

ソフトウェアテストシンポジウム2026北海道

生成AIに開発プロセスの全成果物を作らせるには ～コンテキスト設計で出力品質のばらつきを抑えた実践事例～

2026年7月24日

株式会社NTTデータ
重道大河・川上蒼太

自己紹介



重道大河

株式会社NTTデータ
テクノロジーセグメント AI事業本部 AIテクノロジー事業部

経験・専門分野

- 2024年入社
- 入社以来、ソフトウェア開発および生成AI活用領域に従事
- 生成AIを活用した開発プロセスの改善や生産性向上に向けた取り組みを推進

自己紹介



川上蒼太

株式会社NTTデータ
テクノロジーセグメント AI事業本部 AIテクノロジー事業部

経験・専門分野

- 2021年入社
- 2021～2023：社内のプロジェクト管理の改善支援に従事
- 2023～：生成AIを活用したシステム開発の効率化のR&Dや支援に従事
- 開発/管理プロセスの改善に向けた取り組みに携わってきました

背景・目的



所属する部署

生成AIを活用したシステム開発を全社的に分野を横断して推し進める部署である、AI事業本部に所属している。

対外公表組織機構図（2026年7月1日付）



所属部署から見える、プロジェクトごとの生成AI活用のばらつき

生成AIの活用は進んでいる一方、生成AI活用スキルの適用範囲・効果はプロジェクト間で大きなばらつきがあり、十分な効果を得られていないプロジェクトが多数を占めている。

全社のプロジェクトでの生成AI活用の実態（イメージ）

効果が大きいプロジェクト

一部にとどまる



- ✓ 生成AI活用スキルを持つ人材がプロジェクトにいる
- ✓ 適用範囲が**広く**、開発プロセス全体で**活用できる**
- ✓ 成果物の品質・生産性が**大きく向上**

十分な効果を得られていないプロジェクト

多数を占める

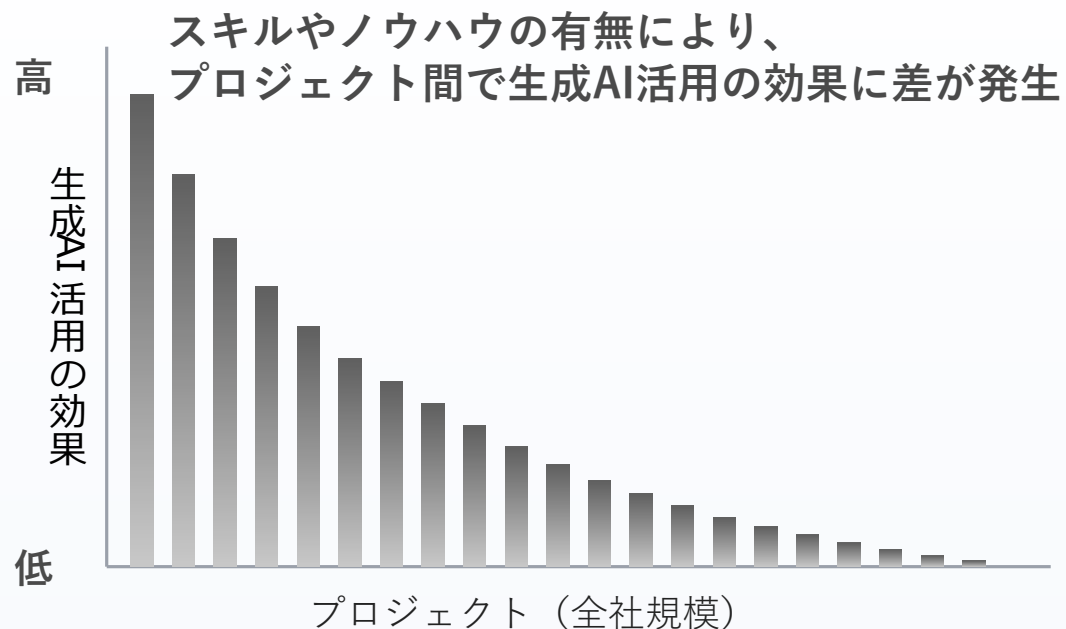


- × 生成AI活用スキルを持つ人材がプロジェクトに**いない／少ない**
- × 使い方が**個人の知識・経験に依存**
- × 十分な効果が出ず、**品質向上・生産性向上につながらない**

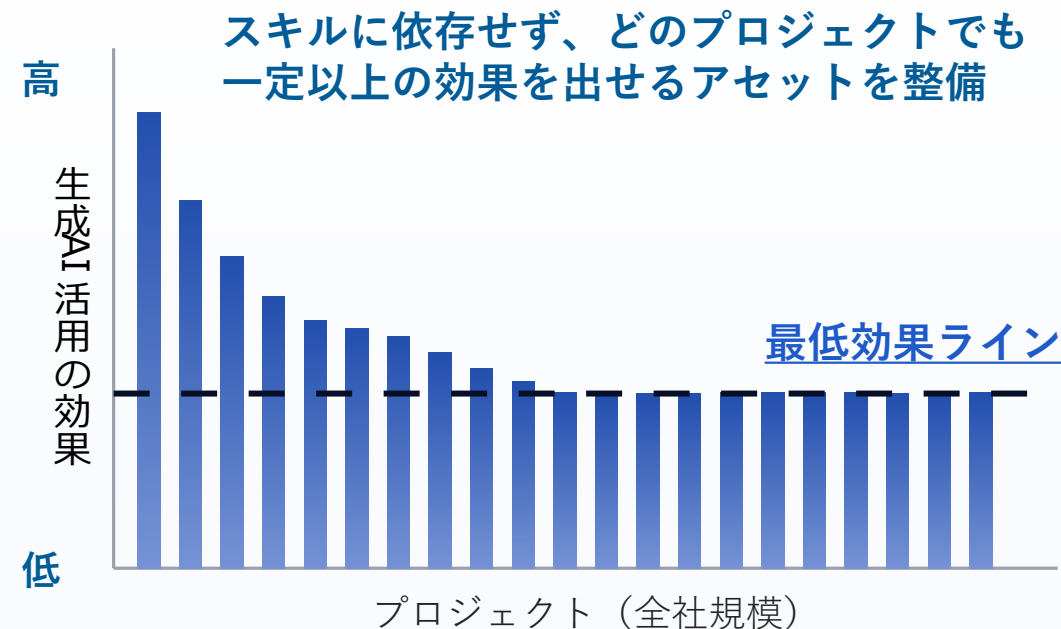
ばらつきを解消するためのアセットの整備

プロジェクトごとの生成AI活用スキルや適用範囲の違いに対応するため、誰でも開発プロセスの全工程で効果を発揮できるアセット※を整備した。

課題：スキルのばらつきが大きい



解決策：アセットの整備



※アセット：業務やシステム開発で再利用できる、ツール・テンプレート・ノウハウなどの資源

今回作成するアセットに求める条件

今回作成するアセットでは、「成果物が社内標準に準拠していること」、「誰が使っても一定の出力品質であること」、「誰でも使いやすいこと」の3点を満たすべき条件として定めた。

1. 成果物が社内標準に準拠していること

- ✓ 多くのPJの多くの工程で効果を発揮するため

2. 誰が使っても一定の出力品質であること

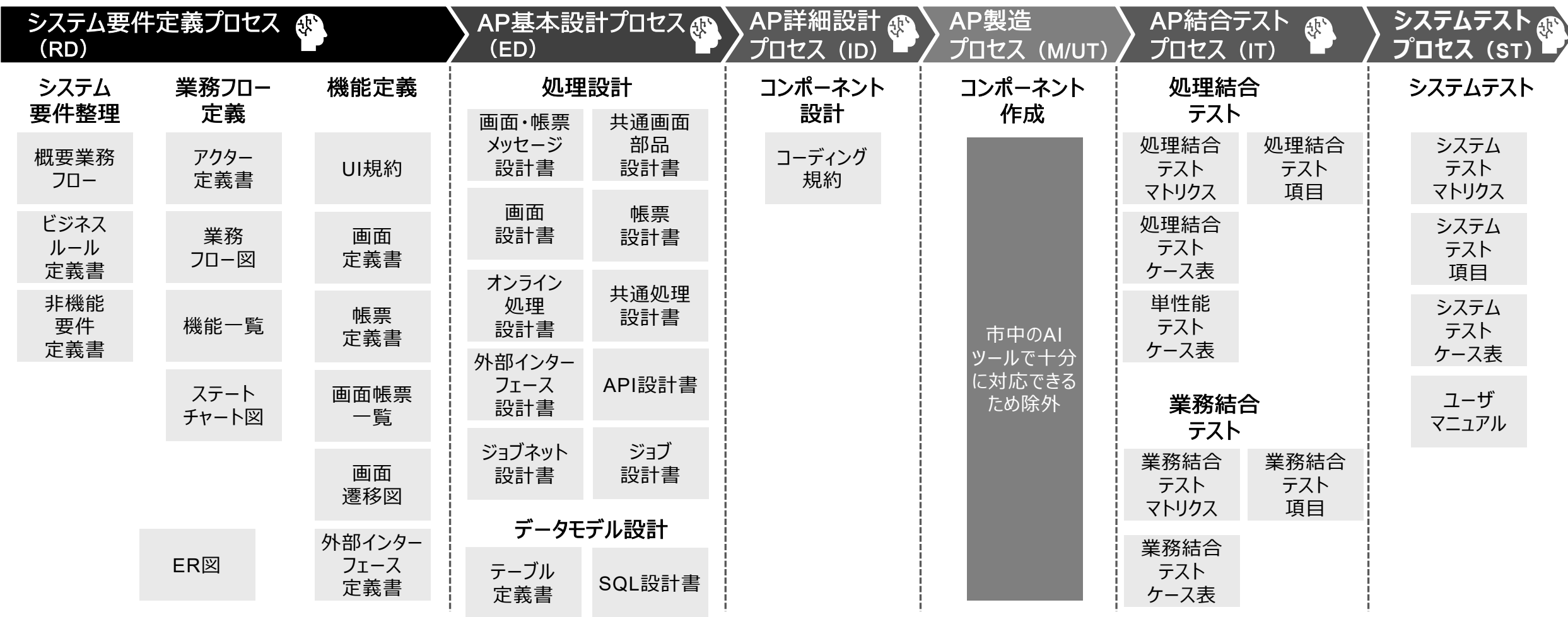
- ✓ スキルによらず、一定の効果を出すため

3. 誰でも使いやすいこと

- ✓ 多くのユーザに抵抗なく使ってもらうため

初期リリースの対象成果物スコープ

社内標準で定められている130種以上の成果物のうち、
まずは、小規模開発向けに定められている必要最低限の全39種を対象成果物とした。

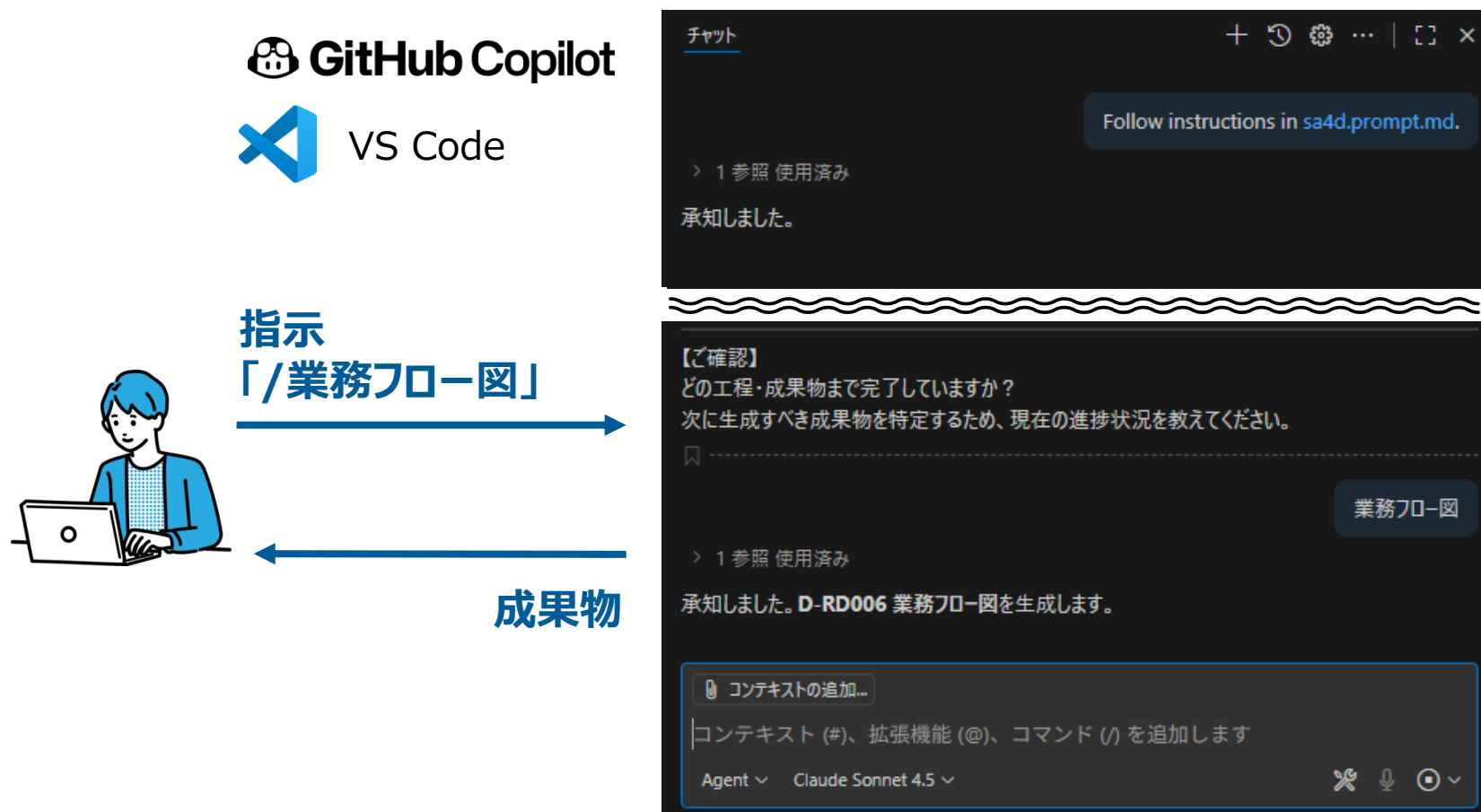


アセットの構築



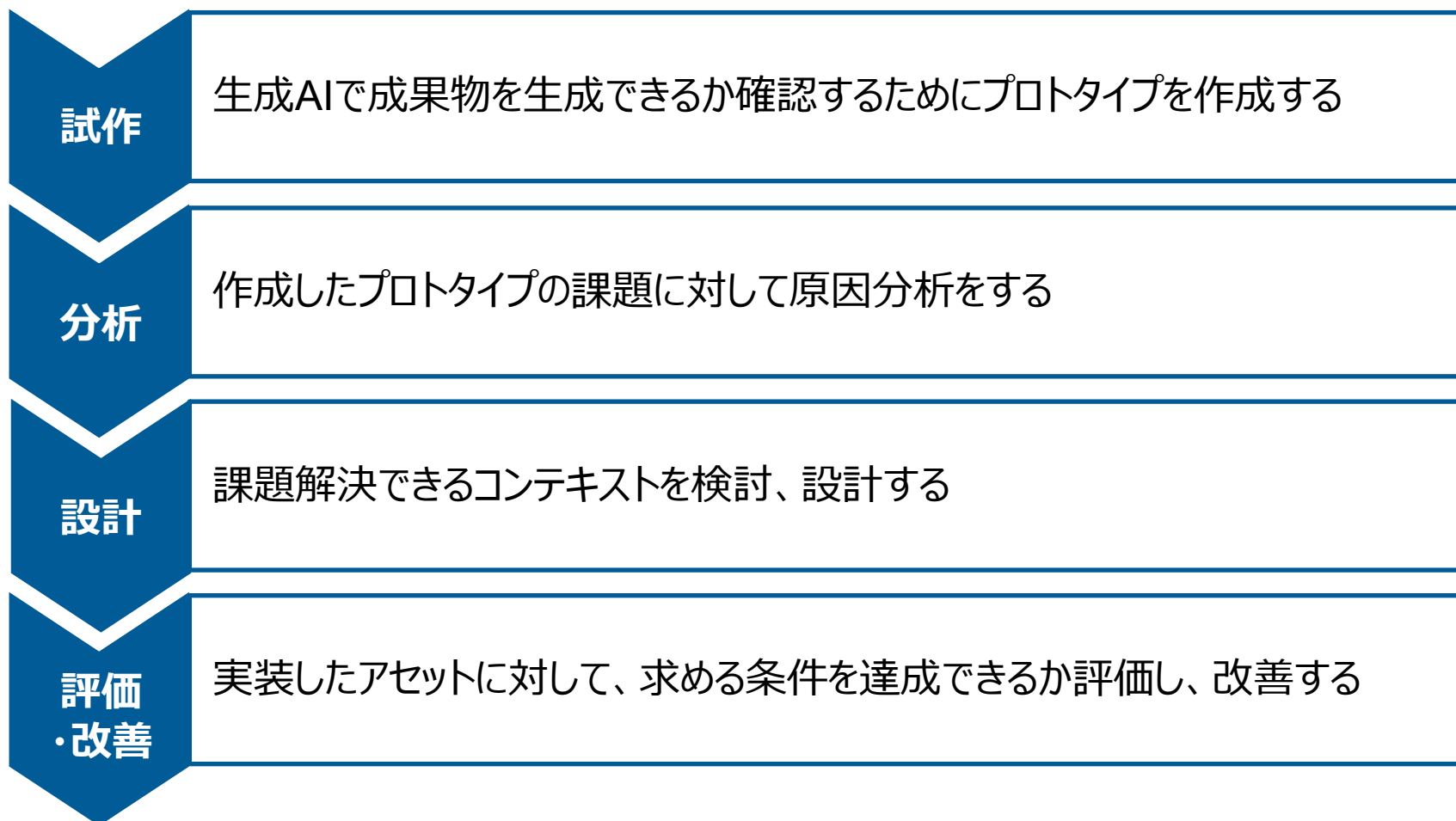
アセットの動作イメージ

3つの条件を満たすために、社内で広く普及してるGitHub Copilotを採用し、スラッシュコマンドだけで任意の成果物を出力できるアセットを目指す。



アセット実装プロセス

試作、分析、設計、評価を繰り返し行うことでアセットを実装する。



プロトタイプの実装

単一のプロンプト（*.prompt.md）に任意の成果物を作成するためのフローを記述した。

プロンプト記載内容

```
# 目的
# 基本動作原則
# 成果物フロー
# 成果物作成の実行ルール
  ## 1. 成果物生成開始時の確認
  ## 2. 入力情報の確認方法
  ## 3. プロンプトファイルの参照
  ## 4. 成果物の生成
  ## 5. 成果物の命名規則
# プロンプト抽出のマッピング
# ユーザーとの対話例
# 品質保証
# エラーハンドリング
# 注意事項
# 使用開始方法
```

目的

ユーザーからの依頼に応じて、成果物フローに従って、システム開発に必要な成果物を順番に生成します。各成果物の生成には、ワークフローから抽出したプロンプトを使用します。

基本動作原則

1. ****フロー順守****: 成果物フローの順番に従って成果物を生成する
2. ****入力確認****: 各成果物生成前に、必要な入力情報をユーザーに確認する
3. ****プロンプト活用****: ワークフローのYMLファイルから抽出されたプロンプト（.mdファイル）を参照する
4. ****テンプレート準拠****: `template/RD/` 配下のテンプレートに従った形式で成果物を生成する
5. ****成果物保存****: 生成した成果物は `attrs/RD/` 配下に保存する

成果物フロー

以下の順番で成果物を生成します：

No	アクティビティ	成果物ID	成果物名	入力
1	システム要件整理	D-RD001	概要業務フロー図	ユーザの指示
2	システム要件整理	D-RD002	ビジネスルール定義書	ユーザの指示、ユーザが提供する関連ドキュメント
3	システム要件整理	D-RD003	非機能要求定義書	ユーザの指示、PJ計画書
4	システム要件整理	D-RD004	システム全体構成図	(TBD)
5	業務フロー定義	D-RD005	アクター定義書	D-RD001
6	業務フロー定義	D-RD006	業務フロー図	D-RD001, D-RD005
7	業務フロー定義	D-RD007	機能一覧	D-RD006
8	業務フロー定義	D-RD102	ステートチャート図	D-RD006, D-RD012
9	機能定義	D-RD008	UI規約	ユーザ指示、ユーザが提供する過去のUI規約
10	機能定義	D-RD010-1	画面一覧	D-RD006, D-RD012, D-RD008
11	機能定義	D-RD010-2	帳票一覧	D-RD006, D-RD012
12	機能定義	D-RD009-1	画面定義書	生成対象の画面名, D-RD012, D-RD008
13	機能定義	D-RD009-2	帳票定義書	生成対象の帳票名, D-RD006, D-RD012, D-RD008
14	機能定義	D-RD011	画面遷移図	D-RD010-1, D-RD006, D-RD012, D-RD102, D-RD008
15	機能定義	D-RD103	外部インターフェース設計書	D-RD012, D-RD005
16	データモデル定義	D-RD012	ER図	ユーザからのエンティティなど、ER図作成に必要な情報

成果物生成の実行ルール

1. 成果物生成開始時の確認

0. 一番最初に、
- ```
```text
1. システム要件整理 - D-RD001 概要業務フロー図
2. システム要件整理 - D-RD002 ビジネスルール定義書
3. システム要件整理 - D-RD003 非機能要求定義書
4. システム要件整理 - D-RD004 システム全体構成図
5. 業務フロー定義 - D-RD005 アクター定義書
```

プロトタイプで発生した品質問題

テンプレートに準拠しない、品質チェックが実行されない、前提条件が揃わないまま生成が始まる、といった問題が発生。
アセットに求める条件である「一定の出力品質」が満たせない。

出力がブレている例

テンプレート (xml形式)

```
<!-- 画面遷移図 -->
<mxfile host="65bd71144e">
  <diagram name="ScreenFlow Template (Generic)" id="0">
    <mxGraphModel dx="1053" dy="787" grid="1" gridSize="10" guides=
connect="1" arrows="1" fold="1" page="1" pageScale="1" pageWidt
pageHeight="2100" math="0" shadow="0">
      <root>
        <mxCell id="0"/>
        <mxCell id="1" parent="0"/>

        <!-- メタデータオブジェクト (右下) -->
        <object label="【メタデータ】&#xa;ID: {artifact_id}&#xa;#
Name: {name}&#xa;#a;YAML: .../{prompt_filename}&#xa;Commit
#xa;Date: {generated_date}&#xa;Agent: {generator_agent}
#xa;Input: {input_artifacts}"
          artifact_id="{artifact_id}"
          phase="{phase}"
          name="{name}"
          source_prompt="{source_prompt}"
          source_commit="{source_commit}"
          generated_at="{generated_at}"
          generator_agent="{generator_agent}"
          model="{model}"
          input_artifacts="{input_artifacts}"
          id="metadata_object">
          <mxCell style="shape=note;whiteSpace=wrap;html=1;ba
darkOpacity=0.05;size=10;fillColor=#fff2cc;strokeCo
vertex="1" parent="1">
            <mxGeometry x="2670" y="1920" width="280" heigh
as="geometry"/>
          </mxCell>
        </object>
      </root>
    </mxGraphModel>
  </diagram>
</mxfile>
```

AIの出力 (ASCII形式)

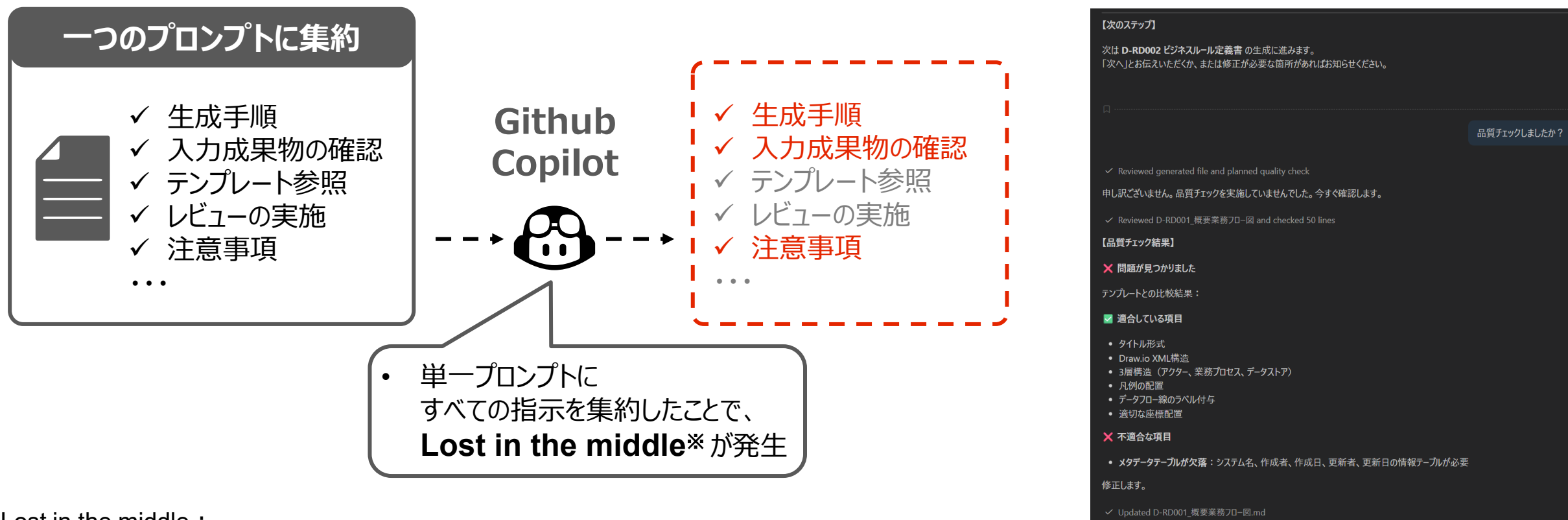
```
atrs > RD > D-RD011_画面遷移図.md > abc # 画面遷移図
1 # 画面遷移図
11 ## 画面遷移の全体フロー
14
15
16 SC001
17 フライト検索
18
19 [検索]
20
21
22 SC002
23 検索結果一覧
24
25 [便選択] [検索条件変更]
26
27
28 SC003
29 予約情報入力
30
31 [確認へ]
32
33 SC004
34 予約確認
35 [[修正]T[確定]]
36
37 [修正時] SC005
38
39 [計算エラー]
```

テンプレート非準拠率

	非準拠数	非準拠率
1回目	6/14 件	42.9 %
2回目	5/13 件	38.5 %
3回目	3/14 件	21.4 %
合計	14/41 件	34.1 %

原因分析

単一プロンプトへ指示が集約していることで、Lost in the middle※が発生し、GitHub Copilotが一部の指示を省略していた。そこで、プロンプトの分割が解決のカギだと考えた。

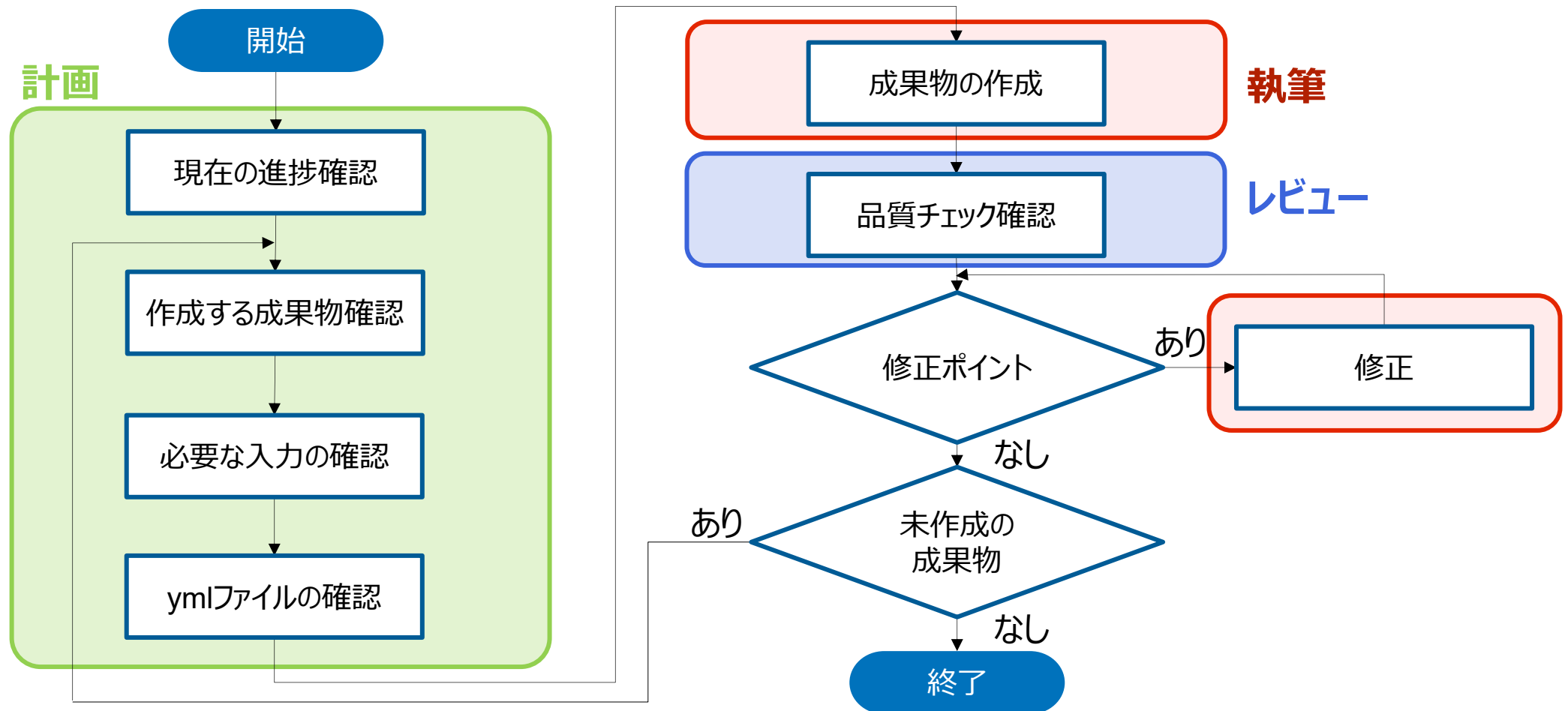


※Lost in the middle :
コンテキストの中間部分に配置された情報を効果的に利用できず、コンテキストの始めと終わりに配置された情報をより優先する現象

単一プロンプト構造を解消することで「一定の出力品質が満たせない」問題を解決

単一プロンプトの分割

単一プロンプトの成果物作成フローを確認すると、大きく3段階に分かれていることが判明した。



単一プロンプト分割の実装

単一プロンプトの分割により、一定の出力品質はクリアできたが、
使用者が正しい順序で実行することに依存する設計になり、「誰でも使いやすいこと」がクリアできなくなった。

指示を複数ファイルへ分割した結果、構成と実行手順が複雑に

```
sa4d_split/
├── sa4d.prompt.md          # 入口（短いルータ：フェーズ判定＋次に読む1つ＋参照証跡）
├── phases/
│   ├── 01_plan.prompt.md  # 計画：対象成果物ID決定・依存関係確認
│   ├── 02_intake.prompt.md # 入力ゲート：不足入力の洗い出し・質問のみ
│   ├── 03_write.prompt.md # 執筆：テンプレート枠固定→本文生成
│   └── 04_review.prompt.md # レビュー：品質ゲート
├── rules/
│   ├── flow.md            # 成果物フロー（依存関係の唯一の正）
│   ├── quality.md         # 品質ゲート詳細（チェック観点＋判定基準＋結果フォーマット）
│   ├── repo_conventions.md # 保存先・命名・テンプレート場所
│   └── errors.md          # 例外処理（不足/テンプレート不整合/参照欠落/矛盾）
├── refs/                  # 成果物ID↔ワークフロー対応表（移植用）
├── templates/
│   └── md/                # 成果物固有：必須入力・テンプレート枠・記載方針（個別差分）
└── examples/              # 対話例
```

実行手順：phase01~04のスラッシュコマンドを順に実行、修正があれば適宜03,04を再実行

解決した課題

誰が使っても
一定の出力品質であること

新たに出てきた課題

誰でも使いやすいこと

新たな課題に対する解決策

新たな課題の解決策を模索する中でGitHub Universe2025で提唱されたContext Engineeringに着目した。

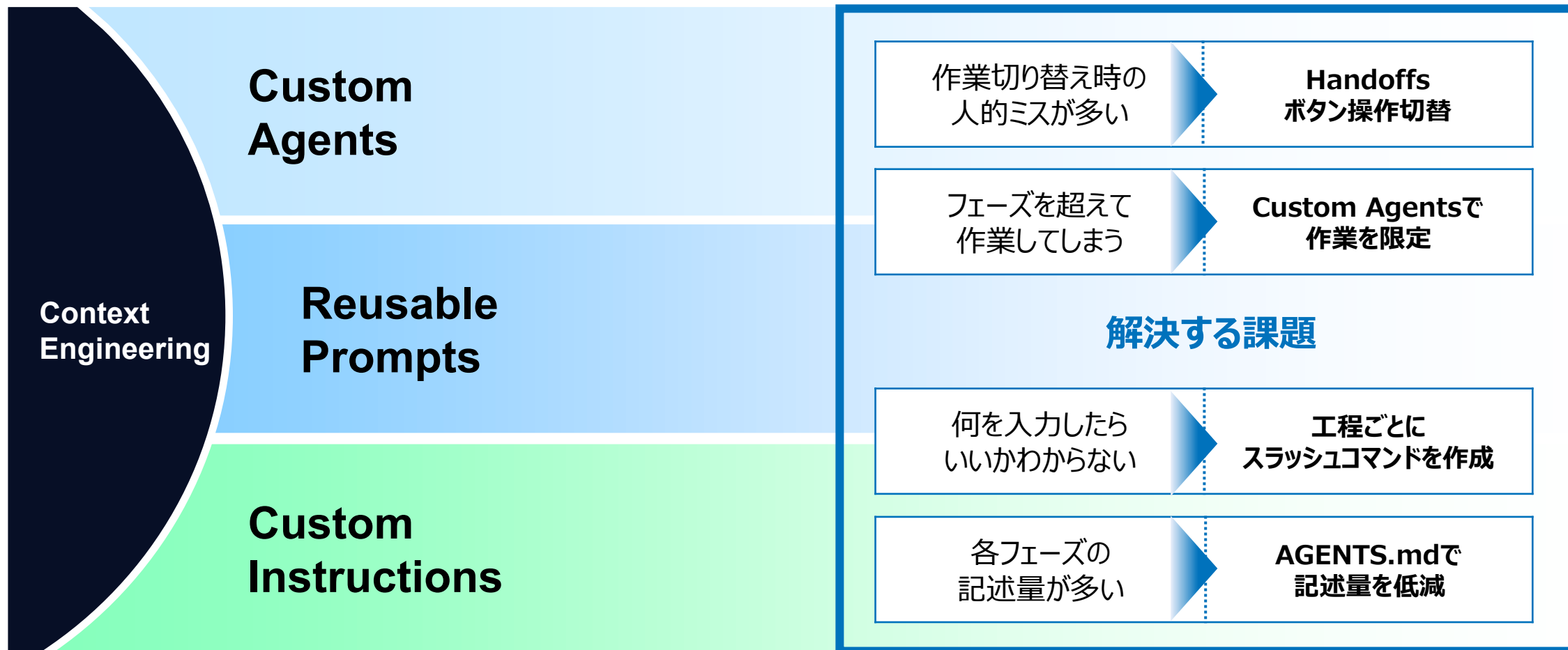
GitHub Universe 2025で発表された考え方を、アセット開発中の2025/11に取り入れた

Context Engineering Primitives			
	Custom Instructions *.instructions.md <i>Rules of engagement</i>	Reusable Prompts *.prompts.md <i>Executable commands</i>	Custom Agents *.agents.md <i>Personas and workflows</i>
Primary purpose	Always-on guidelines that influence all AI interactions	Specific tasks you run when needed	Different working contexts with specialized tools and instructions
Best for	- Coding standards - Language-specific rules - Documentation guidelines	- Generate components - Increase test coverage - Atomic commit	- API Architect - SRE Debugger - Security Review
Usage	Automatically applied to all requests	User runs slash command (soon: auto-run)	User switches agent (soon: auto-suggested)

Harald Kirschner. Your codebase, your rules: Customizing Copilot with context engineering.
<https://www.youtube.com/watch?v=0jEzUhU8bLc&list=PL0lo9MOBetEFKNIPHNouEmVeYeyoyGTXC&index=8&t=164s>. 2025/11/13.

Context Engineeringでルールを実装

使用者が必ず正しい順序で実行するための、使用者への「ルール」をコンテキストエンジニアリングで実装した。



Custom Agents(*.agent.md)

計画、実装、レビューの三段階はCustom Agentsを活用して分割した。

Custom AgentsのHandoffs機能でボタン操作のみで正しい順序を実行できるように。

```
agents/  
├─ planner.agent.md  
├─ writer.agent.md  
└─ reviewer.agent.md
```

```
---  
name: Planner  
description: "artifacts-map.ymlを参照し、次に作る成果物を特定・要件確認する"  
ツールの構成...  
tools: ['search', 'todo', 'web/githubRepo']  
handoffs:  
  - label: "生成へ (Writer) "  
    agent: Writer  
    prompt: "直前の合意内容に基づき、対象成果物をテンプレに沿って生成し、結果をchat_logsに記録し  
            (1) Frontmatter付与 (.github/config/frontmatter.schema.yml に準拠) (2) テンプレ必須セク  
            は TODO を明示) (3) 図は Draw.io (Mermaid も可) ■想定：依存関係は合意済み。不足があれば質  
            send: true  
    argument-hint: " (任意) PJ名や前提を入れると質問が減ります"  
---
```

Planner から続行

生成へ (Writer) ▼ Autopilot を使用して開始する

+ RD.prompt.md

(任意) PJ名や前提を入れると質問が減ります

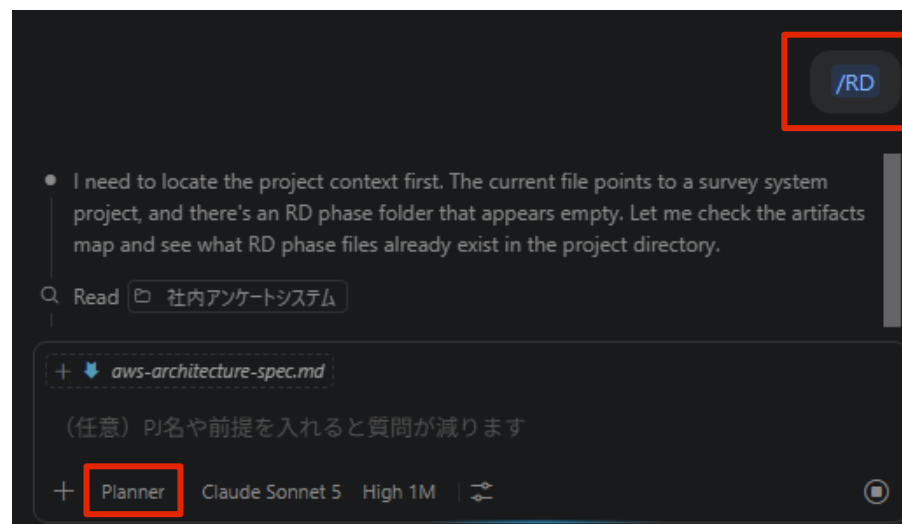
+ Planner * Claude Sonnet 5 High 1M

Reusable Prompts(*.prompt.md)

各工程のエントリーポイントをプロンプトファイルで定義することで、
各工程、同じスラッシュコマンドで任意の成果物を作成できるように。

```
prompts/  
├── RD.prompt.md  
├── ED.prompt.md  
├── ID.prompt.md  
├── IT.prompt.md  
└── ST.prompt.md
```

```
RD.prompt.md X  
.github > prompts > RD.prompt.md > abc # /RD  
1 ---  
2 name: RD  
3 description: '要件定義 (RD) フェーズを開始します'  
4 agent: Planner  
   ツールの構成...  
5 tools: ['search', 'web/githubRepo']  
6 argument-hint: ' (任意) プロジェクト名 (例: sales_system) や前提条  
   件を入力すると質問が減ります'  
7 ---  
8
```



Custom Instructions(AGENTS.md)

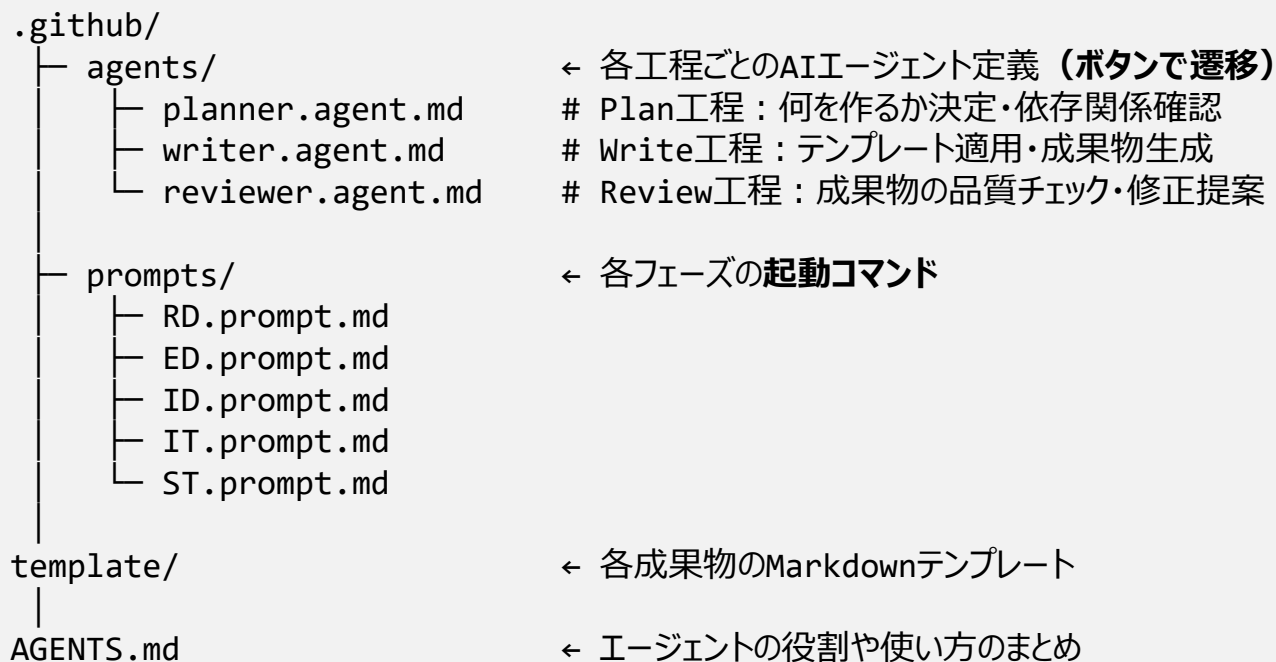
AGENTS.mdに共通事項を集約することで、エージェント(*.agent.md)とプロンプト(*.prompt.md)に重複した記述を削減でき「Lost in the middle」を防止した。

```
AGENTS.md
AGENTS.md > abc ## 🤖 エージェント構成 (ドキュメント作成ワークフロー)
5  ## 🤖 エージェント構成 (ドキュメント作成ワークフロー)
6
7  > 詳細なフロー・Handoffs・循環依存・反復作成のルールは ``.github/agents/*.agent.md`` を参照。
8
9  | エージェント | 役割 | 禁止事項 |
10 | :-- | :-- | :-- |
11 | **Planner** | 次に作る成果物を特定し、必要情報をユーザーから引き出す | 成果物ファイルの直接編集・生成 |
12 | **Writer** | Plannerの指示に基づきドキュメントを生成・修正する | ``{project_name}/`` 以外への書き込み、テンプレート構造の破壊 |
13 | **Reviewer** | 成果物の品質チェック・指摘を行う | 成果物ファイルの直接編集 (軽微な修正も Writer へ差し戻し) |
14
15 ---
16
17 ## 📏 Ground Rules (共通ルール)
18
19 1. **Single Source of Truth (SSOT)**
20 | - 成果物の定義・順序はすべて ``.github/config/artifacts-map.yml`` に従う。
21 | - 勝手なファイル名やIDで生成しない。
```

アセットの実装構成

Context Engineeringを用いることで、求める条件をクリアしたアセットを作成することができた。

ディレクトリ構成



成果物が社内標準に準拠していること

- ✓ 各成果物のテンプレートを用意

誰が使っても一定の出力品質であること

- ✓ コンテキストを分割し、Lost in the middleを防止

誰でも使いやすいこと

- ✓ *.prompt.mdで工程ごとに起動コマンドを固定
- ✓ *.agent.mdのHandoffsボタンを設定

実行手順のBefore： phase01~04のスラッシュコマンドを順に実行、修正があれば適宜03,04を再実行

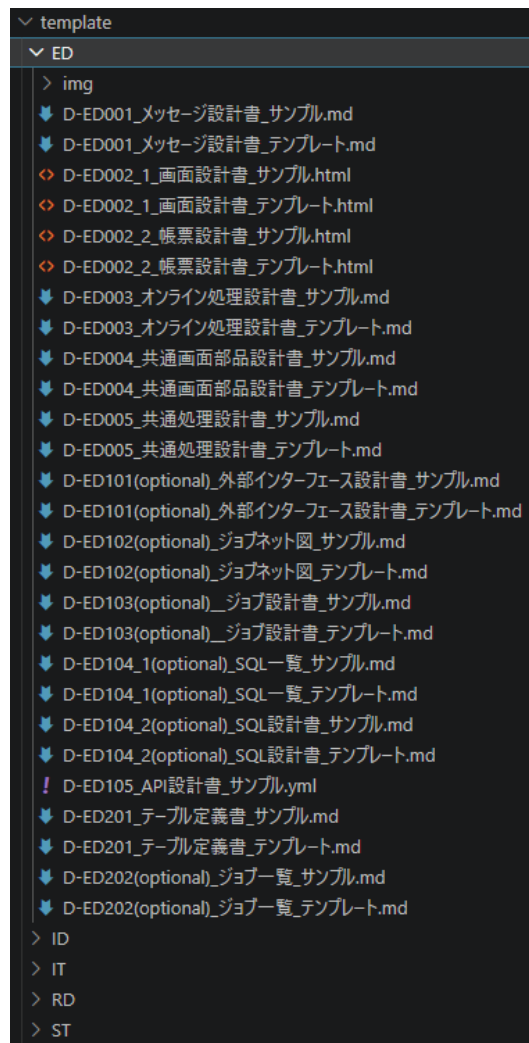
実行手順のAfter： 工程ごとにスラッシュコマンドを1回実行、あとはAIの指示に従ってチャットに表示されたボタンを押下

アセットの評価



アセットの評価（成果物が社内標準に準拠していること）

予定した全39種の成果物のテンプレートを作成し、それに準拠した出力が得られた。



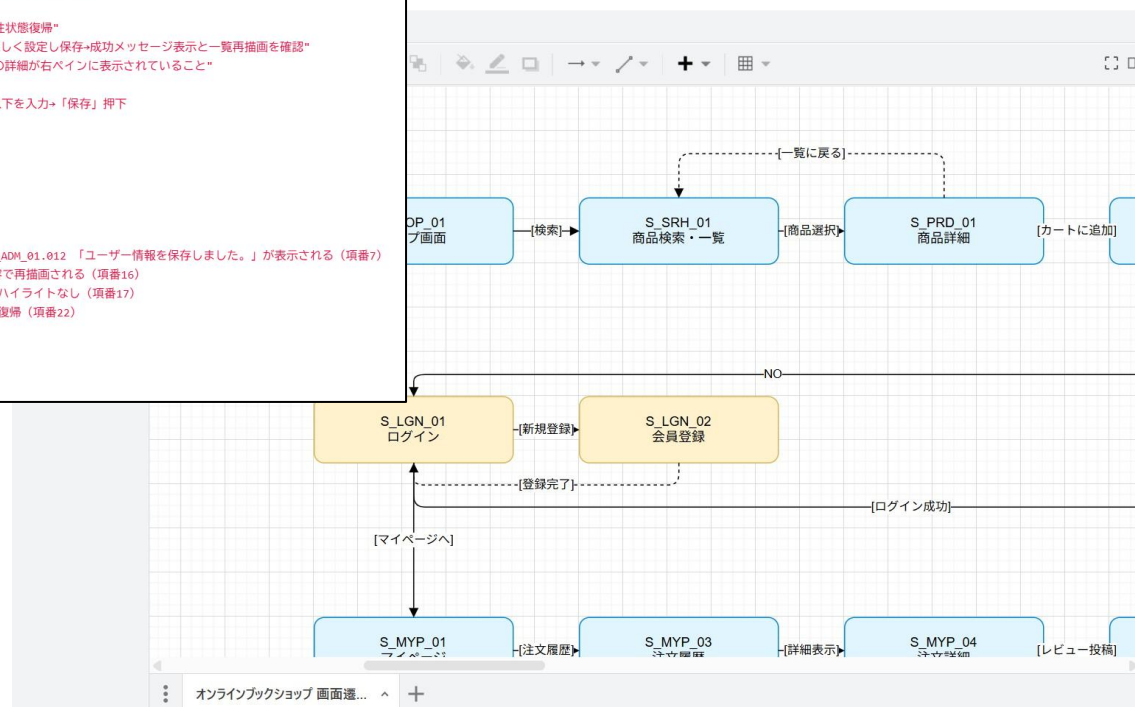
Github
Copilot



AIの出力の一例 (テストケース・Markdown)

```
テストケース:
# --- ケース 1.1: 正常系 (基本) ---
- ケースNo: "1.1"
対応テスト項目:
- テスト項目番号: 7
  テスト項目: "情報メッセージの文言・ID一貫性 (M.INF.ATRS_ADM_01.012) "
- テスト項目番号: 16
  テスト項目: "正常保存後の一覧更新 (更新後再表示) "
- テスト項目番号: 17
  テスト項目: "正常入力でユーザー保存処理の完了動作"
- テスト項目番号: 22
  テスト項目: "保存後の保存ボタン活性状態復帰"
概要: "[Screen]S_ADM_01で編集入力を正しく設定し保存→成功メッセージ表示と一覧再描画を確認"
事前条件: "既存ユーザー user=U001 の詳細が右ペインに表示されていること"
入力値: [
  操作: [Screen]S_ADM_01 右ペインで以下を入力+「保存」押下
  user_name = "管理 太郎"
  email = "admin.taro@example.com"
  department = "企画部"
  status = "有効"
  roles = ["admin", "auditor"]
  force_reset = OFF
]
期待結果: [
  - [Screen]右上に MsgID=M.INF.ATRS_ADM_01.012 「ユーザー情報を保存しました。」が表示される (項番7)
  - [Screen]ユーザー一覧が更新後の内容で再描画される (項番16)
  - 入力値が右ペインに反映され、エラーハイライトなし (項番17)
  - 保存完了後、保存ボタンは活性状態に復帰 (項番22)
]
単機能テストの実施環境: "IT環境"
本番稼働時の性能目標値: "2秒"
単機能目標値: "2秒"
備考: ""
```

AIの出力の一例 (概要業務フロー図・draw.io)



成果物テンプレートの一覧

アセットの評価（誰が使っても一定の出力品質であること）

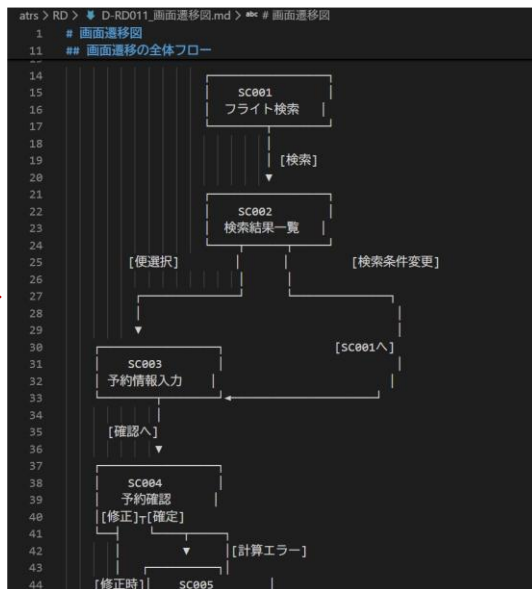
小規模なサンプルプロジェクト（約3kstep）で検証を行った結果、想定 of 振る舞いを100%再現できることを確認できた。リリース後もテンプレートに準拠しないといった報告はされていない。

✗ Before : テンプレート非準拠

テンプレート（XML形式）

```
<!-- 画面遷移図 -->
<mxfile host="65bd71144e">
  <diagram name="ScreenFlow Template (Generic)" id="0">
    <mxGraphModel dx="1853" dy="787" grid="1" gridSize="10" guides=
connect="1" arrows="1" fold="1" page="1" pageScale="1" pageWidt
pageHeight="2100" math="0" shadow="0">
      <root>
        <mxCell id="0"/>
        <mxCell id="1" parent="0"/>
        <!-- メタデータオブジェクト (右下) -->
        <object label=" [メタデータ] &#xa;ID: {artifact_id}&#xa;
Name: {name}&#xa;YAML: .../{prompt_filename}&#xa;Commit
&#xa;Date: {generated_date}&#xa;Agent: {generator_agent}
&#xa;Input: {input_artifacts}"
          artifact_id="{artifact_id}"
          phase="{phase}"
          name="{name}"
          source_prompt="{source_prompt}"
          source_commit="{source_commit}"
          generated_at="{generated_at}"
          generator_agent="{generator_agent}"
          model="{model}"
          input_artifacts="{input_artifacts}"
          id="{metadata_object}"
        </object>
        <mxCell style="shape=note;whiteSpace=wrap;html=1;ba
darkOpacity=0.05;size=10;fillColor=#fff2cc;strokeCo
vertex="1" parent="1">
          <mxGeometry x="2670" y="1920" width="280" heigh
as="geometry"/>
        </mxCell>
      </root>
    </mxGraphModel>
  </diagram>
</mxfile>
```

AIの出力（ASCII形式）

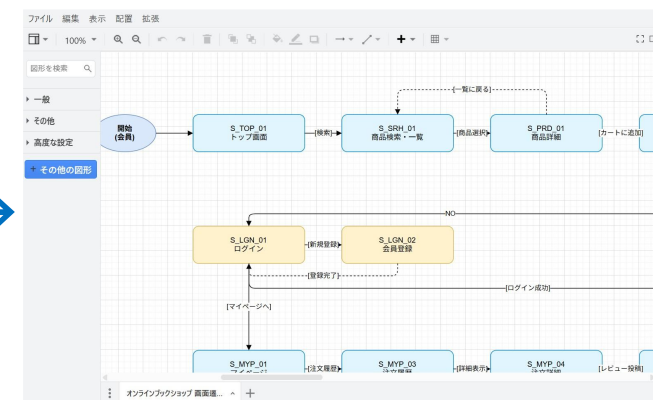


✓ After : テンプレートに準拠

テンプレート（XML形式）

```
<!-- 画面遷移図 -->
<mxfile host="65bd71144e">
  <diagram name="ScreenFlow Template (Generic)" id="0">
    <mxGraphModel dx="1853" dy="787" grid="1" gridSize="10" guides=
connect="1" arrows="1" fold="1" page="1" pageScale="1" pageWidt
pageHeight="2100" math="0" shadow="0">
      <root>
        <mxCell id="0"/>
        <mxCell id="1" parent="0"/>
        <!-- メタデータオブジェクト (右下) -->
        <object label=" [メタデータ] &#xa;ID: {artifact_id}&#xa;
Name: {name}&#xa;YAML: .../{prompt_filename}&#xa;Commit
&#xa;Date: {generated_date}&#xa;Agent: {generator_agent}
&#xa;Input: {input_artifacts}"
          artifact_id="{artifact_id}"
          phase="{phase}"
          name="{name}"
          source_prompt="{source_prompt}"
          source_commit="{source_commit}"
          generated_at="{generated_at}"
          generator_agent="{generator_agent}"
          model="{model}"
          input_artifacts="{input_artifacts}"
          id="{metadata_object}"
        </object>
        <mxCell style="shape=note;whiteSpace=wrap;html=1;ba
darkOpacity=0.05;size=10;fillColor=#fff2cc;strokeCo
vertex="1" parent="1">
          <mxGeometry x="2670" y="1920" width="280" heigh
as="geometry"/>
        </mxCell>
      </root>
    </mxGraphModel>
  </diagram>
</mxfile>
```

AIの出力
(テンプレート準拠・draw.io)



アセットの評価（誰でも使いやすいこと）

アセットを実際のプロジェクトに適用し、要件定義、基本設計工程で一定の生産性向上効果を確認した。さらにアセットを使用したプロジェクトメンバは1日で習熟できたことから、誰でも使いやすいことを判断した。

概要	詳細
目的	2026/04から開始する開発案件に、アセットを導入できるか判断する
背景・課題	- 現行の設計工程は人手で実施しており、品質のばらつきや属人化が課題 - アセットによる設計効率化・品質向上が期待されるが、効果が未検証
期間	2026/03（1か月）
対象プロジェクト	公共案件の某システムの機能追加
対象成果物	• RD • 画面一覧 • 画面遷移図 • 画面定義書 • ED • オンライン処理設計書

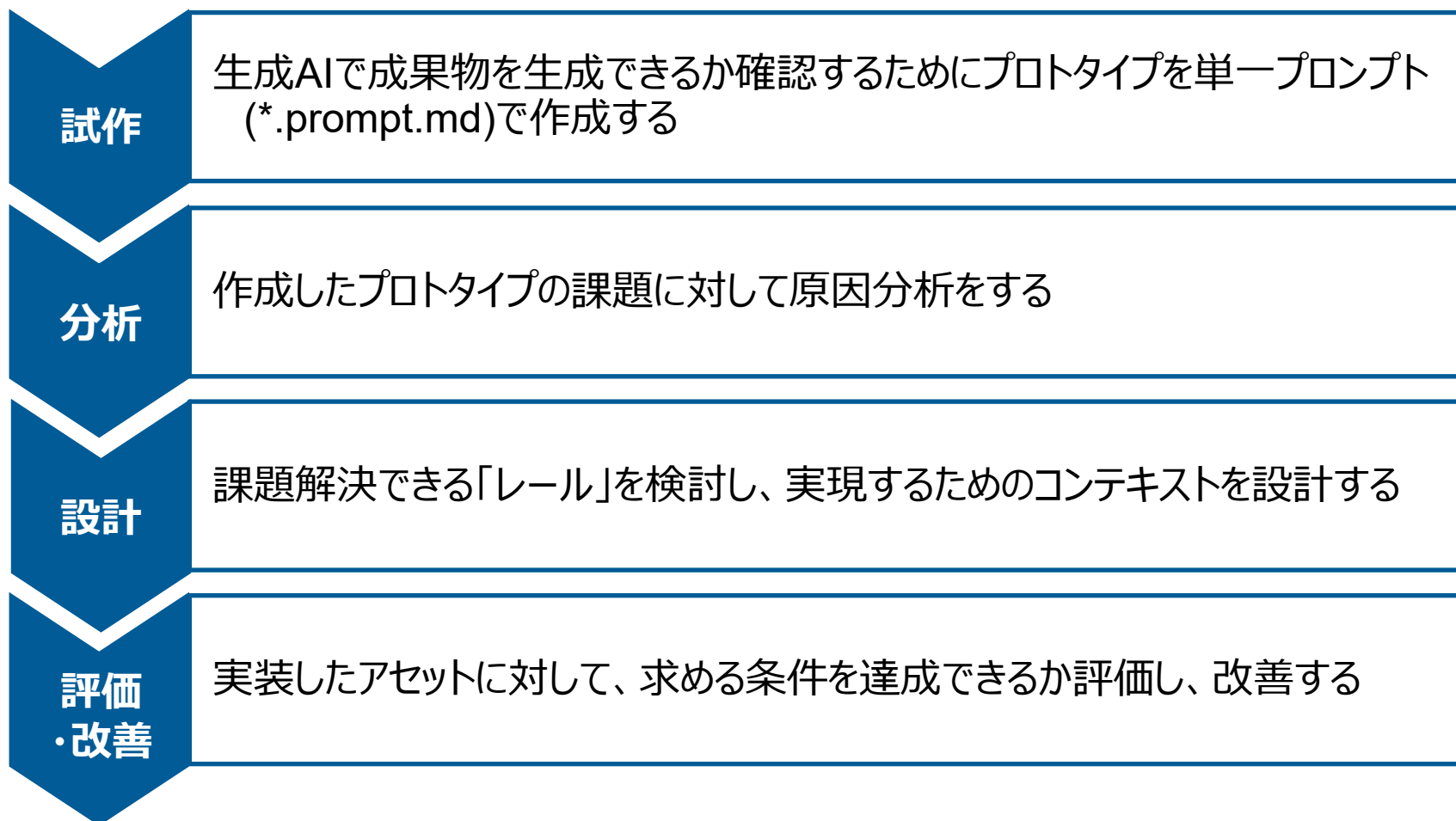
アセットの使い方・設計思想を1日レクチャ、その後手順書などのドキュメント類だけで自走可能に

まとめ



まとめ（アセット実装プロセス）

生成AIに開発プロセスの全成果物を作らせるには、「ルール」を「コンテキスト」に落とし込むマッピング(設計)が重要である。



まとめ（アセットの効果）

求める条件「成果物が社内標準に準拠していること」、「誰が使っても一定の出力品質であること」、「誰でも使いやすいこと」を達成し、実プロジェクトで、要件定義、基本設計工程で一定の生産性向上効果を確認できた。

成果物が社内標準に準拠していること

- ✓ 各成果物のテンプレートを用意

誰が使っても一定の出力品質であること

- ✓ コンテキストを分割し、Lost in the middleを防止

誰でも使いやすいこと

- ✓ *.prompt.mdで工程ごとに起動コマンドを固定
- ✓ *.agent.mdのHandoffsボタンを設定

