

プログラミング・ハーネス

Scalaを中心に、AIの実装と修正を支える

既存機能の再利用

型で利用条件を明確にする

補完コードの構造

合成規則と検査で支える

再利用と補完コードの役割

既存機能を利用

AIが独自に設計する範囲を減らす



追加コードの構造を指定

値・状態・失敗の扱いを揃える



検査で適合を確認

AIの判断と修正の負担を抑える

実装の構造と操作を定め、誤りや逸脱を抑える。

モデル・生成・補完・実行の分担

CML → Textus

実行可能なモデルを直接動かす



AIによる補完

連携コードと業務固有の処理



Cozy → 型付きAPI

入出力と接続条件を明示する



共通の実行基盤

保存・通信・実行管理を担う



型の情報が再利用とグルーコードの作成を支える。

値の意味と扱う種類を型に表す

ドメインの型

顧客番号と注文番号を区別



代数的データ型 (ADT)

結果の種類と場合分けを定義

コンパイラの診断

取り違い・接続不一致・扱い漏れを検出

入力の範囲は値の生成時に検査し、専用の型で条件を引き継ぐ。

型による保証の実務的な効果



カーリー＝ハワード対応を背景に活用する。業務全体の証明ではない。

リアクティブな処理と合成規則

必要な特性

応答性・耐障害性・弾力性・メッセージ駆動



共通の実行機構

非同期実行・通信・障害対応・資源管理



業務コードの合成

型付き操作を依存関係・独立性に沿って接続

実行基盤の機構を、モナド・アプリカティブを通じて利用する。

純粋関数による状態依存の整理

入力を明示

時刻・価格表・規則の版を渡す



不変データで計算

同じ入力から同じ結果を得る

局所的な確認

隠れた状態への依存と変更を抑える

Scalaの型だけで純粋性は保証されない。APIと依存関係も確認する。

Consequenceによる結果と失敗の合成

関数の結果を統一

成功は型付きの値、失敗は構造化したConclusion



依存する処理：flatMap

成功した値を渡す
失敗時は後続処理を止める



独立した検査：zip系API

項目ごとの結果を組み合わせ
複数のエラーを集約する

共通の合成規則に従い、AIによる独自の分岐や集約を減らす。

操作の記述と実行管理の分離

型付きの内部DSL

保存・外部呼出しの
入力と結果を定義



Free Monad

その場では実行せず
操作の記述を組み立てる



UoW

インタプリタを通じ
共通機構で実行を管理

操作の記述にConsequenceの規則を組み込み、結果と失敗を揃える。

UoWの構成がハーネスになる理由

AIが担う記述

型付き操作を純粹に組み合わせる



共通機構が担う実行

UoWで一連の操作を管理する



分担を検査・レビューで維持

AIが個別に作り込む制御を減らす

型は接続を検査し、依存関係の検査は共通経路からの逸脱を確認する。

部品の契約と業務上の意味の確認

検査の手段	確認する条件
型検査・car-lint	値と接続、部品構造と公開契約
テスト	具体的な入力に対する結果と実際の応答
モデルに基づくレビュー	要求・設計・実装の対応と責務

異なる検査を組み合わせ、一つでは捉えられない誤りを補う。

診断を修正と再確認へつなぐ

診断を得る

コンパイル・lint・
対象を絞ったテスト



原因のある箇所を修正

モデル・生成規則・
補完コードを区別



同じ条件を再確認

原因不明や契約変更は
設計・担当者の判断へ

契約を緩めて診断を消さず、変更された契約へ実装を適合させる。

AIによる高度な手法の活用

AIがコードを記述

純粋関数型・モナドを採用しやすい



型と共通APIで支える

出力の構造と利用条件を確認する



必要な品質を確保

個別実装と修正・再レビューの負担を抑える

開発時間とAI利用コストの低減を図る。

プログラミング・ハーネス

モデルの契約を、値・計算・操作の制約へつなぐ

CMLモデル・標準API

MODEL → API → AI

Textusがモデルを直接実行し、Cozyが型付きAPI・接続構造を生成する。
AIは、モデルと共通機構にない固有の処理を補完する。

Scalaで記述する補完コード：同じ値と操作に制約を引き継ぐ

1 型付きの値

入力検査 + ドメイン型

注文IDと顧客IDを型で区別
検査済みの値をAPIへ渡す

取り違えと未検査値の
混入を抑える

2 計算の合成

不変データ + 純粋関数

純粋な計算を型で接続する
Consequenceで結果を合成

暗黙の変更と
失敗の脱落を抑える

3 操作の記述

内部DSL + Free Monad

定義済みの操作を組み立てる
UoWが共通の実行を管理

外部アクセスの処理を
共通の実行経路へ集約する

補完コード全体を、部品の契約でも検査する

CARに契約・実装・依存関係をまとめる。
コンパイラ・car-lint・cbd-supportで検査とレビューを分担する。
診断 → 原因箇所の局所修正 → 再検査

Textusの実行時制約

認可・現在の状態・資源利用を確認
型だけでは決まらない条件を補う
第7回：ランタイム・ハーネス

共通機構が判断と処理を担い、AIの補完範囲とレビュー対象を絞る

必要な品質を支えながら、開発時間とAIコストの低減を図る。

型・API・実装規律・検査・実行経路を組み合わせ、単独の機構に残る制約の抜けを補う。

まとめ

再利用と補完

型で条件を明確にし、誤りが入りにくい構造へ導く

組合せによるハーネス

合成規則・共通実行・検査で分担を維持する

次回：ランタイム・ハーネス

実行時の認可と資源管理を扱う

AIの記述能力を活用し、必要な品質と開発効率の両立を図る。