別添資料1

別添 資料1：基本共通情報モデル仕様案 （検討結果から ChatGPT による仕様書化）

目的：
「日本の多様な医療文書を FHIR 化する際の共通情報モデル」を定義すること。

背景：
従来の FHIR 文書化では、

- 記載項目ごとに FHIR リソースが異なる
- 文書ごとに構造が変わる
- 利用者が取得方法を理解しにくい

という問題がある。
そこで、
文書構造は Composition.section
記載項目は Observation
という単純な原則に統一している。

全体構造
大きく分けると以下の構成である。

```
Bundle
 └ Composition
      └ section
          └ section
              └ entry
                  └ Observation
```

---

1. Composition の役割
Composition は
文書の構造
を表現する。
例えば
患者基本情報

生活状況
　清潔
　　入浴
　　更衣

活動
移乗
移動
という文書なら
Composition.section
└ patientInfo

Composition.section
└ adlSection
├ hygiene
│ ├ bathing
│ └ dressing
└ activity
├ transfer
└ mobility
として表現する。

---

section.text の扱い
この設計では
section.text
は
表示補助用
である。
データ本体を持たせない。
つまり
NG

section.text
= "患者の希望は..."
ではない。

---

2. Observation の役割
すべての記載項目を
Observation
で表現する。
例:
患者の希望

↓
Observation.code = patient-hope

介護者の状況
↓
Observation.code = caregiver-status

栄養課題
↓
Observation.code = nutrition-issue

---

3. JPDocumentItemCodeSystem

Observation.code で使用するコード体系である。
例:
patient-hope
family-hope

family-structure
caregiver-status

adl-transfer
adl-mobility

nutrition-issue
など。

目的は
Observation
WHERE code='patient-hope'
だけで
全医療文書から取得できるようにすることである。

---

4. Observation.valueCodeableConcept

今回の設計の特徴である。
通常 FHIR では
valueString
valueQuantity
valueBoolean
などを使い分ける。

しかし文書では
体重 50kg
もあれば
体重:変わりなし
もある。
そこで
Observation.valueCodeableConcept
に統一する。

例
体重 50kg
↓
valueCodeableConcept
= measured

体重:変わりなし
↓
valueCodeableConcept
= unchanged

---

5. component[typedValue]
実際の数値を保持する。
体重 50kg
↓
component[typedValue]
valueQuantity = 50 kg

BMI 22.3
↓
component[typedValue]
valueQuantity = 22.3

---

6. component[originalText]
原文保持用である。
例えば
患者・家族の希望

「自宅で過ごしたい」
なら

component[originalText]
= ″自宅で過ごしたい″

これにより
section.text
に原文を持たせなくても済む。

---

7. component[otherText]

選択肢の
その他
に対応する。
例:
食事形態

□普通食
□軟菜食
□その他
　　ペースト食
↓
valueCodeableConcept
= other

component[otherText]
= ″ペースト食″

---

8. 数値検索用 Observation

FHIR 検索を使えるようにするため、
別 Observation を生成する。
例:
Observation.code
= body-weight
とは別に
Observation.code
= body-weight-quantity
を作る。

Observation?code=body-weight-quantity
で検索可能になる。

関係は
derivedFrom
で結ぶ。
body-weight-quantity
　derivedFrom
　　body-weight

---

9. 繰り返し項目

例えば
入所時の症状

発熱
咳嗽
呼吸困難
の場合。
Observation を繰り返する。
admission-symptom
　発熱

admission-symptom
　咳嗽

admission-symptom
　呼吸困難

同じコードを複数持つ。

順序は
component[itemOrder]
で保持する。

---

10. 主要 FHIR リソース

以下だけは Observation 化しない。
Patient
Condition
AllergyIntolerance
Procedure
MedicationStatement
MedicationRequest

CarePlan
Goal
Device

つまり
傷病名
は
Condition

看護計画
は
CarePlan

看護目標
は
Goal

である。

このモデルの最大の利点
利用者は
Composition.section
で文書構造を辿り、
Observation.code
だけを見れば、
どの文書から来た情報でも同じ方法で取得できる。
つまり、
患者の希望
介護者状況
栄養課題
ADL
生活状況
などを
Observation
WHERE code='patient-hope'
のように文書横断的に取得できる。
これは 80 文書以上の日本の医療文書を FHIR 化する場合に、実装者・利用者双方にとって極めて扱いやすいモデルになる。

別添資料２

生成AIによる医科様式のFHIR実装ガイド作成について

IG-EcoSystem プロジェクト 概要説明

2026年3月

背景と目的

課題

- 医科様式（診療録・看護記録・栄養評価等）のFHIR準拠が求められている
- FHIR実装ガイド（IG）作成にはFHIR規格・国内プロファイル・FSH記述言語等の高度な専門知識が必要
- 熟練した人材の確保が困難
- 様式数は膨大であり、1様式あたりのIG作成コストが高い

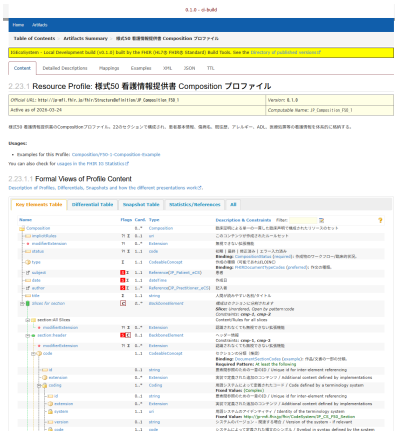
本プロジェクトの目的

- 生成AI（Claude Code）と専用ツール群を活用した半自動的なIG作成ワークフローの構築
- 様式50（看護・栄養）を対象に自動化を実施
- 「FHIR専門家の工数を削減しつつ品質の高いIGを作成できるか」を検証
- 医科様式のFHIR化ワークフローを標準化・半自動化

実施したこと

様式50 看護及び栄養管理に関する情報

生成AI



実装ガイド

プロジェクト規模（様式50 実績）

4,400+
FSHファイル総行数
プロファイル・CS・VS・サンプル

20+
定義済みプロファイル数
Observation・Composition等

30+
CodeSystem定義数
ADL・認知症・食事形態等

30+
ValueSet定義数
各CodeSystemに対応

26
開発フェーズ数
各フェーズにspec/plan/tasks

多数
サンプルインスタンス
Bundle・Composition等

技術スタック

FHIR 基盤

- **FHIR R4 (4.0.1)**
  医療情報標準規格
- **JP Core (1.1.2-clins)**
  日本国内FHIRプロファイル
- **JP-CLINS (1.11.0)**
  電子カルテ情報共有サービス用
- **JP Terminology (1.4.0)**
  日本語医療用語・CodeSystem

開発ツール & AI支援

- **FSH / SUSHI / IG Publisher**
  プロファイル記述・コンパイル・HTML生成
- **Claude Code**
  AI開発支援エージェント（CLI）
- **SpecKit**
  仕様→実装ワークフロー自動化
- **Serena MCP**
  コードベース意味解析・記憶

主要ツール選択の理由

Claude Code

- CLIベースのエージェント動作でIGの実装ループを自動化
- プログラムコーディングの世界でNo1の実力（2026年1月現在）
- MCP対応でSerena等の外部ツールと統合
- FHIR/FSHの知識を保有し高精度な対応
- SUSHIエラーログから修正方針を具体的に提示

SpecKit

- 仕様・計画・タスク・実装を独立した成果物として管理
- 曖昧点を2〜3個程度の質問で早期に明確化
- タスクを最小単位に分解しAIの実装精度を向上
- 共通仕様(Constitution)連携で全フェーズの設計方針を一貫
- 整合性の自動チェックで設計ミスを事前発見

SpecKit ワークフロー

constitution 原則定義 → specify 仕様書 → clarify Q&A → plan 実装計画 → tasks タスク分解 → analyze 整合性検証 → implement FSH実装

5フェーズ開発モデル

	1	2	3	4	5
	様式分析	FHIRマッピング	FSH実装	ビルド検証	IG公開
	PDFから項目抽出 TSV整理・仮マッピング	FHIR要素マッピング確定 リソース選択・VS検討	プロファイル・CS・VS サンプル実装	SUSHI 0エラー確認 エラー修正ループ	IG Publisher HTML生成 qa.html確認
	○生成AI＋確認	○ 生成AI＋確認	◎ 生成AI	◎ 生成AI	手動

推奨チーム構成

FHIRアーキテクト

全体設計策定
プロファイル設計レビュー
JP Core準拠確認

△ 構造設計はチェックが必要

医療情報専門家

様式の内容解釈
臨床的意味の解説
完成物の内容確認

△ 専門知識はチェックが必要

FSH実装エンジニア

FSHコード実装・修正
SUSHIエラー解析
サンプルインスタンス作成

◎ AIが大部分を担当可

プロジェクト管理者

フェーズ管理・進捗管理
ワークフロー運営
成果物レビュー調整

○ タスク管理はAI支援可

品質管理の仕組み

原則	品質ゲート
FHIR R4 準拠	SUSHIビルド検証
JP CLINS > JP Core 優先利用	設計レビュー
SUSHIビルドゲート	0エラーコンパイル必須
インクリメンタル検証	一度作成されたガイドを都度修正
用語バインディング	追加指示によるバインディングの実現
Compositionセクション設計	最大3階層・細粒度分離等の指示を追加

共通仕様はConstitution としてバージョン管理され、問題発生時に原則を追記することで、後続フェーズでの同一問題の再発を防止。

生成AIの課題と対策

限界	具体的な問題	回避策
コンテキストウィンドウの制約	セッションが長くなると命名規則等を「忘れる」	.claudeIgnoreで不要ファイルを除外
ハルシネーション	存在しないプロファイルURLやFSH構文を生成	SUSHIコンパイル等の検証フェーズ 生成AIだけのレビューだとすり抜けてしまう可能性大、目視レビューが必要
セッション間の記憶欠如	前回の設計判断を次セッションで忘れる	Serena MCPの記憶機能、constitution.mdへの明記
リソースや構造の割当違い	汎用プロファイルで実装を止める傾向	明示的な指示を追加
医療ドメイン知識の不足	リソース選択・CodeSystem粒度の誤判断	FHIRアーキテクト・医療専門家によるレビューが本来は必要、生成AIに質問することで多くはカバー可能。

生成AIで実装ガイドを作ってみて

- 文書のあいまいさは、生成AIの判断にも影響を与える。
- 生成AIは指示がないと汎用的なリソースの割り当てなる傾向がある。FHIRリソースの割当や階層構造、用語バインドなどは明確に指示する必要がある。
- 生成AIは相談相手としては優秀、多くの案を提示してくれる。
- 明確な指示さえあれば、ほぼ思い通りのFHIRリソースを記述できる。
- サンプルファイル作成やエラー等の面倒な作業は生成AIの威力は絶大。
- SpecKit等の手順を踏むと、生成AIへの依頼は少なくとも3分以上かかってしまう。細かな修正であれば生成AIよりも手作業のほうが早い。
- Section.entryはデータ型を割り当てられない為、ひとつリソースを割り当てるか、Extensionを利用する必要があり、冗長に感じる。
- 専門家であっても生成AIのサポートがあったほうが断然に楽、記述ミスなどは皆無。

効果と実績

FSH実装の自動化

4,400行のFSHコードの大部分をAIが記述。プロファイル・CodeSystem・ValueSet・サンプルを生成。

文書生成の自動化

仕様書・計画書・説明文・サンプルインスタンスのドキュメントをAIが自動生成。

SUSHIエラー解析の高速化

エラーログをAIに渡すだけで修正方針を提示。エラー修正ループを大幅に短縮。

実装ガイド作成未経験者の参加可能性

FSH記述をAIが担当し、未経験メンバーが設計議論・レビューに集中できる可能性あり（初期メンバ投入）

設計一貫性

SpecKitにより仕様→計画→タスク→実装の流れが明示化。長期開発でも一貫性を維持。

生成AIは「高度な補助ツール」
人間が担当：設計判断・リソース選択・医療的妥当性の確認

今後の展開

高	**他の医科様式への適用**	様式50で確立したワークフローを診療情報提供書等に展開
高	**リソース説明文書の整備**	各FHIRリソース・プロファイルに対応するデータ定義書の作成
中	**CodeSystemの標準化検討**	医療情報標準化団体との調整を検討
中	**共通項目の抽出**	共通部分のプロファイル再利用

生成 AI による医科様式の
FHIR 実装ガイド作成について

概 要 説 明 書

2026 年 3 月 24 日

株式会社ファインデックス

別添資料３

1. はじめに

近年、医療情報の電子化・標準化が急速に進展しており、医科様式の FHIR（Fast Healthcare Interoperability Resources）準拠が求められています。しかし FHIR 実装ガイド（IG）の作成には、FHIR 規格・国内プロファイル（JP Core / JP-CLINS）・FSH 記述言語等の高度な専門知識が必要であり、熟練した人材の確保が困難という課題があります。

本プロジェクト「IG-EcoSystem」では、この課題に対し生成 AI（Claude Code）と専用ツール群（SpecKit・Serena MCP 等）を活用した半自動的な IG 作成ワークフローを構築しました。様式 50（看護・栄養）を対象に 22 フェーズにわたる開発を実施し、その有効性と限界を検証しています。

本書は、このプロジェクトの成果・知見を整理し、生成 AI を活用した FHIR IG 作成の実態と今後の展開に向けた指針を提供することを目的とします。

2. プロジェクト概要

2.1 背景と目的

医科様式（診療録・看護記録・栄養評価等）は医療現場の重要な情報基盤です。これらを FHIR リソースとして標準化することで、医療機関間のデータ交換・二次利用が容易になります。しかし、様式数は膨大であり、1 様式あたりの IG 作成コストは高く、手動での整備には多大な工数が必要です。

本プロジェクトは「生成 AI を活用することで、FHIR 専門家の工数を削減しつつ品質の高い IG を作成できるか」を検証することを主目的とし、医科様式の FHIR 化ワークフローを標準化・半自動化しました。

2.2 対象様式

様式	内容	フィーチャー数	ステータス
様式 50（第 1 号）	看護・栄養（基本情報・ADL・排泄・睡眠・精神・運動感覚 等）	22 フェーズ	実装完了
様式 50（第 2 号）	看護・栄養追加情報	統合済み	実装完了
その他医科様式	診療情報提供書等（将来対応）	－	計画中

2.3 プロジェクト規模（様式 50 実績）

指標	値	備考
FSH ファイル総行数	約 4,400 行	プロファイル・CodeSystem・ValueSet・サンプル
定義済みプロファイル数	20 以上	Observation・Composition 等
CodeSystem 定義数	50 以上	ADL・認知症・食事形態 等
ValueSet 定義数	50 以上	各 CodeSystem に対応
サンプルインスタンス数	多数	Bundle・Composition・各 Observation
開発フィーチャー数	22 フェーズ	各フェーズに spec / plan / tasks.md

2.4 技術スタック

カテゴリ	コンポーネント	バージョン	役割
FHIR 基盤	FHIR R4	4.0.1	医療情報標準規格
FHIR 基盤	JP Core	1.1.2-clins	日本国内 FHIR プロファイル
FHIR 基盤	JP-CLINS	1.11.0	電子カルテ情報共有サービス用プロファイル
FHIR 基盤	JP Terminology	1.4.0	日本語医療用語・CodeSystem
開発ツール	FHIR Shorthand (FSH)	3.0.0	プロファイル記述言語
開発ツール	SUSHI	3.17.0+	FSH → FHIR JSON コンパイラ
開発ツール	IG Publisher	最新版	実装ガイド HTML 生成
開発ツール	Docker	最新	IG Publisher ビルド環境
AI 支援	Claude Code	最新	AI 開発支援エージェント（CLI）
AI 支援	SpecKit	プロジェクト独自	仕様→実装ワークフロー自動化
AI 支援	Serena MCP	MCP サーバ	コードベース意味解析・プロジェクト記憶
AI 支援	Sequential Thinking	MCP サーバ	複雑な設計判断の構造化支援

3. ツール選択の理由

本プロジェクトで採用した各ツールは、医科様式の FHIR IG 作成という要件を踏まえ、複数の選択肢を評価した上で選定しています。

3.1 Claude Code を選択した理由

選択理由	詳細
CLI ベースのエージェント動作	ファイル読み書き・コマンド実行・Git 操作を自律的に行えるため、IG の実装ループ（FSH 記述→SUSHI 実行→エラー修正）を自動化しやすい。
長大なコンテキスト処理	4,000 行以上の FSH ファイルを参照しながら一貫した実装が可能。
MCP（Model Context Protocol）対応	Serena・Sequential Thinking 等の外部 MCP サーバと統合でき、コードベース理解・設計支援の機能を拡張できる。
SpecKit との統合	SpecKit スラッシュコマンドが Claude Code 上で動作するよう設計されており、ワークフローを一元管理できる。

FHIR / FSH の知識保有	学習データに FHIR 仕様・HL7 標準が含まれており、FSH 構文・JP Core プロファイルの扱いについて高い精度で対応できる。
エラー解析能力	SUSHI や IG Publisher のエラーログをそのまま渡すことで、修正方針を具体的に提示できる。

3.2 SpecKit を選択した理由

選択理由	詳細
フェーズの明示的分離	仕様・計画・タスク・実装を独立した成果物として管理するため、AI の出力に一貫性が生まれる。
曖昧点の早期解消	/speckit.clarify により、実装前に仕様の曖昧点を 5 問程度の質問で明確化できる。
タスクの原子化	/speckit.tasks でタスクを最小単位に分解するため、AI の実装精度が向上し進捗管理も容易になる。
Constitution との連携	constitution.md（プロジェクト原則）を SpecKit が参照するため、全フェーズで命名規則・設計方針が一貫する。
軽微修正の省略可能	小規模な変更は /speckit.implement から直接開始でき、フルフローを毎回実行する必要がない。

3.3 Serena MCP を選択した理由

選択理由	詳細
シンボルレベルの検索・編集	特定のプロファイル名・コードをシンボルとして検索・編集でき、ファイル全体を読み込まずに必要箇所のみを参照できる。
プロジェクト記憶の保持	constitution.md 等を Serena の「記憶」として管理し、セッションをまたいで参照可能。毎回の再読み込みが不要。
コンテキスト削減	不要なファイルを読み込まないため、生成 AI のコンテキスト使用量を大幅に削減し実行コストを抑制できる。
参照関係の把握	あるプロファイルを参照している箇所を特定でき、変更の影響範囲を事前に把握できる。

3.4 FSH / Docker を選択した理由

ツール	選択理由
FSH（FHIR Shorthand）	Markdown 風の人間可読な構文で FHIR プロファイルを定義でき、生成 AI との相性が良い。SUSHI コンパイラが品質ゲートとして機能する。
Docker	IG Publisher（Java ベース）の実行環境をコンテナ化し、JP Core 等の FHIR パッケージを同梱することで環境差異のない再現性を確保。

4. 必要な技術メンバー構成

生成 AI を活用しても、IG 作成には人間の専門知識・判断が不可欠な領域があります。以下に推奨するチーム構成を示します。

4.1 推奨チーム構成

役割	主な責務	必須スキル	AI 代替可否
FHIR アーキテクト（リード）	・全体設計の策定 ・プロファイル設計のレビュー ・JP Core / JP-CLINS 準拠確認 ・constitution.md 管理	FHIR R4 の深い知識 JP Core / JP-CLINS の理解 FSH 記述経験	△ 構造設計は代替困難 説明文・サンプルは代替可
医療情報専門家	・医科様式の内容解釈 ・各項目の臨床的意味の解説 ・リソース選択への助言 ・完成物の内容確認	対象様式の業務知識 医療情報の構造化経験 （FHIR 知識は不要）	△ 専門知識は代替困難 定型解釈の支援は可
FSH 実装エンジニア	・FSH コードの実装・修正 ・SUSHI エラーの解析・修正 ・サンプルインスタンス作成 ・IG Publisher の実行	FSH / SUSHI の基本知識 Git 操作 Claude Code 操作	◎ AI が大部分を担当可 エラー修正は特に得意
プロジェクト管理者	・フェーズ管理・進捗管理 ・SpecKit ワークフロー運営 ・成果物レビュー調整	プロジェクト管理の基礎 SpecKit の操作方法	○ タスク管理は AI 支援可 最終判断は人間

4.2 スキルレベル別の活用スタイル

スキルレベル	AI の活用方法	人間の関与ポイント
上級者（FHIR 熟練者）	・プロファイル構造の骨格を人間が設計 ・説明文・サンプルの生成に AI を活用 ・エラー解析の迅速化に AI を使用	・設計判断の全般 ・品質レビューの高速化 ・AI 出力の検証
中級者（FHIR 基礎知識あり）	・SpecKit 全フローで仕様〜実装を管理 ・FSH は AI と対話しながら段階的に作成 ・SUSHI エラーは AI に貼り付けて修正確認	・リソース選択の判断 ・様式項目の解釈 ・最終レビュー
初級者（FHIR 未経験）	・全ての FSH 記述を AI に委任 ・SpecKit のフルフローを遵守 ・constitution.md の規則を AI に遵守させる	・様式の業務的な解釈 ・AI 出力の妥当性確認 ・専門家への確認依頼

4.3 専門家が居る場合と居ない場合の比較

観点	FHIR 専門家あり + AI	FHIR 専門家なし + AI
実装速度	高速（設計即実装）	中程度（対話で段階的に進む）
設計品質	高い（専門判断 + AI 補助）	普通（AI に依存、検証重要）

コスト	人件費 + AI 利用コスト	AI 利用コスト + レビュー外注コスト
リスク	低い	設計ミスのリスクあり（レビュー必須）
適用場面	大規模・品質重視の IG 作成	FHIR 知見がない組織の初期導入

【専門家との速度比較について】
FHIR 熟練の専門家がいる場合、生成 AI による速度向上は限定的になる可能性があります。
生成 AI の最大の価値は「FHIR の知見がないチームでも IG 作成に参加できること」にあります。
ただし、人間によるレビュー（特にリソース選択・マッピングの妥当性確認）は必ず必要です。

5. 開発ワークフロー詳細

5.1 SpecKit ワークフロー

コマンド	成果物	目的・効果
/speckit.constitution	constitution.md	プロジェクト原則・命名規則・品質ゲートを定義
/speckit.specify	spec.md	自然言語での要件を構造化した仕様書に変換
/speckit.clarify	Q&A + spec.md 更新	仕様の曖昧点を最大 5 問で明確化
/speckit.plan	plan.md	実装計画を策定。依存関係・リスクを明示
/speckit.tasks	tasks.md	タスクを原子的単位に分解
/speckit.implement	FSH / Markdown	tasks.md に従い実装を実行
/speckit.analyze	整合性レポート	仕様・計画・タスクの整合性を事前検証
/speckit.checklist	チェックリスト	実装完了後の品質確認項目を自動生成
/speckit.taskstoissues	GitHub Issues	tasks.md から Issue を自動生成

5.2 5 フェーズ開発モデル

フェーズ	名称	主な作業	成果物	AI 貢献度
Phase 1	様式分析	PDF から項目抽出・TSV 整理	extracted/*.tsv *-description.md	○（補助）
Phase 2	FHIR マッピング	FHIR 要素へのマッピング確定	FHIR マッピング仕様書	○（補助）
Phase 3	FSH 実装	プロファイル・CS・VS・サンプル実装	input/fsh/*.fsh	◎（主担当）
Phase 4	ビルド検証	SUSHI 0 エラー確認	fsh-generated/resources/	◎（主担当）
Phase 5	IG 公開	IG Publisher HTML 生成	output/	△（補助）

【2 段階アプローチの推奨】
Phase 1（PDF 解析）と Phase 3（FSH 実装）は独立したセッションで実施すること。
同一セッションで混在させると AI のコンテキストが肥大化し、品質が低下します。

5.3 Constitution による品質管理

原則	内容	品質ゲート
FHIR R4 準拠	全リソースは FHIR R4（4.0.1）準拠	SUSHI ビルド検証
JP Core 優先	JP Core / JP-CLINS プロファイルを積極利用	レビュー
SUSHI ビルドゲート	0 エラーでのコンパイルを必須	SUSHI ビルド自動検証
インクリメンタル検証	フェーズ単位でチェックポイントを設定	フェーズ完了判定
AI 支援開発	Claude Code + MCP サーバのワークフロー標準化	ワークフロー準拠確認
Composition セクション設計	細粒度の意味分離、最大 3 階層	設計レビュー
様式スコープ分離	独立した様式文書ごとに Composition を分離	設計レビュー
文書抽出スコープ	ファイル I/O のみ（ツール/API 開発は対象外）	スコープ判定
CodeSystem バインディング	ValueSet 特化型 Observation プロファイルを使用	レビュー

6. 生成 AI の限界

生成 AI は FHIR IG 作成において大きな可能性を持つ一方、固有の技術的・業務的限界があります。本プロジェクトで実際に確認した限界を系統的に整理します。

6.1 技術的な限界

6.1.1 コンテキストウィンドウの制約

問題	発生状況	回避策
初期指示の忘却	セッションが長くなると命名規則を無視した出力が増える。	.claudeIgnore で不要ファイルを除外。Serena MCP で必要部分のみ参照。
矛盾した出力	前半の定義と後半の実装が矛盾することがある。	長い作業は複数セッションに分割し、開始時に制約を再提示。
大容量ファイルの処理劣化	fsh-generated/ 等を読み込むとコンテキストが汚染される。	.claudeIgnore で除外対象を明示設定。

6.1.2 ハルシネーション（幻覚生成）

問題	発生状況	回避策

存在しないプロファイル URL の生成	JP Core や JP-CLINS に存在しない URL を AI が生成。	SUSHI コンパイルで必ず検証。重要 URL は公式ドキュメントで確認。
古いバージョンの仕様への準拠	知識カットオフ以前のプロファイルや URL を参照。	最新バージョンを明示指定し、packages/ を参照させる。
存在しない FSH 構文の生成	FSH の構文ルールを誤った形で生成。	SUSHI コンパイル検証で検出。エラー時は AI に修正させる。

6.1.3 セッション間の記憶の欠如

問題	回避策
前セッションで修正したルールを次セッションで再び違反。	constitution.md にルールを明記し、セッション開始時に AI が参照する設計とする。
以前のフェーズの設計判断を次フェーズで覚えていない。	Serena MCP のプロジェクト記憶機能で重要な設計判断を保存。
前フェーズで発覚した問題を繰り返す。	spec.md や plan.md に「既知の課題」セクションを設け AI に参照させる。

6.2 section.entry と ValueSet バインディングの問題

本プロジェクトで最も頻繁に確認された設計上の問題が、section.entry の参照先が汎用プロファイル（JP_Observation_Common_F50）にとどまり、ValueSet への正式バインディングが行われない傾向です。

【初期実装（AI が生成しやすいパターン）】

```
// Composition の adl セクション定義
* section[adl].entry only Reference(JP_Observation_Common_F50)
// → ValueSet バインディングなし。どんな値でも入れられる汎用設計。
```

【正しい実装（ValueSet バインディングあり）】

```
// 専用プロファイル JP_Observation_ADL_F50 を定義
Profile: JP_Observation_ADL_F50
* value[x] from JP_VS_F50_ADLLevel (required)  // ValueSet を正式バインド

// Composition セクションは専用プロファイルを参照
* section[adl].entry only Reference(JP_Observation_ADL_F50)
```

原因	詳細
実装の省力化傾向	汎用プロファイルへの参照は実装量が少なく、AI が「省エネ」の選択をしやすい。
ValueSet の事前定義が必	正式バインディングには CodeSystem と ValueSet の先行定義が必要。AI が

要	全体を見通せない場合、後回しにしがち。
指示の粒度不足	「Observation を追加せよ」だけでは専用プロファイル作成まで踏み込まない。「ValueSet required バインディングを含む専用プロファイルを作成せよ」と明示する必要がある。

【設計指針：ValueSet バインディングの完結まで実装せよ】
section.entry の設定と ValueSet バインディングを同一フェーズで完結させることを推奨。
汎用プロファイルへの参照は「暫定」であり、最終的には全セクションが
専用プロファイル + ValueSet バインディングを持つ設計にすべきです。
tasks.md には「専用プロファイル作成」「CodeSystem/ValueSet 定義」「entry 更新」を
個別タスクとして明示的に記載すると、AI が省略せず実装する確率が上がります。

6.3 医療・業務ドメイン知識の限界

問題	影響	回避策
リソース選択の誤り	Observation か Condition かなど臨床的判断を誤ることがある。	FHIR アーキテクト・医療情報専門家によるレビューを必須とする。
否定形の実装ミス	「〜でない場合」を逆に実装することがある。	否定形は肯定形に書き換えて指示。実装後にサンプルで確認。
チェックボックスの混同	単一選択を複数選択（component）として実装することがある。	TSV 抽出時に「単一/複数」列を明示し、AI への指示に含める。
PDF 解析の限界	表の「その他」列等の補助列を見落とす。業界用語を誤認識。	医療情報専門家が PDF 解析に参加し、事前に表構造の説明文を整理。

6.4 設計・品質に関する限界

限界	詳細	対策
意味的妥当性の検証不可	SUSHI は構文エラーを検出するが、リソースの意味的妥当性は検出しない。	FHIR アーキテクトと医療情報専門家によるレビューを必須とする。
繰り返しミス	同一セッション内で特定の命名規則違反等を繰り返す。	ミスを発見したら即座に明示指摘し、constitution.md に記録。
CodeSystem の標準化	プロジェクト固有の CodeSystem は標準化団体の調整なしには標準とならない。	設計時から標準団体との調整を計画に含める。

【SUSHI の 0 エラーは必要条件であり十分条件ではない】
SUSHI コンパイル成功でも以下は保証されません：
① FHIR リソースの意味的妥当性
② ValueSet バインディングの完全性（section.entry から末端まで）
③ サンプルインスタンスの医療的内容妥当性

これらは人間によるレビューでのみ確認できます。

【生成 AI の現時点における位置づけ】
生成 AI は「FHIR IG 作成の主担当者」ではなく「高度な補助ツール」です。
AI が担当：FSH 記述・エラー解析・文書生成・サンプルインスタンス作成
人間が担当：設計判断・リソース選択・ValueSet 設計・医療的妥当性確認・最終レビュー

7. 生成 AI 活用時の注意事項

カテゴリ	注意事項	対策
コンテキスト管理	セッションが長くなると命名規則や設計方針を「忘れる」。	.claudeIgnore 設定。長い作業は複数セッションに分割。
コンテキスト管理	PDF 解析と FSH 実装を同一セッションで行うと品質低下。	フェーズを跨ぐ際はセッションリセット。
ValueSet バインディング	section.entry を汎用プロファイルで終わらせ、ValueSet バインディングまで実装しない傾向。	tasks.md に「専用プロファイル作成」「ValueSet required バインディング」を明示的に記載。
指示の明確化	チェックボックスの単一/複数選択を事前に明示。	TSV 抽出時に選択種別列を設ける。
指示の明確化	否定形の条件指定は混乱しやすい。	肯定形に言い換えるか、実装後に必ず確認。
検証・レビュー	SUSHI の 0 エラーは必要条件。意味的妥当性は別途レビュー必要。	FHIR アーキテクトによる設計レビューを全フェーズに組み込む。
検証・レビュー	AI のチェックは SUSHI 成功確認まで。IG Publisher は人間が管理。	IG Publisher は Docker を用いて人間が実行。

8. Git コミット履歴から見た工夫と改善点

本プロジェクトの Git コミット履歴を分析し、開発過程での主要な改善点を整理します。

8.1 改善経緯の概要

Feature	コミット要約	改善内容
001〜008	様式 50 FSH 初期実装	全セクションで汎用 JP_Observation_Common_F50 を参照する形からスタート。
009	JP-CLINS プロファイル採用 .claudeIgnore 追加	独自定義の Patient/Practitioner を標準プロファイルへ切替。コード量削減と標準準拠を同時達成。fsh-generated/output/等を AI 解析対象から除外。
010	Observation を分割	汎用 Observation→ADL・ケアレベル・入浴・移動等 10 種の専用プロファイルに分割。section.entry を専用プロファイルに切替。

011	Observation の構成見直し	Component Slicing 導入。複数の観察値を持つ Observation を適切に構造化。Constitution 更新（v1.8.0/v1.9.0）。
012	診療情報提供書参照追加	DocumentReference を用いて外部文書を参照するパターンを新規追加。
013	看護上の問題とケア方法を分離	看護上の問題（Condition）とケア方法（Observation）を別リソースに。
014	説明・希望目標をテキスト型へ	自由記述項目は Observation ではなく section.text で表現する設計に変更。
015〜022	各テーマ別の専用プロファイル追加	家族構成・要介護認定・清潔・排泄・睡眠/精神・運動感覚と段階的に追加。各フェーズで CodeSystem/ValueSet を正式バインディング。

8.2 主要な設計変更と学習

8.2.1 JP-CLINS プロファイルへの移行（Feature 009）

変更前	変更後	効果
JP_Patient_F50（独自定義）	JP_Patient_eCS（JP-CLINS 提供）	コード量削減・標準準拠・相互運用性向上
JP_Practitioner_F50（独自定義）	JP_Practitioner_eCS（JP-CLINS 提供）	同上

【学習ポイント】
JP Core / JP-CLINS に既存プロファイルがあれば必ず再利用する。
独自プロファイル定義は「既存で表現できない」場合のみとする。

8.2.2 Observation の分割と専用プロファイル化（Feature 010/011）

分割前	分割後
全セクション → JP_Observation_Common_F50（ValueSet バインディングなし）	section[adl] → JP_Observation_ADL_F50（JP_VS_F50_ADLLevel にバインド） section[excretion] → JP_Observation_Excretion_F50 … セクションごとに専用プロファイル

【学習ポイント】
「汎用プロファイルに全部まとめる」は SUSHI 的に成立するが IG としての品質が低い。
生成 AI は明示的に指示しない限り汎用プロファイルで済ませる傾向がある。
constitution.md および tasks.md に専用プロファイル化を義務として記述することが重要。

8.2.3 .claudeIgnore の導入（Feature 009）

除外ディレクトリ	理由
fsh-generated/	SUSHI コンパイル出力。大容量かつ自動生成。
output/	IG Publisher HTML 出力。大容量かつ自動生成。
docker/	Docker ビルド設定。FSH 実装時は参照不要。
input-cache/ / temp/ /	IG Publisher 一時ファイル群。

template/	

【学習ポイント】
.claudeIgnore は開発初期から設定すべき。特に fsh-generated/ は早期除外が必須。

8.2.4 自由記述項目の section.text への変更（Feature 014）

変更前	変更後	理由
JP_Observation_Common_F50 valueString で自由記述	Composition section.text ナラティブとして格納	自由記述は FHIR のナラティブで表現するのが適切。 Observation の不要な生成を削減しシンプル化。

【学習ポイント】
構造化データ（コード値・数値・日付等）→ Observation（専用プロファイル + ValueSet）
自由記述（テキスト説明・目標・メモ等）→ section.text（Composition ナラティブ）

8.2.5 Constitution のバージョン管理

バージョン	主要追加原則	きっかけ
v1.6.0	初期原則群	プロジェクト開始時の基盤整備
v1.7.0	ドキュメント抽出スコープ原則	スコープ拡大を防ぐため
v1.8.0	CodeSystem バインディング戦略	汎用プロファイルで止まる問題への対処
v1.9.0	Composition セクション設計ガイドライン	セクション設計の一貫性確保
v1.10.0	バインディング詳細規則の強化	Feature 020〜022 での設計精度向上

【学習ポイント】
問題発生時はその場限りの修正で終わらせず、constitution.md に原則として追記する。
これにより後続フェーズ・後続様式での同一問題の再発を防止できる。

8.2.6 PDF 項目抽出の教訓

問題	根本原因	改善策
テーブルの「その他」列を複数回見落とし	主要な数値列にのみ着目し補助列を未確認。AI の「主要項目への注目バイアス」。	全列見出しを列挙→全行見出しを列挙→全行×列の組み合わせを網羅チェック。

9. 効果と考察

9.1 生成 AI 活用の効果（実績）

観点	具体的な効果
FSH 実装の自動化	4,400 行の FSH コードの大部分を AI が記述。
SUSHI エラー解析	エラーログを AI に渡すだけで修正方針を提示。修正ループの大幅高速化。
SpecKit による一貫性	22 フェーズにわたる開発で設計一貫性を維持。
文書生成の自動化	仕様書・計画書・説明文・サンプルインスタンスを AI が自動生成。

FHIR 未経験者の参加	FHIR 未経験メンバーが設計議論・レビューに集中できる体制を実現。

9.2 様式文書への提言

【様式文書の改善提案】

【最重要】様式文書を FHIR リソースを意識した構成に見直すと AI 解析精度が大幅向上。
① 各項目にデータ型（文字列/数値/コード値/真偽値）を明記する。
② チェックボックスに「単一選択」「複数選択可」の種別を明示する。
③ 選択肢の値に CodeSystem 候補を記載する。
④ リソースごとの説明文書（データ定義書）を整備する。
⑤ データの持ち方（1 項目 1 リソース/複数項目を 1 リソース）を記述する。
⑥ 否定形の条件は肯定形に書き直すか、判定ルールを明記する。

10. ディレクトリ構成

ディレクトリ / ファイル	内容・役割
input/fsh/	FSH ソースコード。全開発の中心。
specs/{機能名}/	spec.md / plan.md / tasks.md（SpecKit 成果物）。
.specify/memory/	constitution.md・fsh-style-guide.md 等のプロジェクト記憶。
.serena/memories/	Serena MCP が管理する教訓・設計判断記憶。
extracted/	PDF から抽出した TSV および Markdown。Phase 1 成果物。
forms/	元の PDF 様式ファイル。
fsh-generated/	SUSHI コンパイル出力。.claudeIgnore で除外推奨。
output/	IG Publisher 出力。.claudeIgnore で除外推奨。
docker/	IG Publisher ビルド用 Docker 環境。
packages/	FHIR パッケージキャッシュ。.claudeIgnore で除外推奨。
docs/	プロジェクト説明文書（本書含む）。
.claudeIgnore	AI 解析対象外ファイルの指定。

11. 今後の展開

項目	内容	優先度
他の医科様式への適用	様式 50 のワークフローを他の様式に段階的に適用。	高
リソース説明文書の整備	各プロファイルのデータ定義書を作成。	高
CodeSystem の標準化検討	標準化団体との調整を検討。	中
診療情報提供書 IG の参照	JP-CLINS の診療情報提供書 IG との整合性確認。	中
細かな修正の逆輸入	後続様式で判明した改善を様式 50 に反映。	中
汎用プロファイル参照の解消	残存する汎用参照を全て専用プロファイルに置換。	中

12. 参考・関連情報

リソース	URL / 備考
FHIR R4 仕様書	https://hl7.org/fhir/R4/
JP Core	https://jpfhir.jp/fhir/core/
JP-CLINS	https://jpfhir.jp/fhir/clins/
FHIR Shorthand (FSH)	https://build.fhir.org/ig/HL7/fhir-shorthand/
SUSHI	https://fshschool.org/
IG Publisher	https://github.com/HL7/fhir-ig-publisher
Claude Code	https://www.anthropic.com/claude-code
HL7 International	https://www.hl7.org/