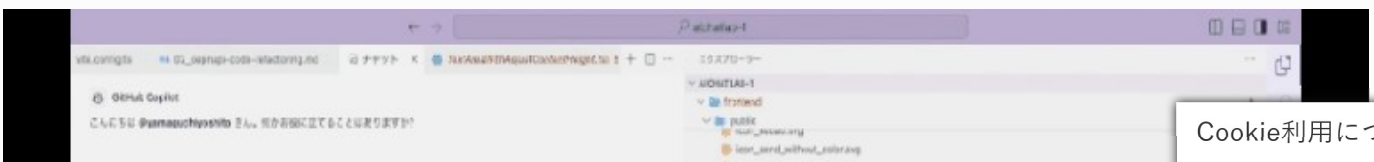
### 1. コマンド簡単呼び出し

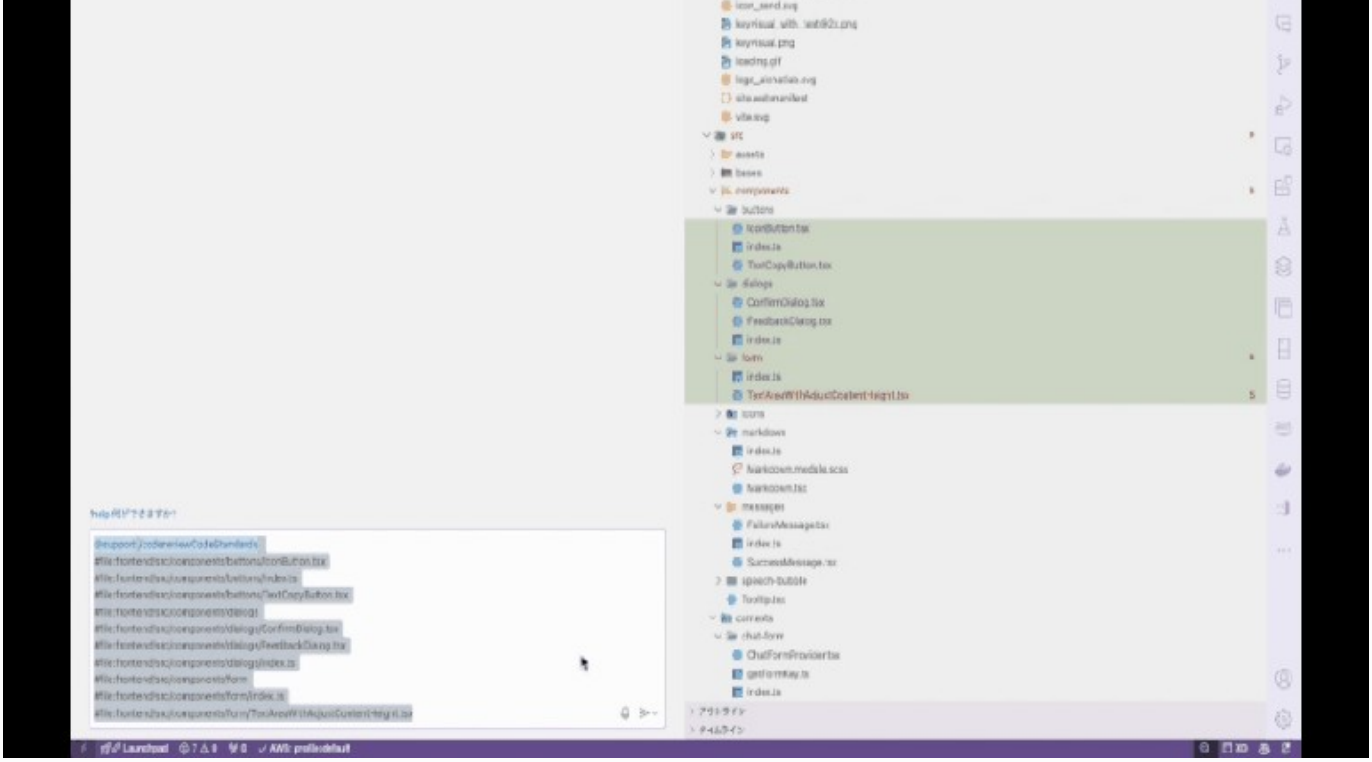
GitHub Copilotを用いた独自ツールを開発し、Visual Studio Code（VS Code）エクステンション「Promptis」 として公開しています。
このツールは事前に用意したレビュー用プロンプトをコマンド一発で呼び出し、連続実行を可能とします。

人間が行う操作はワンタッチで、100以上のコードのレビューのチェック項目を数分で終わらせることができます。




複数のレビュー対象ファイルを順に読み込んで、一括してレビューすることもできます。
このツールはすでに実践導入されていて、高い生産性効果と品質効果を出しています。





複数のReactコードファイルを、一回のコマンド実行で連続レビューさせている

## 2. コードレビュー用プロンプトの整備

TISの開発で使用されることが多い主要なプログラミング言語・アプリケーションフレームワークに適したレビュー用のプロンプトを整備し、AIコードレビューで呼び出しやすい形式（mdファイル）にして配布しています。プロンプトは、AIのレビュー品質の揺らぎを減らすための記述方法に統一しています。

また、AIによるコードレビュー精度を上げるにはレビュー用のプロンプトファイルは細かく分けた方がベターです。TISでは、プログラミング言語・アプリケーションフレームワーク別、コンポーネント別、レビュー観点別の複数のコードレビュープロンプトを順次拡充しています。

## まとめと今後の展望

本記事では、TISにおけるGitHub Copilotを用いたAIによるコードレビューの活用についてご紹介しました。

AIによるコードレビューは人間のチェックを完全に代替するものではありません。また、組織としての最適なレビュープロセスの整備自体は、AIが解決してくれるものではありません。AIコードアシスタントのメリットを最大限享受するには、**AI利用の注意点を認識し、AIツール利用を含む開発プロセスそのものを適切にコントロールする**ことが必要です。

TISでは今後もGitHub Copilot利用を進めていきます。

## 参考資料

1.Stecklein, Jonette M., Dabney, Jim, Dick, Brandon, Haskins, Bill, Lovell, Randy, Moroney, Gregory. “Error Cost Escalation Through the Project Life Cycle“. NASA Technical Reports Server (NTRS). 2004年6月19日 (参照 2025-01-27).↩

2.Eick, Stephen G., Graves, Todd L., Karr, Alan F., Marron, J. S., Mockus, Audris. “Does Code Decay? Assessing the Evidence from Change Management Data“. IEEE Transactions on Software Engineering. 1999年 (参照 2025-01-27).↩

3.Thoughtworks. “Technology Radar Vol 31“. Thoughtworks Technology Radar. 2024年10月 (参照 2025-01-27).↩

/* Recommend */

## 「Generative AI（生成AI）」 のおすすめ記事はこちら

この記事に関連する記事もお読みください。

Generative AI（生成AI） NEW

OctoNihon Forumイベント発表資料「GitHub Copilotを活用した大規模開発の 今 と 未来」公開

2025/06/06 4

Generative AI（生成AI）

インフラコードに対するAIコードレビュー：GitHub CopilotとPromptisの活用

2025/03/26 9

Generative AI（生成AI）

GitHub Copilotを活用した大規模開発 ~オフショア開発での実践と知見~

2025/02/26 16

最近投稿された記事も用意しました。

Generative AI（生成AI） NEW

「金融業界特化型AIエージェントワークショップ」開催レポート

2025/06/13 8

Generative AI（生成AI） NEW

OctoNihon Forumイベント発表資料「GitHub Copilotを活用した大規模開発の 今 と 未来」公開

2025/06/06 4

ITアーキテクチャ NEW

生成AIの新潮流：AIエージェント勉強会を開催しました

2025/05/02 12

「Generative AI（生成AI）」 で最も読まれている記事を以下にまとめています。

Generative AI（生成AI）

GitHub Copilotを用いたAIによるコードレビューの活用

2025/01/28 26

Generative AI（生成AI） NEW

OctoNihon Forumイベント発表資料「GitHub Copilotを活用した大規模開発の 今 と 未来」公開

2025/06/06 4

Generative AI（生成AI）

GitHub Copilotを活用した大規模開発 ~オフショア開発での実践と知見~

2025/02/26 16