~ 2024 ~

Exaptation for the future.

EXA CORPORATION

新RDM構築に生成AIを適用してみた

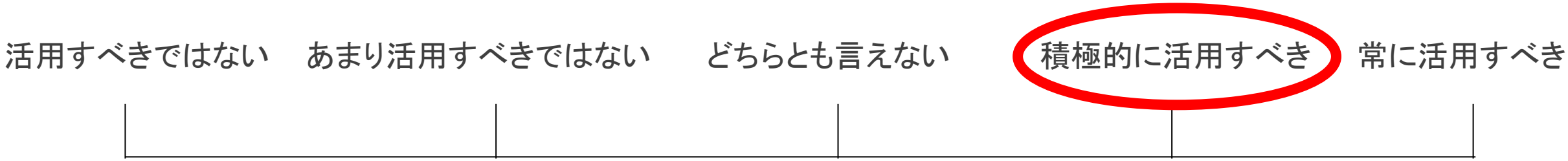
2024/10/24 @EVF2024

デジタルプラットフォームサービスセンター　潮田雄揮

目次

生成AIを業務に活用すべきか？

生成AIを業務に活用すべきか？



気を付けるべき点はあるが、非常に強力なツール

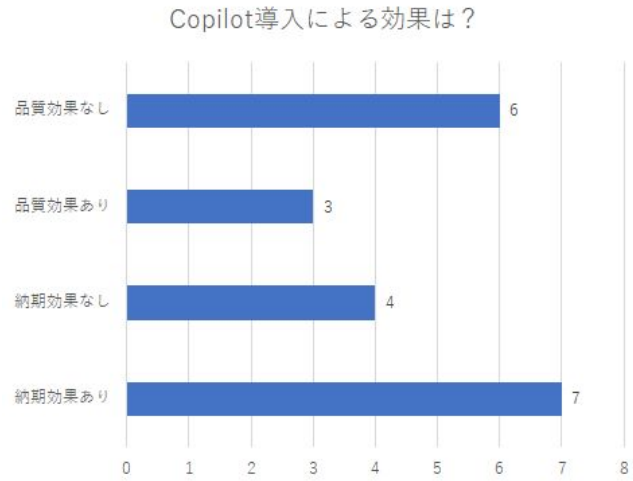
先行研究:コード生成ツールの導入判断のための評価方法の提案

先行研究:コード生成ツールの導入判断のための評価方法の提案

2023年6月に、弊社のSE12人を対象に生成AI(GitHub Copilot)を使用したソフトウェア開発の効率の変動を検証を行った
(デジタルプラットフォームサービスセンター　堀さんにより実施※1)

- 客観評価
  - GitHub Copilotを使用したグループの方が、品質・スケジュールは向上したが、コストはわずかに悪化した。なお、3種類の指標のうち、スケジュール指標がもっとも向上効果が大きかった。
- ユーザ主観評価
  - スケジュール効率については肯定的な回答が多かった反面、品質効果は否定的な回答が多かった。

品質指標	コスト指標	スケジュール指標	
GitHub Copilotなし	67%	57%	11%
GitHub Copilotあり	68%	56%	14%
あり-なし	+1.2%	-0.7%	+3.7%



先行研究:コード生成ツールの導入判断のための評価方法の提案

検証対象者をSEランク別に分類すると、Cランク以外のSEでは品質、コスト、スケジュールのすべての指標で、GitHub Copilot を使用した方が効果があることが判明した。

- CランクのSEは時間単価が最も低い開発者グループであり、技術的知識や保有スキルについても、他のランクと比較して低いと考えられる。
- GitHub Copilot による検索候補を活用するためには、一定のスキルや経験が必要なため、ツールコスト(GitHub Copilot稼働費用等)を超える効果を得ることができなかったと考えられる。

先行研究:コード生成ツールの導入判断のための評価方法の提案

ソフトウェア開発において、生成AIを活用するには相応の技術知識、スキルがないと
十分な(コストに見合った)成果が出ない

設計フェーズは？

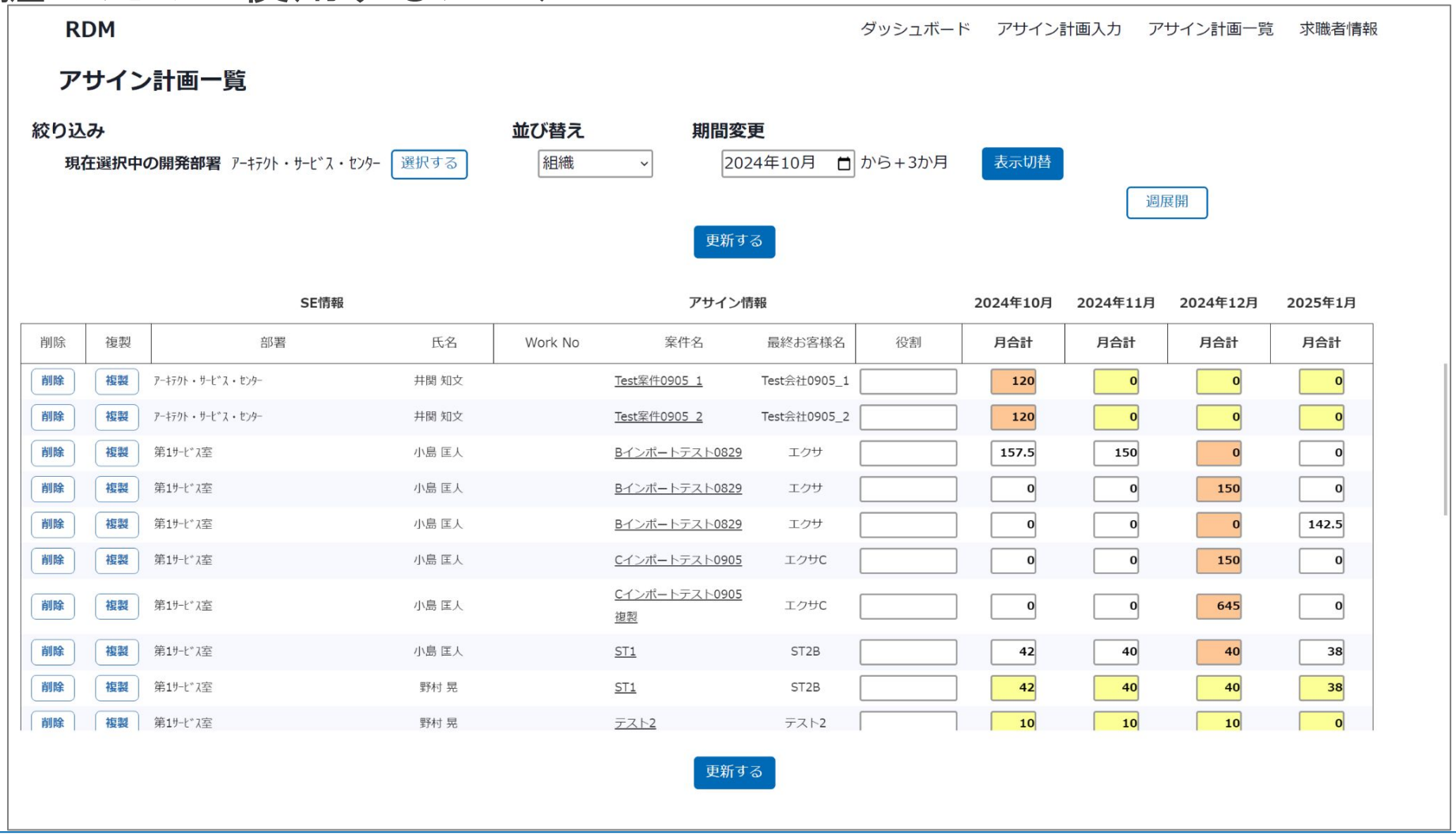
インフラ構築(IaC)は？

どれくらいの知識があればいい？

生成AI(Azure OpenAI ChatGPT4)を使用したシステム構築

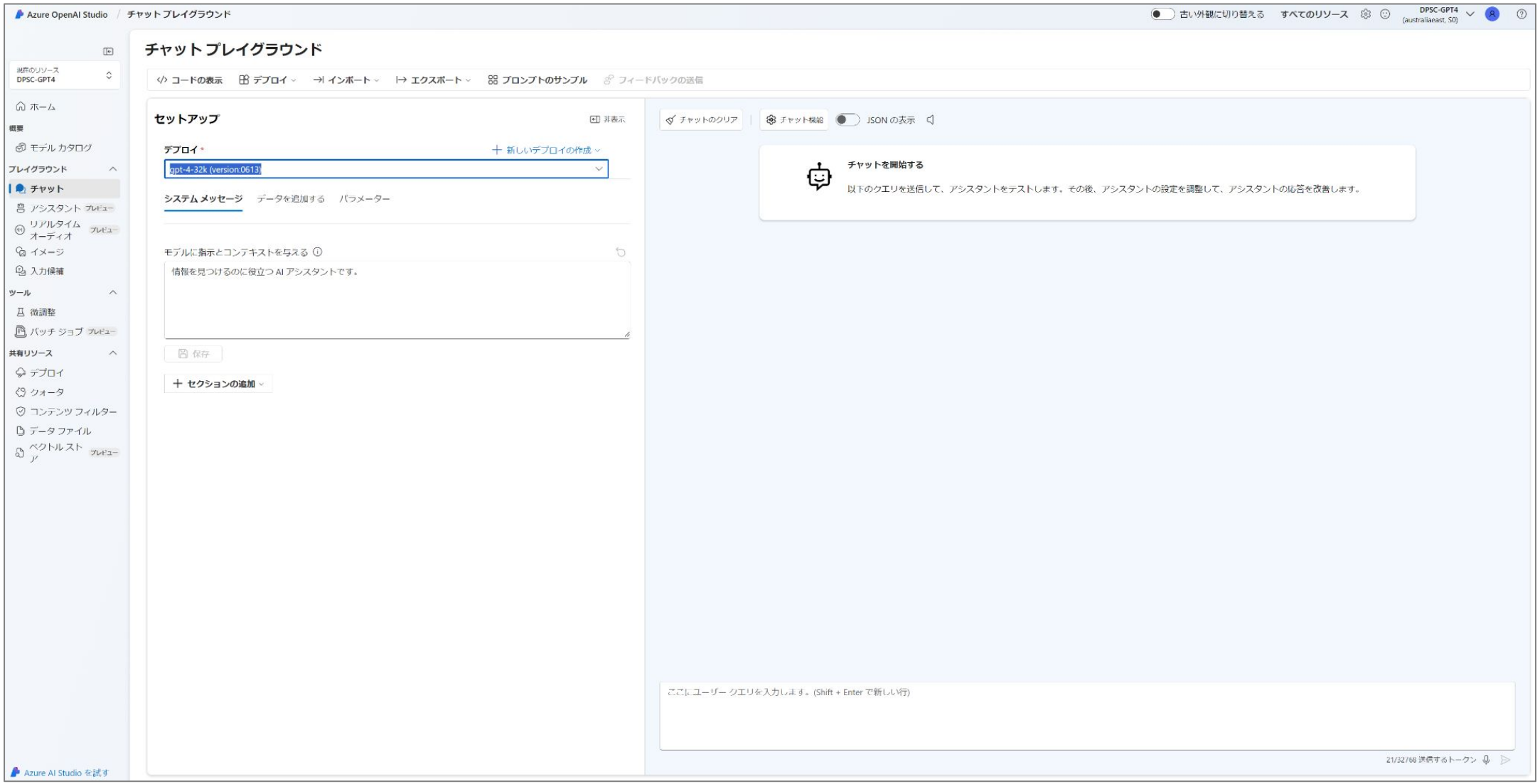
生成AI(Azure OpenAI ChatGPT4)を使用したシステム構築

新RDMシステム：ライン職が配下のアサイン状況の管理および配下外の人員リソース状況の把握のために使用するシステム



生成AI(Azure OpenAI ChatGPT4)を使用したシステム構築

Azure Open AI Studioでgpt-4-32kを使用して開発を行った



設計フェーズにおける生成AI活用事例

設計フェーズにおける生成AI活用事例

業務説明書から概念データモデル(エンティティと属性)を抽出

#命令書:
あなたは優秀なDBスペシャリストです。
以下の#制約条件と#出力形式にしたがって、#業務説明 からリソース管理システムの概念データモデルのエンティティと属性を洗い出してください。
#制約条件:
・#業務説明 から名詞を抽出する
・名詞のなかから、エンティティ、エンティティに含まれる属性を洗い出す
・各エンティティの属性から、主キーを特定する
・外部キーが存在する場合も特定する
#業務説明:
リソース管理システムの要件
1. プロジェクトの目的
　1. 開発部門の要員（リソース）の情報を一元管理するシステムを構築する。要員（リソース）のアサイン情報管理だけに留まらず、全社的なリソースの求人、求職のマッチングに活用する。また、アサインされた案件を業務経歴に保存するなど、社員のスキルの見える化に活用し、社員がよりスキルを発揮できる環境を整備する。
　2. EXA Digital Platformの第一号案件として、新しいシステム開発の手法やツール等を適用し、その経験を他の案件で広く適用できるように、アセットとして整備する。
2. フェーズ1で実現したい機能
　1. 開発部門の要員（リソース）をどの案件にアサインするかを管理する、アサイン計画の機能
　2. まだ案件へのアサインが決まっていない要員（リソース）の情報を提供する、求職情報の機能
　3. 各開発部門のアサイン計画の情報から、リソースの稼働率や損益（O/U）を計算し、表示するダッシュボードの機能
3. フェーズ1で管理するデータ
　1. 社員の属性情報（職番、氏名、組織コード、組織名、直接/間接、職種、ランク、休職中フラグ）
　2. アサイン情報の管理期間（現在を含め4か月、週毎の入力された時間を管理）
　3. アサイン情報の種類（有償、生産的作業、無償）
　4. 案件の情報（最終顧客名、案件名、Work__No、Role、開始日、終了日、各月の作業時間）
　5. 管理情報（SE Rate、稼働率計画、O/U計画、超過労働閾値、稼働日カレンダー）
　6. 顧客情報（正式名称、略称（複数あり得る））
4. フェーズ1で構築するシステムの利用者
　1. 開発部門のラインマネージャー（本部長、部長、室長）
　2. 開発本部のリソース管理担当者
　3. 全社リソース管理担当者
5. フェーズ1での外部システムとの連携
　1. 社員のアサイン計画をGoogle Workspaceのカレンダーに登録する
　2. ENERGYシステムから、アサイン予定の案件の情報を取得する
6. フェーズ1で提供する画面
　1. アサイン情報入力、更新画面

#エンティティと属性

以下に業務説明から洗い出したエンティティとそれぞれの属性を示します。

##エンティティ: 社員
- 属性: 職番 (主キー), 氏名, 組織コード, 組織名, 直接/間接, 職種, ランク, 休職中フラグ

##エンティティ: 組織
- 属性: 組織コード (主キー), 組織名

##エンティティ: アサイン情報
- 属性: 管理期間, 種類 (有償、生産的作業、無償), 社員の職番 (外部キー)

##エンティティ: 案件
- 属性: 最終顧客名, 案件名, Work_No (主キー), Role, 開始日, 終了日, 各月の作業時間, 社員の職番 (外部キー)

##エンティティ: 管理情報
- 属性: SE Rate, 稼働率計画, O/U計画, 超過労働閾値, 稼働日カレンダー

##エンティティ: 顧客
- 属性: 正式名称 (主キー), 略称

##エンティティ: アサイン計画
- 属性: アサイン先案件のWork_No (外部キー), 社員の職番 (外部キー), 管理期間, 種類, 作業時間

##エンティティ: 求職情報
- 属性: 社員の職番 (主キー), アサイン待ち状態

##エンティティ: ダッシュボード
- 属性: 表示対象期間, 表示対象組織の組織コード (外部キー)

##エンティティ: ラインマネージャー
- 属性: 社員の職番 (主キー), 担当組織の組織コード (外部キー)

注意: 主キーは各エンティティを一意に識別するための属性であり、外部キーは他のエンティティへの参照を表す属性です。

設計フェーズにおける生成AI活用事例

関係スキーマを作成して生成AIで修正

#命令書:
以下の #関係スキーマ は、抽出されたエンティティと属性をもとに関係スキーマ形式で記述し直したものです。#制約条件 と #出力形式 に従って、関係スキーマを修正し出力してください。
#制約条件:
・「組織」の属性を「組織負担コード」、「組織名」、「親組織負担コード」、「組織の種類」のみとする
・「組織」の主キーを「組織負担コード」にし、ほかのエンティティの属性にある「組織コード」を「組織負担コード」という名称に変更する
・「社員」から「組織名」をはずす
・「アサイン情報」を「アサイン実績」に変更
・「アサイン情報」および「アサイン計画」の主キーは「アサインID」のみとする
・「アサイン情報」および「アサイン計画」の属性「職番」を外部キーにする
・「WORK_NO」というエンティティを新設し、主キーを「WORK_NO ID」、属性に「WORK_NO」と「部門管理コード」を追加する
・エンティティのなかで属性として「WORK_NO」をもつものは「WORK_NO ID」に変更する
#関係スキーマ:
- 社員(**職番** , 氏名, 組織コード, 組織名, 直接/間接, 職種, ランク, 休職中フラグ)
- 組織 (**組織コード** , 組織名)
- アサイン情報(管理期間, 種類 (有償、生産的作業、無償), *職番*)
- 案件(最終顧客名, 案件名, **Work_No** , Role, 開始日, 終了日, 各月の作業時間, *職番*)
- 管理情報(SE Rate, 稼働率計画, O/U計画, 超過労働閾値, 稼働日カレンダー)
- 顧客(**正式名称** , 略称)
- アサイン計画(*Work_No*, *職番* , 管理期間, 種類, 作業時間)
- 求職情報(**職番** , アサイン待ち状態)
- ダッシュボード(表示対象期間, *組織コード*)
- ラインマネージャー(**職番** , *組織コード*)
> **太字** : 主キー、 *斜体* : 外部キー
#出力形式:
関係スキーマ形式でMarkdown表記してください。
・先頭はエンティティ名、カッコ内に属性群を記述する
・主キーは **太字** 、外部キーは *斜体* 表記にする

#修正後の関係スキーマ:

- 組織(**組織負担コード** , 組織名, 親組織負担コード, 組織の種類)
- 社員(**職番** , 氏名, 組織負担コード, 直接/間接, 職種, ランク, 休職中フラグ)
- アサイン実績(**アサインID** , 管理期間, 種類 (有償、生産的作業、無償), 職番)
- アサイン計画(**アサインID** , *WORK_NO ID*, 職番 , 管理期間, 種類, 作業時間)
- 案件(**案件ID** , 最終顧客名, 案件名, 役割*ID*, 開始日, 終了日, 各月の作業時間, 職番, *WORK_NO ID*)
- 管理情報(SE Rate, 稼働率計画, O/U計画, 超過労働閾値, 稼働日カレンダー)
- 顧客(**顧客ID** , 正式名称, 略称)
- 求職情報(**職番** , アサイン待ち状態)
- ダッシュボード(表示対象期間, 組織負担コード)
- ラインマネージャー(**職番** , 組織負担コード)
- WORK_NO(**WORK_NO ID** , WORK_NO, 部門管理コード)
- 役割(**役割ID** , 役割名)
- 有償(**アサインID**)
- 生産的作業(**アサインID**)
- 無償(**アサインID**)
- 教育(**アサインID**)
- 休暇(**アサインID**)
- プロジェクト直課(**アサインID** , 顧客*ID*, *WORK_NO ID*, 役割*ID*)
- プロジェクト付替(**アサインID** , 顧客*ID*, *WORK_NO ID*, 役割*ID*)
- 共通原価（保守）(**アサインID**)
- 投資(**アサインID**)
- 情報システム部支援(**アサインID**)
- 営業サポート(**アサインID**)

太字 : 主キー、 *斜体* : 外部キー

設計フェーズにおける生成AI活用事例

関係スキーマから関連を抽出

#命令書:
あなたは優秀なDBスペシャリストです。
以下の#制約条件と#出力形式にしたがって、#関係スキーマ 間の関連をすべて洗い出してください。
#制約条件:
・#業務説明 から動詞を抽出して、関連を洗い出し関連名をつける
・#関係スキーマ に定義されているエンティティ間の関連を洗い出し、連結元エンティティ名、連結先エンティティ名を特定する
・#業務説明 から関連の多重度が判明するものは表記する（不明なものは表記しなくて良い）
#業務説明:
リソース管理システムの要件
1. プロジェクトの目的
　1. 開発部門の要員（リソース）の情報を一元管理するシステムを構築する。要員（リソース）のアサイン情報管理だけに留まらず、全社的なリソースの求人、求職のマッチングに活用する。また、アサインされた案件を業務経歴に保存するなど、社員のスキルの見える化に活用し、社員がよりスキルを発揮できる環境を整備する。
　2. EXA Digital Platformの第一号案件として、新しいシステム開発の手法やツール等を適用し、その経験を他の案件で広く適用できるように、アセットとして整備する。
2. フェーズ1で実現したい機能
　1. 開発部門の要員（リソース）をどの案件にアサインするかを管理する、アサイン計画の機能
　2. まだ案件へのアサインが決まっていない要員（リソース）の情報を提供する、求職情報の機能
　3. 各開発部門のアサイン計画の情報から、リソースの稼働率や損益（O/U）を計算し、表示するダッシュボードの機能
3. フェーズ１で管理するデータ
　1. 社員の属性情報（職番、氏名、組織コード、組織名、直接/間接、職種、ランク、休職中フラグ）
　2. アサイン情報の管理期間（現在を含め4か月、週毎の入力された時間を管理）
　3. アサイン情報の種類（有償、生産的作業、無償）
　4. 案件の情報（最終顧客名、案件名、Work_No、Role、開始日、終了日、各月の作業時間）
　5. 管理情報（SE Rate、稼働率計画、O/U計画、超過労働閾値、稼働日カレンダー）
　6. 顧客情報（正式名称、略称（複数あり得る））
4. フェーズ１で構築するシステムの利用者
　1. 開発部門のラインマネージャー（本部長、部長、室長）
　2. 開発本部のリソース管理担当者
　3. 全社リソース管理担当者
5. フェーズ１での外部システムとの連携
　1. 社員のアサイン計画をGoogle Workspaceのカレンダーに登録する
　2. ENERGYシステムから、アサイン予定の案件の情報を取得する
6. フェーズ１で提供する画面
　1. アサイン情報入力、更新画面
　2. 求職情報管理画面
　3. リソースの稼働率や損益（O/U）を表示する画面

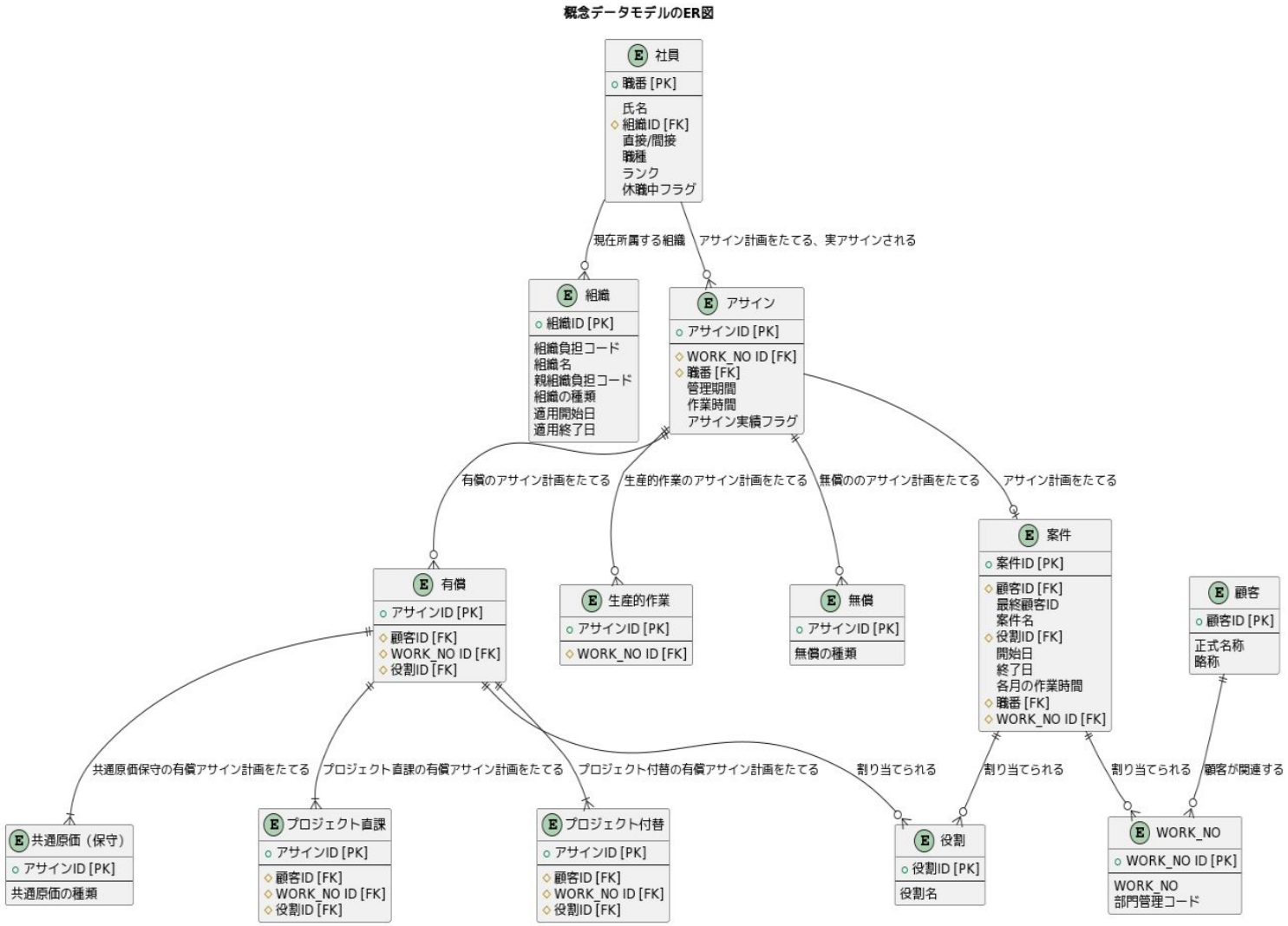
以下に#業務説明 と #関係スキーマ から洗い出した関連のリストを記述します。

関連名	連結元エンティティ名	連結先エンティティ名	多重度
組織に所属する	社員	組織	多対1
アサインされる	社員	アサイン	1対多
案件がアサインされる	アサイン	案件	1対1
顧客が関連する	有償	顧客	1対1
役割が割り当てられる	有償	役割	1対1
タスクが割り当てられる	有償	プロジェクト直課	1対1
タスクが割り当てられる	有償	プロジェクト付替	1対1
タスクが割り当てられる	有償	共通原価（保守）	1対1
顧客が関連する	案件	顧客	1対1
役割が割り当てられる	案件	役割	1対1
社員が関連する	案件	社員	1対1
WORK_NOが関連する	案件	WORK_NO	1対1
顧客が関連する	顧客	WORK_NO	1対1
アサインが派生する	アサイン	有償	1対1 継承
アサインが派生する	アサイン	生産的作業	1対1 継承
アサインが派生する	アサイン	無償	1対1 継承
有償が派生する	有償	プロジェクト直課	1対1 継承
有償が派生する	有償	プロジェクト付替	1対1 継承
有償が派生する	有償	共通原価（保守）	1対1 継承

設計フェーズにおける生成AI活用事例

関係スキーマと関連からER図(PlantUML)を作成

#命令書:
あなたは優秀なDBスペシャリストです。
以下の#関係スキーマと#関連一覧をもとに#出力形式にしたがって概念データモデルのER図を出力してください。
#関係スキーマ:
1. 組織(**組織ID**, 組織負担コード, 組織名, 親組織負担コード, 組織の種類, 適用開始日, 適用終了日)
   - 組織の種類：本部・部門、部、室
2. 社員(**職番** , 氏名, *組織ID*, 直接/間接, 職種, ランク, 休職中フラグ)
   - 職種：SE、ラインマネージャー、スタッフ、役員
3. アサイン(**アサインID** , *WORK_NO ID*, *職番* , 管理期間, 作業時間, アサイン実績フラグ)
   - アサイン実績フラグ：真→アサイン実績、偽→アサイン計画
4. 有償(**アサインID** , *顧客ID*, *WORK_NO ID*, *役割ID*)
   - アサインのサブエンティティ
5. 生産的作業(**アサインID** , *WORK_NO ID*)
   - アサインのサブエンティティ
6. 無償(**アサインID** , 無償の種類)
   - アサインのサブエンティティ
   - 無償の種類：教育、休暇
7. プロジェクト直課(**アサインID** , *顧客ID*, *WORK_NO ID*, *役割ID*)
   - 有償のサブエンティティ
8. プロジェクト付替(**アサインID** , *顧客ID*, *WORK_NO ID*, *役割ID*)
   - 有償のサブエンティティ
9. 共通原価（保守）(**アサインID**, 共通原価の種類)
   - 有償のサブエンティティ
   - 共通原価の種類：投資、情報システム支援、営業サポート
10. 案件(**案件ID** , *顧客ID*, 最終顧客ID, 案件名, *役割ID*, 開始日, 終了日, 各月の作業時間, *職番*, *WORK_NO ID*)
11. 顧客(**顧客ID** , 正式名称, 略称)
12. WORK_NO(**WORK_NO ID** , WORK_NO, 部門管理コード)
13. 役割(**役割ID** , 役割名)
14. 管理情報(SE Rate, 稼働率計画, O/U計画, 超過労働閾値, 稼働日カレンダー)
   - 社員ごとの稼働率をあらわす別カテゴリに属するエンティティ
15. 求職情報(**職番** , アサイン待ち状態)
   - 非稼働社員を一覧表示するための別カテゴリに属するエンティティ
16. ダッシュボード(表示対象期間, *組織ID*)
   - 1から13までのエンティティの表示対象期間ごとの集計情報を表示させるための別カテゴリに属するエンティティ
> **太字**：主キー、 *斜体*：外部キー
#関連一覧:
関連名	連結元エンティティ名	連結先エンティティ名	多重度



設計フェーズにおける生成AI活用事例

- 業務説明書から概念データモデルを直接作成することは難しい
- エンティティと属性、関係スキーマ、関連のようにタスクを分割して生成すると比較的良い成果物ができる
- 概形を作るだけなら圧倒的に高速化できる
- どれだけ細かくステップを刻んでも、間違った回答は出る(7、8割は正解しているため間違いを見つけることも割と手間がかかる)
- 根気強く生成AIの回答を添削する必要がある
- ER図などそれ以降の工程に対して手戻りのインパクトが大きいものはより慎重に人の手で精査する必要がある
- 根本的に間違ってしまう場合は手動で作成する方が早い
- 概形を前もってイメージできない場合は参考資料として優秀

うまく扱うためには かなりの技術知識や経験が必要

インフラ構築における生成AI活用事例

インフラ構築における生成AI活用事例

概要からCloudFormationテンプレートを生成

#命令書:
あなたは優秀なAWSスペシャリストです。
すでにAurora(Postgres)が作成済みの環境に対しGlueジョブを実行するための設定として以下のセキュリティグループの追加・変更を行うCloudFormation定義ファイルを作成してください。
・Glueジョブ実行用セキュリティグループを新規作成する
自分自身のセキュリティグループに対し0~65535/tcpを許可
・すでにAuroroaに紐づいているセキュリティグループに以下のルールを追加する
5432/tcp ：Glueジョブ実行用セキュリティグループ(先に作成したもの)に対し5432/tcpを許可
#出力方法:
・YAML形式とする
・すでに作成済みのAuroraに紐づくセキュリティグループをパラメータとして定義できるようにする

```
AWSTemplateFormatVersion: '2010-09-09'
    Description: 'AWS CloudFormation for AWS Glue Job and Aurora
PostgreSQL'
Parameters:
    ExistingAuroraSecurityGroup:
    Description: 'The ID of the existing Aurora security group'
    Type: 'AWS::EC2::SecurityGroup::Id'
Resources:
    GlueJobSecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
        GroupDescription: 'Security group for AWS Glue Job'
        SecurityGroupIngress:
```

...省略...

インフラ構築における生成AI活用事例

エラーメッセージからCfnテンプレートの修正箇所を検出

エラーメッセージ

BatchServiceRole: Type: 'AWS::IAM::Role' Properties: RoleName: !Sub "${NamePrefix}-batch-service-role" AssumeRolePolicyDocument: Version: '2012-10-17' Statement: - Effect: Allow Principal: Service: - batch.amazonaws.com Action: - sts:AssumeRole Path: '/' CLIENT_ERROR - User: arn:aws:sts::004133839690:assumed-role/rdm-backend-dev-batch-service-role/aws-batch is not authorized to perform: ecs:ListClusters on resource: * because no identity-based policy allows the ecs:ListClusters action (Service: AmazonECS; Status Code: 400; Error Code: AccessDeniedException; Request ID: 5077539b-9a07-46e8-b49e-051baeae337f; Proxy: null)

エラーメッセージから、AWS BatchがECSのListClusters操作を実行する権限がないことがわかります。これは、BatchServiceRoleに必要なポリシーが付与されていないために発生しています。
BatchServiceRoleに以下のようなポリシーを追加すると、この問題が解決するはずです：

```
BatchServiceRolePolicy:
      Type: 'AWS::IAM::Policy'
      Properties:
      PolicyName: BatchServiceRolePolicy
      Roles:
            - !Ref BatchServiceRole
      PolicyDocument:
            Version: '2012-10-17'
      Statement:
            - Effect: Allow
      Action:
      - ecs:ListClusters
      Resource: "*"
```

このポリシーは、BatchServiceRoleにECSのListClusters操作を許可するものです。これにより、AWS BatchがECSクラスタをリストアップするための権限を得ることができます。
ただし、このポリシーは全てのリソースに対してListClusters操作を許可するものなので、セキュリティ上の懸念がある場合は、Resourceを具体的なリソースARNに置き換えるか、または適切な条件を追加してください。

インフラ構築における生成AI活用事例

クラウドベンダー特有の操作(EBSボリューム追加時)の操作ガイドを生成

EBSのボリュームをAWSコンソール上から増やしたが、OSに認識されない

EBSボリュームをリサイズした後、その変更をオペレーティングシステムに認識させるためには、ファイルシステムを拡張する必要があります。このプロセスは、ボリュームの新しい空間を利用できるようにするために必要です。この問題を解決するには、拡張されたEBSボリュームのパーティションテーブルを更新する必要があります。
以下に、/dev/nvme0n1（この例ではrootボリューム）のパーティションテーブルを更新する手順を示します。
1.まず、lsblkコマンドを使用してパーティションのレイアウトを確認します：

```
lsblk
```

2.次に、growpartコマンドを使用してパーティションテーブルを更新します。ここでは、1はパーティション番号を示します：

```
sudo growpart /dev/nvme0n1 1
```

3.最後に、ファイルシステムの拡張を行います。Ext4の場合、以下のコマンドを使用します：

```
sudo resize2fs /dev/nvme0n1p1
```

インフラ構築における生成AI活用事例

- 既存リソースが無くてもCfnテンプレートを生成してくれるため、yamlを作成する際にヒューマンエラーが無くて済む
- 大まかな指示に沿ってひな型を作成してくれるため詳細部分に労力を割くことができる
- 間違った回答を返すことはある(7、8割程度)
- CloudFormationでデプロイの試行を繰り返すことが可能であれば最終形を作り出すまでの時間を短縮できる
- OSレベル、ミドルレベルの設定に関しても回答してくれる

基本的な知識があり、やりたいことを意識できていれば 実現手段を素早く提示 してくれる

フロントエンド開発における生成AI活用事例

フロントエンド開発における生成AI活用事例

自然言語で指示をすることでライブラリを使用しつつ実装例を提示

モーダルが表示されているときに背後が暗くならないようにはできますか。

はい、それは可能です。Chakra UI の Modal コンポーネントは、ModalOverlay コンポーネントを使用して背後を暗くしています。この ModalOverlay の背景色を透明に設定することで、モーダルが表示されているときに背後が暗くなるのを防ぐことができます。
以下にその変更を反映したコードを示します。

```
export const RegistSuccessModal:FC<Props> = memo((props) => {
const { isOpen, onClose, onContinue, onJumpAssignPlanList } = props;
…中略...
return (
<Modal isOpen={isOpen} onClose={onClose} autoFocus={false} size="5xl">
      <ModalOverlay bg="transparent" />
      <ModalContent pb={6} bg="#CCE1F1">
      <ModalCloseButton />
      <ModalBody mx={4}>
      …中略...
</Modal>
)
})
```

このコードでは、ModalOverlay コンポーネントの bg プロパティに "transparent" を指定しています。これにより、モーダルが表示されているときに背後が暗くならなくなります。

フロントエンド開発における生成AI活用事例

(ReactなどのModelとViewの概形を並べるだけでViewへの値埋め込みを実装

提示したModelを表示するViewを作成してください

あなたが望むようにアサインされた月の情報が正しく表示されるように、コードを変更するためには、AssignPlanDetailコンポーネント内のdata.assigned_monthsに対するマッピングを調整する必要があります。
具体的には、ヘッダーと同じ順序で月のアサイン情報を表示するために、data.assigned_monthsのマッピングを、data.months_headerに基づいて行うと良いでしょう。
以下にその方法を示します:

```
{data.months_header.map((headerMonth, index) => {
const monthData = data.assigned_months.find((month) => month.month === headerMonth);
if (monthData) {
        return (
                <React.Fragment key={index}>
                        <td><input type='number' value={monthData.weekly_assigned_hours.week1} style={{ width: "40px" }}/></td>
                        <td><input type='number' value={monthData.weekly_assigned_hours.week2} style={{ width: "40px" }}/></td>
                        <td><input type='number' value={monthData.weekly_assigned_hours.week3} style={{ width: "40px" }}/></td>
                        <td><input type='number' value={monthData.weekly_assigned_hours.week4} style={{ width: "40px" }}/></td>
                        <td><input type='number' value={monthData.weekly_assigned_hours.week5} style={{ width: "40px" }}/></td>
                        <td><input type='number' value={monthData.monthly_total} style={{ width: "64px" }}/></td>
                </React.Fragment>
                );
        } else {
                …中略...
        }
})}
```

このコードは、months_headerで定義された順番に従って各月の情報を表示します。その月の情報がassigned_monthsに存在しない場合は、空の入力フィールドを表示します。

フロントエンド開発における生成AI活用事例

テストコード概形の実装

この関数(省略)をMCCかつ外れ値テスト、境界値テストを実施するテストコードを書いて

了解しました。以下にreduceOldestMonth関数のテストコードを示します。このテストでは、正常なデータ、境界値、および外れ値を使用してテストを行います。

```
import { DateStringUtil } from '../DateStringUtil';
import { AssignPlanDetailData } from '../types/AssignPlanDetailData';
describe('DateStringUtil', () => {
describe('reduceOldestMonth', () => {
        // Normal test case
        it('should return the oldest month', () => {
                const data: AssignPlanDetailData[] = [
                        {
                                // omit other properties for brevity
                                assigned_months: [{ month: '202年12月' }, { month: '203年01月' }]
                        },
                        {
                                // omit other properties for brevity
                                assigned_months: [{ month: '202年11月' }, { month: '202年01月' }]
                        }
                ];
                expect(DateStringUtil.reduceOldestMonth(data)).toBe('20年11月');
        });
        // Boundary test case
        it('should handle single data point', () => {
                const data: AssignPlanDetailData[] = [
                        {
                                // omit other properties for brevity
                                assigned_months: [{ month: '202年12月' }]
                        }
                ];
                expect(DateStringUtil.reduceOldestMonth(data)).toBe('20年12月');
        });
        …後略...
```

この例は基本的なもので、テスト対象の関数の完全なカバレッジを提供するわけではありません。テストケースを追加したり、必要に応じて既存のテストケースを調整して、全てのシナリオとエッジケースをカバーするようにしてください。

フロントエンド開発における生成AI活用事例

テストフレームワークの構築をサポート

PASS src/utils/DateStringUtil.test.ts FAIL src/components/pages/AssignPlanList.test.tsx ● AssignPlanList › renders without crashing The error below may be caused by using the wrong test environment, see https://jestjs.io/docs/configuration#testenvironment-string. Consider using the "jsdom" test environment. ReferenceError: document is not defined 5 | describe('AssignPlanList', () => { 6 | it('renders without crashing', () => { > 7 | render(<AssignPlanList />); | ^ 8 | expect(screen.getByText('アサイン計画入力')).toBeInTheDocument(); 9 | }); 10 | }); at render (node_modules/@testing-library/react/dist/pure.js:234:5) at Object.<anonymous> (src/components/pages/AssignPlanList.test.tsx:7:11)   Test Suites: 1 failed, 1 passed, 2 total Tests: 1 failed, 11 passed, 12 total Snapshots: 0 total Time: 4.863 s Ran all test suites.

このエラーメッセージは、JestがDOM環境でテストを実行していないことを示しています。Reactコンポーネントをテストするには、JestがDOM APIにアクセスできるように設定する必要があります。
Jestはデフォルトで"jsdom" 環境を使用しますが、何らかの理由で設定が変更されている可能性があります。これを解決するにはJestの設定で "testEnvironment" オプションが "jsdom" に設定されていることを確認してください。
Jestの設定は jest.config.js ファイルまたは package.json ファイルの jest セクションで行うことができます。
例えば、以下のように設定します:

```
// jest.config.js
module.exports = {
      // 他の設定...
      testEnvironment: 'jsdom',
};
```

または：

```
// package.json
{
      "jest": {
            // 他の設定...
            "testEnvironment": "jsdom"
      }
}
```

この設定を行った後、テストを再度実行してみてください。

フロントエンド開発における生成AI活用事例

- ModelとViewの紐づけなど、事務的なコーディングを高速化できる
- 画面表示によって実装結果が異なるケースをすぐに確認できるため、生成AIの実装間違いにすぐ気づくことができる
- テストフレームワークなどの構築に関しては設定をくみ上げるまでが時間がかかることが多かったが、生成AIによりエラー対処が非常に高速になる
- テストケースの概形づくりなど、事務的なコーディングを高速化できる
- テスト駆動開発と相性が良い

生成AIの間違いにすぐに気づき修正することができるため、完成形をイメージできていれば作業が 大幅に高速化 される

バックエンド開発における生成AI活用事例

バックエンド開発における生成AI活用事例

OpenAPI定義を生成

あなたは優秀なシステムエンジニアです。
リソース管理システムの #アサイン計画一覧画面 でアサイン計画一覧を描画するために必要なAPIを提案してください。
REST APIは #ER図 で示したデータベースに対しての操作のみおこなうものとします。
#アサイン計画一覧画面
画面上には以下の項目が列挙されています:
・アサイン計画一覧
・SE情報
・開発本部
・部署
・職番
・氏名
・職位（複数の職位を持つ場合は複数表示）
・案件情報
・Work.No
・案件名
・アサイン情報
・役割
・今月から3か月先まで計4か月分のアサイン時間(h)を以下の形式で月ごとに表示
・1Wのアサイン時間
・2Wのアサイン時間
・3Wのアサイン時間
・4Wのアサイン時間
・5Wのアサイン時間
・月合計
アサイン計画一覧画面を描画する際に必要な機能は以下です
・アサイン計画一覧画面では、自分の部下全員のアサイン計画を今月から3か月先まで計4か月分確認、変更できます。
・Work.No、案件名で絞り込むことができます。
・自分の部下のアサイン計画を一覧で見ることができます
#ER図:
@startuml
title リソース管理システム 物理データモデル(第3スプリント) アサイン+社員+案件カテゴリ
entity "assignments\nアサイン" as assignments {
#ERROR!
--
case_id INT [FK]
assigned_on DATE NOT NULL
employee_id INT NOT NULL
assignment_type_code VARCHAR
role_in_case VARCHAR(255)
organization_code CHAR(4)
…後略...

以下に、必要なAPIのOpenAPI 3.0.0 フォーマットのYAML形式を提供します

```yaml
openapi: 3.0.0
info:
title: リソース管理システムAPI
version: 1.0.0
paths:
  /assignmentPlans:
    get:
      summary: アサイン計画一覧を取得
      description: 自分の部下全員のアサイン計画を今月から3か月先まで計4か月分取得します。
      parameters:
        - name: workNo
          in: query
          description: Work.Noで絞り込みます
          required: false
          schema:
            type: string
            example: O94EK0
        - name: caseName
          in: query
          description: 案件名で絞り込みます
          required: false
          schema:
            type: string
            example: XXXX次期システム開発要件定義
      responses:
        200':
          description: アサイン計画一覧の取得に成功
          content:
            application/json:
              schema:
                type: array
                items:
                  type: object
                  properties:
                    seInfo:
                      type: object
                  properties:
                    jobNumber:
                      type: string
                      example: 12345
…後略...
```

バックエンド開発における生成AI活用事例

API入出力およびER図からSQLの概形を生成

以下にassign_plansテーブルのER図を示します。
---ER図---
@startuml
title リソース管理システム　論理データモデル　第3スプリントのみ　アサイン計画ビュー
entity "アサイン計画 :View" as assignplans {
　　organization_id INT [組織id]
　　organization_code CHAR(4) [組織負担コード]
　　full_organization_name VARCHAR(255) [完全組織名]
　　job_number VARCHAR(5) [職番]
　　employee_name VARCHAR(50) [氏名]
　　case_id INT [案件ID]
　　rank_list VARCHAR(255) [職位リスト]
　　…中略…
--- ER図おわり---
assign_plansテーブルを使って以下の jsonを返すAPIを作りたいです。
まずはこの APIで使用するSQLを考えてみてください
--- json ---
[{
　　development_division: "開発本部1",
　　department: "開発部1",
　　job_number: "12345",
　　name: "江草 一郎",
　　ranks: ["S-ITA", "APS"],
　　work_no: "O94EK0",
　　case_name: "XXXX次期システム開発要件定義 ",
　　assign_type: "有償",
　　role: "PM",
　　assigned_months: [
　　{
　　　　month: "2021年12月",
　　　　weekly_assigned_hours: {"week1": 20, "week2": 20,
　　　　"week3": 20, "week4": 20, "week5": 20},
　　　　monthly_total: 100
　　},
…後略…

以下に、提供された ER図を基に、指定された JSON形式のデータを取得するための SQLクエリーを作成します。ただし、 "development_division"と"department"の詳細な情報が ER図には存在していないため、これらのフィールドは省略します。
また、"weekly_assigned_hours"と"monthly_total"は週ごとおよび月ごとの合計時間を計算する必要がありますが、これは　SQLの集計機能を使用して行います。しかし、現状では 1週間ごとの時間を特定する情報が欠けているため、これらのフィールドも省略します。

```sql
SELECT
　　job_number AS "job_number",
　　employee_name AS "name",
　　rank_list AS "ranks",
　　work_no AS "work_no",
　　case_name AS "case_name",
　　CASE
　　　　WHEN assignment_type_code = '10' THEN '有償'
　　　　WHEN assignment_type_code = '20' THEN '生産的'
　　　　WHEN assignment_type_code = '30' THEN '無償'
　　END AS "assign_type",
　　role_in_case AS "role",
　　to_char(to_date(year::text || LPAD(month::text, 2,
'0'), 'YYYYMM'), 'YYYY年MM月') AS "month"
FROM
　　assign_plans
WHERE
　　job_number = '12345' AND
　　employee_name = '江草 一郎';
```

このSQLは、"job_number"が'12345'で、"employee_name"が'江草 一郎'のレコードを assign_plansテーブルから取得します。また、"assignment_type_code"に基づいて"assign_type"を決定します。"month"は年と月のフィールドを組み合わせて生成します。
ただし、このクエリは完全な解答ではなく、あくまで参考の一例です。必要に応じて、実際のデータベース構造や要件に合わせて調整が必要です。特に週ごとおよび月ごとの時間の集計については、具体的な要件やデータベースの構造により、異なる実装方法が考えられます。

バックエンド開発における生成AI活用事例

SQLから返ってくる値を格納するデータモデルを生成

以下のSQLを実行するメソッドを SQLAlchemyを使って書いています。この SQL実行結果を受け取る Modelを作ってください

```
SELECT
	organization_code AS "organization_code",
	full_organization_name AS "full_organization_name",
	job_number AS "job_number",
	employee_name AS "name",
	rank_list AS "ranks",
	work_no AS "work_no",
	case_name AS "case_name",
	CASE
		WHEN assignment_type_code = '10' THEN '有償'
		WHEN assignment_type_code = '20' THEN '生産的'
		WHEN assignment_type_code = '30' THEN '無償'
	END AS "assign_type",
	role_in_case AS "role",
	to_char(to_date(year::text || LPAD(month::text, 2, '0'), 'YYYYMM'), 'YYYY年MM月') AS "month",
	MAX(CASE WHEN week = 1 THEN weekly_assignment_minuts/60 END) AS "1W",
	MAX(CASE WHEN week = 2 THEN weekly_assignment_minuts/60 END) AS "2W",
	MAX(CASE WHEN week = 3 THEN weekly_assignment_minuts/60 END) AS "3W",
	MAX(CASE WHEN week = 4 THEN weekly_assignment_minuts/60 END) AS "4W",
	MAX(CASE WHEN week = 5 THEN weekly_assignment_minuts/60 END) AS "5W",
	SUM(weekly_assignment_minuts)/60 AS "monthly_total"
FROM
	assign_plans
GROUP BY
	organization_code,
	full_organization_name,
	job_number,
	employee_name,
	rank_list,
	work_no,
	case_name,
	assignment_type_code,
	..後略...
```

以下に、各列をそのままの形でモデルにマッピングするバージョンを示します。これは　SQLの各列をPythonの属性としてそのまま表現します。

```python
from sqlalchemy import Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base
Base = declarative_base()
class AssignPlan(Base):
	__tablename__ = 'assign_plans'
	organization_code = Column('organization_code', String, primary_key=True)
	full_organization_name = Column('full_organization_name', String)
	job_number = Column('job_number', String)
	employee_name = Column('employee_name', String)
	rank_list = Column('rank_list', String)
	work_no = Column('work_no', String)
	case_name = Column('case_name', String)
	assignment_type_code = Column('assignment_type_code', String)
	role_in_case = Column('role_in_case', String)
	year = Column('year', Integer)
	month = Column('month', Integer)
	week = Column('week', Integer)
	weekly_assignment_minuts = Column('weekly_assignment_minuts', Integer)
	__table_args__ = (
		{
			mysql_charset': 'utf8',
		},
	)
```

このモデルは、SQLの各列をPythonの属性としてそのまま表現しています。また、 organization_codeを主キーとして設定しました。それ以外のフィールドは、適切なデータ型を指定しています。

バックエンド開発における生成AI活用事例

テストコードの実装

以下のメソッドのテストコードをpytestで書いてください。

```python
async def get_organization_hierarchy(job_number: str, session: AsyncSession)->
List[OrganizationModel]:
    query = text("""
        WITH RECURSIVE organization_hierarchy AS (
        SELECT
            O.id, O.name, O.code, O.upper_id
        FROM
            employees E
        JOIN belongs B ON E.id = B.employee_id
        JOIN organizations O ON B.organization_id = O.id
        WHERE
            E.job_number = :job_number
        AND
            O.closed_on is null
        UNION
        SELECT
            O.id, O.name, O.code, O.upper_id
        FROM
            organizations O
        JOIN organization_hierarchy OH ON OH.id = O.upper_id
        AND
        O.closed_on is null
        )
        SELECT * FROM organization_hierarchy
    ")
    result = await session.execute(query, {job_number": job_number})
    result = result .fetchall()
    # レコードをOrganizationModelに変換します。
    org_list = []
    for row in result:
        org_model = OrganizationModel(
            id=row.id,
            name=row.name,
            code=row.code,
            upper_id=row.upper_id
        )
        org_list.append(org_model)
    return org_list
```

以下に、指定されたメソッドのテストコードをpytestを使用して書いてみました。テストでは、モックオブジェクトを使用して実際のデータベースへの接続を避け、想定される返り値をテストしています。

```python
import pytest
from unittest.mock import AsyncMock, Mock
from sqlalchemy.engine import ResultProxy
from your_module import get_organization_hierarchy # your_module  は実際のモジュール名に置き換えてください
@pytest.mark.asyncio
async def test_get_organization_hierarchy():
    # モック結果の作成
    mock_result = Mock(spec=ResultProxy)
    mock_result.fetchall.return_value = [
        {"id": 2, "name": " ｱｰｷﾃｸﾄ・ｻｰﾋﾞｽ・ｾﾝﾀｰ ", "code": "XE70", "upper_id": 1},
        {"id": 5, "name": " ﾃﾞｼﾞﾀﾙ・ﾌﾟﾗｯﾄﾌｫｰﾑ・ｻｰﾋﾞｽ・ｾﾝﾀｰ ", "code": "XE80", "upper_id": 1},
        {"id": 3, "name": " 第1ｻｰﾋﾞｽ室", "code": "XE71", "upper_id": 2},
        {"id": 4, "name": " 第2ｻｰﾋﾞｽ室", "code": "XE72", "upper_id": 2}
    ]
    # モックセッションの作成
    mock_session = AsyncMock()
    mock_session.execute.return_value = mock_result
    # テスト対象メソッドの実行
    result = await get_organization_hierarchy("88733", mock_session)
    # 結果の検証
    assert len(result) == 4
    assert result[0]["name"] == " ｱｰｷﾃｸﾄ・ｻｰﾋﾞｽ・ｾﾝﾀｰ "
    assert result[1]["code"] == "XE80"
    assert result[2]["upper_id"] == 2
    assert result[3]["id"] == 4
```

このテストでは、モックのAsyncSessionを使用して、実際のデータベースからのフェッチをエミュレートします。結果が期待通りに返されることを確認します。

バックエンド開発における生成AI活用事例

- OpenAPI定義などを高速に作成できる
- シンプルなSQL文の概形を高速に作成できる
- SQLに紐づくデータモデルの定義を高速に作成できる
- 実装コードに対するテストコードを高速に作成できる
- 複雑なビジネスロジックに関しては間違った回答を生成することが多い
- 複雑なビジネスロジックに関するテストケースの洗い出しは生成AIに任せるのは少々不安(抜け漏れをチェックする機能は薄い、かつ生成AIもテスト設計者も見逃している点に気づきにくくなる)

事務的なコーディングを非常に高速にこなせる　反面、
複雑なビジネスロジック への生成AIの適用は難しい
(複雑なビジネスロジックの開発に集中できる　)

傾向抽出：生成AIが生産性向上に寄与するケース

傾向抽出：生成AIが生産性向上に寄与するケース

- 用途に関する傾向
  - 設計資料からSQLへの変換 ◎
  - 事務的なコーディング ◎
  - OpenAPIを使用したバックエンド・フロントエンドの橋渡し ◎
  - 実装結果をすぐに確認しやすいフロントエンドにおける開発の高速化 ◎
  - 単純なテストコードの生成(テスト駆動開発の補助) ◎
  - 要件資料から設計資料への落とし込み △
- データに関する傾向
  - インターネットに情報が多い場合の検索の高速化(絞り込み) ◎
  - 公式ドキュメント(英語のみの場合など)の日本語化、実装例提示 ◎
  - エラーメッセージの説明と解決策提案 ◎
  - プロンプト内にヒントがある場合のデータ変換 ◎

傾向抽出：生成AIが生産性向上に寄与しないケース

傾向抽出：生成AIが生産性向上に寄与しないケース

- 用途に関する傾向
  - 複雑なビジネスロジックの生成　×
    - 細かいステップに分けてプロンプトを作成する場合は　○
  - 複雑なSQL文の生成　×
- データに関する傾向
  - 学習モデル作成時よりも新しい情報を取得できない　×
    - Reactやviteなど、公式情報の更新が早い場合、対応できないケースがある
    - Cfnテンプレートにおいても古い記法を提示することがある
  - インターネットに情報が出回っていない場合、的外れな回答をする場合がある　×
    - vitestにおけるmock対象のimport順など、ヒント程度の記事しかネットにない場合は生成AIも良い回答は返してくれない
    - AWS batchのサービス起動時のsystemctlコマンドの実行順など、AWSサポートに聞く以外に情報がない場合は生成AIも良い回答は返してくれない
  - 生成AIが気を回しすぎた結果(yaml定義の文面を自然言語的に解釈してしまう場合など)、まと外れな回答をすることがある　×
    - Cfnテンプレートにおいて存在しない(それっぽく見えるような)プロパティを回答に含めるケースがある

傾向抽出：生成AIを使用した開発における注意点

傾向抽出：生成AIを使用した開発における注意点

- 回答に間違いが含まれるケースがあるため、そのままプロダクトに入れることが難しい
- フレームワークに関するトラブルにも回答してくれるため、多用するとフレームワークへの理解が薄れる
- トラブルシューティング時に生成AIとのやりとりに集中しすぎると、想定していない原因を見落としがち(生成AIが会話の内容に重点を置いて回答するあまり、想定外のアイデアを提供してくれることが少ない)
- 生成AIとのやり取りに集中するよりも、インターネットで直接検索する方が素早く解決できることがある(ボーダーを見つけづらいため、「5回聞いてダメだったら自分で調べる」のようにしたほうが良い)

傾向抽出：生成AIを使用した開発における問題点

傾向抽出：生成AIを使用した開発における問題点

- 生成AIを使って開発した成果物であることを正しく認知し、納品までのチェックを行うワークフローを整備する必要がある
  - 7，8割は正しい成果物であるため他者が生成AIで作成したものを添削する際は細心の注意を払う必要がある
  - チェックを行う人員の負荷は低くならない
- 成果物に対する責任を自身が負っていることを深く認知して納品(コミット)するべき

考察：生成AIを活用する際に求められるスキル

考察：生成AIを活用する際に求められるスキル

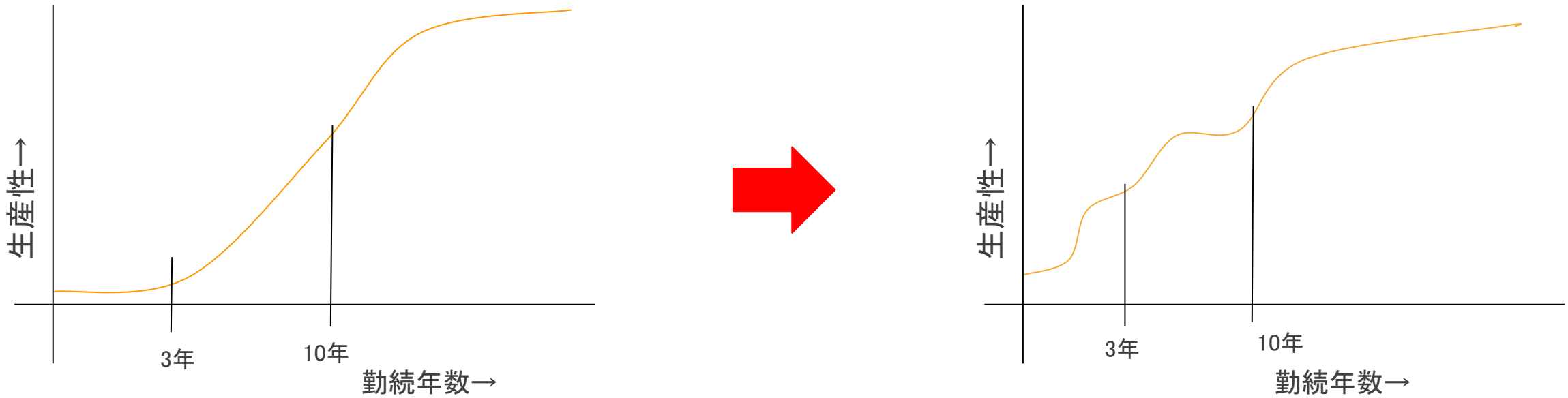
- 生成AIが提示した成果物の稼働メカニズムを探求する姿勢
  - 同じことを何度も聞かなくても良いように、深く理解する必要がある
  - 成果物の稼働メカニズムを説明できるようになっているべき
    - 例)「ReactのuseEffectの仕様として今回のニーズにはこの実装が最適です」など、公式ドキュメントに書かれている内容程度は身に着けておくべき
- 生成AIが提示した成果物を添削する基礎知識(以下、スキルレベル例)
  - 設計
    - 豊富な現場経験
  - インフラ
    - AWS Solution Architect Associate + 2、3年の実務経験
      - もしくは AWS Solution Architect Professional相当
  - フロントエンド
    - Javascriptでの開発経験 + 2，3年のVue、Reactなどのフレームワークを使用した開発実務経験
  - バックエンド
    - 2，3年のJava、Pythonもしくは各プロジェクトが重点を置いている言語の開発経験

考察：生成AIが最も効能を発揮する活用シーン

考察：生成AIが最も効能を発揮する活用シーン

業務では 生成AIを活用するにはある程度の基礎知識を持っている必要がある

学習においては 基礎知識を獲得するまでの時間を **短縮**できる



考察：生成AIが最も効能を発揮する活用シーン

業務では 生成AIを活用するにはある程度の基礎知識を持っている必要がある

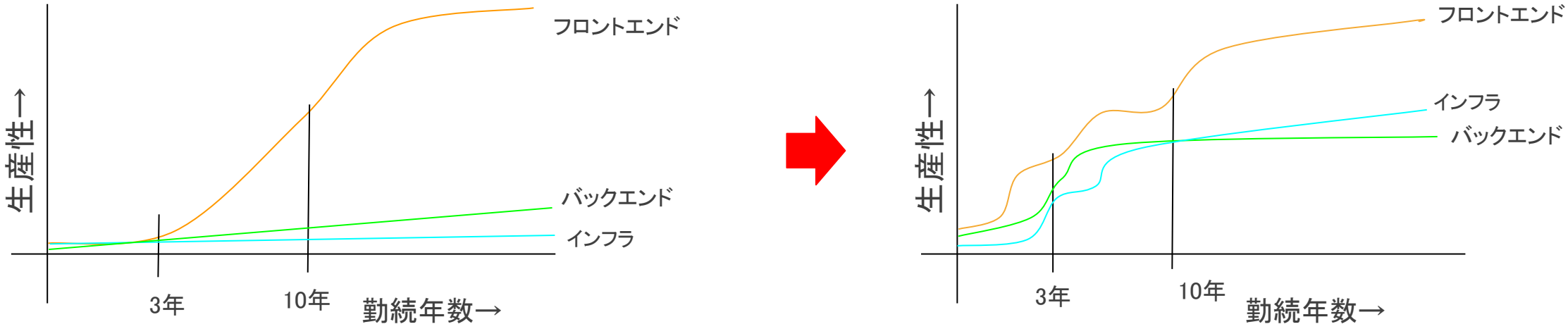
学習においては 基礎知識を獲得するまでの時間を **短縮**できる

＋

社内においてはインフラ、アプリ開発(フロントエンド、バックエンド)は異なる知識系統になっている

＝

複数分野をまたいで中堅レベルの知識を身に着けるのに役立つ

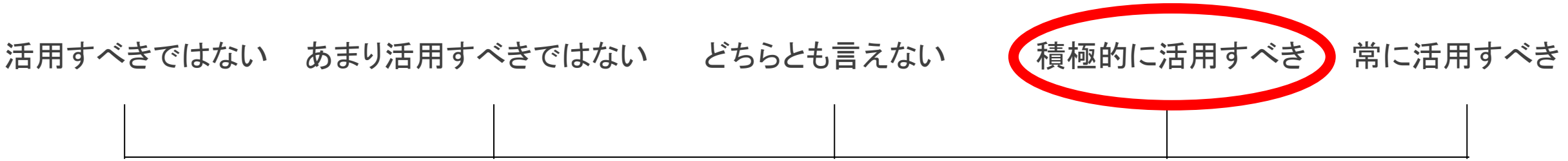


考察：生成AIが最も効能を発揮する活用シーン

複数分野をまたいで中堅レベルの知識を身に着けるのに役立つ

全社的にAWS資格保有者を増やそうとしている今のエクサの取り組みに合致する!

アプリもわかるインフラエンジニア
インフラもわかるアプリエンジニア
が集まった柔軟で強力なエンジニア集団になれる！



参考資料

※1 コード生成ツールの導入判断のための評価方法の提案
（https://exa-knowledge.exa-corp.co.jp/open.knowledge/view/2622）