

ソフトウェア開発における 生成AI活用の現在地と 品質面での考慮点





服部 佑樹 @yuhattori

シニア アーキテクト
GitHub

ヴァイスプレジデント
The InnerSource Commons Foundation



ソフトウェア開発における 生成AI活用の現在地



GitHub Platform

安全なソフトウェアを
ビルド/スケール/リリース
できる AI 駆動の開発基盤





Powered by AI



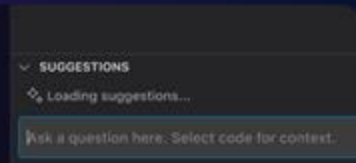
Copilotで開
発業務が
55%速く



75%の
エンジニアが
より満足

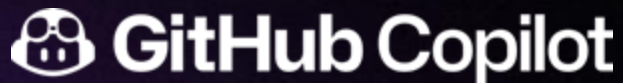


20,000+ の企
業にて
採用済み



繰り返し作業の精
神的労力が
87%低下

利用者が
100万人以上



自分のプロジェクトに 特化したコード提案を提示

✓ コメントをコードに変換

✓ 繰り返しコードの自動補完

✓ 代替手段の提示

TS sentiment.ts

write_sql.go

parse_expenses

```
1 #!/usr/bin/env ts-node
2
3 import { fetch } from "fetch-h2";
4
5 // Determine whether the sentiment of
6 // Use a web service
7 async function isPositive(text: string) {
8   const response = await fetch(`http://
9     method: "POST",
10    body: `text=${text}`,
11    headers: {
12      "Content-Type": "application/x-www
13    },
14  });
15   const json = await response.json();
16   return json.label === "pos";
17 }
```

Copilot



人気のコードエディタ向けの拡張機能



Visual Studio Code



All JetBrains IDEs

IntelliJ IDEA, WebStorm, PyCharm, Rider, etc.








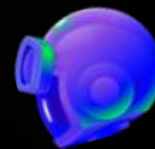
Visual Studio



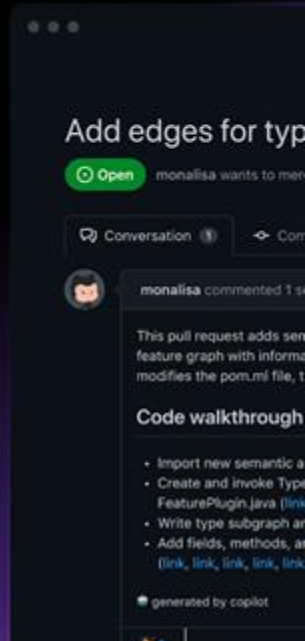
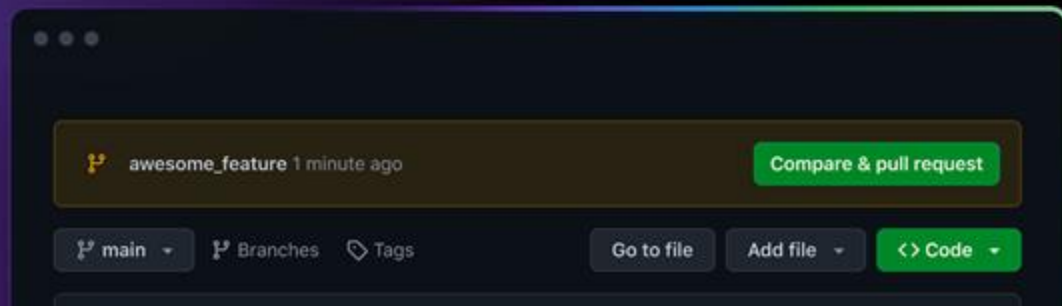
neovim



 .eslintignore	improve getting started docs	6 days ago
 .gitignore	improve getting started docs	6 days ago
 .npmrc	improve getting started docs	6 days ago
 jest.config.js	improve getting started docs	6 days ago
 package.json	update nodegit	3 days ago

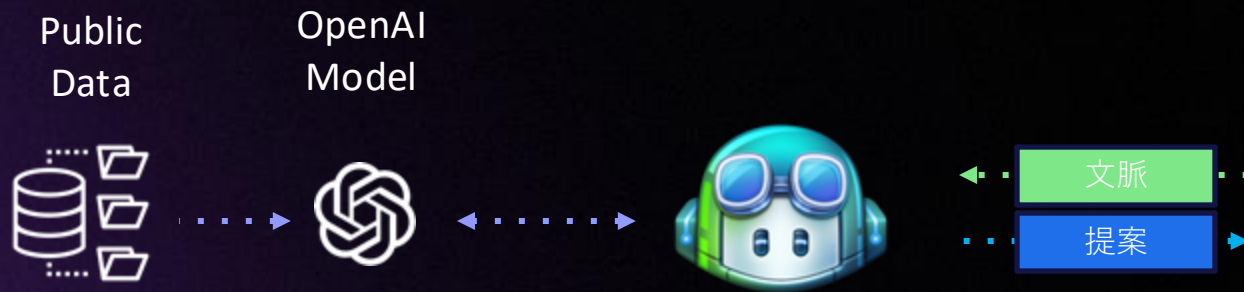


Copilot Completion Demo





概要



```
33 translation = trans
34 assert translation
35
36
37 def juxtapose(image, text):
38     """create a new image
39     Make the new image
40     Display the text in
41     # create a new image
42     new_image = Image.new
43
44     # paste the original
45     new_image.paste(image,
46
47     # get a font
48     font = ImageFont.tru
49
50     # get a drawing con
51     draw = ImageDraw.Dra
52
```

PROBLEMS OUTPUT COMMENTS

```
ryanjsalva in ~/Code/copilot_de
$ source /Users/ryanjsalva/Code

ryanjsalva in ~/Code/copilot_de
$ /Users/ryanjsalva/Code/copi
Traceback (most recent call la
  File "/Users/ryanjsalva/Code
    from google.cloud import ti
ImportError: cannot import nam

ryanjsalva in ~/Code/copilot_de
$
```

Python 3.10.0 64-bit ('.venv': venv) 0.0



データの流れ





35%

提案採用率

46%

Copilotが生成した
コードの割合



95人の開発エンジニアを集め
二つのグループに分けました

JavaScript でウェブサーバーを定義するタスクを与えました

Copilotなし



Copilotあり

50

エンジニア数

45

70%

完了した割合

78%

2時間 41分

完了までの平均所要時間

1時間 11分

Copilot プロンプトのコツ

Context
文脈

AI がタスクを
よりよく理解するのに
役立つ

Intent
意図

プロンプトを
作成する際の
具体的な目標や目的

Clarity
明瞭さ

曖昧さがなく
理解しやすいこと

Specificity
具体性

プロンプトの
具体性と
正確さのレベル



Copilot Chat

文脈を認識したCopilotとのChatGPT式チャット機能

単体テストの生成

Copilot Chatはエディタ内にある既存コードやエディタで選択されているコード部分に基づいて単体テスト用のコードスニペットを生成することが可能。

コード解説

Copilot Chatは選択されているコードに対して自然言語で対象のコードの性能や意図が含まれた解説文を生成することが可能。

修正コードの提案

エラーや問題の文脈に応じてCopilot Chatからバグに対する修正コードや解決案を提案することを補助することが可能。

コードに対する質問への回答

特定のコード問題に対して助けや補助をCopilot Chatにお願いすることが可能で、返事は自然言語またはコードスニペットとして受信可能。

開発言語間の変換

Copilot Chatは一つの開発言語から別の開発言語に変換することを補助することが可能。

可読性向上

Copilot Chatは抜けているコメントの追加やより適切な変数名の提案等でコード全体の可読性向上を補助することが可能。

GITHUB COPILOT: CHAT



monalisa

Write a set of unit test functions for the selected code



GitHub Copilot

```
import unittest
import datetime

class TestParseExpenses(unittest.TestCase):

    def test_parse_expenses_with_valid_input(
        expenses_string = '2023-01-02 -34.01'
        expected_output = [(datetime.datetime
        self.assertEqual(parse_expenses(expen

    def test_parse_expenses_with_empty_input(
        expenses_string = ''
        expected_output = []
        self.assertEqual(parse_expenses(expen

    def test_parse_expenses_with_comments(sel
        expenses_string = '# This is a commen
        expected_output = [(datetime.datetime
        self.assertEqual(parse_expenses(expen

if __name__ == '__main__':
    unittest.main()
```

The code assumes that the datetime module is imported. The test cases cover different scenarios such as valid input, empty input, with comments, invalid date format, and invalid value format.

Analyze and debug

Ask a question or type '?' for topics



Copilot Chat

文脈を認識した Copilot とのチャット機能

スラッシュコマンド

`/explain` → 選択されたコードの仕組みを解説

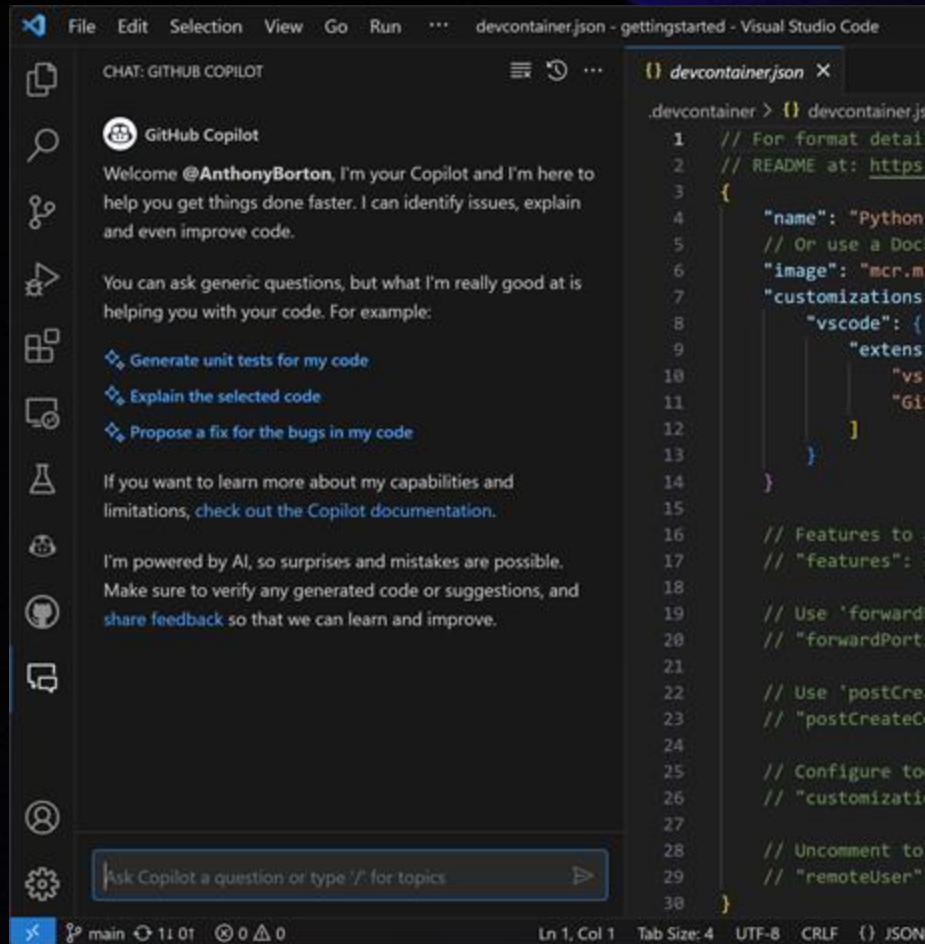
`/fix` → 選択されたコードの問題点に対し修正方法を提案

`/help` → GitHub Copilot に関する一般的なヘルプ






`/tests` → 選択されたコードに対して単体テストを提案

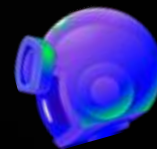
`/vscode` → VS Code に関する質問への回答

`/clear` → セッションをクリアする

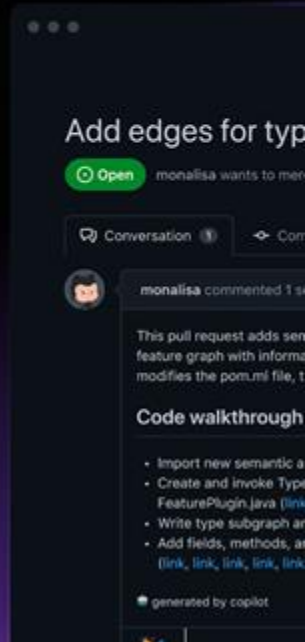
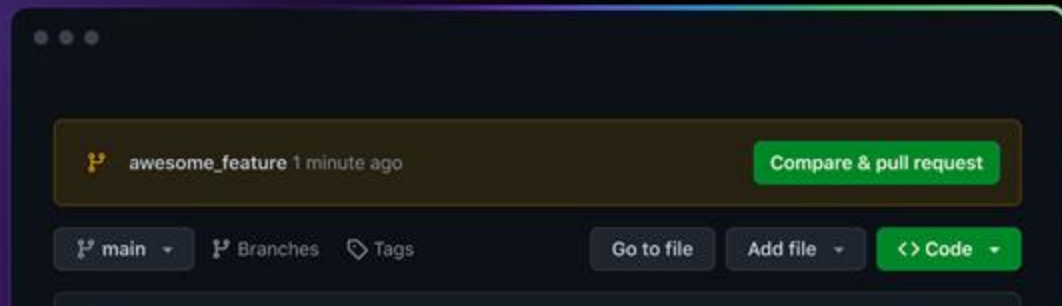




 .eslintignore	improve getting started docs	6 days ago
 .gitignore	improve getting started docs	6 days ago
 .npmrc	improve getting started docs	6 days ago
 jest.config.js	improve getting started docs	6 days ago
 package.json	update nodegit	3 days ago



Copilot Chat Demo



GitHub Copilot の特徴を理解

ツールに求めるべき質 は モデルに求めるべき質 とは異なる

ユーザーの入力を
最小化

インクリメンタルな実装
手戻りの許容

迅速なレスポンスと
集中力の維持

確実に
安全な出力

ツールの質は、ユーザーからの明示的なプロンプトなしに、望まれる出力をどれだけ正確に予測できるかに依存

Copilot Completion はアジャイルなやり取りをサポート

段階的コード出力とユーザーによる逐一の採否決定が可能

自動補完型ツールの利点は、迅速なレスポンスによる集中力の維持

世界最大/最新のオープンソースデータベースと瞬時にコードマッチング、ライセンスに関する問題発生を抑制

ツールの違いを知る

	自動補完型	対話型	エージェント型
入力	エディタへの入力	フォームへの入力	フォームの入力、ファイルアップロードなど
出力内容	コード / コメント	コード / コメント / 解説文	編集可能なコードを含む成果物
出力形式	コードの補完	コードを含む返信	成果物の出力
コードサイズ	小規模 (1行~20行程度)	中規模 (1行~数百行)	大規模 (数百行~数千行)
明示的なプロンプト	0行 ~ 数行程度	数行 ~ 数十行	大量
応答速度	数百ミリ秒 ~ 数秒	数秒 ~ 数十秒	数十秒 ~
ツールのフォーカス	応答速度、開発時の集中力	精度、プロンプト構築補助	ソリューションとしての完成度
類似体験	ペアプログラミング モブプログラミング	Sack や Teams での技術質問 軽量のプルリクエスト受け取り	ローコードツールの利用 テンプレートエンジンの利用
ツール例	GitHub Copilot (Completion)	ChatGPT GitHub Copilot Chat	Copilot Workspace



GitHub Copilot Future





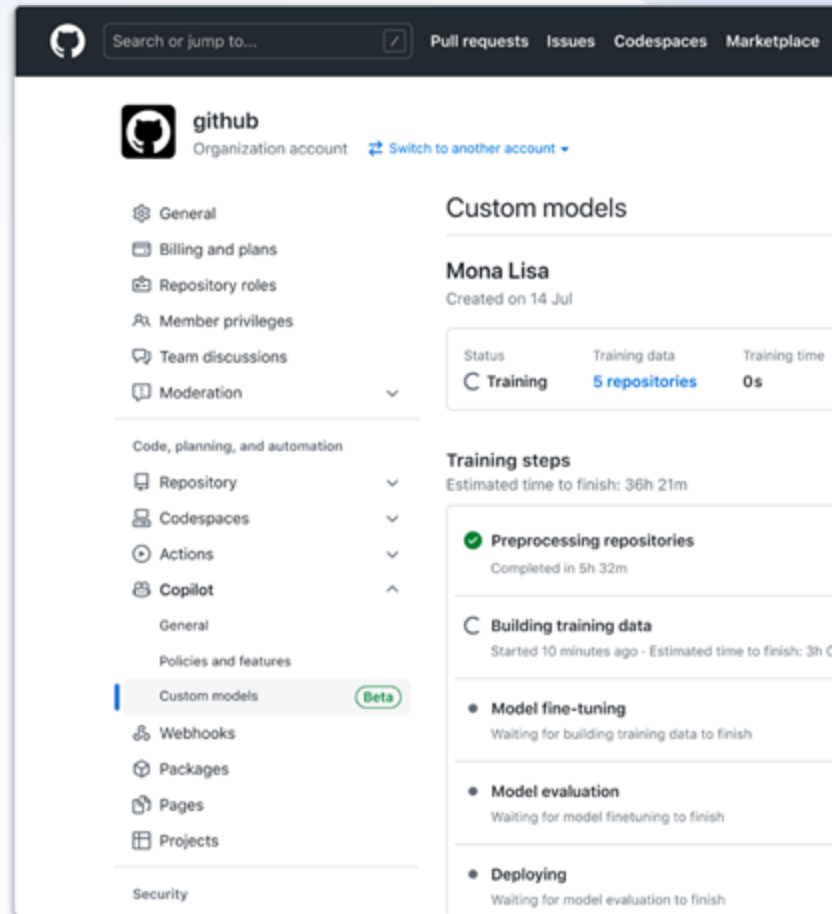
パーソナライゼーション モデルのファインチューニング

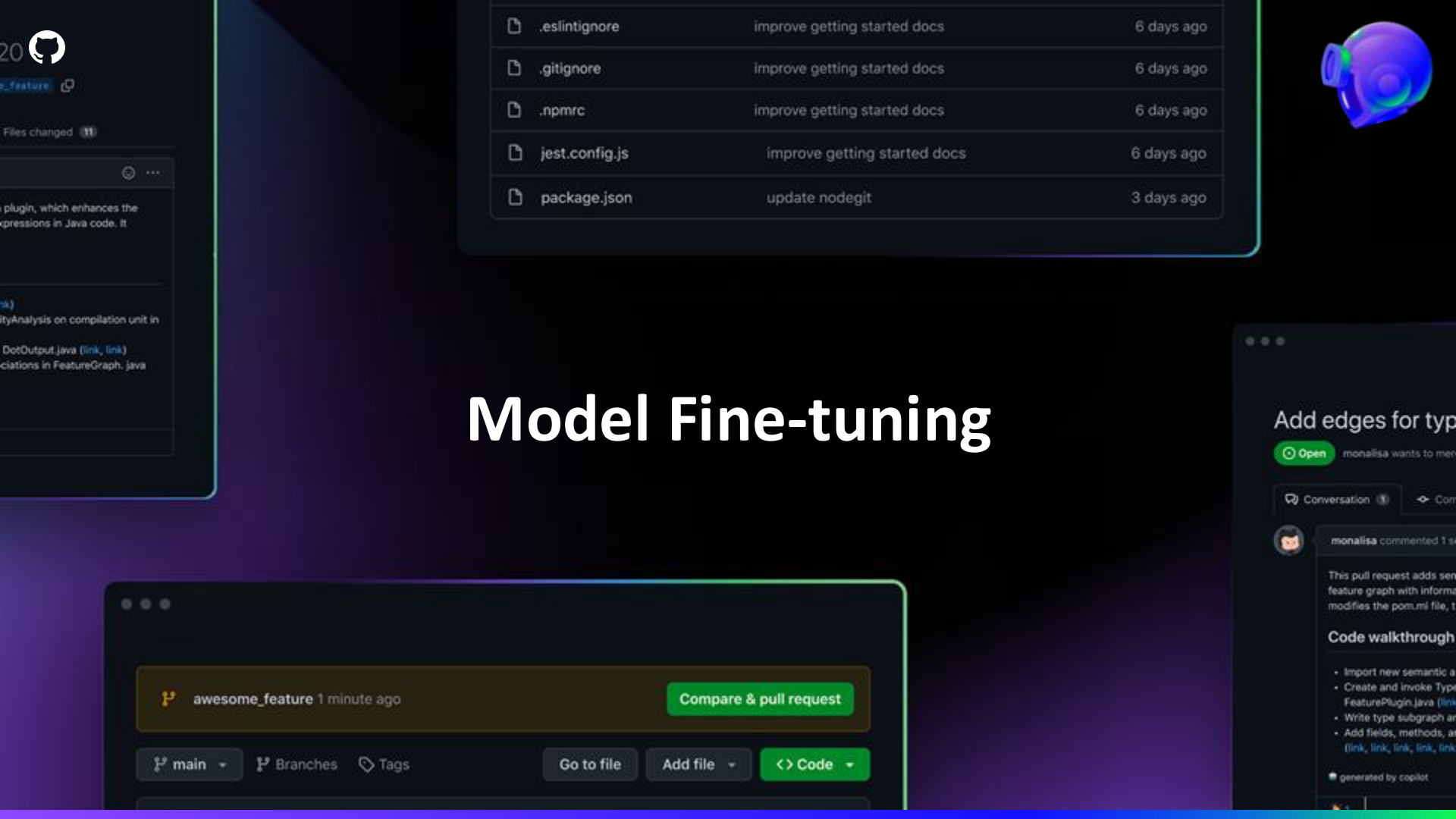
- ✓ より適切でカスタマイズされたコード提案を表示
- ✓ より早く、より少ないエラーでコードを記述可能
- ✓ コード参照、モデリング、ホスティング、参照はプライベート
- ✓ 独自言語や関連性の低い言語に基づくコード提案



Copilot Enterpriseをご利用のお客様専用のアドオン






Add-on

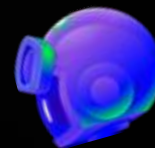




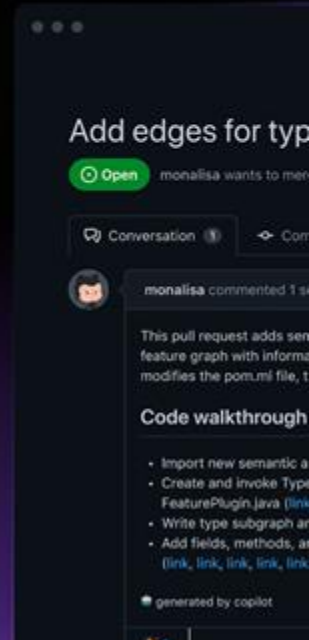
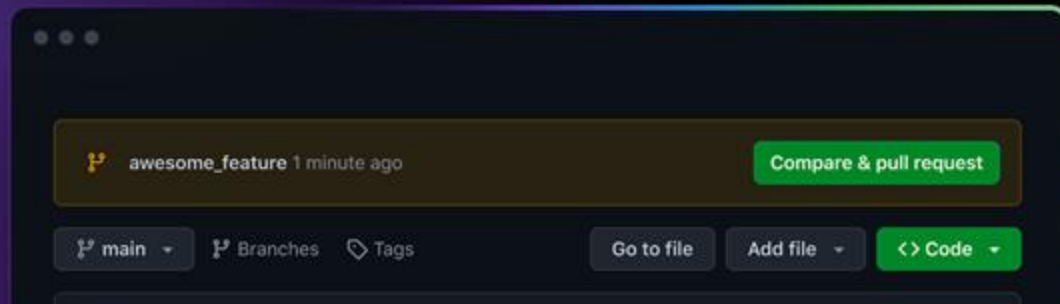
Model Fine-tuning



 .eslintignore	improve getting started docs	6 days ago
 .gitignore	improve getting started docs	6 days ago
 .npmrc	improve getting started docs	6 days ago
 jest.config.js	improve getting started docs	6 days ago
 package.json	update nodegit	3 days ago



Copilot Workspace

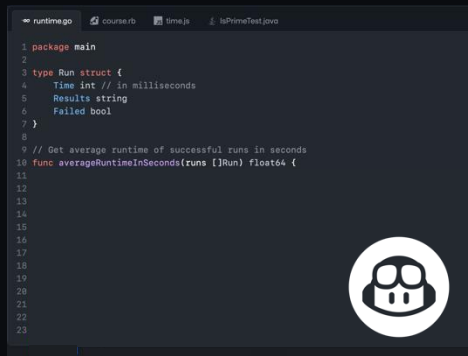




GitHub Copilot Behind the Curtain

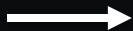


GitHub Copilot の 3 Layer



Copilot Client

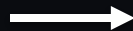
Githubによって開発及び保守。
カスタムプロンプト作成と自動補完ユ
ーザーエクスペリエンスを提供



API

Copilot Platform

Githubによって開発及び保守。
認証、セキュリティ、プライバシー



OpenAI Model

OpenAI によって開発
Azure でホスト
スケーラブルなデリバリー

* 上記は GitHub Copilot のアーキテクチャを抽象化した図です



Prompt Crafting

- Language Marker: プログラミング言語情報
- Path Marker: 現在のファイルへのパス
- Neighboring Tabs: 非アクティブなオープンしているタブ



Prompt Crafting: Markers

Path:

foo/foo.py

Language Marker

HTML: <!DOCTYPE html>

Python: #!/usr/bin/env python3

Ruby: #!/usr/bin/env ruby

ファイルパスおよび言語マーカーは、強力なシグナルを言語モデルに提供し、出力の構文と文法を調整します。



Prompt Crafting: 隣接ファイル

Open Tab 1

```
func main() {  
  
    var a [5]int  
  
    var twoD [2][3]int  
    for i := 0; i < 2; i++ {  
        for j := 0; j < 3; j++ {  
            twoD[i][j] = i + j  
        }  
    }  
    fmt.Println("2d: ", twoD)  
}
```

Open Tab 2

```
package main  
  
import "fmt"  
  
func main() {  
  
    var a [5]int  
  
    // put 10 to 50 with 10 increments (inclusive) into the array u  
    fmt.Println("emp:", a)  
  
    a[4] = 100  
    fmt.Println("set:", a)  
    fmt.Println("get:", a[4])  
  
    fmt.Println("len:", len(a))  
  
    b := [5]int{1, 2, 3, 4, 5}  
    fmt.Println("dcl:", b)  
  
    var twoD [2][3]int  
    for i := 0; i < 2; i++ {  
        for j := 0; j < 3; j++ {  
            twoD[i][j] = i + j  
        }  
    }  
    fmt.Println("2d: ", twoD)  
}
```

他のオープンタブからのコード
スニペットは類似性を検索し、
プロンプトに追加のコンテキス
トとして注入されます。

Jaccard Similarityでファイル内を検索

- GitHub Copilot は 現在 Jaccard Similarity を活用して類似性を求める
 - 関数名はシンプルかつ、意味のあるものにする必要性
 - AB テストなどでこの方法は変わります / カーソル履歴など
- 一貫して言えること
 - 一貫性のある命名規則ときれいなコード
 - 現在何をしているのかにフォーカスしながら作業



生成AIとソフトウェアエンジニアリング

～ 品質の高いソフトウェアを生み出すために ～

[参加者配布短縮版]

9/19 発売

Special Thanks (敬称略)

森崎修司, 和田卓人, 山口鉄平, 金子昌永,

牛尾剛, 三田雅人, 吉田素文, 黒崎優太, 百田涼佑

bit.ly/CodeAndAI

ChatGPTやGitHub Copilotに代表される生成AIの躍進は
ソフトウェア開発を一変させた。
AIとともに進めるコーディング、コードリーディング、
ドキュメンテーション、要件定義、設計、レビュー。
新時代の必須スキルを習得する。

コード



服部佑樹

ソフトウェア開発者のための
生成AI実践入門

AI

生成AIの
真価を発揮する
知識と実践

100以上のPracticeを掲載

生成AI時代の開発の在り方
AIの理想と現実
AIはエンジニアの仕事を奪うのか?
補完・対話・エージェント
開発者向けAIツールの分類
プロンプトエンジニアリング
GitHub Copilot / GitHub Copilot Chat /
GitHub Copilot Workspace
ChatGPT / Claude
AIエージェント
AI活用ロードマップ
組織導入
プロンプト最適化戦略
プロンプトリーディング
AI可読性を高める
効果の測定・評価
エディター連携

技術評論社

トピック

- 生成AIがエンジニアリングの常識を変える
- プロンプトで生成AIを操る
- プロンプトの実例と分析
- AIツールに合わせたプロンプト戦略
- AIと協働するためのコーディングテクニック
- AIの力を引き出す開発アプローチ
- 生成AIの力を組織で最大限に引き出す
- 開発におけるAI活用Tips
- AI時代をリードするために

bit.ly/CodeAndAI

ChatGPTやGitHub Copilotに代表される生成AIの躍進は
ソフトウェア開発を一変させた。
AIとともに進めるコーディング、コードリーディング、
ドキュメンテーション、要件定義、設計、レビュー。
新時代の必須スキルを習得する。



AIによる作業単位の最適化

関心の分離によるコード最適化

コードを適切に分割し、AIに与える情報を最適化する。

クラスを関心ごとに分割し、シンプルな構造にすることで、生成されるコードの品質向上を図る。

```
class DataManagementUtil:
    def save_data(self, path, data):
        # 中略

class DataAnalyzer:
    def analyze_foo_data(self, data):
        # 中略

    def analyze_bar_data(self, data):
        # 中略

# 関連する処理を適切に分割
```


コードのAI可読性向上

AIとの協働を意識した命名

変数や関数に具体的で説明的な名前を採用する。

適切な命名により、AIが提案するコードの品質を向上させ、人間の開発者とAIの両方が理解しやすいコードを作る。



```
# 英語で名前をつける  
property_id = "BK0012"  
# 判別可能な名前  
administrative_fee = 12000  
# 単位を含めて具体的な名前をつける  
price_per_square_meter = 180
```

コードのAI可読性向上

検索最適化された命名戦略

統一された命名規則を採用し、
検索にヒットしやすいコードを
書く。

AIツールが適切なコードを提案
するために、一貫性のある命名
を心がける。

```
# bukken_controller.py  
bukken_id = "R001"
```

```
# property_model.py  
property_id
```

```
# migration.py  
self.id =  
"R001"
```


AIと協働する際のコーディングスタイル

スタイルガイドの明示的提供とカスタマイズ

AIによるコード生成時に標準的なスタイルガイドに従うよう指示する。

標準的なスタイルガイドをベースに、必要に応じて最低限のカスタム規約セットを作成する。

AIへのコーディング時の規約伝達を最小限に抑え、効率的な連携を実現する。



```
// Google Java Style Guide に  
// 従ったコードにすること
```

AIに適したコードアーキテクチャ

将来の拡張を考慮したコード設計

既存コードを改変せずに新しいコードを追加できるよう設計する。

コードの保守性や拡張性が向上し、AIによる開発スピードを妨げない。

(OCP原則)

```
interface PaymentGateway {  
    void handleTransaction(double amount);  
}
```

```
class CreditCardPaymentHandler implements  
PaymentGateway {  
    @Override  
    public void handleTransaction(double  
amount) {  
        // クレジットカードの支払い処理  
    }  
}
```

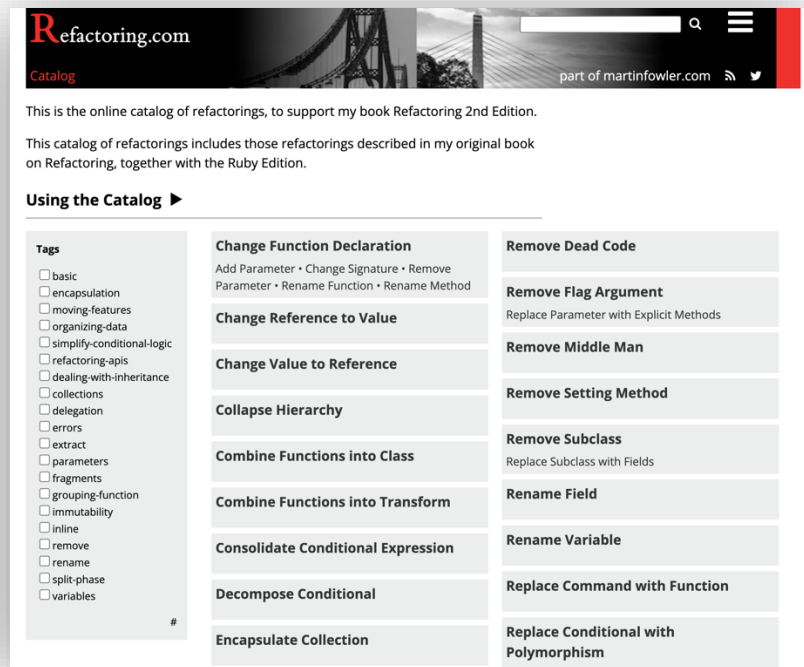
```
class QRPaymentHandler implements  
PaymentGateway {  
    @Override  
    public void handleTransaction(double  
amount) {  
        // QRの支払い処理  
    }  
}
```

AIに適したコードアーキテクチャ

体系的なリファクタリング手法の適用

リファクタリングカタログなどを活用し、AIにより具体的な提案を引き出す。

効果的なリファクタリングと品質の高いコード開発につながる。

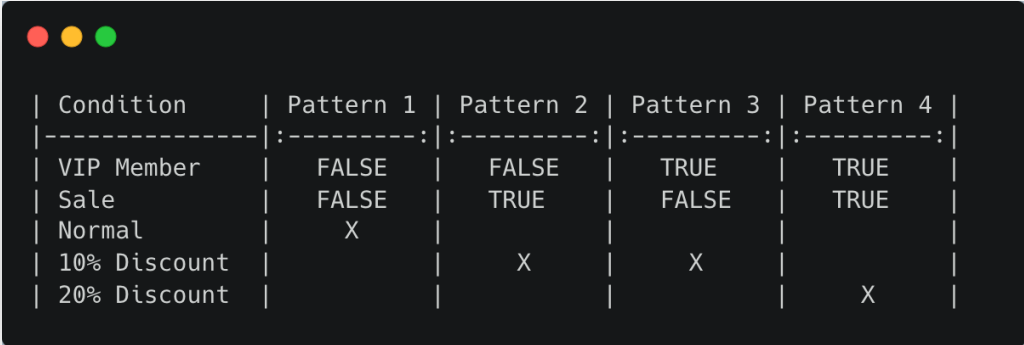


AIを活用したコード品質向上

網羅的テスト設計のためのデシジョンテーブル活用

デシジョンテーブルを作成し、それにもとづいてテストコードを生成する。

より網羅的で有効なテストコードの作成が可能になる。

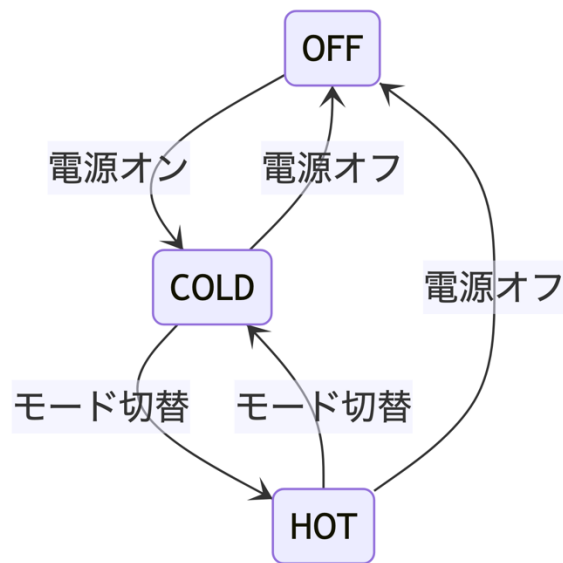


Condition	Pattern 1	Pattern 2	Pattern 3	Pattern 4
-----	-----	-----	-----	-----
VIP Member	FALSE	FALSE	TRUE	TRUE
Sale	FALSE	TRUE	FALSE	TRUE
Normal	X			
10% Discount		X	X	
20% Discount				X

AIを活用したコード品質向上

状態遷移図を経由したテストコード生成

状態遷移図を作成し、それにもとづいてテストケースを確認し、テストコードを生成する。視覚的に状態遷移を確認し、より確実なテストコードを生成できる。



AIを活用したコード品質向上

AI時代にはシフトライトが必要になるのか？

シフトライトに関する言説には「今後エージェント型の生成ツールがあらわれて、ソリューションを一気に作ったら」という枕詞が隠されている。

将来、AIの生成するコードの質が飛躍的に向上し、一気にアプリケーションを生成できるようになれば、シフトライトの考え方が有効になるでしょうが、現時点ではまだそのレベルには達していないのが実情です。

SHIFT RIGHT?



AI時代の競争優位性を高めるための開発組織戦略

オープンソースの文化を組織に取り入れる

ここであらためてAIが読みやすいコードの特徴をまとめると、以下のようなことが挙げられます。

- コードが記述的かつ文脈が明確で、初見でも理解しやすいこと。
- 比較的メジャーなバージョンや技術スタックで書かれている。
- 継続的にメンテナンスされ、常に使用可能な状態であること。

AI時代の競争優位性を高めるための開発組織戦略

インナーソースの原則



オープン

ソースコードは組織内で公開され、誰もが自由にアクセスできます。これにより、開発者は他のチームのコードを参照し、学ぶことができます。知識の共有が活発に行われ、全体のスキルアップにつながります。



透明性

コードだけでなく、議論の過程も公開されています。つまり、意思決定がどのように行われたかが透明になっています。これにより、他チームの開発者も容易に参入でき、プロジェクトは組織全体の共有物と認識されます。



優先的な メンターシップ

新しい開発者が参加しやすいよう、優先的にサポートが行われます。ゲストチームの貢献をフォローして開発参入の障壁を下げ、プロジェクトへの新規貢献が継続的に生まれていきます。



自由意志による 貢献

プロジェクトへの貢献は強制されるものではありません。プロジェクトからのサポートも自由意志で行われます。各チームは互いを尊重し合い、協調して開発を進めていきます。

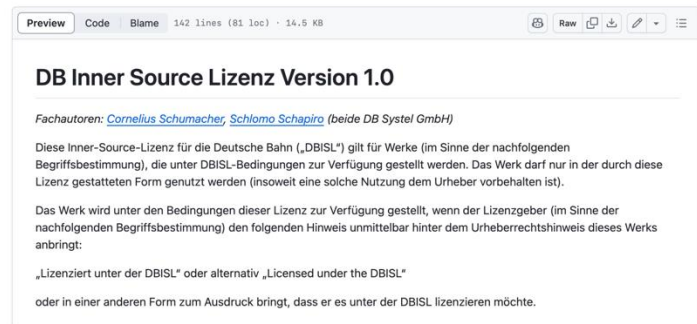
AI時代の競争優位性を高めるための開発組織戦略

組織内コード共有のルール化

AIによるコードの活用には、ソースコードの自由な利用が前提となります。

その「自由な利用の範囲」は、組織全体ではなく、特定の部門やプロジェクトかもしれませんが、少なくともその範囲を明確にすることが大切です。

また、フリーソフトウェアの4つの自由である「使用する自由」「変更する自由」「共有する自由」「変更したソフトウェアを再配布する自由」を社内限定して適用できるようにライセンスに統合することも考えられます。



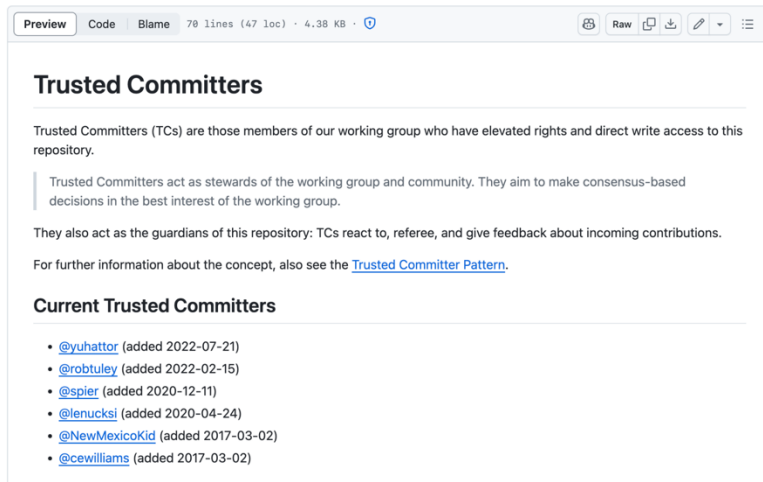
AI時代の競争優位性を高めるための開発組織戦略

メンテナーの明確化

社内のリポジトリ管理者を明確にするには
「トラステッドコミッター」という概念が有効。

一般的なメンテナーやコミッターの概念に加え、以下のような社内事情を考慮している点が特徴。

- 組織内のチーム間の貢献を認識するための仕組みや言語を提供する。
- ビジネスの優先順位の変化に対応するため、メンテナーのフォーカスのずれを考慮する。
- 従業員の評価に組み込めるよう、公式な役割として定義する。
- 従業員でなくなる等による退任プロセスを考慮する。
- 組織内での公式な認定プロセスを設定する。



Thank you

bit.ly/CodeAndAI

ChatGPTやGitHub Copilotに代表される生成AIの躍進は
ソフトウェア開発を一変させた。
AIとともに進めるコーディング、コードリーディング、
ドキュメンテーション、要件定義、設計、レビュー。
新時代の必須スキルを習得する。



服部佑樹

ソフトウェア開発者のための
生成AI実践入門



生成AIの
真価を発揮する
知識と実践

100以上のPracticeを掲載

生成AI時代の開発の在り方
AIの理想と現実
AIはエンジニアの仕事を奪うのか?
補完・対話・エージェント
開発者向けAIツールの分類
プロンプトエンジニアリング
GitHub Copilot / GitHub Copilot Chat /
GitHub Copilot Workspace
ChatGPT / Claude
AIEージェント
AI活用ロードマップ
組織導入
プロンプト最適化戦略
プロンプトリーディング
AI可読性を高める
効果の測定・評価
エディター連携

技術評論社