Wagby 11

Overview

ローコード × AI で、未来のシステム開発を加速する

2026 年 7月

株式会社ジャスミンソフト

AIを使った開発の課題

注意が必要！

1

プロンプトから完全に動作する
アプリケーションを開発することはできる。

プロンプト例
「顧客管理システムを作成してください。
顧客の登録・一覧・検索・編集・削除ができる Web アプリケーションにしてください。」

顧客一覧

顧客 ID	会社名	利用者	電話番号
1001	株式会社 ABC	山田 太郎	03-xxx-xxxx
1002	株式会社 DEF	鈴木 花子	03-yyy-yyyy
1003	株式会社 GHI	佐藤 次郎	03-zzz-zzzz

動作するアプリが作れる

2

しかし
同じプロンプトであっても
生成されるソースコードは大きく異なり、
一貫性がない。

プロンプト例
「顧客管理システムを作成してください。顧客の登録・一覧・検索・編集・削除ができる Web アプリケーションにしてください。」

生成結果 A

```
// 構成・技術スタック
- React
- Node.js/Express
- MongoDB
- REST API 構成
...
```

生成結果 B

```
// 構成・技術スタック
- Vue.js
- Laravel (PHP)
- MySQL
- MVC 構成
```

生成結果 C

```
// 構成・技術スタック
- Spring Boot 3
- Java 21
- PostgreSQL
- クリーンアーキテクチャ
```

同じプロンプトでも、構成・技術・設計・実装が大きく異なる

生成結果がバラバラで一貫性がない！

3

一回作成するならよいが、
継続して保守することが難しい。

- コードの構成や設計が毎回異なり、理解・把握に時間がかかる
- 命名規則や構造が不統一で、修正や機能追加が困難
- 同じ機能でも実装方法が異なり、修正箇所の特定が難しい
- 引き継ぎやチーム開発が難しく、属人化しやすい

継続保守が困難！

課題への対応方法

設計情報とアーキテクチャに基づき、テンプレートを活用して一貫性のある開発を実現します

1 統一されたフォーマットで設計情報を与える

会話ベースの開発ではなく、統一されたフォーマットで記述された設計情報を与える。

設計情報（例）

```
entities:
 - name: 営業日報
   attributes:
     - name: 日報日付
       type: date
     - name: 担当者 ID
       type: string
     - name: コメント
       type: text
relations:
 - name: 日報コメント
   type: one-to-many
...
```

YAML / JSON / DSL など

2 アーキテクチャ方針を与える

設計情報をどのような構成でプログラムにするかという方針（アーキテクチャ）を与える。

アーキテクチャ方針（例）

- プレゼンテーション層 (Web UI / REST API)
- アプリケーション層 (Service / UseCase)
- ドメイン層 (Entity / Repository)
- インフラストラクチャ層 (DB / 外部連携 / 共通基盤)

・レイヤードアーキテクチャ
・命名規約、パッケージ構成のルール
・技術スタック、使用ライブラリ など

3 テンプレートコードを自動生成（業務処理部は空白）

設計情報とアーキテクチャに対応したテンプレートコードを自動生成する。ただし業務処理部分だけは空白とする

生成されるテンプレートコード（例）

```
src/
└── com/example/
    ├── domain/
    │   └── 営業日報 .java
    ├── repository/
    │   └── 営業日報 Repository.java
    ├── service/
    │   └── 営業日報 Service.java
    └── controller/
        └── 営業日報 Controller.java
    ...
```

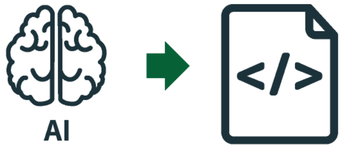
営業日報 Service.java

```
public class 営業日報 Service {
  @Autowired
  private 営業日報 Repository repo;

  public 営業日報 create( 営業日報 req) {
    // ===== 業務処理部（実装は空白）=====
    // TODO: 業務処理を実装してください
  }

  public void update(Long id, 営業日報 req) {
    // ===== 業務処理部（実装は空白）=====
    // TODO: 業務処理を実装してください
  }
  // ... その他のメソッドの雛形
}
```

4 AI が業務処理コードを生成（コードの揺れを抑える）

AI は、設計情報に応じた業務処理コードを生成する。テンプレートコードに合わせることで、コードの揺れを抑える。

AI による業務処理コード生成



テンプレートに合わせて実装

- 命名規約・構成が統一
- 可読性・保守性が向上
- 継続開発・チーム開発を支援

実現すること

設計情報 + アーキテクチャ + テンプレートで、
一貫性のある高品質なコードを効率的に生成・保守できる！


一貫性


生産性向上


保守性向上

Wagby11 の考え方

発想を**逆転**し、AI の力を最大限に活かす！

これまでの Wagby ありきのアプローチ

AI に Wagby のルールを説明し、Wagby 流のコードを作成させる

Wagby のルールを AI に説明

- 業務ルール
- データモデル
- 権限管理
- 命名規則
- 設計方針
- 実装パターン
- コーディング規約

…

AI が理解・解釈

AI

理解の揺れや
解釈のばらつきが発生

従来のコードを生成

現行 Wagby の制約に
縛られたまま

Wagby11 のアプローチ（発想を逆転）

AI が学習済みの膨大な知見を活かす形で、
Wagby の**設計思想**を組み込む

AI が学習済みの膨大な知見（インターネット上の豊富な情報）

- 最新の技術情報・ベストプラクティス
- オープンソースの実装例
- 設計パターン・アーキテクチャ
- クラウド・インフラの知見
- 開発コミュニティの知見・ノウハウ

…など、無限の知見

＋
組み込み

Wagby の設計思想（組み込む要素）

- 業務ルール
- データモデル
- 権限管理
- 命名規則
- 設計方針
- コーディング規約

…

最新アーキテクチャを活用した Wagby のコード生成

一貫性が高く、
保守性・拡張性に優れた
高品質なコード

Wagby11 の目指す姿

最新アーキテクチャを活用した、AI 時代の Wagby

最新技術の活用
AI が学習済みの最新技術・アーキテクチャを積極的に取り入れる。

設計思想の継承
Wagby の設計思想を核として、一貫性のある開発を実現。

高い保守性
統一された構成・規約により、長期的な保守・継続開発を容易にする。

生産性の向上
AI の力を最大限に活用し、開発スピードと品質を大幅に向上。

未来への対応
変化の激しい技術トレンドやビジネス要求に柔軟かつ迅速に対応。

AI の力 × Wagby の設計思想 = これからのローコード開発プラットフォーム

新・設計思想「ローコード × AI」

設計情報を中心に、最新アーキテクチャと AI の力を組み合わせ、人間を介さない完全自動化を実現します。

1 1 つの設計情報からすべてのソースコードを生成する Single Source of Truth（SSoT）

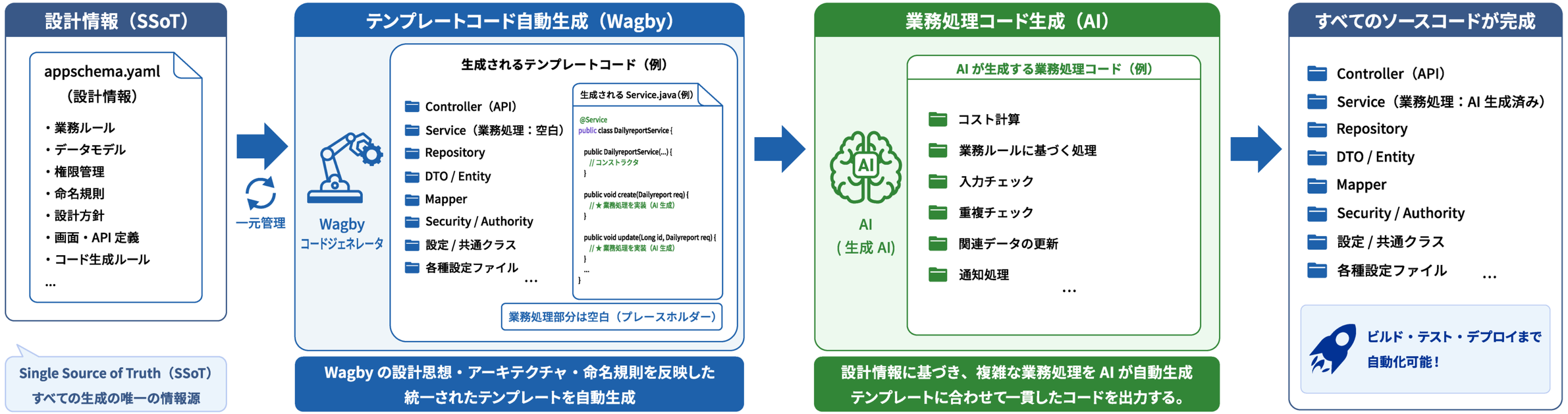
設計情報を一元管理し、すべてのコードやドキュメントを自動生成する。変更は設計情報を修正するだけで全体に反映される。

2 Java21 / Spring Boot 3 / OpenAPI Spec など AI が学習済みの知見を活かす

最新・標準技術を採用し、AI が持つ膨大な知見を最大限に活用する。高品質で保守しやすいコードを生成する。

3 Wagby のテンプレート + AI の業務処理生成によるハイブリッドアプローチで完全自動化を実現

テンプレートコードは Wagby が自動生成し、複雑な業務処理を AI が生成する。人間を介さない開発プロセスを実現する。



このアプローチの効果

一貫性のあるコード
設計情報 + テンプレートでコードの揺れを最小化

高い保守性
統一された設計・構造で保守・継続開発が容易

開発スピード向上
統一された構成・規約により、長期的な保守・継続開発を容易にする。

属人化の排除
標準化されたプロセスで誰でも同品質を実現

最新技術の活用
AI が学習した最新知見を積極的に取り入れる

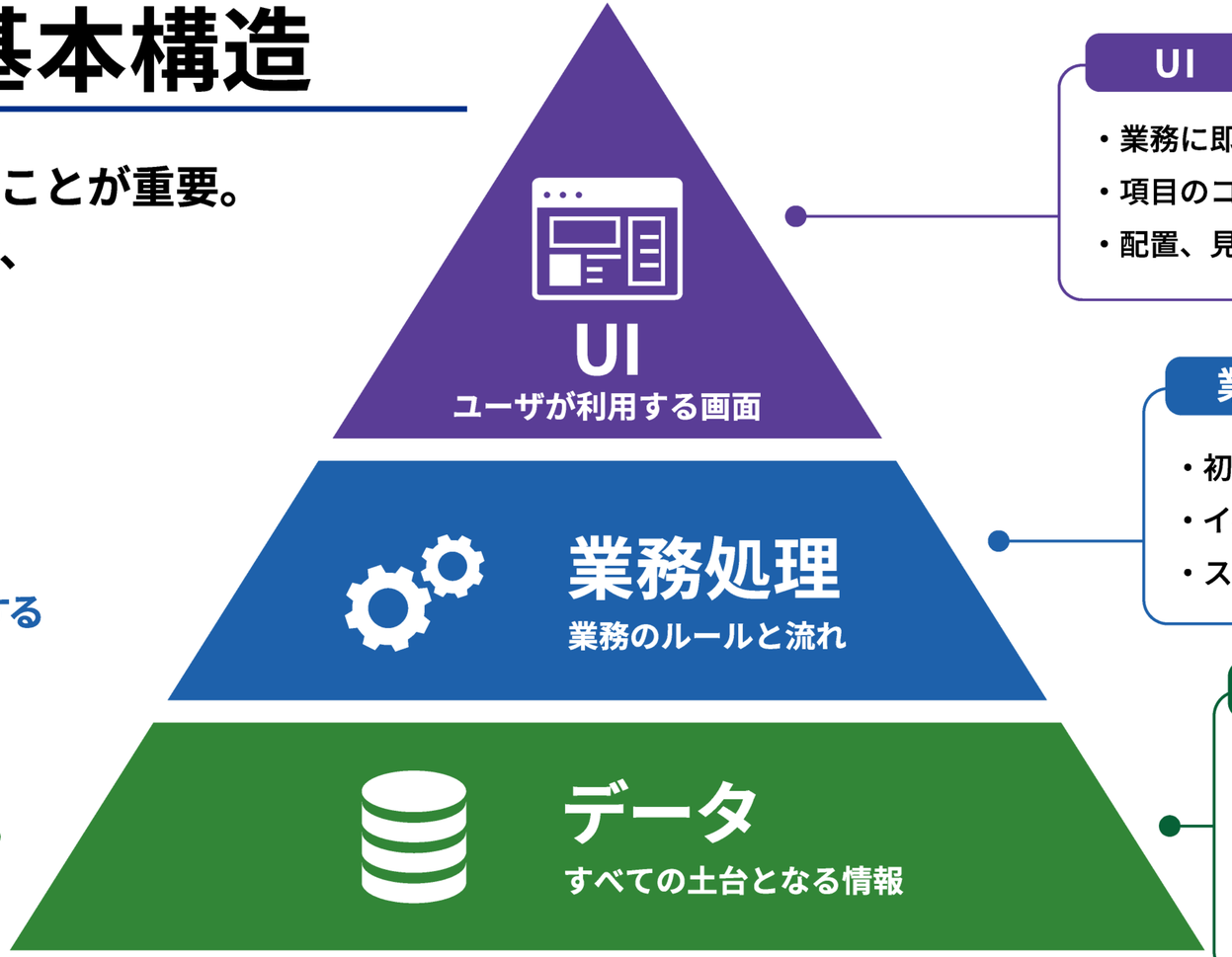
設計情報の基本構造

はじめにデータを整備することが重要。
データがあれば業務処理が、
そのあとで UI を設計する。

❸ UI を設計する
ユーザが使う
画面・操作

❷ 業務処理を定義する
データを活用する
ルールと流れ

❶ データを整備する
すべての土台



UI
- 業務に即した画面フロー
- 項目のコンポーネント
- 配置、見栄え

業務処理
- 初期値設定
- イベントに応じた値の更新
- ステータス管理

データ
- マスタ / トランザクション / 選択肢
- DB 保存項目と導出項目
- データ同士の関連
- 利用するユーザの「ロール」と「権限」に対応したアクセス制御

成功の鍵は「データの整備」
正確で一貫性のあるデータ定義が、
業務処理と UI の品質を左右します。

❶ データを整備する
正確・一貫・
関連性の確保

→

❷ 業務処理を定義する
ルール・イベント・
ステータスを設計

→

❸ UI を設計する
画面フロー・配置・
見栄えを最適化

データ → 業務処理 → UI の順序で設計することで、保守性・拡張性の高いシステムを実現します

AIの利用 [1]　設計情報の作成支援

① AI に設計情報の「構造」をあらかじめ伝えておく

例）エンティティ、項目、業務ルール、
　　、画面、権限、検索条件　など

② AI に、どういう順番で設計情報を作成するかという考え方も伝えておく。

全体から詳細へ、データから画面へ、
段階的に作成していくことを共有する。

③ AI が会話の主導権を握り、開発者（人間）が答える形で設計情報を埋めていく。

質問に答えていくことで、
設計情報が完成していく。

設計情報を作成する流れ

① 全体設計	② データ設計	③ 権限・検索設計	④ 業務ルール設計	⑤ 画面設計
AI からの質問 「このシステムで実現したい業務は何ですか？」	**AI からの質問** 「必要なマスタやトランザクションは何ですか？」	**AI からの質問** 「誰がどこまでのデータを、どのように検索しますか？」	**AI からの質問** 「登録や更新時のルールはありますか？」	**AI からの質問** 「どのような画面が必要ですか？」
開発者が回答 「○○業務を効率化したい」	**開発者が回答** 「顧客、商品、受注、明細など」	**開発者が回答** 「ロールごとの権限、検索条件、スコープ」	**開発者が回答** 「必須チェックや計算ルールなど」	**開発者が回答** 「一覧、詳細、登録、検索など」

対話を繰り返しながら、設計情報を埋めていく。

完成した設計情報（Single Source of Truth）
この設計情報をもとに、すべてのソースコードを自動生成

AI に「構造」と「作り方」を伝え、人間が業務の内容を答えていくことで、精度の高い設計情報を効率的に作成できる！

AIの利用[2]　コード生成

AI を活用して、設計情報に基づいた業務アプリケーションのソースコードを効率的に生成・実装する。

1 AI に Wagby11 の構造を伝えておく

レイヤ構成、パッケージ構成、命名規則、共通基盤との関係、コード生成ルールなどを事前に共有する。

2 AI に必要な情報とカスタマイズガイドを渡す。

設計情報　Open API (YAML)　基本コード (Spring Boot 3)

- 設計情報（appschema.yaml 等）
- OpenAPI が生成した Java コード
- Wagby が生成した Spring Boot 3 ベースの基本コード
- カスタマイズガイド（実装ルール、方針、注意点）

3 設計情報に記載された「業務処理」を、すでに用意されたコードに実装する。

- 初期値設定、イベント処理、更新ロジック、ステータス管理、バリデーションなど
- 既存の基本コードに追記、拡張する形で実装
- AI が生成するコードがぶれないよう、ルールと枠組みを明確にする

AI によるコード生成の流れ

AI へ渡す情報

設計情報	Open API	基本コード	カスタマイズガイド
appschema.yaml など	・openapi.yaml ・OpenAPI が生成した Java コード	Wagby が生成した Spring Boot 3 ベースの基本コード	実装ルール、方針、注意点など

AI

AI が生成・実装するもの

業務処理の実装コード

- 初期値設定
- イベントに応じた処理
- 値の更新
- ステータス管理
- バリデーション　など

Wagby11 の構造（レイヤ構成の例）

Controller（API）	Service	Repository	Entity（JPA）	Database
・REST API エンドポイント ・リクエスト / レスポンス ・バリデーション	・業務ロジック ・トランザクション制御 ・権限チェック	・データアクセス ・Spring Data JPA	・JPA エンティティ ・マッピング定義	・RDB（PostgreSQL など）

AI と開発者の協働により、品質の高いコードを一貫した構造のもとで効率的に生成・実装できる！

構造・ルールを共有することで、AI が生成するコードがぶれない

基本コードを活用することで、品質・保守性を確保

業務処理の実装に集中でき、開発スピードが向上

Wagby 生成コード / AI 生成コードの関係

Wagby 生成コードをベースに、AI 生成コードは限定された領域を担当

Wagby が自動生成する領域

共通性が高く、
パターン化できる処理を
Wagby が自動生成する。
構造が統一されており、
安定・保守しやすい。

Wagby 生成コード

モデルごとの基本 CRUD 処理

- 検索・一覧
- 登録
- 更新
- 削除
- 共通バリデーション
- 権限チェック
- ログ出力 など

既存の構造・インターフェースの上に、
AI が必要な処理を実装

担当領域を限定することで、
ぶれないコード生成を実現

AI 生成コード

計算 / 複雑な更新

- 計算処理
- 複雑なビジネスロジック
- 複数項目の連動更新
- 外部連携処理 など

AI が実装する領域
（限定されたカスタム処理）

設計情報に基づく
業務固有の処理のみを
AI が実装する。
範囲が明確なため、
コードのぶれを最小化。

Wagby 生成コードを前提とした AI のコード生成

- 構造が統一されているため、
  AI が理解しやすい
- 担当領域を限定することで、
  コードがぶれない
- 保守性・拡張性が高く、
  継続的な開発が可能

Wagby11 基本構成

設計情報を起点に、すべてのコンポーネントを自動生成し、
様々なクライアントから利用できる API 中心の構成を実現します。

設計情報（appschema.yaml）

appschema.yaml

- AI と会話しながら作成
- 開発者による直接修正もできる

これが
SSoT
（Single Source of Truth）
となる。

OpenAPI 仕様

設計情報から自動生成される。REST API の仕様となる。

REST API サーバ

設計情報の「データ」「業務処理」を担当する。

他のサービスから REST API として利用される。

Web UI（対人向け）

対人向けの
ユーザーインターフェース

REST API クライアントとして動作する。

MCP Server（対 AI 向け）

AI エージェントが利用するためのサーバ。

REST API クライアントとして動作する。

利用例
(REST API)

- モバイルアプリ
- 外部システム
- クラウドサービス

利用者
(REST API)

エンドユーザー
(ブラウザ)

利用者
(REST API)

AI エージェント
（対話・自動処理）

1 設計情報（appschema.yaml）

AI と会話しながら作成でき、開発者による直接修正もできる。
これが SSoT（Single Source of Truth）となる。

2 OpenAPI 仕様

自動生成。REST API 仕様となる。

3 REST API サーバ / Web UI / MCP Server

- 設計情報の「データ」「業務処理」を担う。
  他のサービスから利用される
- Web UI は対人向け。REST API クライアントとなる
- MCP Server は対 AI 向け。REST API クライアントとなる

ポイント

- 設計情報（SSoT）を起点にすべて自動生成
- API 中心の構成で、様々なクライアントから利用可能
- AI にも人にも最適なインターフェースを提供

Coming Soon...

Wagby 11
Next Generation Low-Code Platform

AI 時代のローコード開発、いよいよ始まります。

想像をカタチに。未来のシステム開発へ、さあ出発しよう。

1. Preview 版の提供

8 月下旬（予定）

現行 Wagby ユーザ
（Corporate ライセンス）へ、
8 月下旬に Preview 版を
提供します。

2. 正式版の発表

2026 年 11 月 12 日（木）

正式版は、
Wagby Developer Day 2026
で発表します。

2025 年度ポスター

3. アップグレード特別キャンペーン

現行 Wagby ユーザ
（Project / Unlimited / Team ライセンス）向け、
Corporate ライセンス アップグレード
特別キャンペーンを開始します。

詳細は Wagby 販売パートナーへ
お問い合わせください。

今が
チャンス！

Wagby11 が、あなたのビジネスを次のステージへ。