Hitachi Solutions

HITACHI

リバースエンジニアリング支援 および リファクタリングツール導入支援の提案

株式会社日立ソリューションズ
ＤＸソリューション本部 開発イノベーション部
品質保証ＤＸソリューショングループ

本支援の背景

HITACHI

社会課題１

■ システムの維持保守が困難

例）発電所は20年以上同じシステムを維持することも…

■ レガシーコードの習得者も減少

社会課題２

■クラウド型の生成AIを使いたいが、
技術情報の流出リスクや
国際的な輸出規制の制限

弊社提案

「リバースエンジニアリング支援環境」で、
旧システムの維持保守作業を支援

サービス骨子

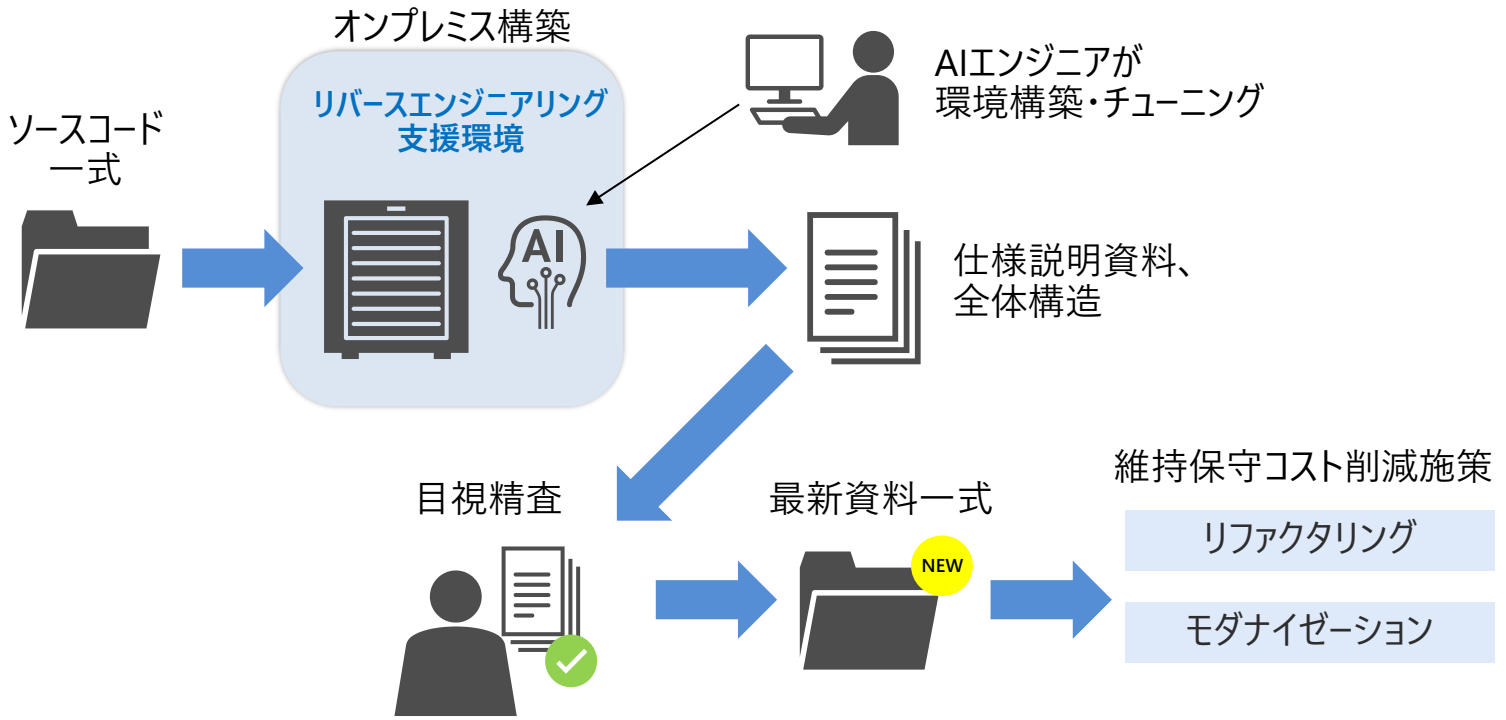
リバースエンジニアリング支援環境を構築し、各種作業を支援

お客様課題

プログラムが**文書化されておらず**、
リバースエンジニアリングに
時間を要する。

プログラム言語が古く、
技術者確保が困難。
（FORTRAN、Pascal、C言語など）
対応する解析ツールも少ない。

弊社提案

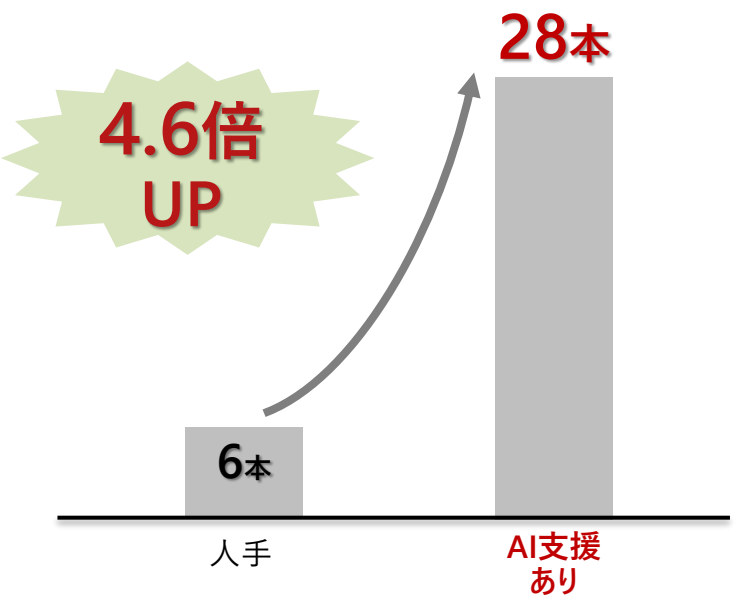


リバースエンジニアリング作業での適用効果（弊社試行PJでの実績から算出）

Case：ソースコードから仕様解説資料を作成

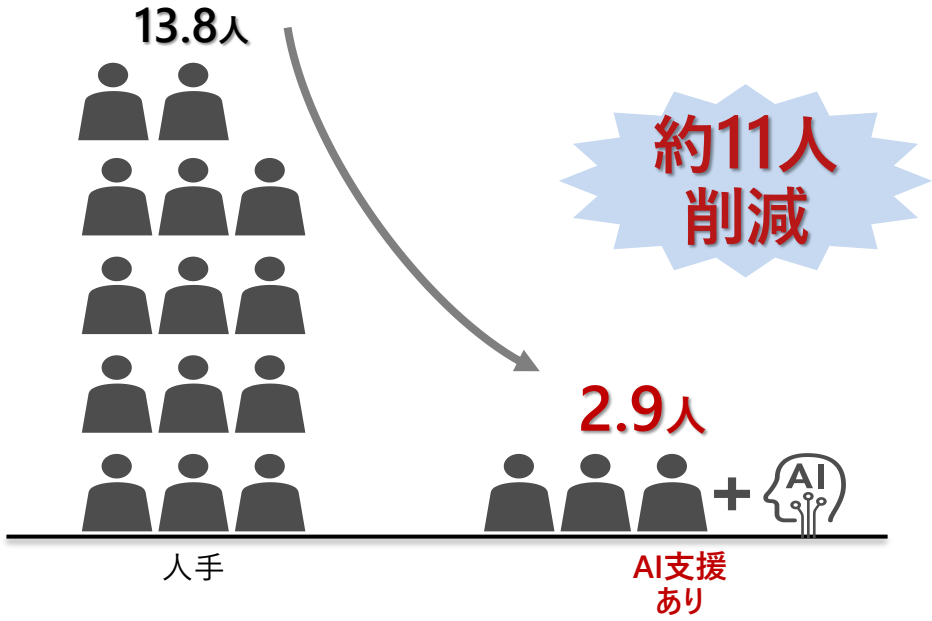
スピードアップ

※1名が1週間で書きおこせるプログラム本数
（平均：0.3Ks）



人員コスト削減

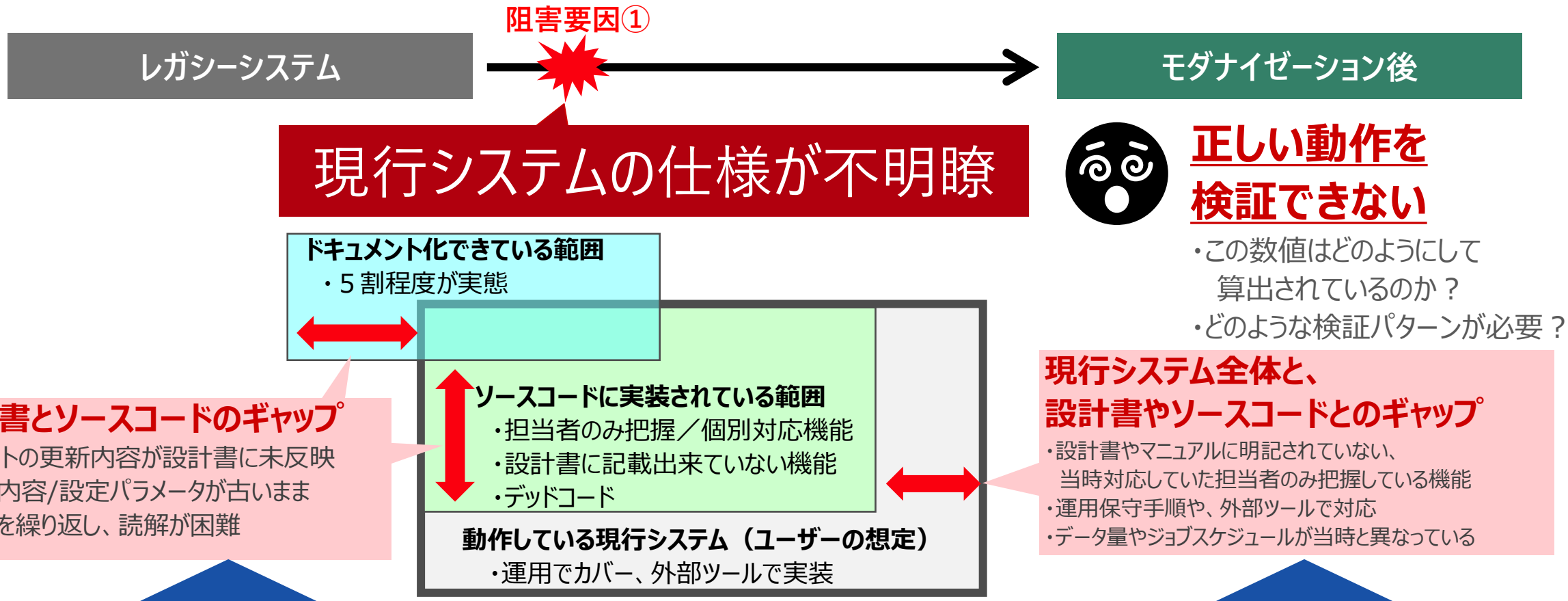
※300KSのソースコードを3ヶ月で
仕様整理するために必要な人数



※ソースコード内容や構造により解析結果に差異が生じます。

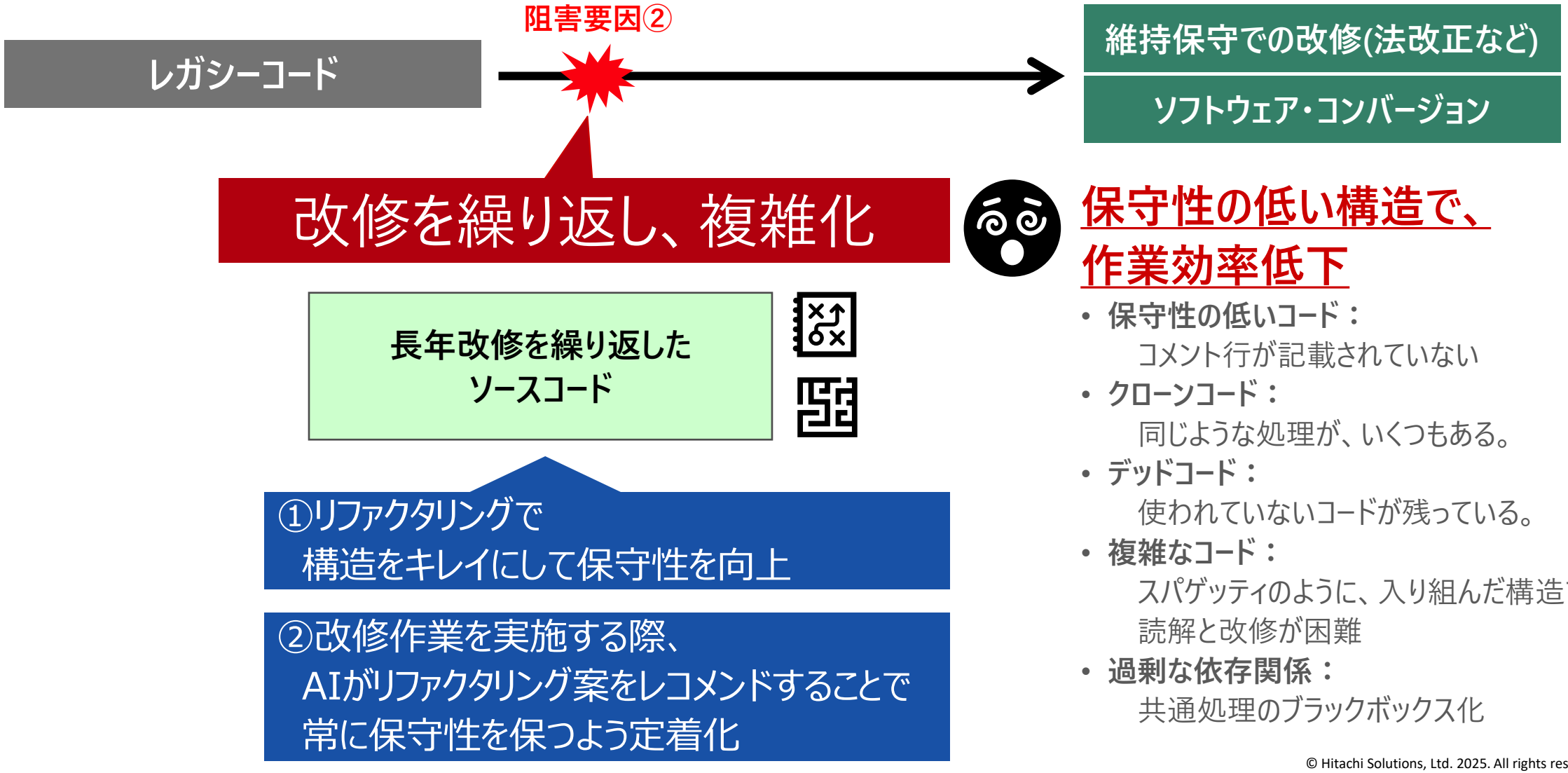
ユースケース１：よくある課題と、リバースエンジニアリングの必要性

HITACHI



ユースケース２ ：よくある課題と、リファクタリングの必要性

HITACHI



保守性の低い構造で、作業効率低下

- **保守性の低いコード：**
  コメント行が記載されていない
- **クローンコード：**
  同じような処理が、いくつもある。
- **デッドコード：**
  使われていないコードが残っている。
- **複雑なコード：**
  スパゲッティのように、入り組んだ構造で読解と改修が困難
- **過剰な依存関係：**
  共通処理のブラックボックス化

リファクタリングとは

HITACHI

リファクタリングとは、正常に動いているプログラムのソースコードを整理し、冗長なコードを削除したり、分かりにくいコードを分かりやすく変更すること

＜原則＞
・バグの修正とは異なり、**プログラムの動作は変えない**。
※変えてはならない

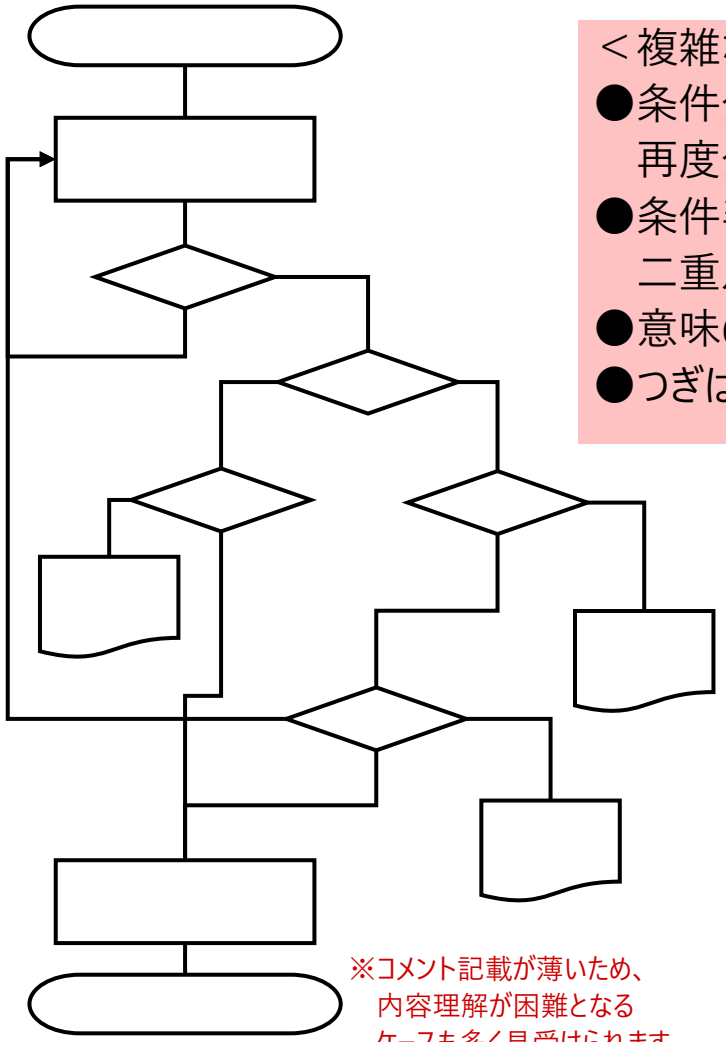
＜効果＞
・プログラムの内部構造を分かりやすくすることでの効果
・将来のコード変更時に**コード理解のスピードアップ**
・複雑度が低下し、新たな**バグを作り込みにくくなる**
・**メンテナンス性が向上する**

＜要点＞
・複雑な構造をもったプログラムを、**そもそも作らせない／残さない**
・後から改修するには、**大変労力を要する**
・変更前後で動作が変わっていないことの検証のため、**テストコードを必ず用意**する

参考：リファクタリング効果について

HITACHI

複雑怪奇なコード



＜複雑なコード＞
- ●条件分岐の先に、再度分岐がいくつもある
- ●条件判定での二重ループもある
- ●意味の通じないコメント
- ●つぎはぎだらけのロジック

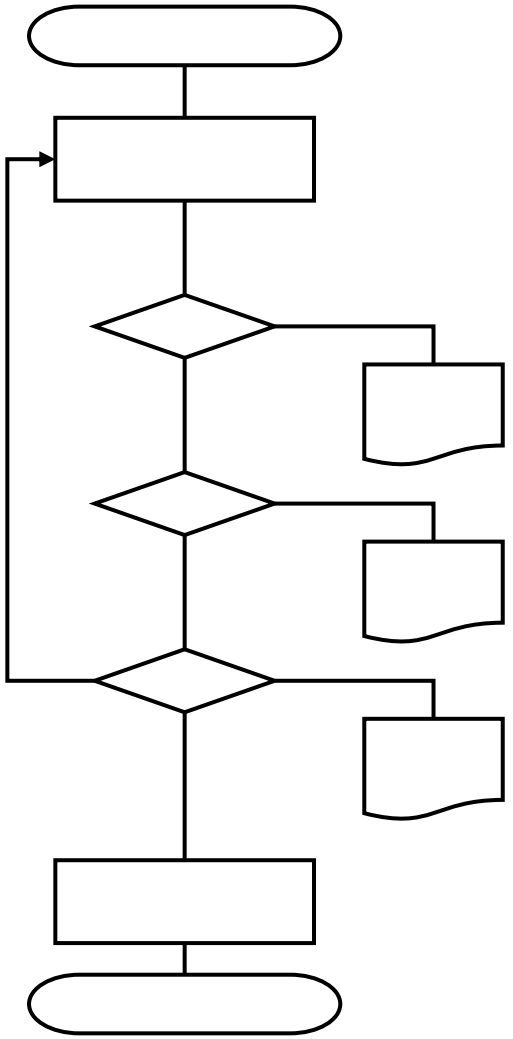
- 内容理解が困難
- テストも困難
- バグが残存
- 修正も困難

※コメント記載が薄いため、内容理解が困難となるケースも多く見受けられます。

リファクタリング後のコード

＜リファクタリング＞
- ●内部構造を見直し
- ●判定処理の最適化
- ●モジュールを分割/簡素化

- 内容理解が容易
- テストが容易
- バグが低減
- 修正も容易



参考：リファクタリングによる、生産性向上効果

HITACHI

1. 可読性の向上
コードが読みやすくなり、他の開発者が
理解しやすくなります。
命名の改善、関数の分割、コメントの
整理などが含まれます。

2. 保守性の向上
バグの修正や機能追加がしやすくなります。
テストのしやすさも向上します
（テストコードの追加も容易に）。

3.再利用性の向上
汎用的な関数やクラスに分離することで、
他のプロジェクトでも再利用可能になります。

4. パフォーマンスの改善（場合による）
無駄な処理や冗長なロジックを削除することで、
実行速度が向上することもあります。

5. バグの予防
複雑なロジックを単純化することで、
潜在的なバグの発生を防ぎやすくなります。

指標	リファクタリング前	リファクタリング後
コード行数（LoC）	10,000行	7,500行 (約20～25%削減)
関数の平均長	50行	20行 (複雑度30%削減)
重複コード率	15%	3%
単体テスト カバレッジ	60%	85% (品質向上効果あり)
バグ修正時間 （平均）	3日	1日

引用元：奈良先端科学技術大学院大学
論文「リファクタリングが
ソフトウェア品質に及ぼす影響の
実証的評価に関する研究」
https://naist.repo.nii.ac.jp/record/10787/files/R012296.pdf

リファクタリングを実施する上での課題と、対応事例

HITACHI

<table>
<tr><th>実施項目</th><th>課題</th><th>対応事例</th></tr>
<tr><td>全体構造理解</td><td rowspan="2">・ドキュメントが陳腐化し、
ソースコードから仕様理解が必要
・ドキュメントが残っていても、
結局はソースコードの
内容理解が必要。</td><td>【understand(※)を活用】
・ソースコード構造解析ツールで
全体像や複雑度の高いコードを把握</td></tr>
<tr><td>機能理解</td><td rowspan="3">【リバースエンジニアリング環境を活用】
・オンプレミスで動作する生成AIにより、
ソースコードから各種資料を自動生成
・機能説明資料
・シーケンス図、マーメイド図
・リファクタリング案
・unit testコードを自動生成
⇒ コード生成に特化したLLMを導入
※ソースコードをお預かりして、
解析結果をお渡しする対応も可
※ソースコードの内容により、
アウトプットが変わることもあります。</td></tr>
<tr><td>コード改修</td><td>・下記対応には大変労力を要する
・コード規約に沿った改修
・コメント行の追記／補正
・変数名を、目的が明確な名称へ変更
・関数の機能分割／整理
・冗長コードの排除
・類似コードの共通化
・複雑なネスト構造の最適化</td></tr>
<tr><td>修正後確認</td><td>・unit testのコードを作成する必要がある</td></tr>
<tr><td>作業内容記録</td><td>上記作業記録や資料作成に労力を要する</td><td>・ツールで生成することで、生産性向上</td></tr>
</table>

※「understand」は、テクマトリックス株式会社のソースコード構造解析ツールです。

リバースエンジニアリング支援環境での出力結果について

内容理解 （＝リバースエンジニアリング）

HITACHI

コード生成に特化したLLMにより、
コメントの無いコードからも処理内容や構造を解析して可視化

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <stdbool.h>

#define N 5

int graph[N][N] = {
    {0, 1, 1, 0, 0},
    {1, 0, 1, 1, 0},
    {1, 1, 0, 1, 1},
    {0, 1, 1, 0, 1},
    {0, 0, 1, 1, 0}
};

int path[N];
bool visited[N];

void *receive_data_A(void *arg) {
    system("curl -X 'POST' 'http://127.0.0.1:8000/download_file?api_key=11
    pthread_exit(NULL);
}

void *receive_data_B(void *arg) {
    system("curl -X 'POST' 'http://127.0.0.1:8000/download_file?api_key=11
    pthread_exit(NULL);
}

bool is_valid(int v, int pos) {
    if (graph[path[pos - 1]][v] == 0) return false;
    for (int i = 0; i < pos; i++) if (path[i] == v) return false;
    return true;
```

コメントの無い
複雑なプログラム

仕様説明資料

概要

このプログラムは、グラフのハミルトン閉路を探索するC言語の実装です。また、並行処理を用いてデータを受信する機能も含まれています。

定数とグローバル変数

- N: グラフの頂点数を定義します（ここでは5）。
- graph[N][N]: グラフの隣接行列を表します。
- path[N]: ハミルトン閉路を格納する配列。
- visited[N]: 頂点の訪問状態を管理する配列（未使用）。

関数

`void *receive_data_A(void *arg)`

- データを受信するためのスレッドAの処理。
- curlコマンドを使用して、指定されたURLからデータをPO

`void *receive_data_B(void *arg)`

- データを受信するためのスレッドBの処理。
- receive_data_Aと同様の処理を行います。

`bool is_valid(int v, int pos)`

- 頂点vがハミルトン閉路の次の位置posに追加可能かを判定
- 条件:
  - 頂点vが前の頂点と隣接していること。
  - 頂点vが既に閉路に含まれていないこと。

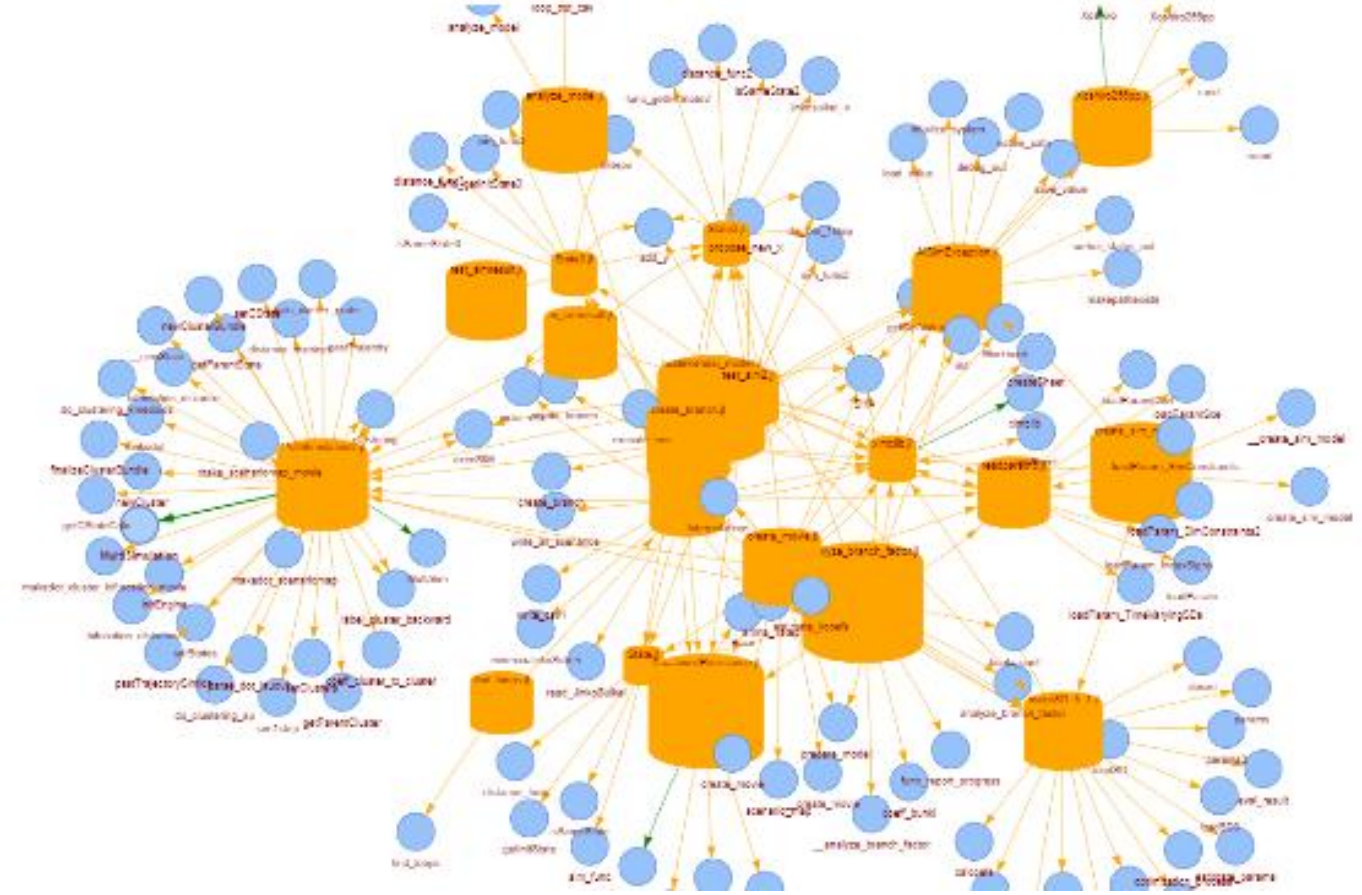
`bool hamiltonian_cycle_util(int pos)`

- 再帰的にハミルトン閉路を探索します。
- 基本ケース: 全ての頂点が閉路に含まれている場合、最初の頂点に戻れるかを確認します。

全体構造可視化（維持保守向け）

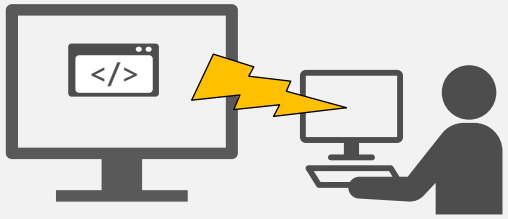
HITACHI

ソースコードからモジュール関連図を自動生成。
全体構造理解や、修正時の影響範囲を確認可能。



リバースエンジニアリング支援環境 提供形態

HITACHI

作業形態	概要	メリット	懸念点／課題
環境構築 および 活用方法 レクチャー	リバースエンジニアリング支援環境をお客様環境へ構築します。お客様自身でご利用頂けるよう支援します。 	・現場に密着することで、 より適切な結果を出すことが可能。 ・社内で随時、 いつでも何本でも解析が可能。	動作に必要な環境はお客様で用意して頂く必要があります。 リモートワークの場合は 専用端末または VPN接続アカウントを お借りすることがあります。
外部委託 (アウトソーシング)	プライベートクラウド経由で分析データを弊社へ提供頂き、 解析結果をオンラインで報告。 	基本事項をヒアリングさせて頂いた後、 お客様は解析したいデータを用意するのみ。 余計な手間やコストはかかりません。 ※分析ツールや動作環境は、 弊社で用意します。	解析本数に応じて、追加費用がかかります。 また、データをお預かりしてから結果を返すまでリードタイムを要します。

※ソースコード１本あたり、500行までが目安

よくある質問と回答

HITACHI

#	質問	回答	補足
1	対応する開発言語は？	下記に挙げます。 ・C、C+、C++、C#、Java、Python、COBOL、FORTRAN、 Visual Basic、PHP、PL/SQL	左記以外も 個別対応可能
2	どれぐらいの規模（行数）まで 解析可能？	開発言語や構造の複雑さによりますが、1000行までは解析可能です。	事前PoCで 見極め推奨
3	GitHub Copilotと、どう違う？	クラウドのAIサービスを扱わないため、下記利点があります。 ・輸出規制への配慮が不要 ・クラウド利用申請が不要 ・サービス利用量やトークンに応じた課金額が固定 また、一括処理が可能なため、現行システムリプレース前の現行調査作業の効率化が期待できます。	大量データを処理する際は、当支援が適しています。
4	３か月の短期間だけ、現行調査作業で ハードウェアを利用したい。	機器レンタルも対応可能です。 事前相談ください。	

リファクタリング支援について

リファクタリング作業での課題と、全体構成

HITACHI

リバースエンジニアリング環境内のAIで、各開発作業を支援

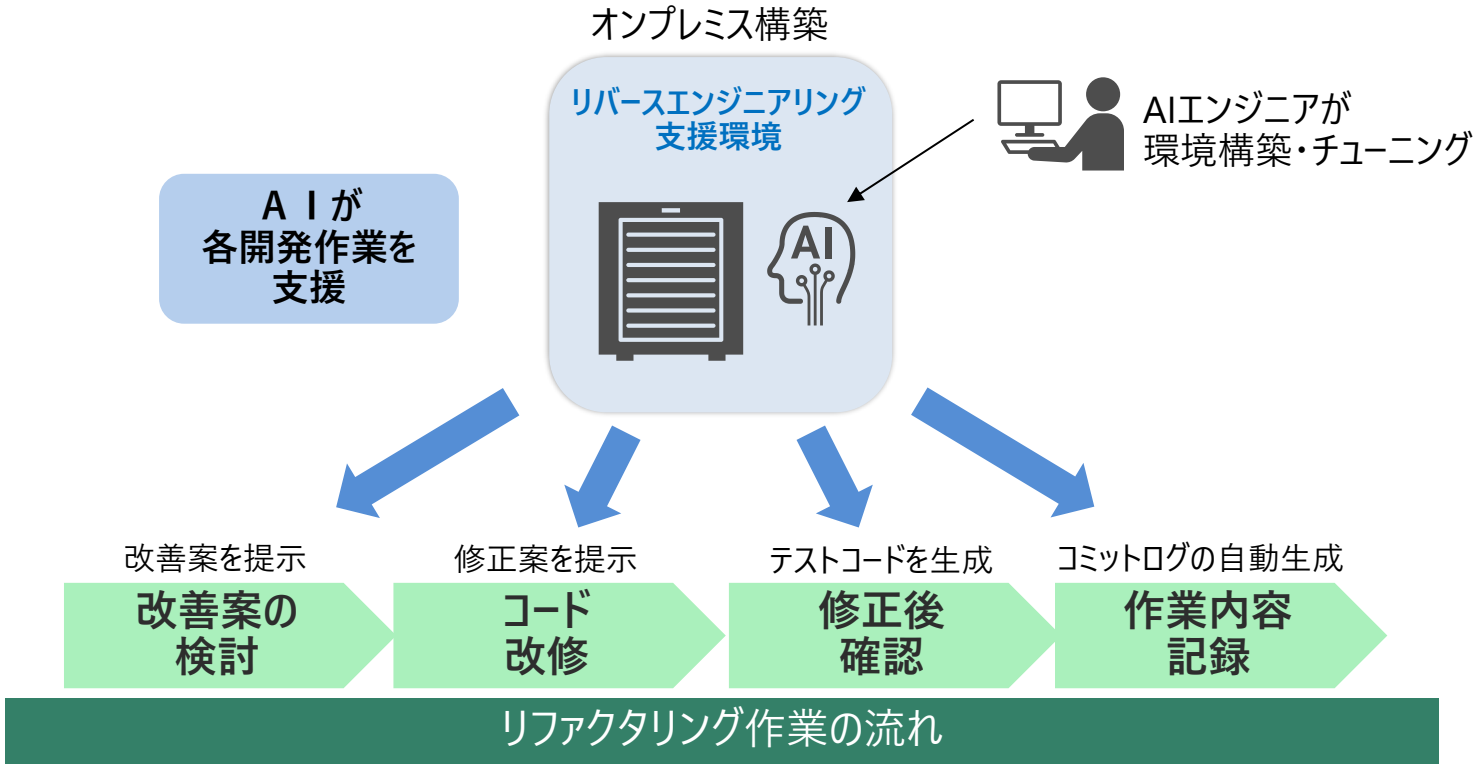
お客様課題

何を、どう修正すれば
良いのかわからない

修正前後の結果を検証するには
テストコードの作成が必要であるが
工数および時間を要する

作業者によって、
作業結果にバラつきがあり
コードレビューに時間を要する。

弊社提案



対応事例：リファクタリング

HITACHI

ソースコードの対象範囲を選択し、
リファクタリングをAIへ依頼すれば、改修箇所の提案が数秒で得られる。

リファクタリングのポイントは以下です。

- 関数化: [redacted]_request 関数を定義して、コードを再利用しやすくしました。
- 変数名の変更: 不要な変数や長い変数名を短縮しました。
- エラーハンドリングの統一: リトライ処理やエラー表示のロジックをまとめました。
- print文の削除: [redacted]_request 関数が返す値を、外部でprintするようにした。

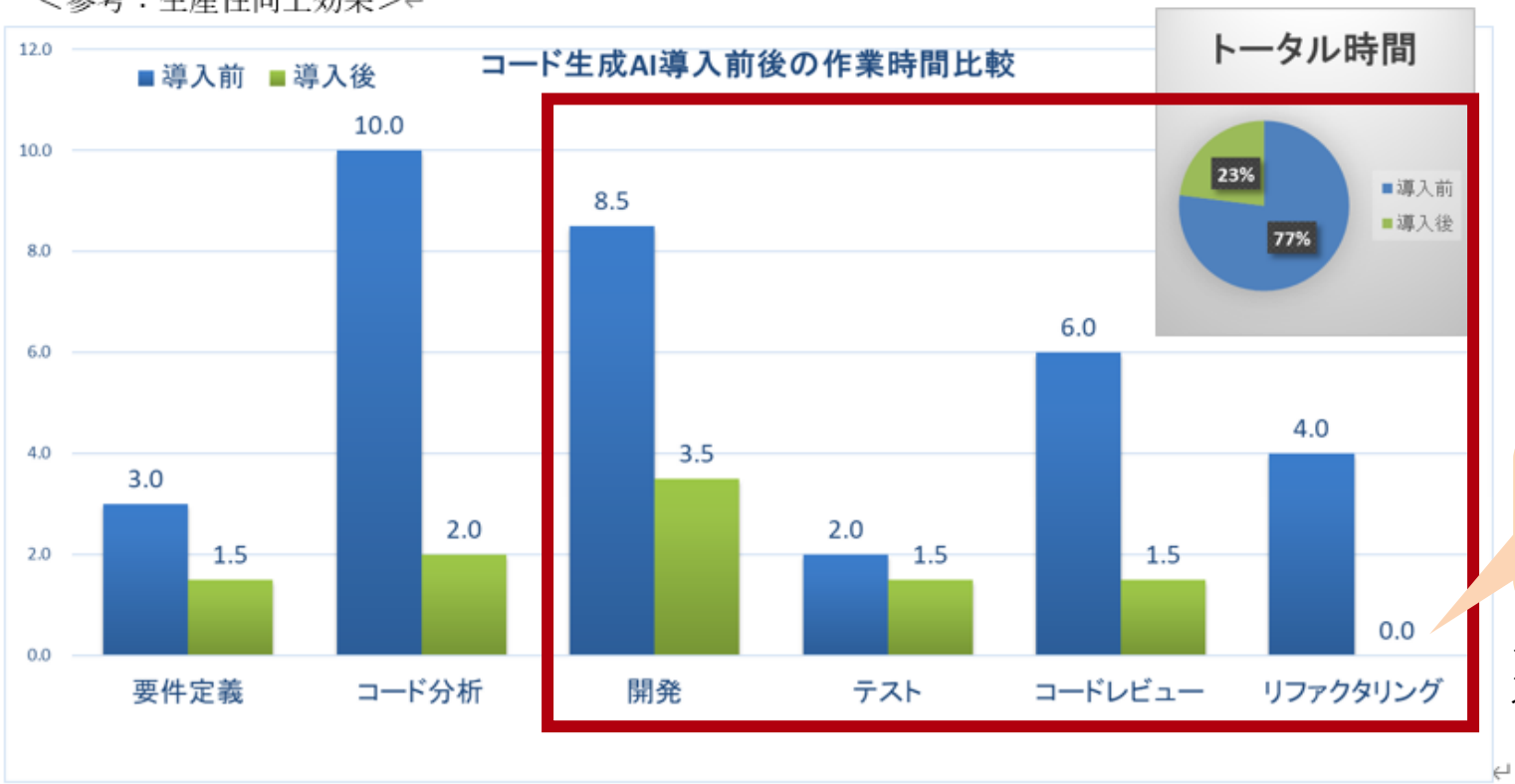
※リファクタリング案の採否は、人間が判断する必要があります。

適用効果

HITACHI

対象システムや分量により異なりますが、
下記生産性向上効果が一例として挙げられます。

＜参考：生産性向上効果＞



※日経コンピュータ 2024.3.21 日号(No.1116) P.21 から数値を引用

平素の開発時に
常にリファクタリングを行うため、
後からリファクタリングを行う作業は
不要となる。

※非効率的なコードを残さないことで、
今後の維持保守作業が容易となります。

Appendix

付録１：現場でよくある課題

HITACHI

下記課題への対策には、リバースエンジニアリング/リファクタリングが有効です

#	現場課題	概要
1	有識者／技術者の不在	**当時の技術者がおらず、ブラックボックス化。ドキュメントも整備されていない。**
2	ドキュメント不足	コードの変更履歴や**設計意図が十分に記録されておらず、新しいメンバーが参入しにくい**
3	コードの可読性の低下	複雑なコードやコメント不足により、**他の開発者が理解しにくい**
4	メンテナンスの困難さ	**コードがスパゲッティ化**しているため、バグ修正や機能追加が難しい
5	パフォーマンスの低下	**非効率なアルゴリズム**や冗長な処理が原因で、システムのパフォーマンスが悪化
6	テストの難しさ	テストが困難なコード構造や依存関係が多く、ユニットテストや自動テストの実施が難しい
7	技術的負債の蓄積	**一時的な解決策や回避策が積み重なり**、長期的な問題を引き起こす
8	依存関係の複雑化	**モジュール間の依存関係が複雑**で、変更が他の部分に影響を及ぼしやすい
9	コードの重複(クローン)	**同じ機能が複数の場所で実装**されており、修正時に一貫性を保つのが難しい

付録２：リバースエンジニアリングを進める上での課題と対応

HITACHI

課題	対応
知的財産権、契約上の制約 メーカー独自仕様や知的情報を 解析すること自体が法的違反となる。	**お客様資産のソースコードが対象** 知的財産権や、メーカーとの契約に 問題が生じないことをお客様で確認ください。
時間制限 メーカーの保守期限が迫っており、 タイムリミットを意識した計画が必要。	**当支援を活用** 人手での解析作業をAIが支援することで、 下記効果が期待できます。 ●所要期間の短縮 ●必要要員の削減 ●要員のスキル不足を補佐
コスト的な課題 大量のコードを解析するには工数がかかり、 かなりの費用を要する。（●千万円）	

※上記効果はソースコードの言語種別や構造によって変わるため、
事前に実用評価を頂くよう、検討をお願いします。

Hitachi Solutions

HITACHI

リバースエンジニアリング支援 および
リファクタリングツール導入支援の提案

株式会社日立ソリューションズ
ＤＸソリューション本部 開発イノベーション部
品質保証ＤＸソリューショングループ


HITACHI