Hitachi Solutions

HITACHI

社内サービス保守運用における作業時間短縮のための テスト自動化と生成AI活用実践

2025/10/22

株式会社日立ソリューションズ
技術革新本部
生産技術部
飯塚　陸斗

HITACHI

目次

HITACHI

1. 背景と改善の目的

1-1. 生産技術部のミッションと施策

1-2. 高速開発基盤サービス提供のための作業

1-3. 保守運用の課題と改善の目的

1-1. 生産技術部のミッションと施策

HITACHI

ミッション『社内の開発プロジェクトの開発効率化』

施策1 新たな開発技法のノウハウ整備

施策2 高速開発基盤サービスの提供

- ソースコード管理 GitLab
- CI/CD基盤 GitLab CI/CD
- タスク管理ツールなど

弊部署が用意した
インフラリソース上で社内提供

1-2. 高速開発基盤サービス提供のための作業

HITACHI

サービス拡充

方針検討 計画策定 導入

- 開発効率化のための新しいサービスの提供方針を検討する
- 本番環境への導入計画を策定し、導入する

運用保守

定期的なバージョンアップ

- システム安定運用のため、半年に1回、バージョンアップ作業(VUP)および動作確認のためのテストを実施する
- バージョンアップの際は、サービスの提供を1~2日停止する

運用保守

関連ドキュメント作成

- 利用促進のため、操作手順を説明するマニュアルを提供する
- バージョンアップによりサービスの画面UIが変更された際はマニュアルの記述をメンテナンスする必要がある

1-3. 保守運用の課題と改善の目的

必要な作業であるブラウザ操作を手作業で実施

課題

- 作業に時間がかかる
- 今後のサービス拡充により作業時間が増加する可能性がある

改善目的

- これらの作業にかかる時間を削減し、負荷を軽減
- サービスの可用性向上

2章　テストの改善 → 3章　マニュアルメンテナンスの改善

HITACHI

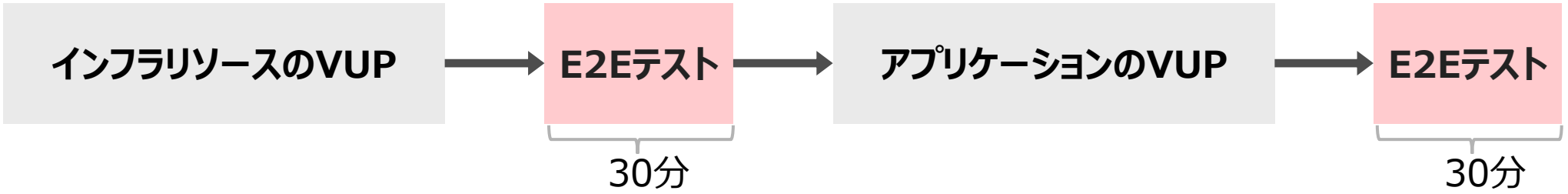
2. E2Eテスト自動化の実践

- 2-1. 運用における課題
- 2-2. 対策内容
  - 2-2-1. テスト作業の改善
  - 2-2-2. 導入のための実施事項
  - 2-2-3. 使用するツール・環境
  - 2-2-4. 導入における工夫点： テストコードの作成
- 2-3. 効果

2-1. 運用における課題

HITACHI

サービスバージョンアップにおけるテストの実施箇所



本発表におけるE2Eテストとは、システム全体を通して各機能が正常に動作するか確認するテストを指す。本サービスでは、アプリケーションの基本的な機能のテストを通してインフラリソースの疎通を確認できるため、インフラリソースとアプリケーションのVUP後に同じテストを実施している。

課題 **E2Eテストに時間がかかる**

要因分析①	要因分析②
テストに必要なブラウザ操作や結果の確認を毎回、手作業で実施している	サービスバージョンアップ時、またその準備・検証時において同じテストを複数回繰り返す

2-2. 対策内容

HITACHI

対策　E2Eテストツールを用いた自動化

目的　**サービス運用保守作業におけるテスト時間の短縮**

要因分析①

テストに必要なブラウザ操作や結果の確認を毎回、手作業で実施している

期待する効果

ブラウザ操作が自動化され、1回当たりのテスト時間が短縮

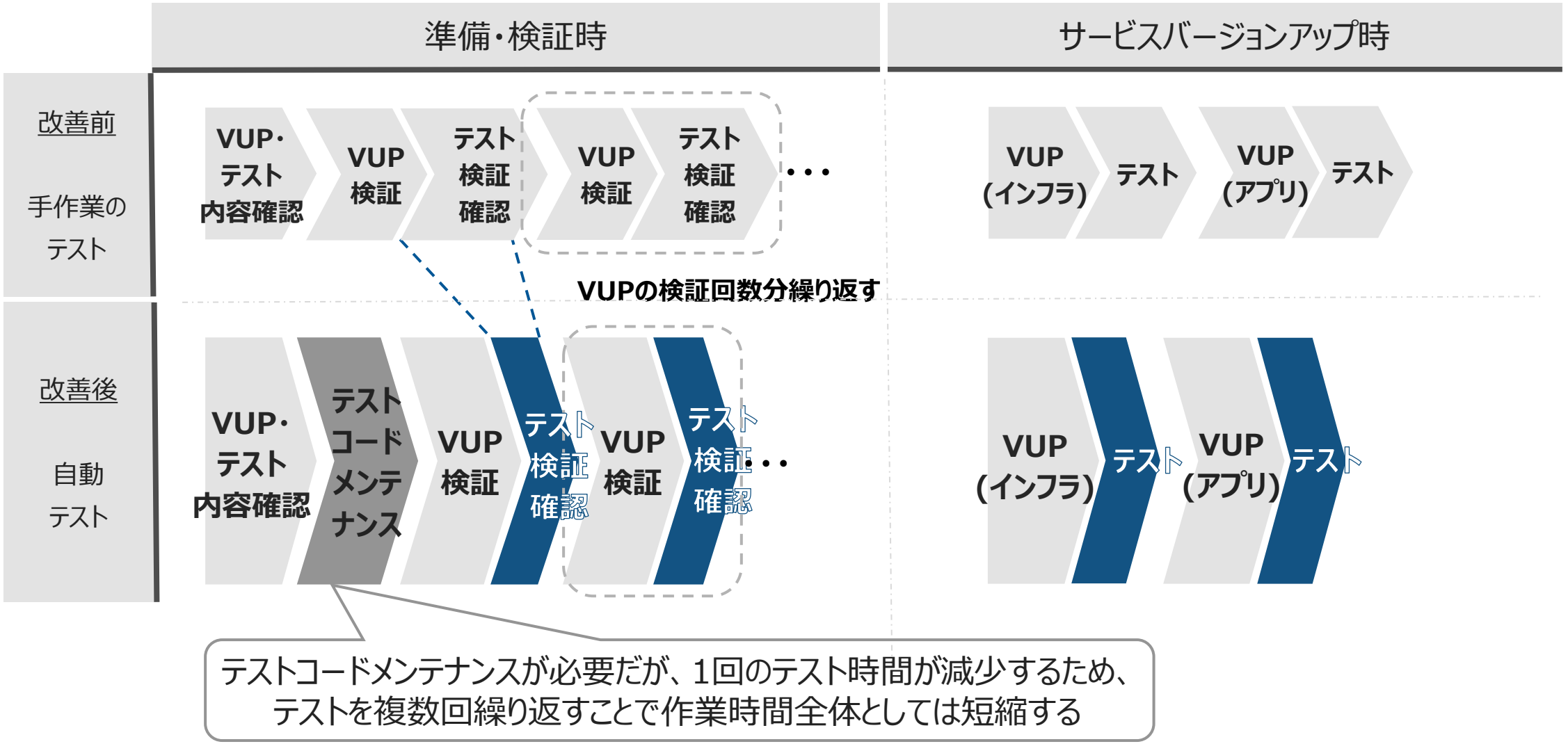
要因分析②

サービスバージョンアップ時、またその準備・検証時において同じテストを複数回繰り返す

期待する効果

複数回のテストが容易に実行可能

2-2-1. テスト作業の改善



2-2-2. 導入のための実施事項

HITACHI

テスト内容の明確化

GitLabの機能を確認するための
24個のテストケースに対し、
実施手順（テストシナリオ）
を明確化

CI/CDを実行できる

1. サンプルプロジェクトを開く
2. サイドバーのPipelinesの
   リンクをクリックする
3. パイプライン一覧が表示され
   ることを確認する
4. New Pipelineボタンをクリックする
5. ・・・

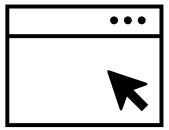
ツールの選択

E2Eテストツールを調査し、
非機能・機能要件を検証した結果、
Playwrightを選択

	Play wright	Cypress	mabl
提供元信頼	○	○	○
無償	○	○	×
採用率	○	△	-
コード生成機能	○	上記の非機能要件の比較結果に応じて項目を除外	
2FA対応	○		
マルチブラウザ	○		
スクリーンショット	○		

テストの作成

明確化した24個のテストケースに
対し、テストコード生成機能により
画面を操作してテストコードを作成



テストに合わせ
ブラウザ操作

テストコードが
生成される

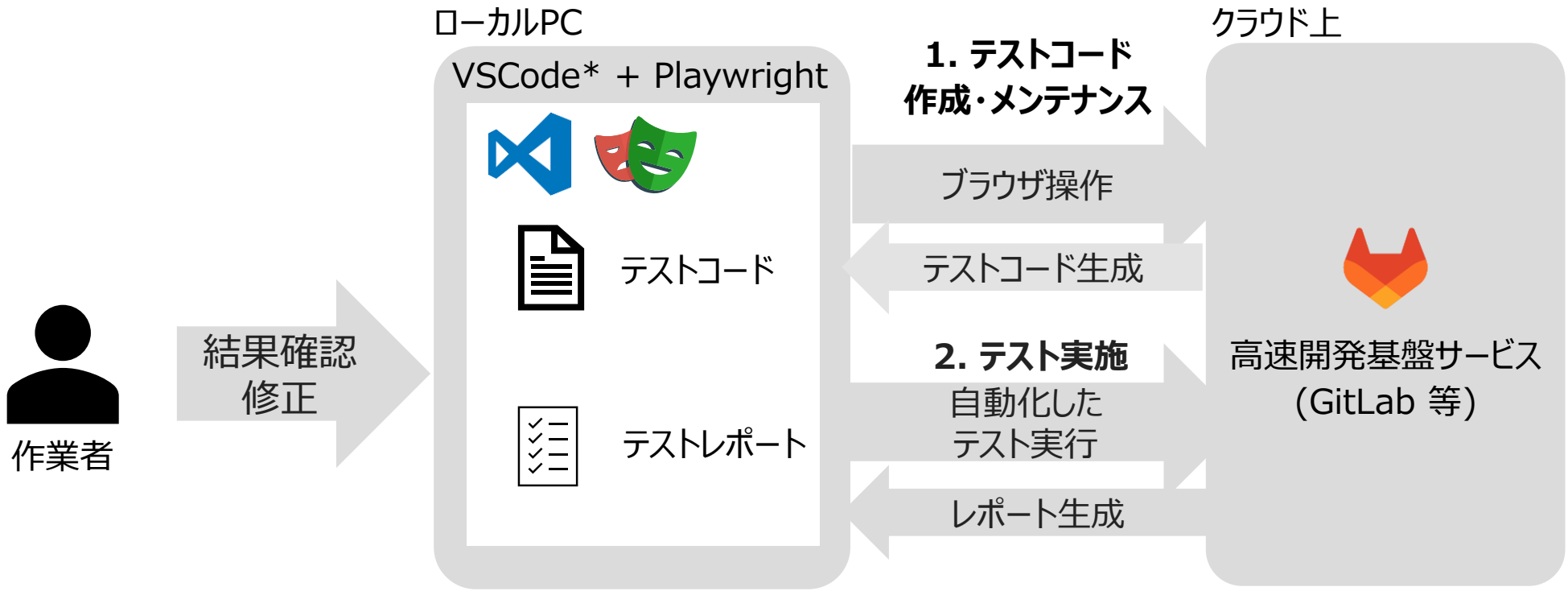
2-2-3. 使用するツール・環境

HITACHI

Playwright

特徴

- Microsoftが提供しているテスト自動化フレームワーク
- E2Eテストに必要な画面操作や画面要素の確認を自動化可能
- コード生成機能を用いて、ブラウザ操作からテストを生成可能
- 近年ツールの採用率が向上



* Visual Studio Code

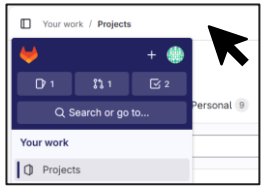
2-2-4. 導入における工夫点： テストコードの作成

HITACHI

発生した問題	Playwrightのコード生成機能のみでは、クリックやキーボード入力を伴わないテストの作成や特定のテスト設定を追加できず、テストが失敗
工夫点	部分的にテストコードをコーディングし、すべてのテストケースのテストを作成

① クリックやキーボード入力等の簡単な画面操作によるテストを作成

クリックやキーボード入力



テストコード生成

```
await expect(page
  .getByTestId(
    'XXXXXXXXX')
  .toBeVisible ({});
```

テスト実行
→失敗

② テスト失敗箇所であるコード生成機能によるテスト作成ができない操作をコーディング

要因：クリックを必要としない操作でコードが生成されない
対応：該当操作を実施するPlaywrightの関数により、操作を追加

```
await page.getByRole('XXX').hover();
```

要因：全テスト共通で待機の上限時間を設定すると一部のテストが失敗
対応：各テストもしくは操作ごとに待機時間を設定

```
await expect(page.getByRole('XXX')
  .toBeVisible ({timeout:15000});
```

要因：同じ名前のリンク・ボタンで競合し操作対象の要素を特定できない
対応：要素を特定するための条件を追加

```
await expect(page.getByRole('XXX')
  .getByRole('YYY').toBeVisible ({});
```

例　・階層：サイドバー配下のAという名前の要素　・他属性：ボタン要素かつAという名前の要素

2-3. 効果 ①サービスバージョンアップ時間の短縮

HITACHI

- **1回当たりのテスト実施時間の19分短縮**

  改善前　30分 ➡ 改善後　11分（約65%削減）

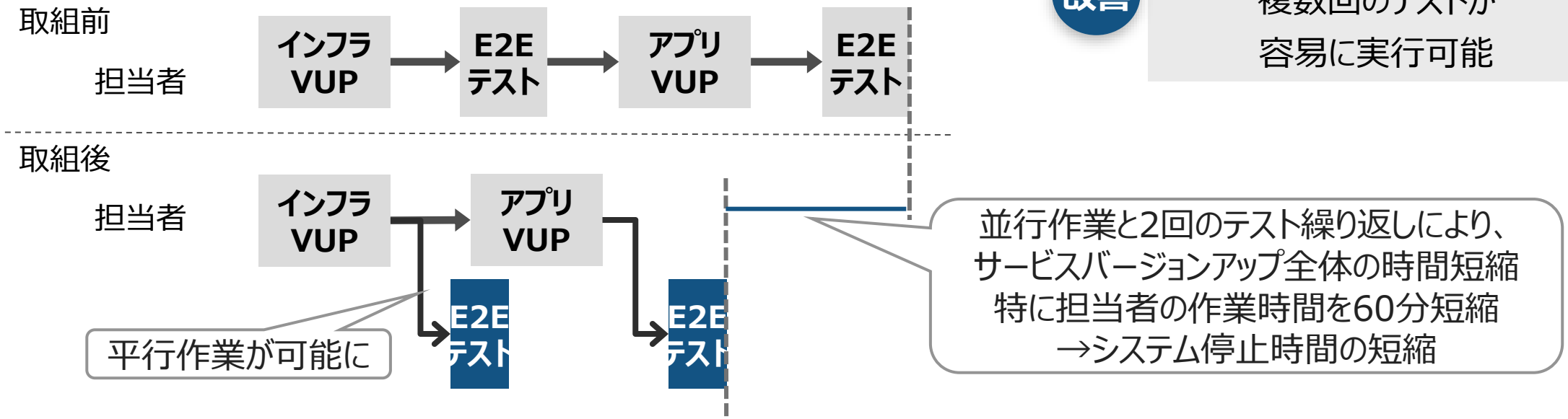
改善

期待する効果

ブラウザ操作が自動化され、
1回当たりのテスト時間が短縮

- **テストの属人性が排除され、テスト実施時の並行作業が可能に**

≫ **サービスバージョンアップに与えた効果**



改善

期待する効果

複数回のテストが
容易に実行可能

2-3. 効果 ②サービスバージョンアップ1回当たりの削減効果

HITACHI

サービスバージョンアップごとに繰り返す

初回導入	1回のVUP当たりの削減効果	…

実施した作業時間

- テスト内容明確化
- ツール検討
- テスト初期作成

+14.6時間

+14.6時間

改善前に対する作業時間

- テストコードメンテナンス　+2時間
- 準備・検証時の自動テスト実施　各4回　- 1.27時間（19分×4）
- サービスバージョンアップ時の自動テスト実施　- 1時間

合計 -0.27時間（16分）

複数回繰り返すことで回収 (現状、55回)

今後、テストケース数の増加やテストコードのメンテナンスの効率化により削減効果が増加する可能性がある

その他の効果

- 人によるテストの操作ミス削減によるテスト品質の向上
- 本施策を通して得たノウハウの横展開によるテスト自動化技術の適用拡大

HITACHI

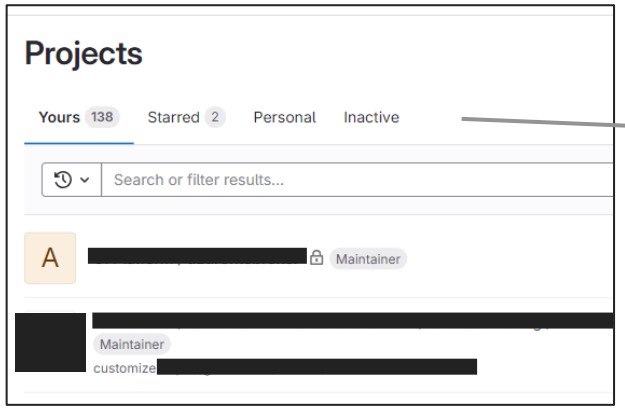
3. 生成AIを用いたマニュアルメンテナンスの実践

- 3-1. 運用における課題
- 3-2. 対策内容
  - 3-2-1. マニュアルメンテナンス作業の改善
  - 3-2-2. 使用するツール・環境
  - 3-2-3. 導入のための実施事項
  - 3-2-4. 導入における工夫点: プロンプトテンプレートの作成
- 3-3. 効果

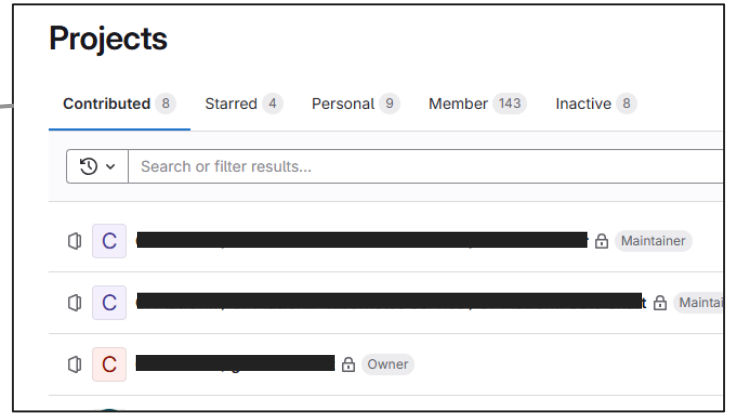
3-1. 運用における課題 - マニュアルメンテナンスの必要性と手順 -

HITACHI

マニュアルメンテナンスの必要性

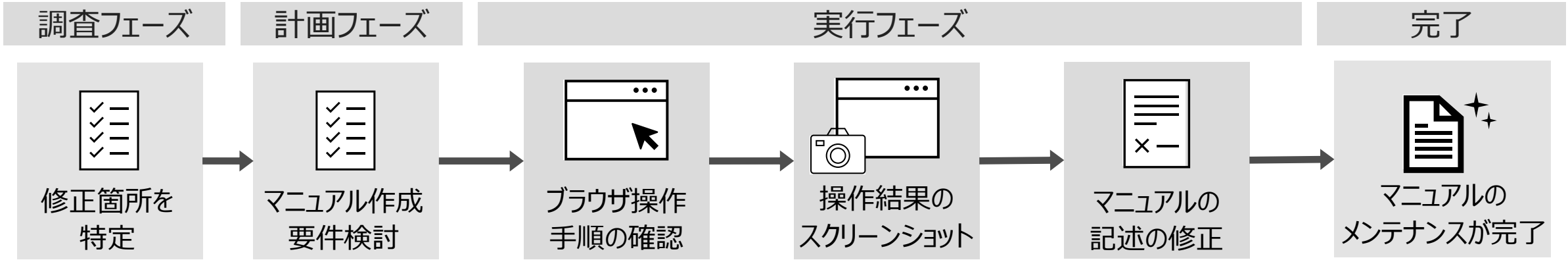


サービスバージョンアップに伴うUI部品の変更や
ページ遷移フローの変更



マニュアルの手順やスクリーンショットを
バージョンアップ後の画面に更新する必要がある

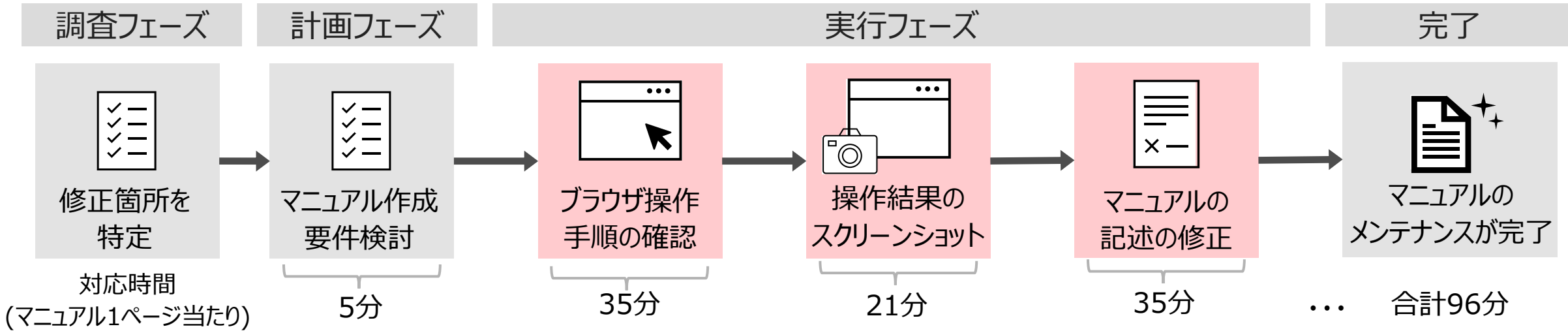
マニュアルメンテナンスの手順



3-1. 運用における課題 - 課題と要因分析 -

HITACHI

マニュアルメンテナンスの手順



課題 **マニュアルのメンテナンスに時間がかかる**

要因分析①	要因分析②
手順確認のためのブラウザ操作やスクリーンショットをメンテナンスの度に手作業で実施している	マニュアルの記述修正やスクリーンショット配置を手作業で実施している

3-2. 対策内容

HITACHI

対策 生成AIを用いてマニュアルメンテナンス作業を自動化

目的 サービス運用保守作業におけるマニュアルメンテナンス時間の短縮

要因分析①

手順確認のためのブラウザ操作やスクリーンショットを
メンテナンスの度に手作業で実施している

期待する効果

ブラウザ操作やスクリーンショットが自動化され、
かかる時間が減少する

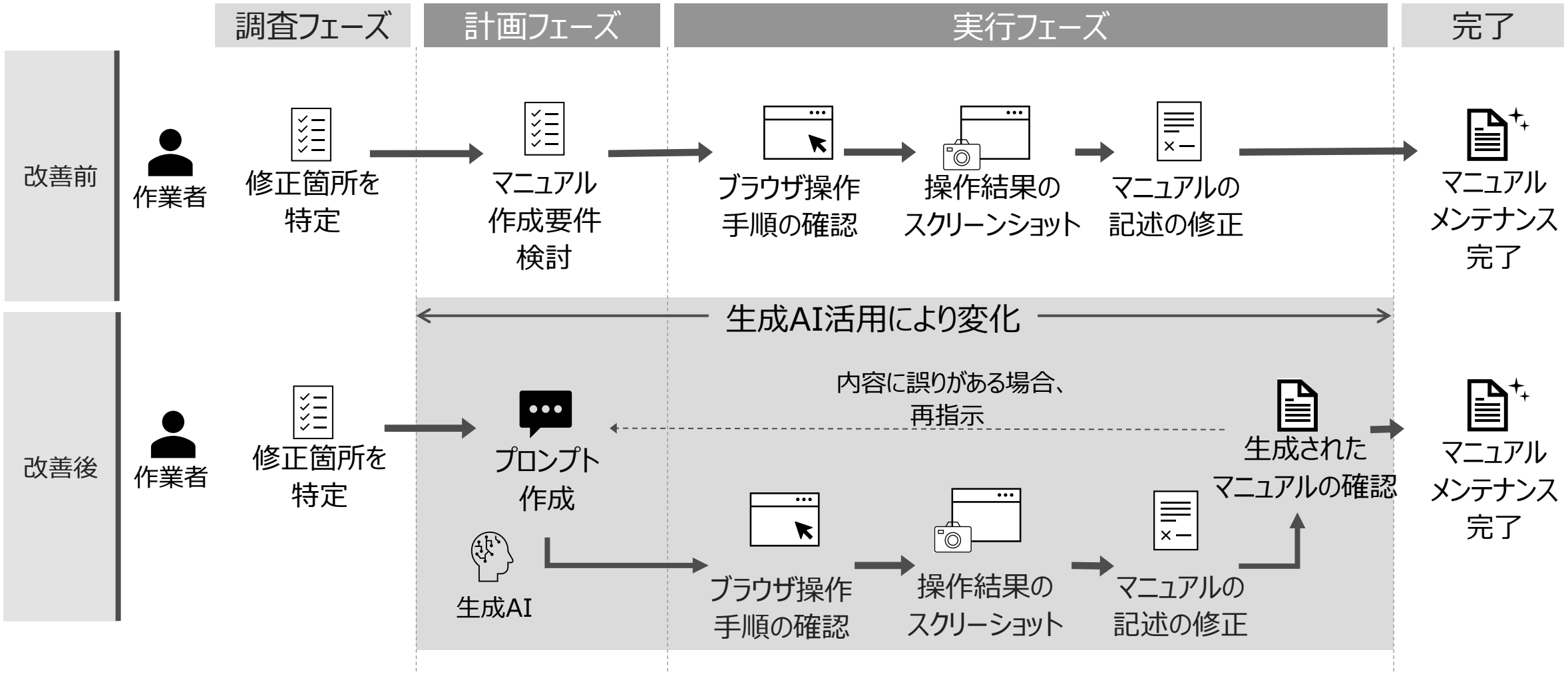
要因分析②

マニュアルの記述修正やスクリーンショット配置を
手作業で実施している

期待する効果

記述作成等が自動化され、
作業は生成AIによる出力を確認するのみ

3-2-1. マニュアルメンテナンス作業の改善



3-2-2. 使用するツール・環境

HITACHI

- マニュアル記述修正　自動化
- ブラウザ操作指示

Cline

特徴
- AIエージェント型の開発支援ツール
- AIによるコードの生成・修正・エラー解析などを実行できる

- ブラウザ操作 自動化

Playwright MCP Server

特徴
- MicrosoftがPlaywrightのMCP Server
- 生成AIを通してPlaywrightを呼び出し、ブラウザ操作、ブラウザ情報取得、スクリーンショット等ができる
- 今後のテストコード作成等へのノウハウ再利用を考慮しツールを選択

マニュアルメンテナンス作業と各種ツールの関係

3-2-3. 導入のための実施事項

HITACHI

環境構築

ローカル環境でPlaywright MCPが動作するための環境構築

プロンプトテンプレート作成 検証

今後のマニュアルメンテナンスを見据え、生成AIを利用した新規マニュアルの作成を通しプロンプトテンプレートを作成

骨子作成

まずは骨子作成として以下を実施

1. マニュアルに記述する機能の調査
2. 機能一覧からマニュアルの骨子を生成を用いて作成

内容詳細化

各章に対し、Playwright MCPを活用して手順を確認し、内容を詳細化

ポイント　ドキュメント体裁統一のため、生成条件を含めたプロンプトを作成

条件　（一部抜粋）
- マニュアルに書かれている内容・情報は必ず新マニュアルにも含めてください。
- 個人を特定しうる情報は以下のようなダミーの情報と差し替えてください
  - ユーザー名：xxxxx.yyyyy
- 1文の長さが60字以内
- 文末表現のスタイルが敬体（です・ます調）
- 句読点として、【、】【。】を用いる

テンプレート作成

マニュアル作成を通して得たノウハウを元に受け入れ率の高い出力となるプロンプトのテンプレートを作成
(3-2-4で説明)

3-2-4. 導入における工夫点: プロンプトテンプレートの作成

HITACHI

発生した問題	・プロンプトの作成に時間がかかる　・生成AIからの回答の精度にばらつきがある
工夫点	生成AIに与えるプロンプトのテンプレートを作成し、効率化および精度向上

マニュアルの章ごとにプロンプトを与える

[〇章]の[○○する手順]を現状のバージョンをブラウザ操作し、正しく書き換えてください。[最初の手順]から[完了を確認する手順]までの工程を書き換えてください。

（その他、指示が必要な際は追加する）
例：手順として複数の方法があり、両方含めたい時）
マニュアルには[1つ目の方法]の場合と[2つ目の方法]の場合を含めてください。
例：手順に加えて画面内の情報などの詳細に説明が必要な場合
入力フォームについてはそれぞれ何を入力するのか解説してください。

すべてのプロンプトに共通する事項はルール定義ファイルに格納して一括管理する

前提事項
- 本ディレクトリは～～のマニュアルです。マニュアルは作成時点の～
ワークフロー
- ブラウザ操作の際は～した後、毎画面スクリーンショットを取得してください
使用する機能
- ブラウザ操作の際はPlaywright MCPを用いてください
条件
-文末表現のスタイルが敬体（です・ます調）
【その他、## ファイル命名規則、リンク集、 ディレクトリ構造 等】

3-2-4. 導入における工夫点: プロンプトテンプレートの作成

HITACHI

発生した問題	・プロンプトの作成に時間がかかる　・生成AIからの回答の精度にばらつきがある
工夫点	生成AIに与えるプロンプトのテンプレートを作成し、効率化および精度向上

マニュアルの章ごとにプロンプトを与える

[〇章]の[○○する手順]を現状のバージョンをブラウザ操作し、正しく書き換えてください。[最初の手順]から[完了を確認する手順]までの工程を書き換えてください

（その他、指示が必要な際は追加する）
例：手順として複数の方法があり、両方含めたい時）
マニュアルには[1つ目の方法]の場合と[2つ目の方法]の場合を含めてください。
例：手順に加えて画面内の情報などの詳細に説明が必要な場合
入力項目についてはそれぞれ何を入力するのか解説してください。

マニュアルの一括更新では、
プロンプトが複雑となり、
提案に対する修正も増加する

章ごとに特筆すべき点があれば追加指示

すべてのプロンプトに共通する事項はルール定義ファイルに格納して一括管理

前提事項
- 本ディレクトリは～～のマニュアル。マニュアルは作成時点の～
ワークフロー
ブラウザ操作の際は～～した後、毎画面スクリーンショットを取得してください
使用する機能
- ブラウザ操作の際はPlaywright MCPを用いてください
条件
-文末表現のスタイルが敬体（です・ます調）
【その他、## ファイル命名規則、リンク集、 ディレクトリ構造 等】

生成AI標準のブラウザ操作機能を使用せずPlaywright MCPの使用を明確に指示

文章表記の統一を指示

3-3. 効果　①メンテナンス時間の短縮

HITACHI

マニュアルメンテナンス時間 45分の短縮

高速開発基盤サービスのマニュアル PDF7ページ

- 修正対象総行数 41行
- スクリーンショット総数12枚

メンテナンス時間	計画フェーズ	実行フェーズ				合計
		ブラウザ操作	スクリーンショット	記述の修正	生成AI結果確認	
改善前	5分	35分	21分	35分		96分
改善後	12分	7分	4分	7分	21分	51分

約45%削減

改善

期待する効果

ブラウザ操作やスクリーンショットが自動化され、かかる時間が減少する

改善

期待する効果

記述作成等が自動化され、生成AIによる出力を確認するのみ

3-3. 効果　②生成AIの受け入れ率

HITACHI

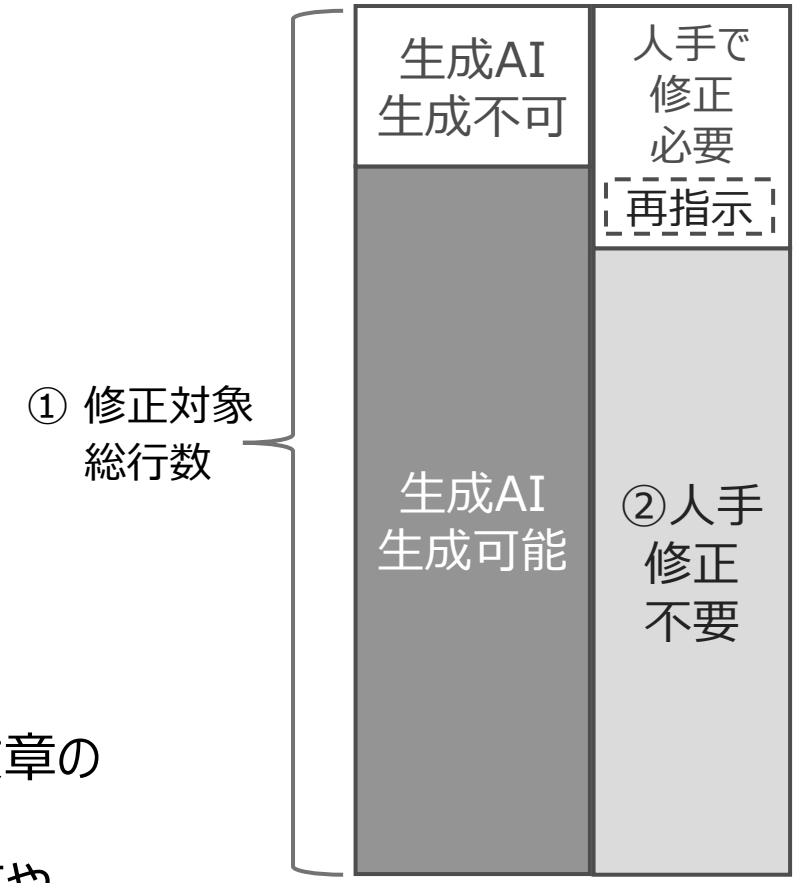
生成AIによる回答の受け入れ率

高速開発基盤サービスのマニュアル PDF7ページ

- 修正対象総行数 41行（①）
- スクリーンショット総数12枚

$$\text{受け入れ率} = \frac{\text{②人手での修正不要な行数}}{\text{①修正対象総行数}} = \frac{36\text{行}}{41\text{行}}$$

$$= \textbf{約90\%}$$



その他の効果

- ドキュメント生成条件における体裁の指定により、文章の表記ミスが削減しドキュメントの品質向上
- 本施策を通して得たノウハウの横展開による生成AIやPlaywright MCP技術の適用拡大

4. まとめと今後の展望

4-1. まとめ

4-2. 今後の展望

4-1. まとめ

HITACHI

運用保守 定期的なバージョンアップ

課題 テストに時間がかかる

運用保守 関連ドキュメント作成

課題 マニュアルメンテナンスに時間がかかる

改善目的

- **これらの作業にかかる時間を削減し、負荷を軽減**
- **サービスの可用性向上**

結果 作業時間 61分削減（テスト作業16分・マニュアルメンテナンス作業45分）

テスト自動化

E2Eテストツールを用いて自動化

- 1回当たりのテスト実施時間の短縮
  - メンテナンス時間の減少やテストケースの増加等で削減効果を増やす必要
- 属人化の解消や人的ミスの削減
- システム停止時間の短縮
- テスト自動化技術の適用拡大

マニュアルメンテナンス

生成AIを用いてマニュアルメンテナンス作業自動化

- ブラウザ操作
- スクリーンショット
- 修正後の記述作成

（自動化）

- マニュアルメンテナンス時間の短縮
- VUP後のUIにおけるマニュアル提供時間の短縮
- 生成AIやPlaywright MCP技術の適用拡大

4-2. 今後の展望

HITACHI

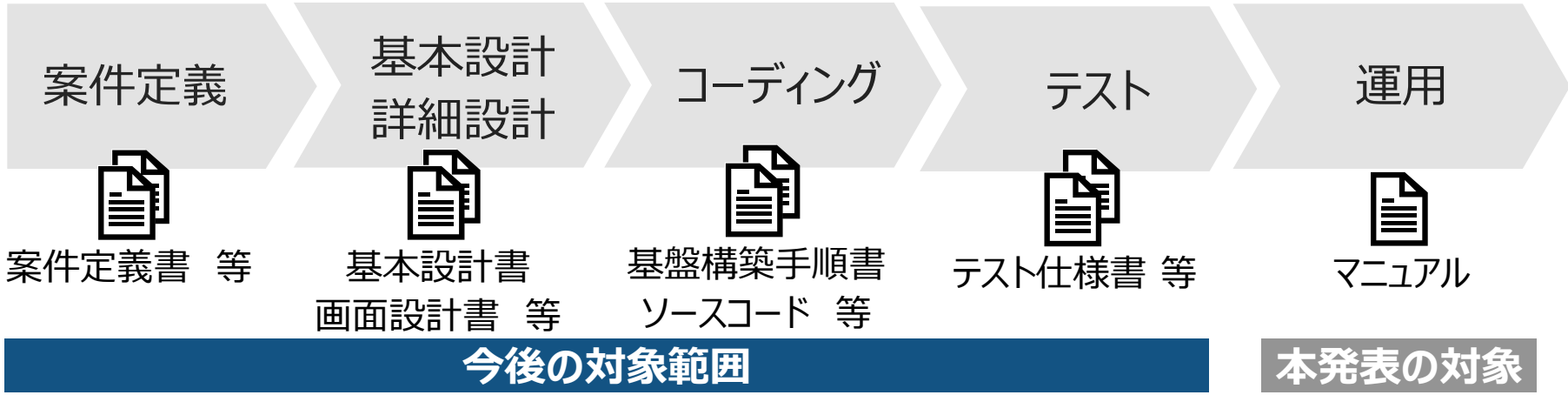
テスト自動化

- テストコードの初回作成およびメンテナンス作業の効率化

現状	今後
•コード生成機能とコーディングでテストコード作成 •テストケース数が多い場合、時間がかかる	Playwright MCPを用いて自然言語（テストケース）から一括生成する手法の検討

マニュアルメンテナンス

- 外部ドキュメント読み込みによるドキュメント生成精度向上
- 各開発工程におけるドキュメント作成の効率化



Hitachi Solutions

HITACHI

- GitLab は、GitLab B.V. の米国およびその他の国における商標または登録商標です。
- Clineは、Cline Bot Inc. の米国およびその他の国における商標または登録商標です。
- Microsoft（Azure、Playwright 、Visual Studio 、VS Code）は、マイクロソフトのグループ企業の、米国およびその他の国における商標または登録商標です。
- その他記載の会社名、製品名は、それぞれの会社の商号、商標もしくは登録商標です。

社内サービス保守運用における作業時間短縮のための テスト自動化と生成AI活用実践

2025/10/22

株式会社日立ソリューションズ
技術革新本部
生産技術部
飯塚　陸斗


HITACHI