

本資料は「NTT Tech Conference 2024」(2024年3月11日開催)における講演資料です。



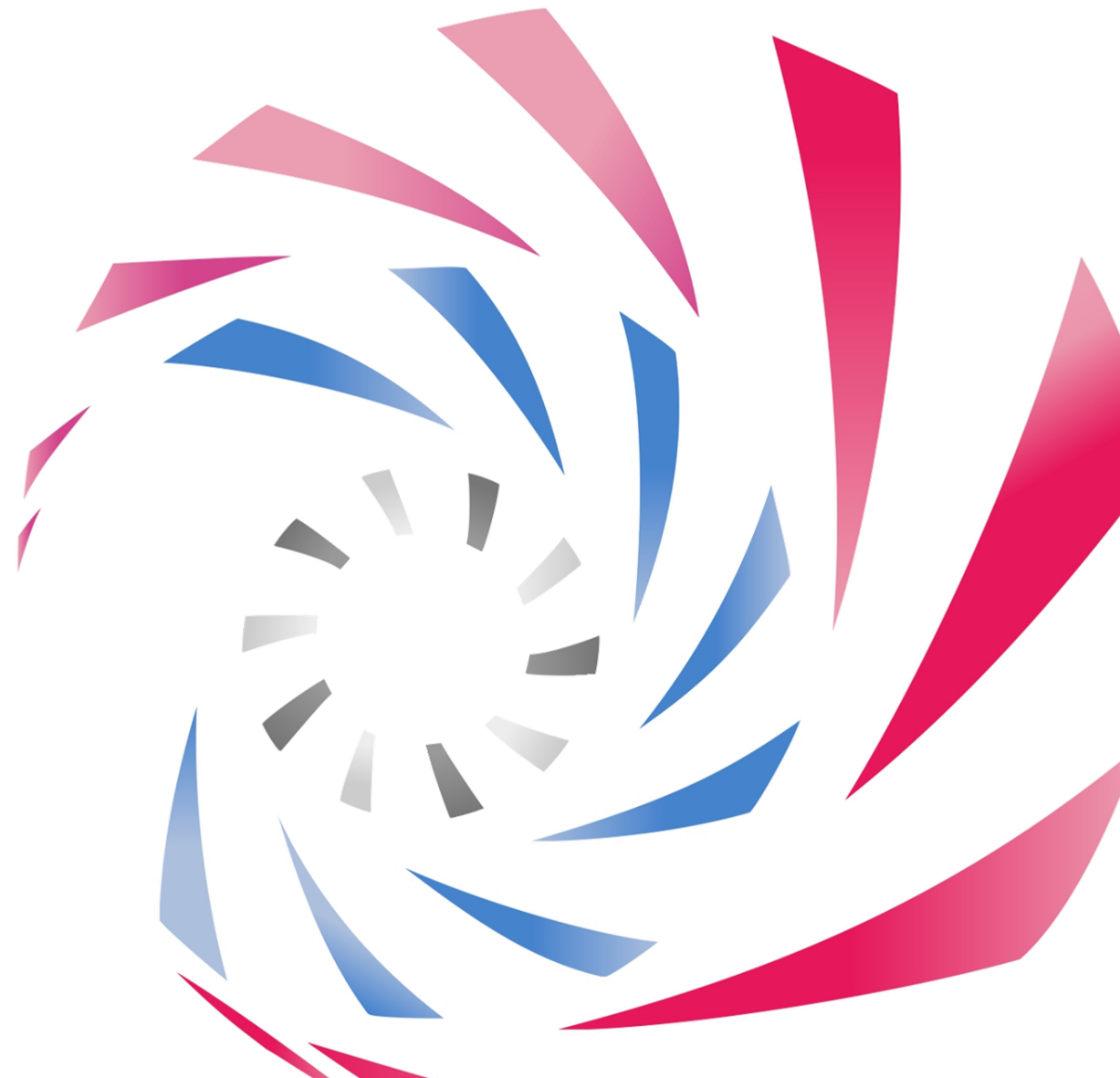
# 生成AIを活用したソフトウェア 開発効率向上の取り組み紹介

2024年3月11日

株式会社NTTPCコミュニケーションズ

サービスクリエイション本部 第一サービスクリエイション部  
サービスクリエイション担当

川尻 真寛



# ▶ Index

1. イントロダクション	.....P. 03
2. 取り組み紹介	.....P. 15
3. 効果測定	.....P. 19
4. トライアル	.....P. 27
5. まとめ	.....P. 38

# ▶▶ 1. イントロダクション

NTT PC コミュニケーションズ サービスクリエーション本部 第一サービスクリエーション部 サービスクリエーション担当で今年度取り組んだ、生成AIサービス活用によるソフトウェア開発効率の改善効果検証の結果について共有します。

利用した生成AIサービスは下記の通りです。

- 「GitHub Copilot(GitHub社)」
  - 「ChatGPT (OpenAI社)」
- 
- 今回お伝えできること
    - 生成AIを活用することで、どの程度開発スピードが向上するのか。
    - 生成AIは開発者体験をどのように変えるのか。
    - 生成AIは若手エンジニアの育成をどのように支援するか。
  - 留意点
    - 本資料に記載される内容については、再現性を保証するものではありません。

今回の取組によって、生成AIをソフトウェア開発に導入すると次のような効果を発揮することを確認した。

- ソフトウェア開発スピードの向上
  - 特定ケースの効果測定において、従来方式と比較して**4割**時間短縮できる。
  - 通常業務のトライアルにおいて、**1日あたり約68分**の作業時間が短縮できる。
- 開発者体験の改善
  - 生成AI利用者の**8割**が業務に対するモチベーションアップを感じている。
- 若手エンジニアの育成支援
  - 若手エンジニアの**8割**が自身が携わる案件に対する貢献度の向上を感じている。
  - 若手エンジニアは**知識の獲得**に生成AIを活用する機会が多く、そこに対して強力な支援ツールとして機能している。

名前：川尻 真寛（かわしり まさひろ）

所属：NTTPCコミュニケーションズ サービスクリエーション本部 第一サービスクリエーション部  
サービスクリエーション担当

経歴：2007年度PC入社

以降、様々なソフトウェア開発に従事

- ・「Master'sONE®」、データセンタ系業務システム開発
- ・フレッツ・モバイル帯域制御システム開発  
など。

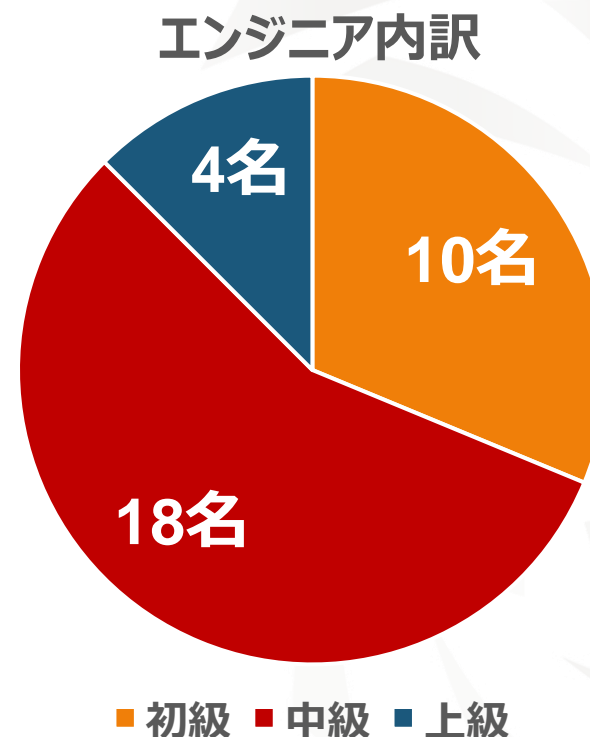
趣味：ゲーム、麻雀

## ▶ 2. 取り組み紹介

- 自社サービスのソフトウェア開発を担う部隊。
- アジャイル開発の推進や脆弱性対応等への即応性向上のため、ここ数年内製化率を高めている。現在60%程度。
- 市場変化への追従・競争力維持のためソフトウェア開発効率の追求に日々取り組む一方で、上級エンジニアが不足しており、その他層のスキルアップひいては上級エンジニアの増員が喫緊の課題。

- 今回の取組では、次の2つをテーマとして効果を検証。

1. ソフトウェア開発効率の向上
2. 初級エンジニアのスキルアップ



初級	補助を受けながらコーディングできるレベル 主に若手(1年目~3年目)
中級	コーディングには補助が不要だが、要件整理や設計には補助が必要なレベル
上級	上記を補助なくこなすことができ、他のメンバーの支援・育成ができるレベル

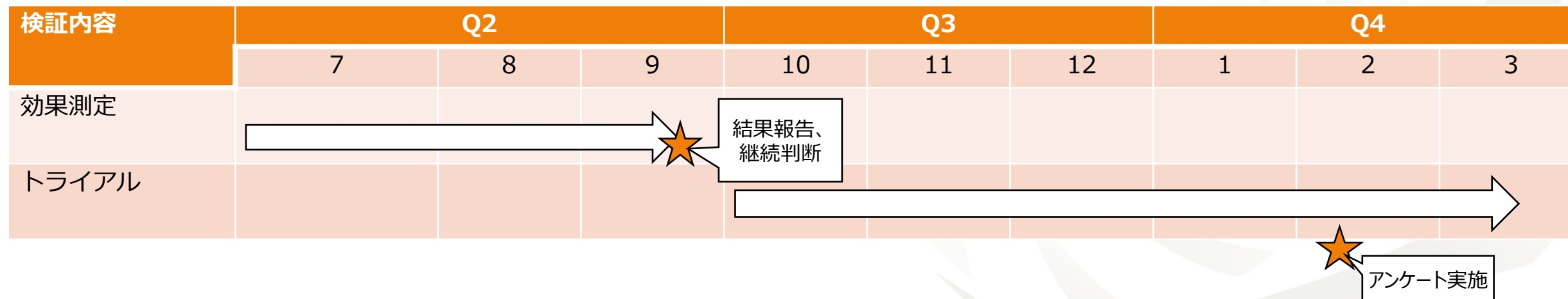


## ▶ 取り組み内容

- 以下の通り、2段階構成で実施。
  - 1段階目は検証として開発時間の短縮効果を測定。(同じ作業を2回実施)
  - 2段階目は開発ツールとして通常業務の中で利用し、厳密な効果測定せずにアンケートで主観的な効果・効能を集計。

フェーズ	被験者	検証項目	検証対象	内容	期待する効果
効果測定	自チーム	ソフトウェアメンテナンスの効率化	GitHub Copilot	言語やフレームワークのバージョンアップ時のコード変更をCopilotが対応してくれるか。	コード変更にかかる時間の30%削減
		エンジニアの自己学習の加速	GitHub Copilot	ソフトウェア実装時のスキル・知識不足をCopilotが補い、独力で問題解決できる機会を増やせるか。	初級エンジニアがチューターの支援なしにソフトウェアの実装ができる割合の20%増加
トライアル	担当内希望者	通常業務での効果確認	GitHub Copilot ChatGPT	導入範囲を拡大しても同様の効果が得られるか。	-

- 全体スケジュールは以下の通り。Q2での効果測定結果を受けて、Q3からトライアルを実施している。



現状は、NTTコミュニケーションズのガイドラインや利用中サービスのプライバシーポリシーを踏まえて、下記ルールで利用中。

- 前提条件  
社内のクラウドサービス利用ルールに則り、情報セキュリティ部門のアセスメントを経て利用すること。
- GitHub Copilot  
privateリポジトリで管理されているプロジェクトのみ利用を許可する。  
(ソースコードの学習防止)
- ChatGPT  
情報資産に該当する情報(社内情報・設計書・ソースコードなど)の投入は禁止。

## ▶ 3. 効果測定

## ▶ 効果測定概要

- 目的
  - GitHub Copilotがもたらすソフトウェア開発効率改善効果を具体的に把握する。
  - 自社の開発スキームの中でも世の中と同じような改善効果を発揮するかを確認する。
- 測定項目

測定項目	目的	期待する効果
ソフトウェアメンテナンスの効率化	言語やフレームワークのバージョンアップ時のコード変更をGitHub Copilotが対応してくれるか。	コード変更にかかる時間の30%削減
エンジニアの自己学習の加速	ソフトウェア実装時のスキル・知識不足をGitHub Copilotが補い、独力で問題解決できる機会を増やせるか。	初級エンジニアがチューターの支援なしにソフトウェアの実装ができる割合の20%増加

- 目的：コード改修作業の時間短縮効果の検証
- 題材：動作環境(OS・言語・フレームワーク)のバージョンアップに対する既存ソフトウェアの適応
- 被験者：3年目社員
- 検証方法：GitHub Copilotを使用しない場合と使用した場合での作業時間の比較

GitHub Copilotを使わなかった場合：

1. バージョンアップ後の動作環境でテストを実行
2. テスト失敗箇所のリストアップ
- 3. 失敗原因の調査及び修正**

GitHub Copilotを使った場合：

1. バージョンアップ後の動作環境でテストを実行
2. テスト失敗箇所のリストアップ
- 3. コードの修正依頼及び微修正**

- 検証結果：GitHub Copilotを使用した方が**4割**程度時間を短縮。
- 考察
  - 起きている事象・修正要望を入力するだけでコードが修正できるため、大幅な時間短縮に繋がった。
  - 入力内容によっては期待するコードを出力しない事態にも遭遇しており、逆に時間がかかるケースの発生も否定できない。ある程度の知識や判断が求められる。

```
# @return [String] 変更後のパスワード
def change_password
  password = new_password
  request_body = {:id => @icsa.login_id, :oldpassword => @icsa.login_pass, :newpassword => password}

  res = @conn.post do |req|
    req.url 'addrlist/api/modpassword.php'
    req.headers['Content-Type'] = 'application/x-www-form-urlencoded'
    req.headers['Accept'] = '*/*'
    req.body = request_body
  end

  case res.status
  when 200
    begin
      @icsa.update_attributes!(:login_pass => password)
    rescue Exception => e
      raise e
    end
    logger.info("change password finished successfully.")
    return res.status
  when 401
    raise IOError.new("Failed to change password. Authentication failed. (HTTP code: #{res.status}, request_body: #{request_body})")
  when 500
    raise IOError.new("Failed to change password. Internal Server Error. (HTTP code: #{res.status}, request_body: #{request_body})")
  else
    raise IOError.new("Failed to change password. (HTTP code: #{res.status}, request_body: #{request_body})")
  end
end

def new_password
  wdays = ["sun", "mon", "tue", "wed", "thu", "fri", "sat"]
end
```

## ▶ ソフトウェアメンテナンスの効率化 - Copilotデモ -

```
def change_password
  password = new_password
  request_body = {:id => @icsa.login_id, :oldpassword => @icsa.login_pass, :newpassword => password}

  res = @conn.post do |req|
    req.url 'addrlist/api/modpassword.php'
    req.headers['Content-Type'] = 'application/x-www-form-urlencoded'
    req.headers['Accept'] = '*/*'
    req.body = request_body
  end

  case res.status
  when 200
    begin
      @icsa.update_attributes!(:login_pass => password)
    rescue Exception => e
      raise e
    end
    logger.info("change password finished successfully.")
    return res.status
  when 401
    raise IOError.new("Failed to change password. Authentication failed. (HTTP code: #{res.status}, request_body: #{request_body})")
  when 500
    raise IOError.new("Failed to change password. Internal Server Error. (HTTP code: #{res.status}, request_body: #{request_body})")
  else
    raise IOError.new("Failed to change password. (HTTP code: #{res.status}, request_body: #{request_body})")
  end
end

def new_password
  wdays = ["sun", "mon", "tue", "wed", "thu", "fri", "sat"]
end
```

修正箇所(廃止メソッドの更新)

```
def change_password
  password = new_password
  request_body = {:id => @icsa.login_id, :oldpassword => @icsa.login_pass, :newpassword => password}

  res = @conn.post do |req|
    req.url 'addrlist/api/modpassword.php'
    req.headers['Content-Type'] = 'application/x-www-form-urlencoded'
    req.headers['Accept'] = '*/*'
    req.body = request_body
  end

  case res.status
  when 200
    begin
      #@icsa.update_attributes!(:login_pass => password)
      # undefined method `update_attributes'が発生するため、同等処理を実装
    rescue Exception => e
      raise e
    end
    logger.info("change password finished successfully.")
    return res.status
  when 401
    raise IOError.new("Failed to change password. Authentication failed. (HTTP code: #{res.status}, request_body: #{request_body})")
  when 500
    raise IOError.new("Failed to change password. Internal Server Error. (HTTP code: #{res.status}, request_body: #{request_body})")
  else
    raise IOError.new("Failed to change password. (HTTP code: #{res.status}, request_body: #{request_body})")
  end
end

def new_password
```

コメントとしてGitHub Copilotに指示を出す。



```
def change_password
  password = new_password
  request_body = {:id => @icsa.login_id, :oldpassword => @icsa.login_pass, :newpassword => password}

  res = @conn.post do |req|
    req.url 'addrlist/api/modpassword.php'
    req.headers['Content-Type'] = 'application/x-www-form-urlencoded'
    req.headers['Accept'] = '*/*'
    req.body = request_body
  end

  case res.status
  when 200
    begin
      #@icsa.update_attributes!(:login_pass => password)
      # undefined method `update_attributes'が発生するため、同等処理を実装
      @icsa.login_pass = password
    rescue Exception => e
      raise e
    end
    logger.info("change password finished successfully.")
    return res.status
  when 401
    raise IOError.new("Failed to change password. Authentication failed. (HTTP code: #{res.status}, request_body: #{request_body}")
  when 403
    raise IOError.new("Failed to change password. Authentication failed. (HTTP code: #{res.status}, request_body: #{request_body}")
  else
    raise IOError.new("Failed to change password. (HTTP code: #{res.status}, request_body: #{request_body}")
  end
end
```

GitHub Copilotが指示を受けて、最適なコードを提案。

```
def change_password
  password = new_password
  request_body = {:id => @icsa.login_id, :oldpassword => @icsa.login_pass, :newpassword => password}

  res = @conn.post do |req|
    req.url 'addrlist/api/modpassword.php'
    req.headers['Content-Type'] = 'application/x-www-form-urlencoded'
    req.headers['Accept'] = '*/*'
    req.body = request_body
  end

  case res.status
  when 200
    begin
      #@icsa.update_attributes!(:login_pass => password)
      # undefined method `update_attributes'が発生するため、同等処理を実装
      @icsa.login_pass = password
      @icsa.save!
    rescue Exception => e
      raise e
    end
    logger.info("change password finished successfully.")
    return res.status
  when 401
    raise IOError.new("Failed to change password. (HTTP code: #{res.status}, request_body: #{request_body})")
  else
    raise IOError.new("Failed to change password. (HTTP code: #{res.status}, request_body: #{request_body}")
  end
end
```

後は記述するだけ(TABボタンを押すだけ)

- 目的：アプリケーション開発時知識補完効果の検証
- 題材：新人向け学習用アプリケーションの作成
- 被験者：新入社員(配属後3ヶ月)
- 検証方法：GitHub Copilot **Chat**を使用しない場合と使用した場合での作業時間の比較
  
- 検証結果：
  - GitHub Copilot Chatを使用した方が4割程度時間を短縮。
  - 期待通りに回答を得られた割合は6割(3/5)
  
- 考察
  - 手順書レベルで期待通りの回答を得られるケースが多く、大幅な時間短縮に繋がった。
  - 全てのケースで期待通りの回答を得ることはできなかった。相応の情報を入力する必要があり、そこが課題。(プロンプトエンジニアリング以前の問題)

以下の要件でデータベースにあるデータをjson形式でデータを返すAPIをrailsで作りたいと考えています。

1. localhost:3000/csvにgetでリクエストするとDBにあるデータをすべてのデータを返す
2. localhost:3000/csv?name=teaのようにリクエストすると、データベースにあるnameと一致するデータを返す このような仕様をもつAPIサーバを作成するとき、railsではどのようなコマンドを実行すればよろしいですか



GitHub Copilot



以下の手順でAPIサーバを作成することができます。

1. モデルを作成する

```
rails g model Item name:string number:integer
```

2. データベースをマイグレーションする

```
rails db:migrate
```

3. ルーティングを設定する

```
Rails.application.routes.draw do  
  get '/csv', to: 'csv#index'  
end
```

## 4. コントローラを作成する

```
rails g controller Csv index
```

## 5. コントローラにアクションを追加する

```
class CsvController < ApplicationController
  def index
    if params[:name]
      items = Item.where(name: params[:name])
    else
      items = Item.all
    end
    render json: items
  end
end
```

## 6. サーバを起動する

```
rails s
```

これで、`localhost:3000/csv`にGETリクエストを送信すると、データベースにあるすべてのデータがJSON形式で返されます。また、`localhost:3000/csv?name=tea`のようにリクエストすると、`name`が`tea`と一致するデータがJSON形式で返されます。

GitHub Copilotを題材に効果測定を行い、次のような結果を得た。

- 両者ともに作業時間を従来の4割削減できた。  
(ラッキーケースの可能性もあるので、鵜呑みしてはいけない。)
- 期待したコード提案・回答が得られないケースにも遭遇しているため、自身での裏付けや有識者のチェックは必須。

この結果を踏まえて、留意点はあるものの、サービスクリエーション担当においても、生成AIの導入によって「ソフトウェア開発業務の効率の向上」「初級エンジニアのスキルアップ」が期待できる。

※効果測定については上記の通り。 次に、より多くのデータを集めるためトライアルへと移行した。

## ▶▶ 4. トライアル

- 内容：通常業務での生成AI導入効果の確認
- 利用ツール：GitHub Copilot、ChatGPT
- 参加者：のべ14名(GitHub Copilot 13名、ChatGPT 9名)
- 効果測定方法：アンケート(参加者の主観評価)
- アンケートの内容
  - 目的：以下の2点を評価することで、生成AIの導入効果を確認する。
    - 1.業務をどの程度効率化できるか。
    - 2.利用者の行動や心情に有意義な影響を与えるか。
  - 回答数：9名
    - ツール別内訳：GitHub Copilot,ChatGPT両方利用 8名 + ChatGPTのみ利用 1名
    - 年次別内訳：1～3年目：5名 + 4年目以降：4名



アンケート結果から次のことが明らかになった。

- 利用者の**約8割**がGitHub Copilot、ChatGPTに対して好感を持っている。
- GitHub Copilot、ChatGPTを利用することで**1日あたり約68分**の作業時間を削減できている。(ただし、該当業務に従事していた場合に限る)
- GitHub Copilot、ChatGPTを導入したことで、**約7割**の利用者の行動や心情に対して良い変化が出ている。

## ▶ 利用者の評価

GitHub Copilot, ChatGPTそれぞれに対する10段階評価を実施。  
いずれも平均評価が8を超えており、概ね好評を得ている。



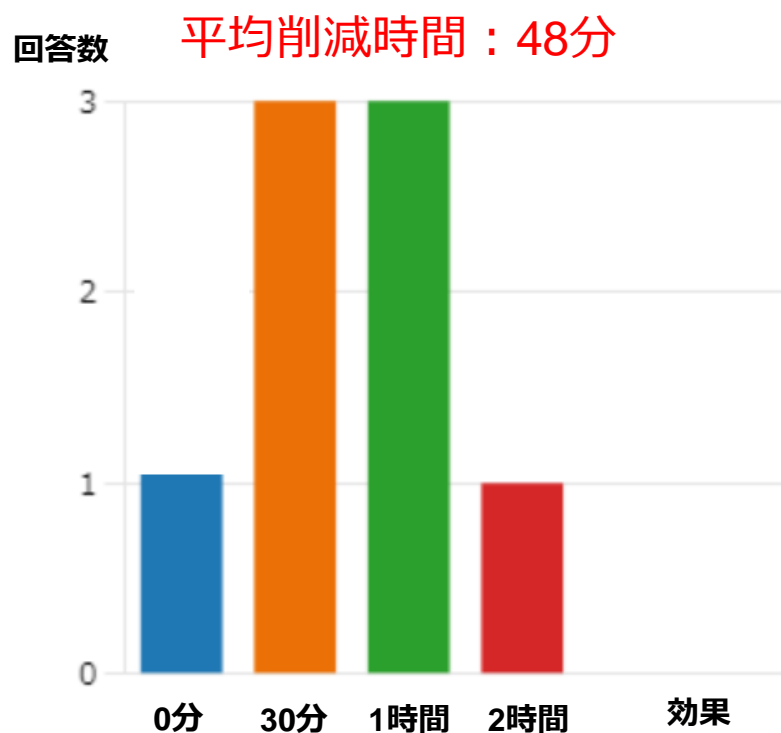
GitHub Copilotに対する10段階評価



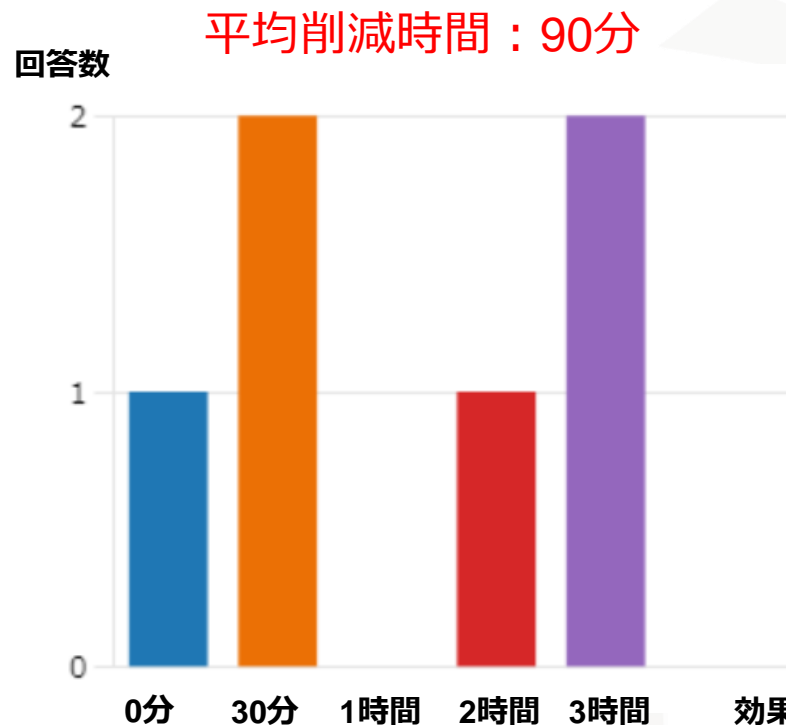
ChatGPTに対する10段階評価

## ▶ 作業効率効果

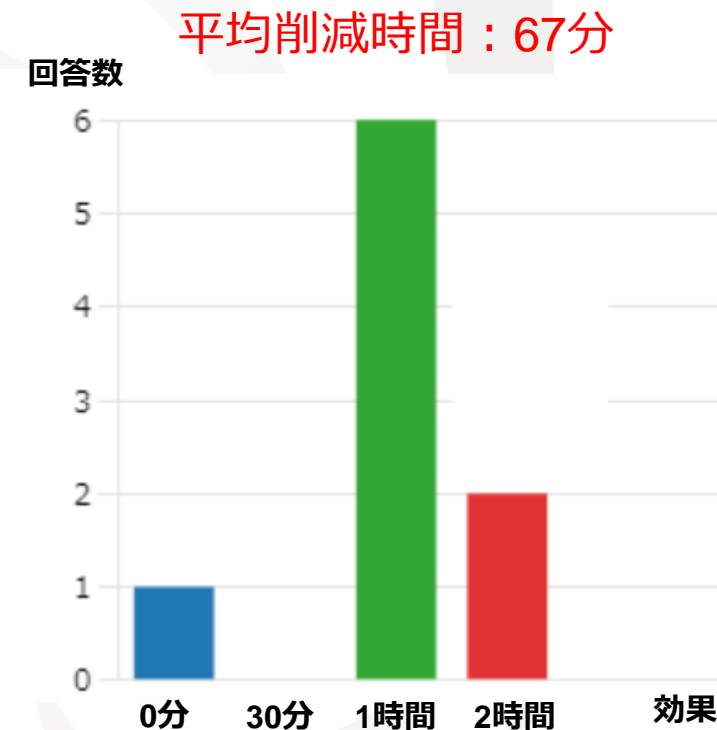
GitHub Copilot, GitHub Copilot Chat, ChatGPTそれぞれの作業時間の削減効果を確認。  
Copilot Chatが一番削減に貢献している。



GitHub Copilotによる  
1日あたりの作業削減効果



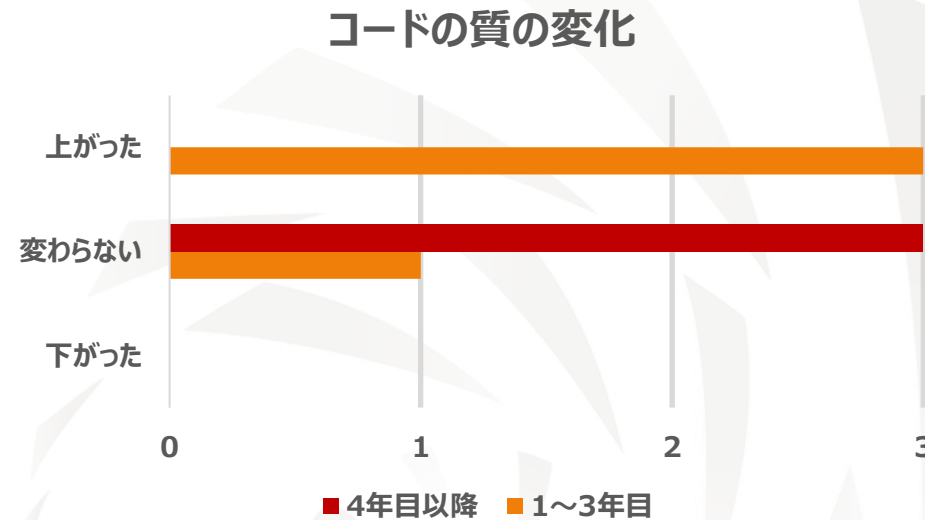
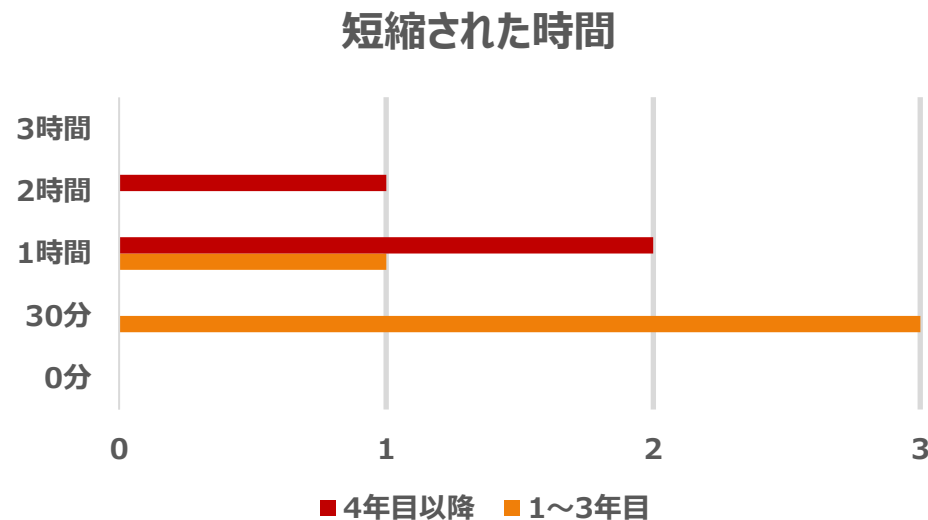
GitHub Copilot Chatによる  
1日あたりの作業削減効果



ChatGPTによる  
1日あたりの作業削減効果

# ▶ 作業効率効果詳細 -GitHub Copilot-

1～3年目と4年目以降にグループ分けし、時間とコードの質に関して結果を比較。

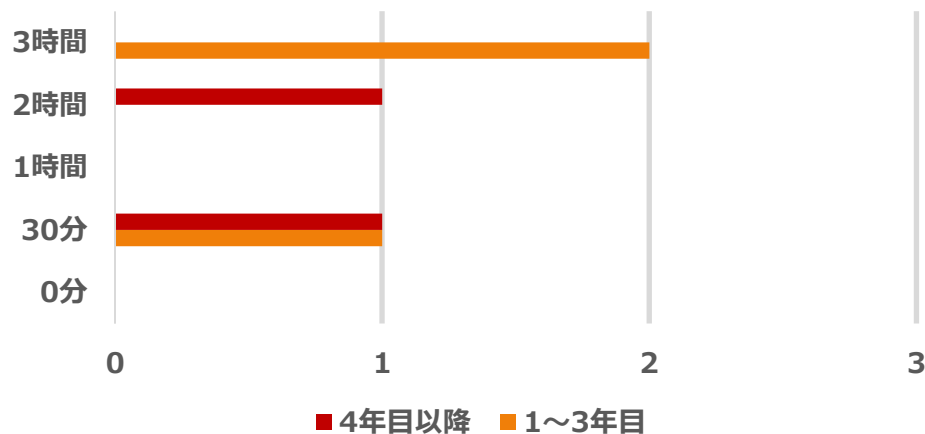


- 時間的な恩恵は4年目以降の方が多く受けている。
- 質的な恩恵は1～3年目の方が多く受けている。
- 考察
  - コーディングの経験値に起因した結果である。
  - ある程度経験を積むと、質を落とすこと無く1日あたり20%程度のスピードアップが期待できそう。

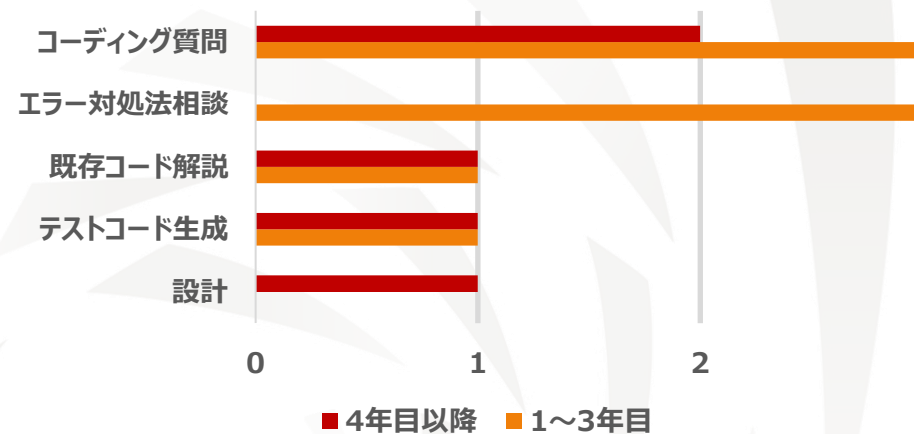
# ▶ 作業効率効果詳細 -GitHub Copilot Chat-

1~3年目と4年目以降にグループ分けし、時間と利用シーンに関して結果を比較。

### 短縮された時間



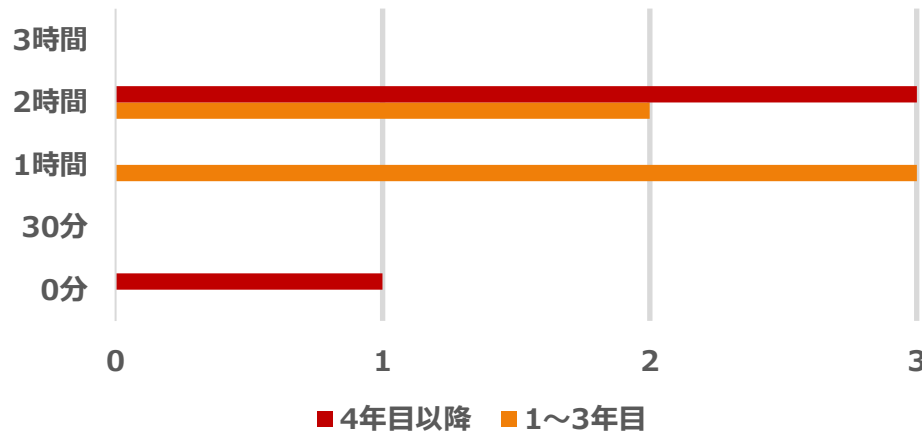
### 利用シーン一覧



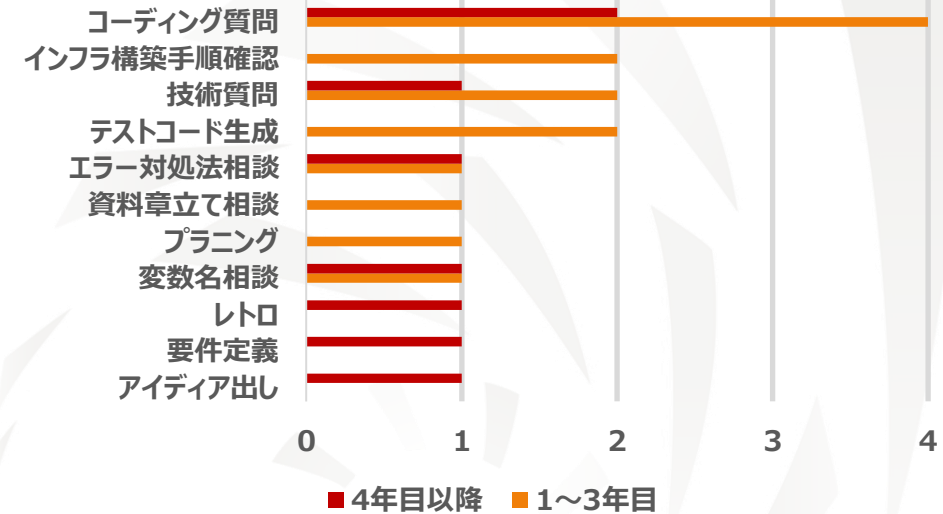
- 時間的な恩恵は1~3年目の方が多く受けている。
- 利用シーンに関しては、1~3年目だけが「エラー対処法相談」に利用している点が大きく違う。
- 考察
  - 1~3年目の方がタスクを完了させるために知識の獲得が必要になる機会が多い。
  - 知識の獲得に関して生成AIが強力な支援ツールになっている。

Copilot Chat同様、1~3年目と4年目以降にグループ分けし、時間と利用シーンに関して結果を比較。

### 短縮された時間



### 利用シーン

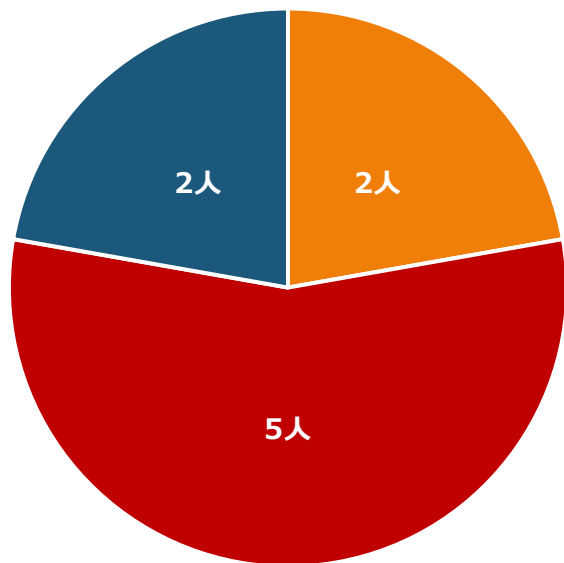


- 1~3年目の方が時間的な恩恵を大きく受けている。
- 利用シーンについては年次区別無く多岐に渡っている。
- 考察
  - 汎用性に関してはChatGPTに軍配が上がる。
  - Copilot Chatの時だけ既存コードの解説で利用するケースがある。そこが強みだと言える。

各種ツール導入後の利用者の行動や心情についても調査を実施。

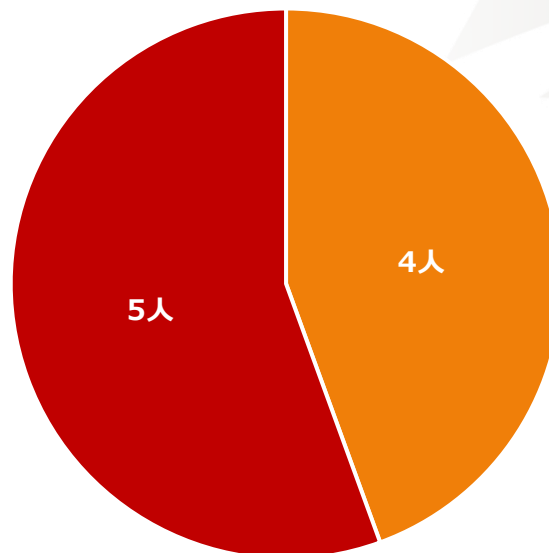
独力で課題解決できる機会・案件に対する貢献度・業務に対するモチベーションアップが確認できた。

### 独力でタスクをこなせる機会の変化



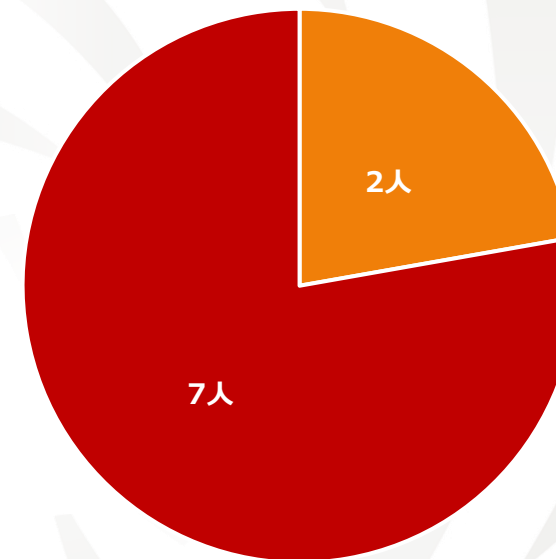
■ 変わらない ■ 増えた ■ かなり増えた

### 案件への貢献度



■ 変わらない ■ 上がった

### 業務に対するモチベーション



■ 変わらない ■ 上がった

各質問に併設されている理由の欄から要因を分析。次のような結果となった。

対象	要因	詳細
独力で課題解決できる 機会の増加	作業効率の向上	<ul style="list-style-type: none"><li>未知の問題に対して素早く提案が得られる。</li><li>効率的な作業方法を提案してもらえる。</li></ul>
案件への貢献度の向上	作業効率の向上	<ul style="list-style-type: none"><li>調査時間が短縮でき、開発スピードがあがったため。</li><li>アウトプットの量が増えたため。</li></ul>
モチベーションアップ	案件への貢献度の向上	<ul style="list-style-type: none"><li>こなせるタスクの量が増えた。</li></ul>
	精神的負荷の低下	<ul style="list-style-type: none"><li>自力でできることが増えたので気が楽になった、楽しくなった。</li><li>新たな取り組みに対するハードルが下がった。</li></ul>

### • 考察

- 生成AIがもたらす作業効率の向上がモチベーションのアップに繋がっている。
- 1～3年目は精神的負荷の低下を理由に挙げているのが特徴的。  
(仕事に真摯に取り組んでくれている証拠。伸びしろあり。)



- 作業効率について
  - 効果測定ほどではないが、通常業務においても実装・知識獲得のスピードアップが見込める。
  - 実装に関しては熟練であるほど、知識獲得に関しては経験が浅いほど大きな恩恵を受けられる。
  - ハルミネーションや利用者の思考停止の懸念もあるが、自身での裏取りや有識者チェックを前提とすることで回避可能。(そうすることで学習効果も期待できる)
- 行動・心情への影響について
  - 生成AIにはエンジニアの行動・心情にも良い影響を及ぼす。
  - 一種の開発者体験の向上。
  - 生成AIはエンジニアのエンゲージメントを向上させる可能性を秘めているかもしれない。(飛躍しすぎ?)

## ▶ 5. まとめ

今回の取組によって、生成AIの導入はサービスクリエーション担当が抱えている課題解決の一助となることを確認できた。

具体的に得られた知見は以下の通り。

- コーディング・テストの工程で作業効率を向上させることができる。
- 不慣れもしくは知識が不足している分野での作業であっても、独力での解決の可能性を高めることができる。
- 上記2つの効果によって、エンジニアの精神的負荷を低減し、モチベーションを高めることができる。
  
- GitHub Copilotが生成するコードは4年目以降のエンジニアと同程度である。
- GitHub Copilot ChatとChatGPTは共存できる。
- ハルミネーションは必ず発生するので、自身での裏付けや有識者によるチェックは必須である。

現状のテーマに対して取り組みを継続し、組織としてのソフトウェア開発力を底上げしていきたい。  
具体的な取り組み内容については、次の通りを予定。

- ソフトウェア開発効率の向上
  - 利用範囲の拡大  
要件定義や設計などにも拡大させる。
  - RAGの活用  
特定業務に関する知識を学習させて、生成AIの支援が受けられる範囲を拡大させる。
- 初級エンジニアのスキルアップ
  - 自己学習での利用促進  
積極的に生成AIを活用して自身で課題を解決させる。  
ただし、生成AIから得られた情報の裏付けや検証を必須にする。
  - 支援方法の更改  
現状よりも高度・応用的な問題を共同で解決していく形に変えていく。  
一緒に質を高めていく支援体制を構築する。  
※生成AIの活用というよりは、人の問題



ご清聴ありがとうございました。

GitHubは、GitHub Inc.の商標または登録商標です。  
Copilotは、米国Microsoft Corporation.の米国およびその他の国における登録商標です。  
GPTは、OpenAI Global, LLCの商標または登録商標です。