TOSHIBA

ソフトウェア品質シンポジウム 2024 パネルディスカッション
「どのようなソフトウェア開発活動で生成AIが活かせるか？」

生成AIを用いたソフトウェア開発効率化 東芝における取り組みの紹介

2024/9/12

株式会社 東芝
デジタルイノベーションテクノロジーセンター
先端ソフトウェア技術室 ソフトウェアエンジニアリング技術部

藤本 宏

自己紹介：藤本 宏

- 経歴
  - 2003年 株式会社 東芝 ソフトウェア技術センター 入社
    医療機器, MFP, デジタルテレビ, 他多数の製品開発プロジェクトに担当者として参画し、
    アーキテクチャ再設計, 開発環境整備, 設計実装担当 に従事
  - 2014年～2022年
    主にソフトウェア設計の技術開発と社内展開を担当し、以下の技術に取り組む
    要求仕様化, 形式検証, 設計検証, 構造診断, モジュール化, リファクタリング, コード理解支援, クラウド活用
  - 2023年～
    生成AI活用検討のチームを構築、ソフトウェア開発効率化に向けた生成AIの”使い方”確立に向けた試行と展開を実践

ソフトウェア開発の現場が好き、
製品リリース直前の夜にやってくる謎なハイテンション
難解なデバッグ作業をこなした時の達成感と開放感
開発技術はそれを使いこなせる人がいてこそ
課題に対応する技術と、技術を使う人の育成が両輪
人の作業をサポートし、ソフト開発をより良く/楽に/面白くする技術が目標

概要

- 東芝グループでは、**事業活用・スタッフ業務効率化・設計効率化**など、
  生成AI活用について様々な取り組みを進めています

- ソフトウェア開発業務の効率化に向けては、**大量の既存ソフトウェア資産の活用**を
  重要な課題として取り組んでいます
  - まずは、ソフトウェア開発における各作業について、どの作業に対して、どのように生成AIを活用すると有用か？
  - その先に、生成AI活用を前提としたとき、一連のソフトウェア開発はどのような開発に変わるのか？

ソフトウェア開発全般の様々な作業について試行と評価を大量に実施し
有用な活用方法をベストプラクティスとして整理

このベストプラクティスの一部を紹介します

生成AIを用いたソフトウェア開発効率化の技術

- 生成AIを用いたソフトウェア開発効率化の技術開発ステップ

①仮説構築
現状資産と課題の把握と
生成AI活用の構成検討により
実施可能を確認する

②仮説実装
具体的なモチーフに対して
生成AI活用方法を工夫し
期待した結果が得られる

③仮説検証
類似のモチーフに対して
同じ生成AI活用方法で
期待した結果が得られる

④実地評価
実際の開発プロジェクトで
生成AI活用方法を適用し
効果を計測して評価する

モチーフに対して有効な生成AI活用方法を検討し、**大量のソフトウェア資産への活用展開と業務自動化を志向**する

- ソフトウェア開発工程と対応づけて整理した生成AI活用技術の例

要件定義	設計	開発	テスト	運用・保守	PJ管理
	設計書 生成・修正・補完	ソースコード 生成・修正・補完	テスト生成 (シナリオ,データ,コード)	マニュアル生成	議事録自動生成
		マイグレーション			
設計レビュー支援		コードレビュー支援			
文章チェック					
要求仕様分類			テスト実施 優先度提案		OSS違反検出 OSS提案
要求仕様書 理解支援	設計書 理解支援	ソースコード 理解支援			
ドキュメント検索		ソースコード検索		異常検知	QA生成

：重点対象領域
（成果物作成、チェック、
情報抽出、情報検索）

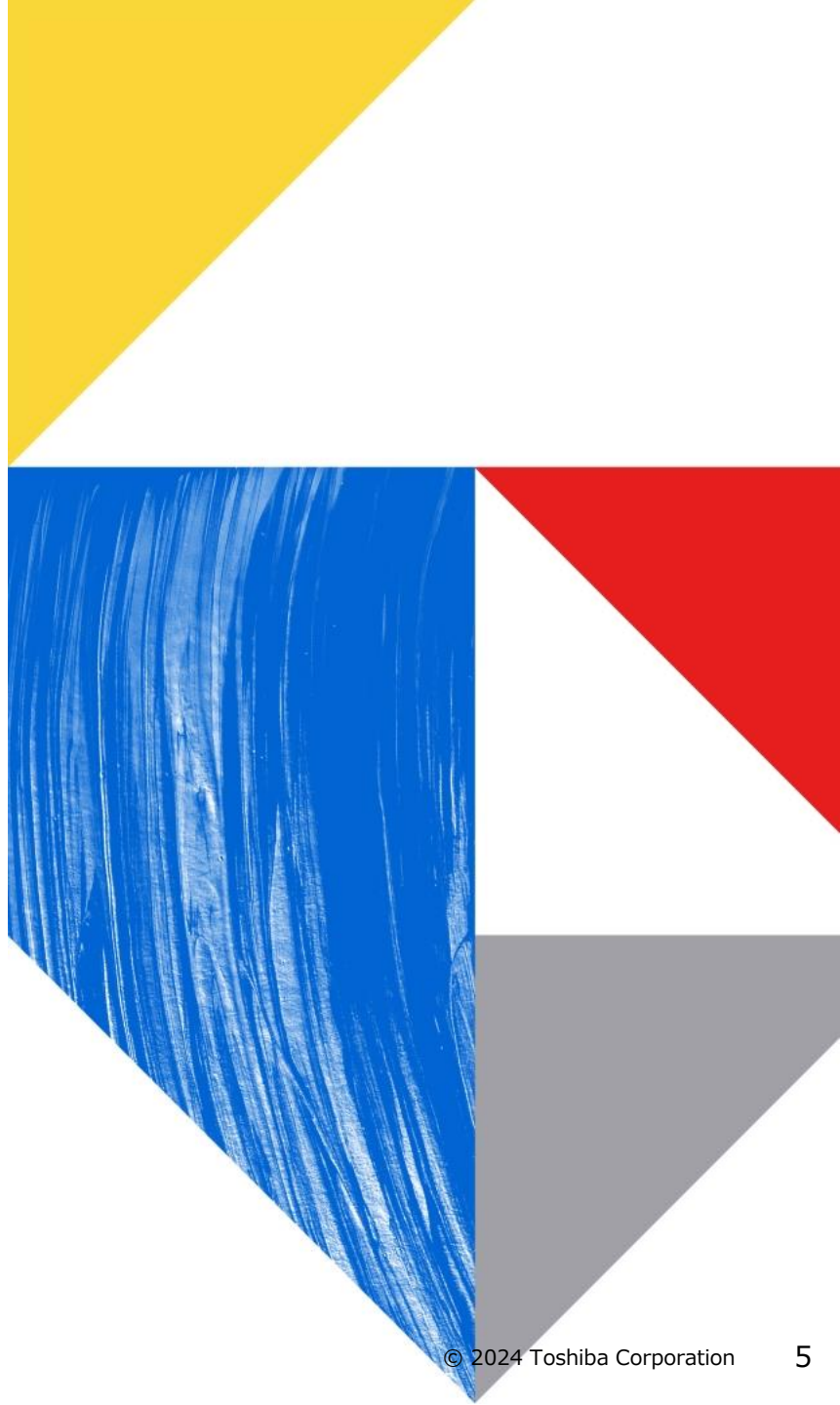
試行評価/活用評価の例

工程	作業	生成AI活用内容
要件定義	要件項目の整理	非機能要件のリストを分類する
設計	設計書の作成	ソースコードから仕様書を生成する
	テーブル設計	正規化されていない表構造を正規化し、物理名を付ける
	方式設計	機能の説明から、実現するための構成を挙げる
	基本設計	仕様文から、状態遷移表を作成する
	設計レビュー	レビューチェックリストに則ったレビューコメントを生成する
	設計書の理解	既存ドキュメント群に対する質問への回答を生成する
開発	ソースコードの実装	仕様文からプログラムコードを作成する
	SQL文の作成	テーブル構造と仕様文から検索SQLを作成する
	Javaプログラムのヘッダの作成	メソッドの内容を説明し、決められた形式で出力する
	別フレームワークへのマイグレーション	既存ソースコードに対して、新フレームワーク適用のための変更を生成する
	リファクタリング	既存ソースコードのロジックを簡潔かつ理解容易に変更する
テスト	単体テストコードの実装	プログラムコードから単体テストコードを作成する
	テストパターンの洗い出し	仕様文からデシジョンテーブルを作成する
運用・保守	ライブラリのバージョンアップ	コードを、指定バージョンに対応するコードに変更する
PJ管理	文章校正	文章中の表記のばらつきを指摘、修正する
	議事メモ要点まとめ	打ち合わせのメモから、要点を抽出する

※ この表には、結果の良否を問わず試行に取り組んだ対象を記載。他にも多数の試行を実施中。

01

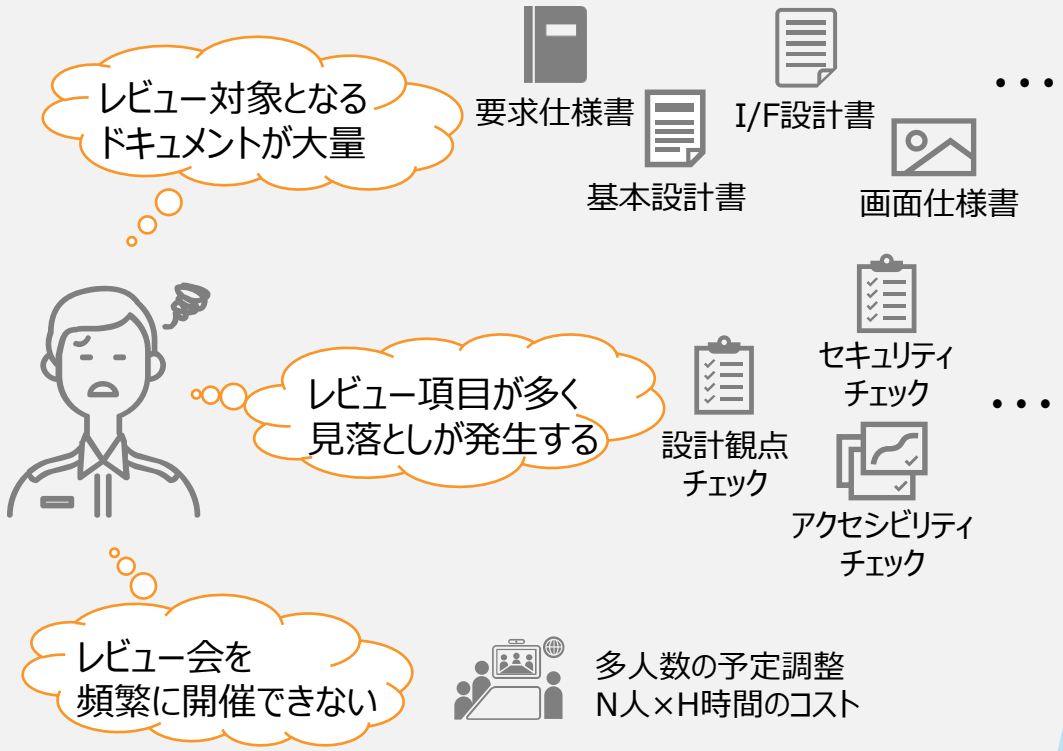
試行例：
ドキュメントレビューの自動化



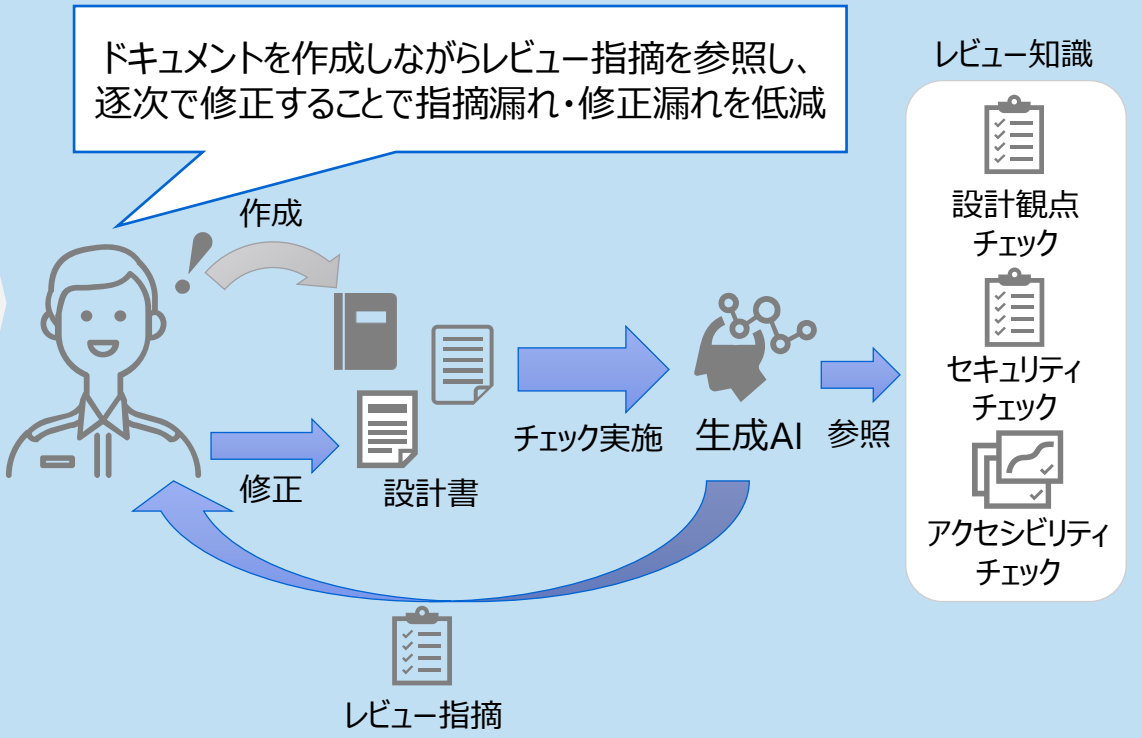
生成AI活用技術：設計レビューの自動化

実施内容	設計書のレビューにおいて、生成AIがレビューチェック項目に則った指摘事項を生成する
期待効果	設計書作成中に担当者自身がセルフレビューとして生成AIを使い指摘事項を修正しておくことで、 有識者レビューの前に問題点を修正することで、レビュー工数と指摘漏れを低減する

設計レビューで問題点を潰しきれない

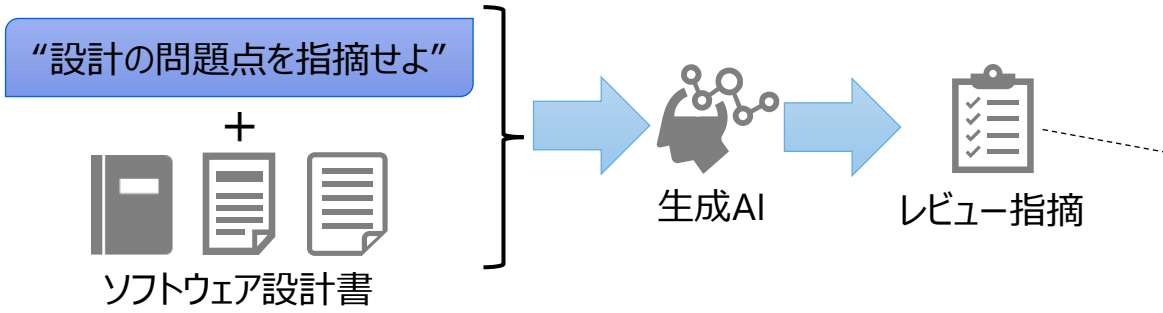


高頻度な生成AIによるレビューで問題点を素早く修正



設計レビューは可能か？

- 基本的方式
  - **簡単な指示でレビューを実施**

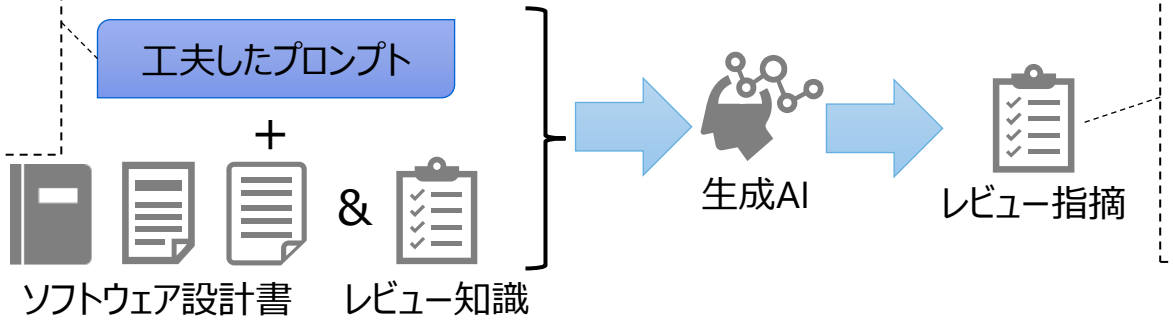


未定義や空欄や「-」などの簡単な指摘や、
HTTPステータスコードの不足といった、
一般的知識に基づく問題点は指摘される。
⇒ 複雑な条件やドメイン特化の問題点は
指摘されない。

良い結果を出すには、どんな情報が必要か？

  - **レビューの質を向上する方法**

プロンプト内の指示を工夫
- レビューに関する知識を含める
  (レビューチェックリストなど)
- ドメイン固有の情報を含める
- 類似ソフトの指摘事項を含める
- …



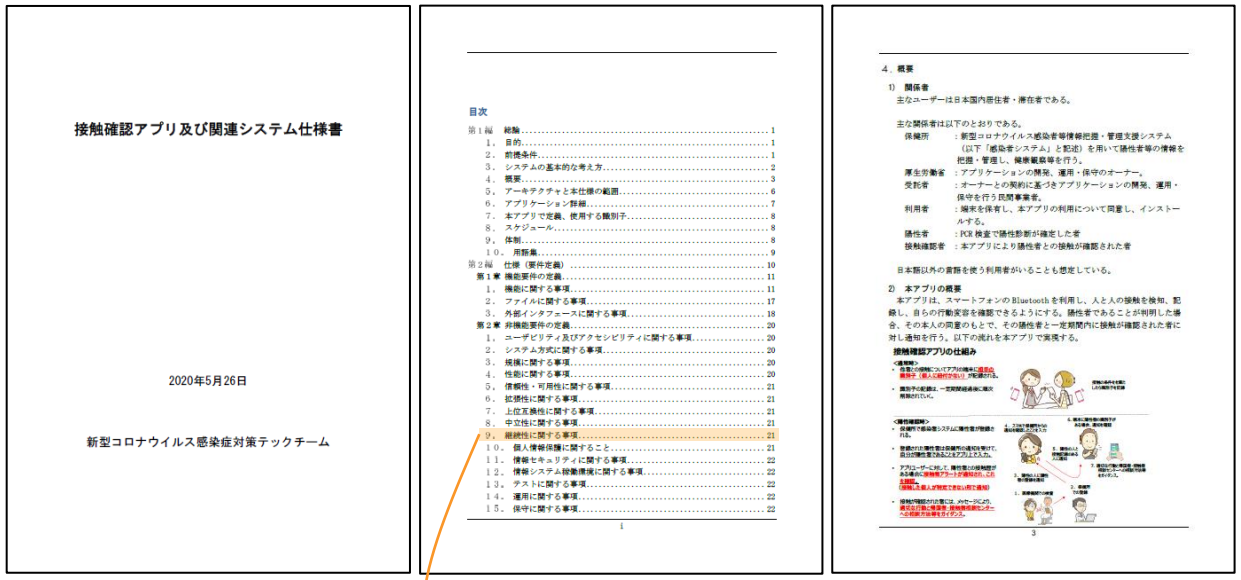
レビュー結果で良い点・悪い点の両方が指摘され、
悪い点については修正の参考が回答されることで
レビューチェックから修正までの役に立つ情報が
生成された。
ドメイン知識やプロジェクトが積み重ねたチェック項目
も指摘可能。

要求仕様書レビューの例

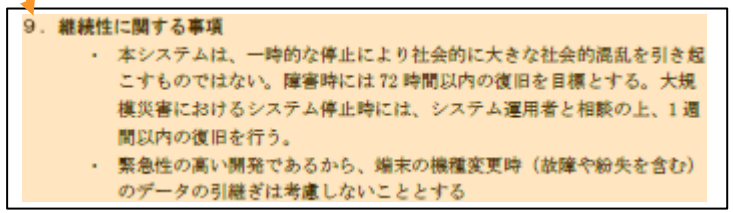
レビュー指摘の評価実験では、レビュー観点に対応する仕様を抽出でき、仕様の不足に関しては記載がない情報を指摘できた。

接触確認アプリ(COCOA)の仕様書

※https://cio.go.jp/node/2613



PDF仕様書に含まれる全てのテキストデータを抽出し、
工夫したプロンプトと合わせて生成AIへレビューを指示



生成AI

レビュー指摘

可用性の継続性に関する記載は以下の箇所にあります：

1. **信頼性・可用性に関する事項**
   - 端末での稼働率は 98%以上とし、サーバーの稼働率は 95%以上とする。
2. **継続性に関する事項**
   - 本システムは、一時的な停止により社会的に大きな社会的混乱を引き起こすものではない。障害時には 72 時間以内の復旧を目標とする。大規模災害におけるシステム停止時には、システム運用者と相談の上、1 週間以内の復旧を行う。
   - 緊急性の高い開発であるから、端末の機種変更時（故障や紛失を含む）のデータの引継ぎは考慮しないこととする。

不足している内容は以下の通りです：

1. **運用スケジュール**
   - システムの稼働時間や定期的なメンテナンス時間についての具体的な情報が不足しています。
2. **業務継続性**
   - 継続性を保証する業務の範囲や、具体的な業務継続計画（BCP: Business Continuity Plan）についての詳細が不足しています。
3. **目標復旧水準**
   - 具体的な目標復旧時間（RTO: Recovery Time Objective）や目標復旧ポイント（RPO: Recovery Point Objective）についての詳細が不足しています。

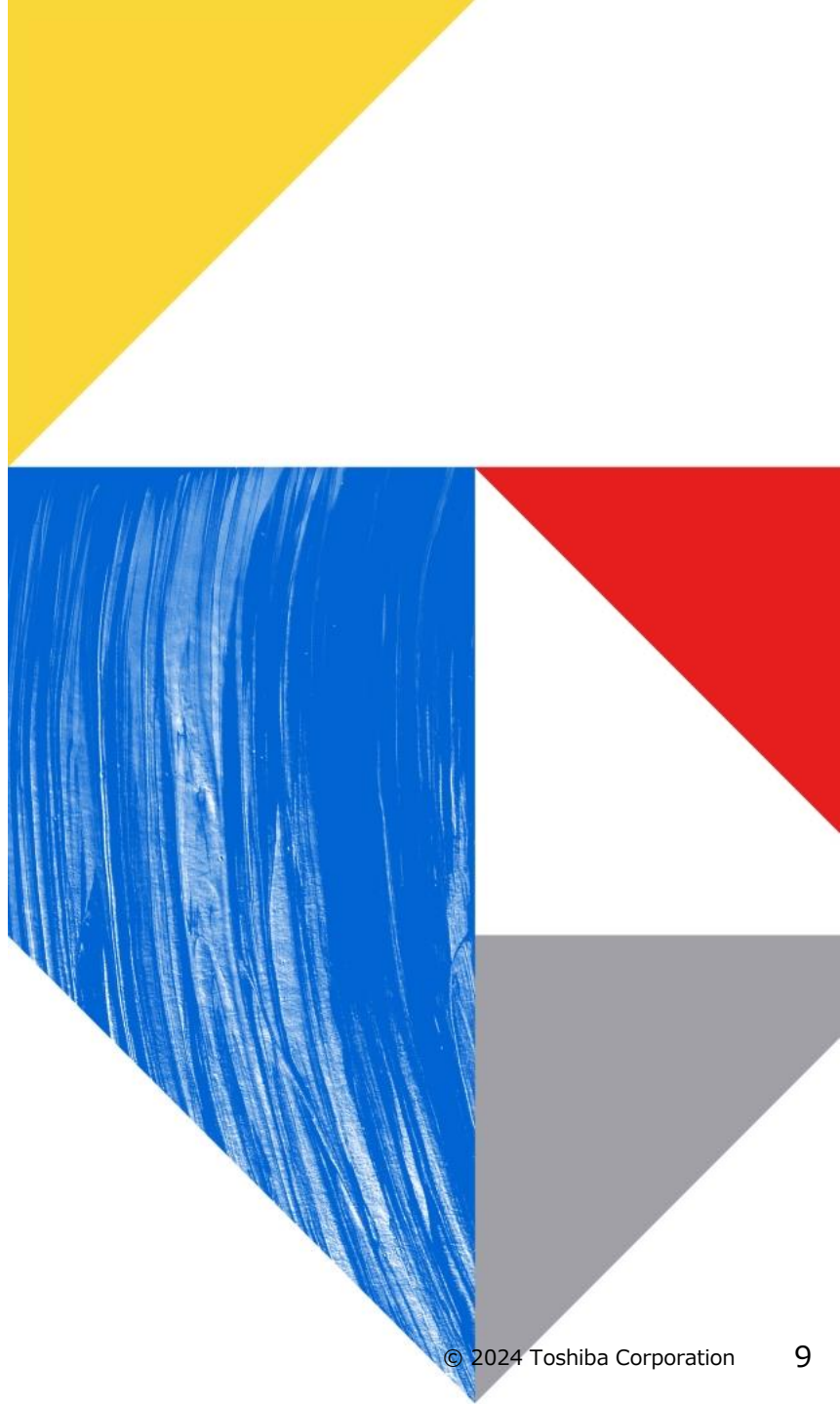
これらの不足内容を補完するためには、以下の情報を追加することが推奨されます：

- **運用スケジュール**：システムの稼働時間、定期的なメンテナンス時間、緊急メンテナンスの手順など。
- **業務継続性**：業務継続計画（BCP）の詳細、災害時の対応手順、代替手段の確保など。
- **目標復旧水準**：具体的なRTOとRPOの設定、復旧手順、データバックアップの頻度と方法など。

これにより、システムの可用性と継続性がより確実に保証されるでしょう。

02

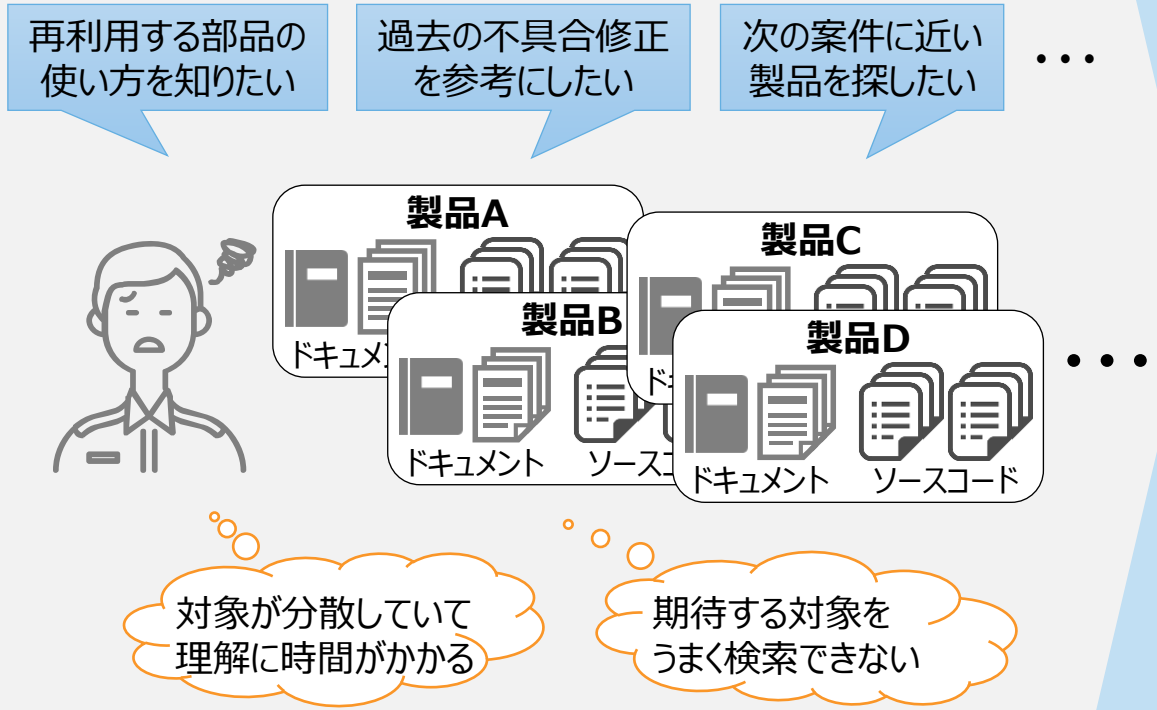
試行例：
ソフトウェア資産の活用



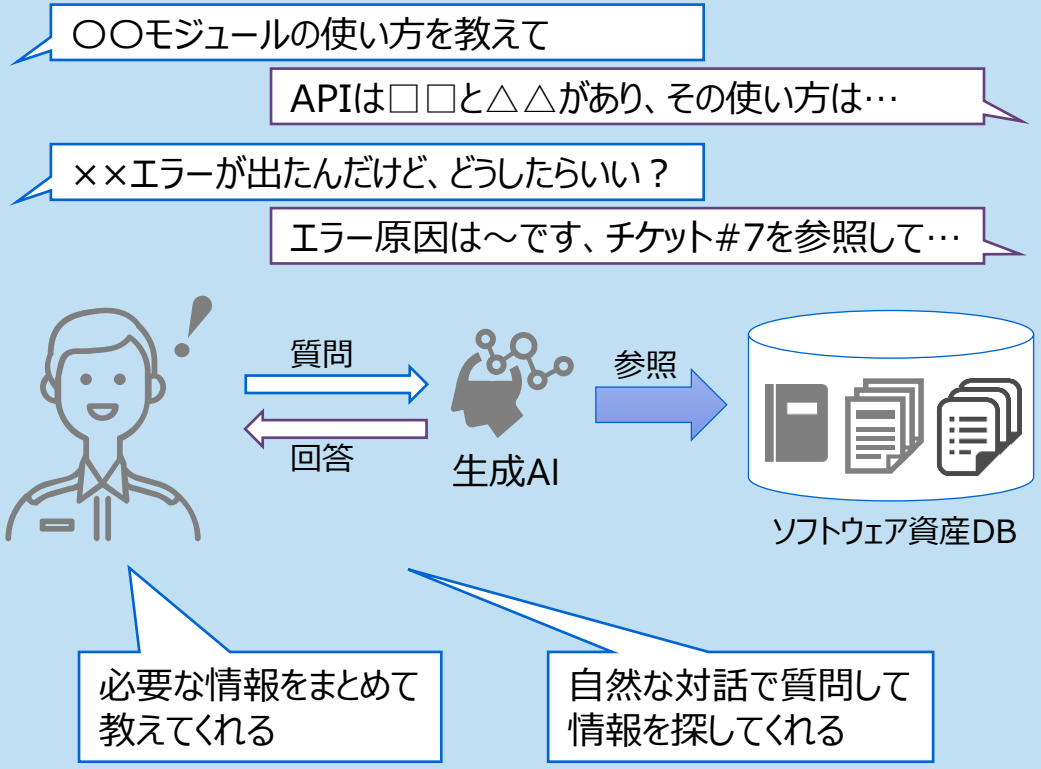
生成AI活用技術：ソフトウェア資産の活用

実施内容	既存の設計書の情報を生成AIに持たせ、設計書内容に関する質問への回答を生成する
期待効果	設計時や実装時に発生した設計内容に対する質問をAIに回答させることで担当者の設計理解を深め、 該当する設計情報のドキュメントを提示させることで大量の設計書から情報を探す時間を短縮する

大量にあるソフトウェア資産を活用できていない

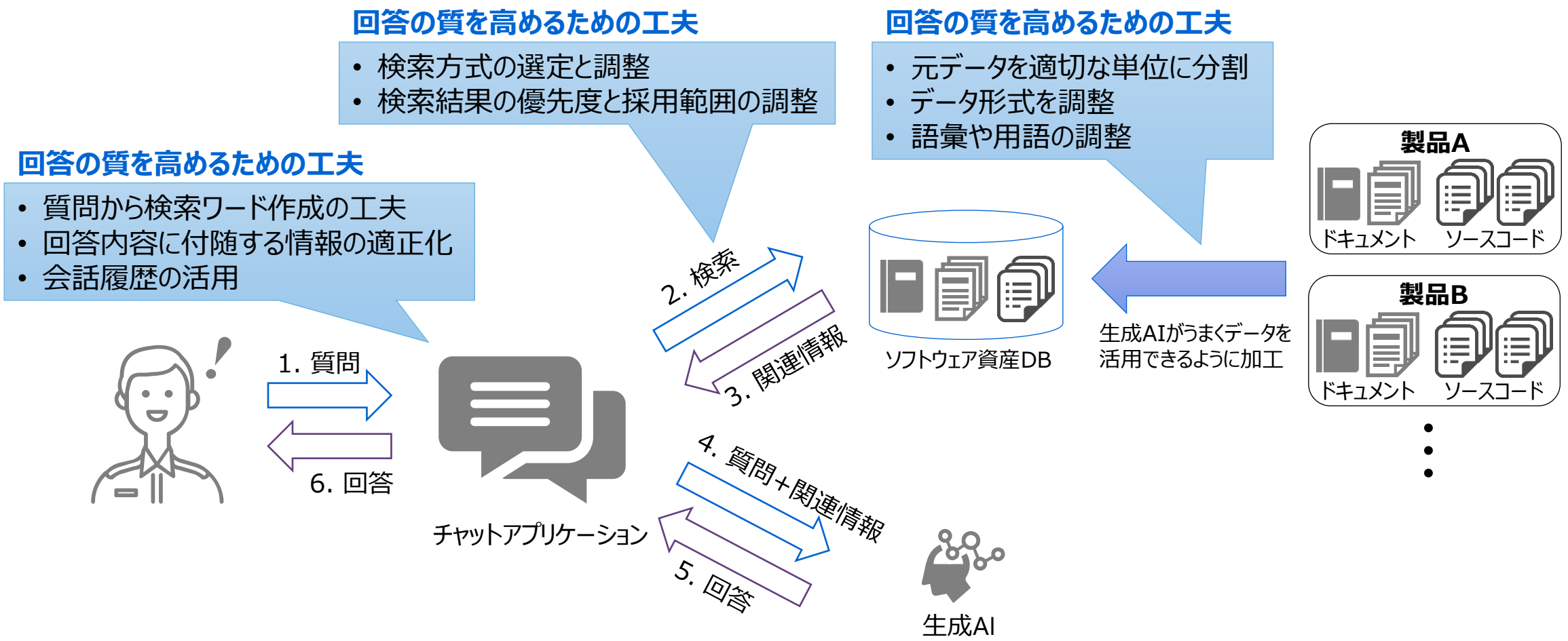


生成AIがソフトウェア資産の活用方法を教えてくれる



大量のソフトウェア資産をどうやって生成AIで活用するか？

- RAG(Retrieval Augmented Generation：検索拡張生成)技術を用いて、
  LLMが作成時に学習した知識に、独自のデータソースを追加して回答を生成



大量のソフトウェア資産を活用したチャット例

RAGの回答内容は必要な情報を含んでおり、
利用者の定性的評価では有用という判断だった。

※ 例では自然言語処理機能を備えたアプリのドキュメントやソースコードをナレッジ化

構文解析部品で提供されるAPIについて説明してください

構文解析部品は、以下のようなAPIを提供しています：

1. **構文解析を実行するAPI**
   - **関数名:** `ParseUniqueSpec`
   - **概要:** UniqueSpec型で与えられた1組の一意な要求仕様を構文解析し、解析結果を公開変数 `parse_result` に追加します。
   - **詳細:**
     - **入力:** UniqueSpec型の要求仕様
     - **出力:** 解析結果が `parse_result` に追加される
     - **エラー処理:** 解析に失敗した場合はエラーコードを返します。
2. **終了処理を行うAPI**

～～～～～～ 途中省略 ～～～～～～

システムの詳細

- **データの流れ:**
  1. **入力:** 一意な要求仕様（日本語の文章）
  2. **形態素解析:** JUMANを用いて形態素解析を行い、形態素情報を取得

～～～～～～ 途中省略 ～～～～～～

引用情報:

1. JUMANKNP解析部品_部品設計書.docx
2. JUMANKNP解析部品_部品化提案資料.pptx

- API名だけではなく、入出力など利用時に必要な情報を回答
- API利用方法を理解するための処理フローを付随
- 情報源のドキュメントを提示

部品の利用方法が分かるコード例を提示して

構文解析部品を利用するためのコード例を以下に示します。このコード例では、部品の初期化、構文解析の実行、結果の取得、終了処理の流れを示しています。

コード例

```
#include <stdio.h>
#include <stdlib.h>
#include "juman_knp_parser.h"  // 構文解析部品のヘッダファイル

// メイン関数
int main() {
```

～～～～～～ 途中省略 ～～～～～～

```
    return EXIT_SUCCESS;
}
```

説明

1. **初期化**
   - `set_output_path` 関数を使用して、解析結果の出力先ディレクトリを設定します。
2. **一意な要求仕様の定義**
   - `UniqueSpec` 構造体を使用して、一意な要求仕様を定義します。この例では、IDが"spec1"で、テキストが"これはテストの文章です。"という要求仕様を定義しています。

～～～～～～ 途中省略 ～～～～～～

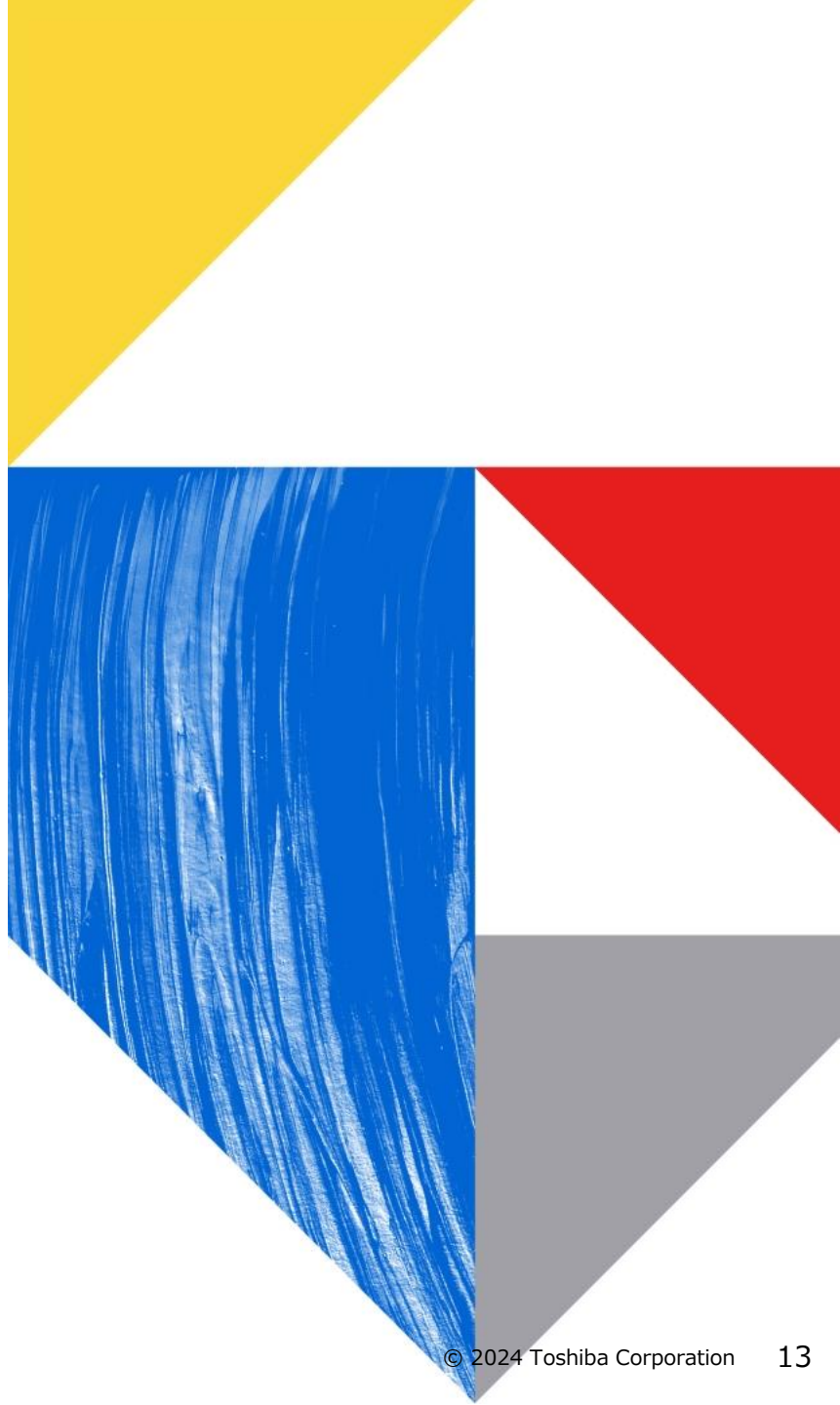
注意点

- このコード例は、構文解析部品のAPIを利用するための基本的な流れを示しています。実際の利用に際しては、エラーハンドリングやメモリ管理などの詳細な実装が必要です。

- 部品を利用するコードから推測したコード例を生成

03

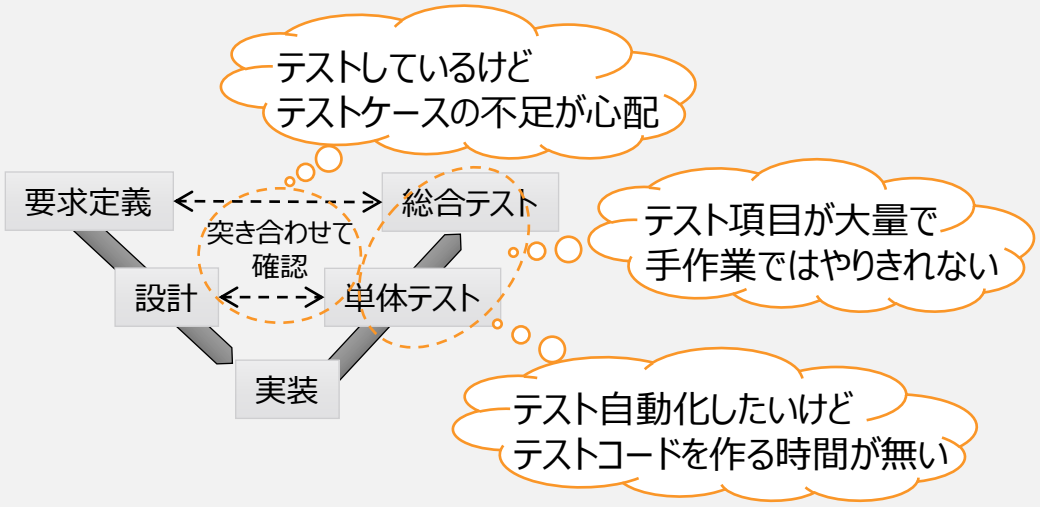
試行例：
単体テストコードの生成

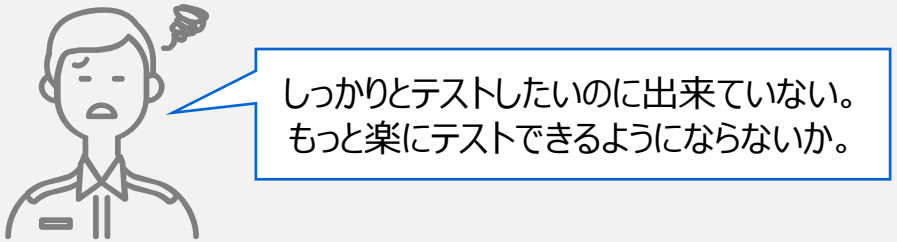


生成AI活用技術：テストコードの生成

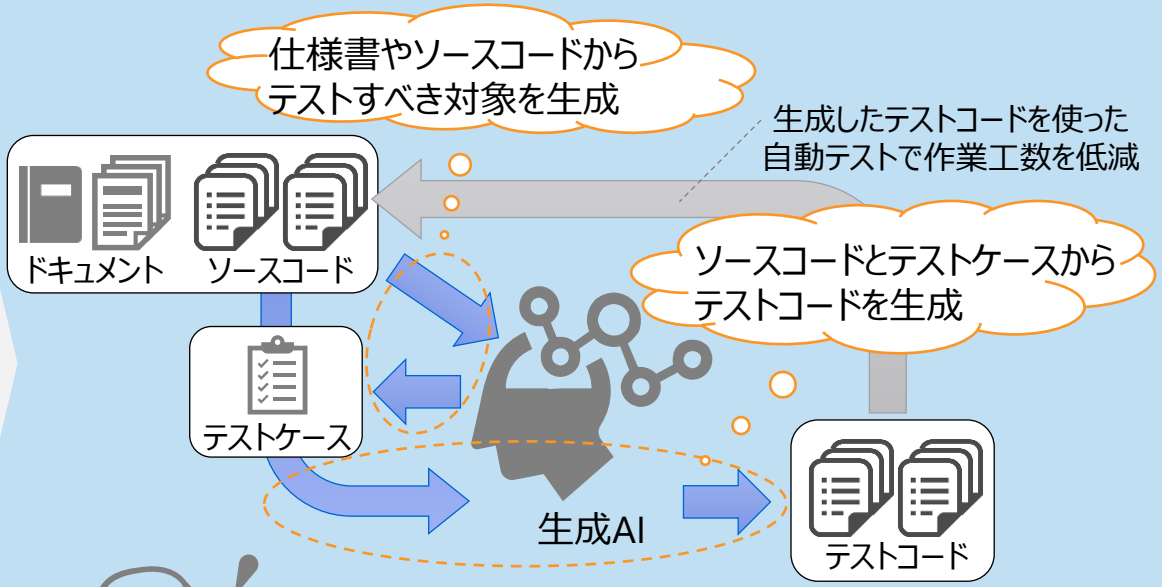
実施内容	ソースコードや仕様書を入力として、生成AIがテストコードを生成する
期待効果	テストケースおよびテストコード作成にかかる工数を低減することで自動テストの実施率を高めて、 リリース後の不具合発生を減少する。

テストは重要だけど、大量のテストをやりきれない



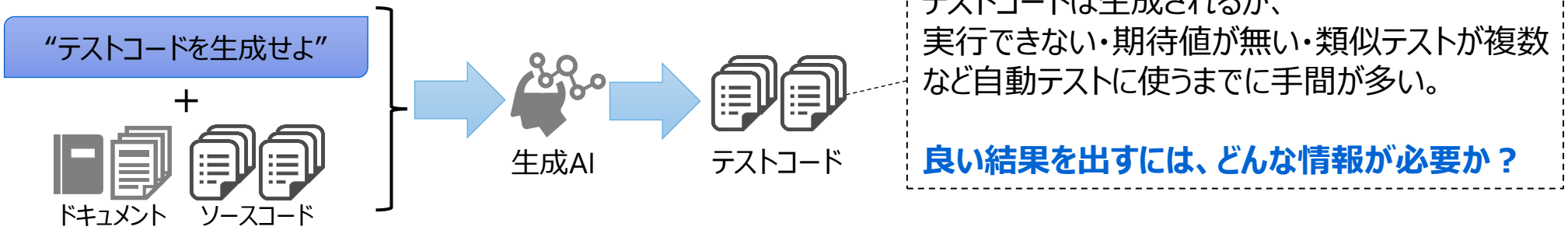


生成AIのテストコード生成により網羅的なテストを実施

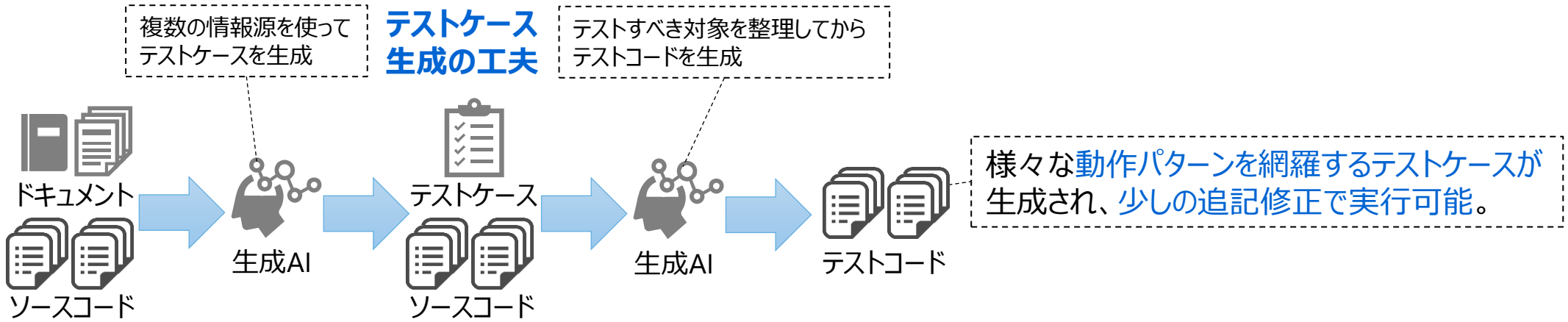


テストコード生成は可能か？

- 基本的方式
  - **簡単な指示でテストコード生成を実施**



  - **テストケースの質を向上する方法**



テストコード生成の例

テストコードの生成実験では、生成されたテストコードの分岐網羅率は平均91%、
人手でのテストコード作成時間と比べて66.7%の作業時間削減となった。

テストケース

1 参照 使用済み >

以下に、指定された制約条件と出力形式に基づいて、`check_data_validity` 関数のテストケースを提案します。

テストケース1: 正常系

事前条件

- `data_path` が有効なパスである
- `data_path` が存在する
- `data_path` に `images.zip` と `annotation.csv` が存在する
- `images.zip` には有効な画像ファイルが含まれている
- `annotation.csv` には、存在する画像ファイル名と0-9の範囲のカテゴリが含まれている
- 画像のピクセル値が0-255の範囲内である

事前条件は、正常系の場合のif文の条件を満たすものが全て出力された。

テストの確認内容

- `check_data_validity` 関数がエラーなく実行される
- `check_data_validity_results.json` が生成される

期待値

- `check_data_validity_results.json` の `result` が `pass` である
- `not_exist_images`, `failed_to_load_images`, `out_of_range_images`, `out_of_range_category` のリストが全て空である

期待値も、正常系の場合の最終的な結果が具体的な値とともに出力された。

テストケース2: 異常系 - `data_path` が存在しない

事前条件

- `data_path` が存在しない

テストの確認内容

- `check_data_validity` 関数がエラーをスローする

期待値

- `data_path {data_path} does not exist.` というエラーメッセージがスローされる

異常系として、事前条件を満たさない場合のテストケースが一通り出力された。

テストコード

1 参照 使用済み >

以下に、指定されたテストケースに基づいてテストを生成します。テストデータは事前条件に基づいて作成され、各テストケースはpytestライブラリを使用しています。

```python
import pytest
import os
import shutil
import pandas as pd
import cv2
import numpy as np
import json
from check_data_validity import check_data_validity

def setup_function():
    if not os.path.exists('test_data'):
        os.makedirs('test_data')

def teardown_function():
    if os.path.exists('test_data'):
        shutil.rmtree('test_data')

def create_test_data(images_exist=True, annotation_exist=True, invalid_image=False, inv
    if images_exist:
        cv2.imwrite('test_data/images.zip', np.random.randint(0, 256, (100, 100, 3), dt
    if annotation_exist:
        df = pd.DataFrame({
            'file_name': ['images.zip'] if not invalid_image else ['invalid.jpg'],
            'category': [5] if not invalid_category else [10]
        })
        df.to_csv('test_data/annotation.csv', index=False)

def test_check_data_validity_case1():
    create_test_data()
    check_data_validity('test_data')
    with open('check_data_validity_results.json', 'r') as f:
        results = json.load(f)
    assert results['results'][0]['result'] == 'pass'
    assert not results['results'][0]['not_exist_images']
    assert not results['results'][0]['failed_to_load_images']
    assert not results['results'][0]['out_of_range_images']
    assert not results['results'][0]['out_of_range_category']

def test_check_data_validity_case2():
    with pytest.raises(Exception, match=r"data_path test_data does not exist."):
        check_data_validity('test_data')

def test_check_data_validity_case3():
    create_test_data(images_exist=False)
    with pytest.raises(Exception, match=r"data_path test_data does not contain images.z
        check_data_validity('test_data')
```

生成されたテストコードは、テスト対象を呼び出す部分の修正のみで実行できた。
テストはソースコードとしては全て成功するはずのもので、テスト結果はPassed3件、Failed6件。
テストコードの修正でPassed8件、Failed1件となった。
(Failedの1件はテストデータの準備が困難なもの)

本日のまとめ

- ソフトウェア開発効率化のための生成AI活用について、東芝グループでは
  大量の既存ソフトウェア資産の活用を重要な課題として試行評価を実施しています
- 試行において有用な効果が見込まれたベストプラクティスをご紹介しました
  1. 設計レビューの自動化
  2. ソフトウェア資産の活用
  3. 単体テストコードの生成
- 有用な生成AI活用に向けた議論トピック

生成AI活用の期待が大きい作業

- コードマイグレーションやリファクタリング
- ソフトウェアの評価と改善提案

技術
プロセス
ノウハウ

生成AI活用における課題

生成結果に確率的に混入する誤り
に対してどのように対処するか？


TOSHIBA