AIモデリング検証

アジェンダ

- 背景と目的
- AVDM (Always-Valid Domain Model) とは
- お題と実験設計
- デモと実験結果
- 失敗パターン（曖昧プロンプト × 非AVDM）
- 成功パターン（AVDM と明確プロンプト）
- 学び・次のアクション

背景と目的

背景

- AIコーディング支援の活用が進む中、ドメインモデルの設計がコード品質に与える影響を定量的に測定したい

目的

- Always-Valid Domain Model(AVDM) が品質に与える影響を検証
- EV充電料金計算ドメインで、**曖昧 vs 明確**プロンプト、**非AVDM(model-a) vs AVDM(model-b)** の4通りを比較

AVDM (Always-Valid Domain Model) とは

コンセプト

- 不正な状態を型レベルで防ぐドメインモデリング手法
- 値オブジェクトに不変条件を組み込み、無効な値の生成を防止

仕組み

- コンストラクタで不変条件を検証
- 検証失敗時は `Result` 型でエラーを返却
- 一度生成されたオブジェクトは常に正しい状態を保証

例: ChargeableEnergy

```
pub fn new(total: KwhMilli, billed: KwhMilli) → Result<Self, ...> {
  if billed > total { return Err( ... ); }  // 不変条件: billed ≤ total
  Ok(Self { total, billed })
}
```

お題:EV充電料金の計算

- ユーザーが充電スタンドで `start` → `snapshot`（途中課金確認）→ `stop` する流れをモデル化
- 課金の基本式：**料金 = 単価（円/kWh） × 課金対象エネルギー**
- 主要ルール（spec.md のシナリオに対応）
  1. **無料5分（scenario1〜3,7）**：開始5分までに相当するエネルギーは請求対象から除外
  2. **按分は時間比例＆切り捨て（scenario4,12）**：エネルギーは時間で均等分布とみなし、小数は floor 処理
  3. **単調増加と決定性（scenario5,11）**：利用時間が伸びるほど料金・エネルギーは減らず、同じ入力なら同じ結果
  4. **停止後課金禁止（scenario6）**：`stop` 完了後に再課金しようとすると拒否
  5. **不正入力防御（scenario8〜10,14,15）**：ゼロ/負エネルギー、時間逆行、上限超過などはエラー

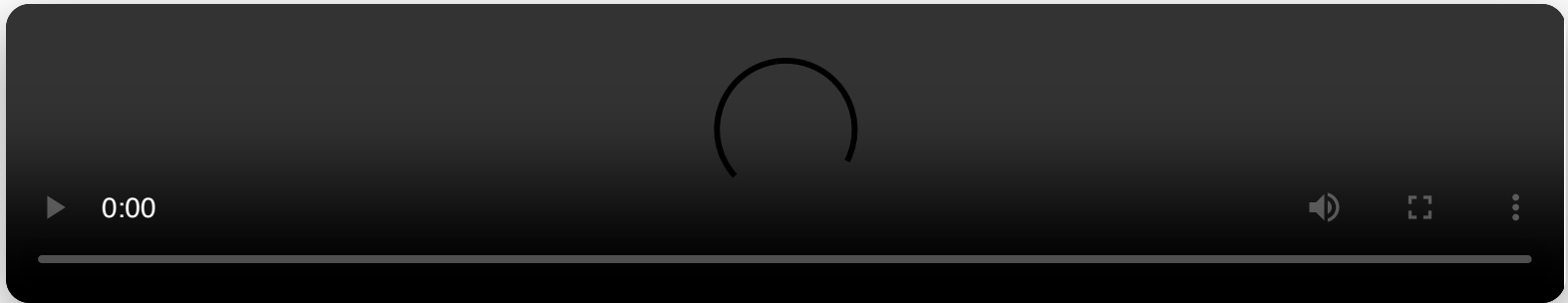
プロンプト定義

- **明確なプロンプト**
  - 目的・変更範囲・ビジネスルール・テスト基準を明文化
  - 例: 「`modules/model-a-non-avdm/src/session.rs` の `stop` を修正し、受入テスト `scenario6` を通す。状態遷移図に合わせて Stop 後の二重課金を防ぐこと」
- **曖昧プロンプト**
  - タスク目標は示すが、変更対象や受け入れ条件が曖昧
  - 「料金計算を直してください」「失敗テストをとにかく通す」など抽象的な指示
  - LLM側の推測が増え、ドメイン制約を踏み外す危険が高い
- LLM の探索余地を適切に絞り、決定的な挙動と再現性を高める

実験設計

- `scripts/run-worktree-all.sh` が4ジョブを並列投入し、各ジョブは `tmp/worktrees/<prefix>-<model>-<mode>-XXXXXX` に作成した worktree で `claude` モードを駆動
- 実行ごとにログと PID を `tmp/logs/<timestamp>` に保存し、必要に応じて worktree を `tmp/worktrees/<timestamp>` に保持（今回のラン: `tmp/logs/run-20251025-231331` / `tmp/worktrees/run-20251025-231331`）
- 成果物：各ログ（`*.log`）、補助 PID（`*.pid`）、保存済み worktree、必要に応じてデモ動画
- 計測指標：テスト結果（単体・受入）、経過時間、代表エラーメッセージ

デモ



実験結果マトリクス

データソース: `experiments/run-20251025-231331`

プロンプト	モデル	単体テスト	受入テスト	経過時間
明確	model-a	✅ 8/8	✅ 9/9	02:14
明確	model-b	✅ 11/11	✅ 9/9	02:54
曖昧	model-a	✅ 16/16	⚠️ 5/9	03:15
曖昧	model-b	✅ 17/17	✅ 9/9	03:22

> ⚠️ `model-a billed energy mismatch` が scenario1/5/11/stop 検証で発生し、曖昧プロンプト × model-a が再び破綻。

明確なプロンプトでの安定化

- 明確な仕様提示により、model-a でも期待通りの修正が入り 9/9 合格
- model-b ではドメインオブジェクトの不変条件がさらに強化
- 所要時間が短縮され、ワークフロー全体の決定性が向上
- 解析ログ: `experiments/run-20251025-231331/precise-a.log` / `experiments/run-20251025-231331/precise-b.log`

```
test result: ok. 9 passed; 0 failed; … finished in 0.00s
elapsed: 02:14 (model-a precise)

test result: ok. 9 passed; 0 failed; … finished in 0.00s
elapsed: 02:54 (model-b precise)
```

曖昧プロンプト × 非AVDM 失敗の実相(1/2)

```
assertion `left == right` failed: model-a billed energy mismatch for scenario1_six_minutes
  left: 2400
 right: 400
```

- 無料5分の按分ロジックが欠落し、6分目以降の課金量が過大になった（scenario1/5/11/stop が失敗）
- 時間が延びれば料金が増えるはず（単調増加）という前提や、同じ入力なら必ず同じ結果になる決定性（Determinism。参照透明性を成立させる前提の一つ）が壊れ、受入テスト 9 本のうち 4 本が連鎖的に失敗

曖昧プロンプト × 非AVDM 失敗の実相(2/2)

```
if elapsed_millis ≤ FREE_DURATION_MILLIS {
    session.billed_kwh_milli = 0;
    // ...
    return Ok(0);
}
// 無料期間を差し引かず全量を課金対象に設定してしまう
session.billed_kwh_milli = session.kwh_milli;
```

- 生成コード: `experiments/run-20251025-231331/worktrees/ambiguous-model-a-claude.iNiOaO/modules/model-a-non-avdm/src/session.rs:73-95` で無料時間を判定後も `session.billed_kwh_milli = session.kwh_milli;` と全量課金。FREE_DURATION 定数を持ちながら、時間按分が未実装のまま残った
- その結果、`stop_scenarios_match_expected`（backtrace: `spec-tests/tests/common.rs:104`）が想定値 400 ミリkWh に対し 2400 ミリkWh を返し、連鎖的に scenario5/11 と決定性検証が破綻
- 解析ログ: `experiments/run-20251025-231331/ambiguous-a.log`
- **非AVDM** では例外を型で防げず、バグが再注入されやすい

曖昧プロンプト × AVDM 成功要因(1/2)

```
test scenario5_progressive_billing_is_monotonic ... ok
test scenario6_rejects_billing_after_stop ... ok
test stop_scenarios_match_expected ... ok
```

- `SessionTimeline` や `ChargeableEnergy` など AVDM の値オブジェクトが不変条件を担保し、AI が生成した `Session` の変更でも型制約が破綻を防いだ
- 編集開始前に AI が値オブジェクト実装を読み込み 不変条件を先に理解してから `Session` 実装へ着手
  - ログ冒頭: `experiments/run-20251025-231331/ambiguous-b.log:10-60`
- 受入テスト `scenario10 invalid timeline is rejected` / `scenario9_negative_energy_is_rejected` / `scenario15 energy over limit is rejected` が全て成功し、終了時刻逆転・負/過大エネルギーを自動的に拒否（ログ: `experiments/run-20251025-231331/ambiguous-b.log:118-128`）
- 受入テスト 9/9 合格、単体テストも完全成功。時間は要したが、ロジック破綻は発生せず（ `elapsed: 03:22` ）
- 解析ログ: `experiments/run-20251025-231331/ambiguous-b.log`

曖昧プロンプト × AVDM 成功要因(2/2)

```
fn compute_bill( ... ) → Result<SessionBill, SessionValueError> {
  let timeline = SessionTimeline::between(started_at, ended_at)?;
  let window = timeline.consume_grace_period(Self::grace_period());
  let energy = ChargeableEnergy::allocate(total_energy, window)?;
  SessionBill::settle(energy, rate)
}
```

コード出典: `experiments/run-20251025-231331/worktrees/ambiguous-model-b-claude.GOkY2q/modules/model-b-avdm/src/session/base.rs:58-84`

- `consume_grace_period` が型レベルで無料5分を控除し、ChargeableEnergy が負値や超過量を拒否

ChargeableEnergy が防ぐバグ

不変条件: `billed ≤ total`（課金対象 ≦ 総エネルギー）を型で強制

```
pub fn new(total: KwhMilli, billed: KwhMilli) → Result<Self, SessionValueError> {
  if u64::from(billed) > u64::from(total) {
    return Err(SessionValueError::EnergyOutOfRange {
      provided: u64::from(billed), max: u64::from(total)
    });
  }
  Ok(Self { total, billed })
}
```

防ぐバグの例:

- ❌ 按分ミス → 全量課金（model-a で発生: `billed_kwh_milli = session.kwh_milli`）
- ❌ 負のエネルギー値（scenario9）
- ❌ 上限超過（scenario15）

学び

- **プロンプト解像度が最重要**: 曖昧指示では非AVDMがバグを再び生む
- **AVDM の防御力**: 値オブジェクトと不変条件で曖昧さを吸収
- **自動化の威力**: `run-worktree-all.sh` により、並列実験とログ収集が確実
- **記録の重要性**: 成果ログ＋動画で検証結果を「再演可能」に保管

付録: デモ運用と自動化の工夫

並列実験の自動化

- `scripts/run-worktree-all.sh` で4パターンを同時実行
- 各実験は独立したworktreeで隔離（競合なし）
- ログ・PID・成果物を自動収集

再現性の確保

- タイムスタンプ付きディレクトリで実験を保存
- worktreeごと保存することで、生成コードを後から確認可能
- 動画録画で視覚的な証拠を残す

運用上の工夫

- 重要な実験結果は `experiments/` へ昇格
- デモは事前録画でリスク回避（ライブ実行の失敗を防ぐ）

ご清聴ありがとうございました