テスト設計コンテスト’25　予選

だんだん動物園入場管理システム テスト設計プロセスについて

考えるアシカbeta

2025/10/4

チーム紹介

2023年の初出場以来、2年ぶり2回目
今年はAI Agentsを仲間に引き連れて参加！！~~(つまりソロ)~~
レギュレーションの人数制限なんて関係なしっ

ところで・・・
なんで、「beta」 なの？

テスト設計のコンセプト

Test Hyperdrive
powered by AI Agent

コンセプトは「Test Hyperdrive」
AI Agentでテストの未来を“ 超加速”させよう

コードエディタとGitリポジトリ内で、テスト設計プロセス全てを完結。

AIと協働することで、 テストをボトルネック にしない
開発スピードが速くなる中で、 テストも進化しなければならない

つまりは、How極振りってこと？

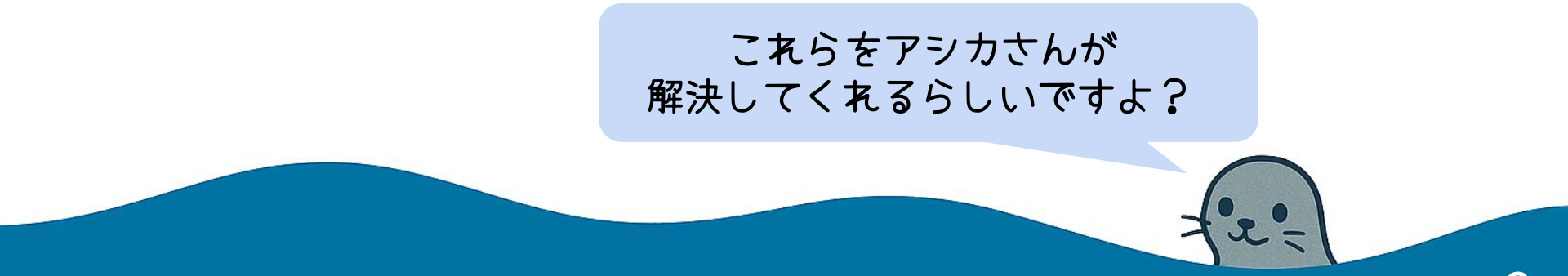
[問い]
AIしかできないことってなんだろう？

［答え］
驚異的なスピードで生成できる

それってあなたの感想ですよね？

驚異的な生成スピードの反面、課題も

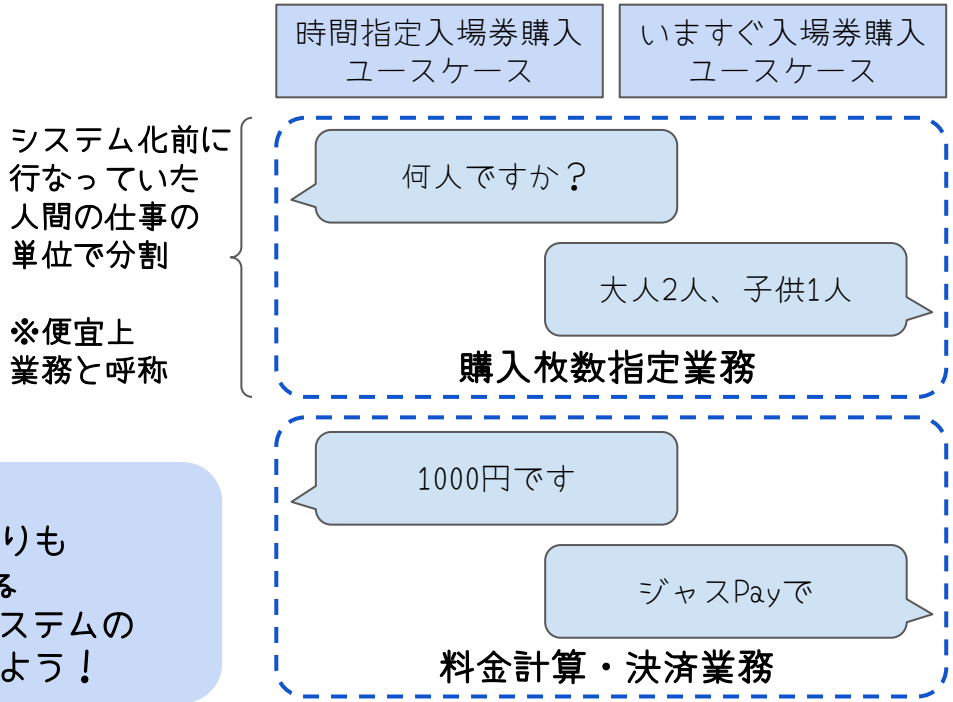
- 与えたコンテキストの中でしか生成されない
  - コンテキストを与えない場合は、一般的な知識をもとに生成される
- 出力されたものをいかにレビューしていくか
  - また、それをチーム内で納得できる形で表現し・合意形成していく必要性がある
- レビューした結果をどのようにフィードバックし、反映していくのか
  - 違和感のある内容を個別にチャットで指示したり、人が手作業で修正していくのは果たして本当に効率的なのか
  - 60~70点の成果物は作成できるが、より高い品質を目指すなら人間の関与が必須



リグレッションリスクの特定 － 前準備

リスクの特定にあたって、どういった単位でリスクを特定するのかを検討。

ユースケース単位でリスクの洗い出しをした結果、共通項が存在することがわかったので、機能的責務ごとにリスク分析を行うことにした。



リスク管理表は継続的に使うから
コンポーネントやユースケースごとに管理するよりも
機能的責務毎にまとめるほうが保守しやすいね
でも、これはあくまで一例。実際にはチーム構成やシステムの
アーキテクチャを考慮して適切な分割単位を検討しよう！

リスクの評価

リスクの抽出・評価 → リスクのレビュー → リスクの見直し

リグレッションリスクの特定 － 前準備

業務分析もAIにたたき台を
作ってもらいました

業務を洗い出して[*]

ほいほーい。

園内チケットシステム 業務分析

No	機能的責務	概要	主要アクター	トリガー	主要な入力	主要な出力
B-Park-001	園内チケットシステム起動業務	発券機、入場ゲートハブ、入場ゲート、入場管理、残数表示インジケータ等のハードウェア初期化を統合して実行し、各種設定情報・当日有効な購入情報・残数閾値・固定文言等を取得してサービス開始状態にする。	動物園管理者	営業開始時の起動操作	設定情報（号機番号等）、購入情報テーブル、残数アイコン切替閾値、固定文言データ	開始画面表示、各機器初期化完了、情報取得完了
B-Park-002	発券機運用・保守業務	発券機の釣り銭切れ・詰まり、紙幣・硬貨の補充、障害発生時の係員呼び出しや復旧対応を行う。	動物園管理者、係員	障害発生、釣り銭切れ・詰まり検知、補充タイミング	障害種別、釣り銭・紙幣・硬貨状態	障害通知、復旧対応、補充完了通知
B-Park-003	時間枠指定業務	入場券購入時に対象となる時間枠を選択し、枠ごとの残数や販売可否を判定する。	入場者	時間枠選択操作	時間枠一覧、残数情報	選択可能枠表示、販売可否判定、枠確保処理
B-Park-004	購入枚数指定業務	いますぐ入場券・時間指定入場券共通で、区分別（おとな・こども）の購入枚数指定を処理する。	入場者	購入枚数選択操作、枚数変更操作	区分別希望枚数、残数制限情報	購入枚数確定、合計金額表示、枚数制限チェック
B-Park-005	予約済み入場券発券業務	事前に購入済みの予約入場券に対してQRコード印刷による物理的な発券を実行する。	入場者（会員）	予約済入場券発券ボタン押下、ログイン完了	会員認証情報、発券対象予約選択	入場券印刷、発券完了確認

リスクの評価

リスクの抽出・評価 → リスクのレビュー → リスクの見直し

*: もちろんこのような単純なone-shot promptingではなく、厳密な定義を与えて出力させています

箸休め - Code Editor上で表形式のデータを扱う

課題感

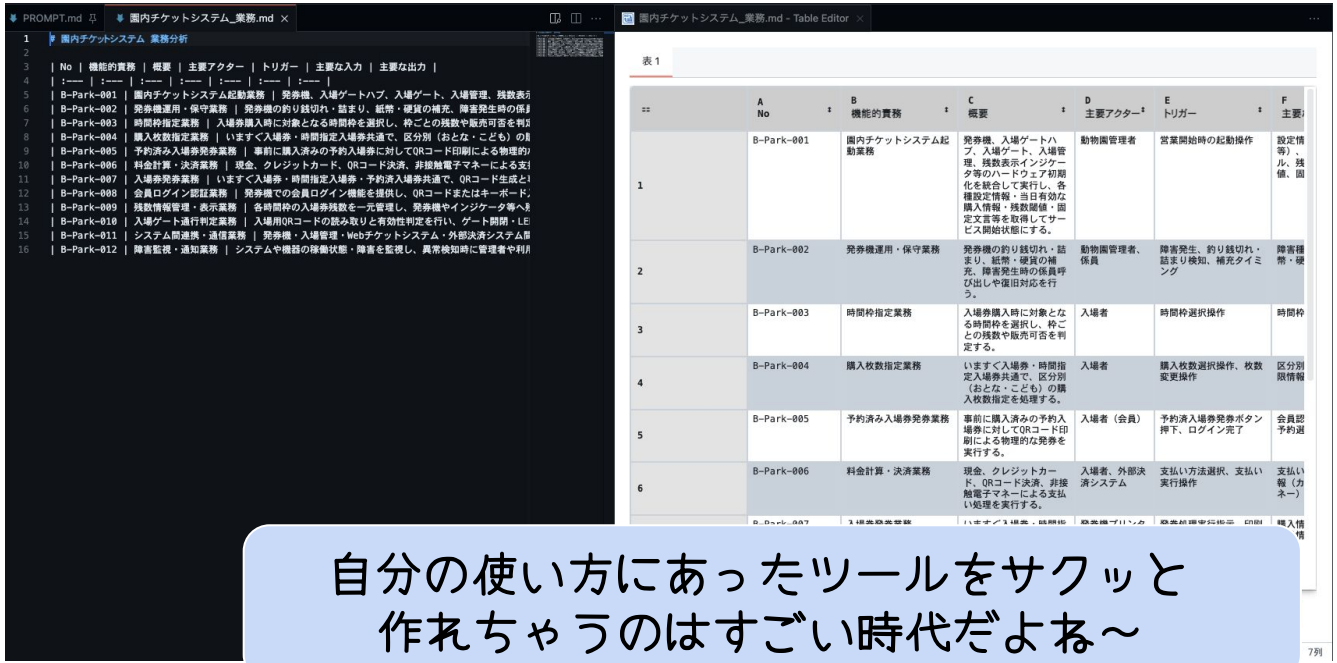
- Excel(Microsoft 365 Copilot)やSpreadSheet(Gemini)などでもAIがインテグレーションされつつある一方でタスク処理の自由度はAI Coding Agentと比較するとやや劣る
- VSCodeやCursorなどのコードエディタではタスク処理の自由度は高い一方、表形式(Markdown, CSV)のデータは可読性や編集面でやや扱いにくい

課題に対する打ち手

- Markdownの表を表計算ソフトライクで編集できるVSCode拡張を開発[*]
  (もちろんVSCodeをforkしたCursor,Kiroなどでも利用可能)

*: 今のところRepositoryは非公開ですが、需要があれば精査の上Publicにします

箸休め - Code Editor上で表形式のデータを扱う

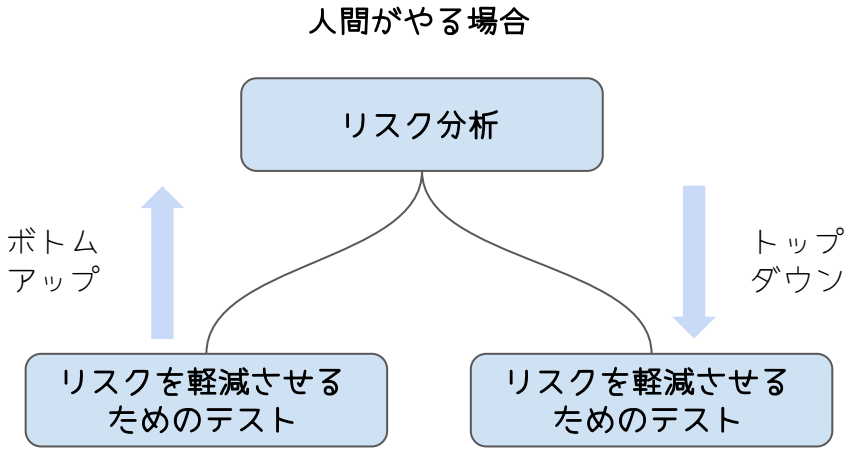


*: 今のところRepositoryは非公開ですが、需要があれば精査の上Publicにします

リスク分析と抽象的テストケースを同時に出力

AIに依頼する場合、個別に依頼すると整合性が取りづらくなるため、リスク内容と抽象的テストケースを一括で**json形式**で出力。

※抽象的テストケースとは、ISTQBのハイレベルテストケース[*]よりもさらに抽象化されたテストケースと定義。
ハイレベルテストケース：抽象的な事前条件、入力データ、期待結果、事後条件、およびアクション（該当する場合）を含むテストケース



*: ISTQB用語集. https://istqb-glossary.page

リスク分析を依頼するときの3つのビュー

単純な指示でも一般的な知識に基づくリスクは洗い出してくれるが、
存在しない架空の機能を作り出されていることも。そこで・・・

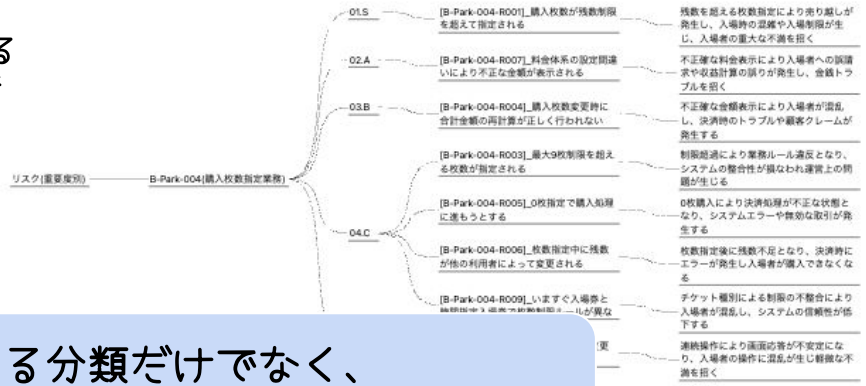
ビューの名称	説明
アーキテクチャビュー	予め作成したUML分析モデルを入力することで、システムの構成やデータ・処理の流れに関するコンテキストを効率的に与え、システムの作りに起因する危うさを洗い出すためのビュー
ユーザビュー	ユーザの操作パターン、誤操作、悪意あるユーザなどを想定したリスクを洗い出すためのビュー
ビジネスロジックビュー	対象システムが扱う業務ロジックからくるリスクを洗い出すためのビュー（例えば、チケットは同時に9枚までしか買えないとか）

リスクのレビュー

jsonで出力させると、さまざまなメタ情報を含めることができるが、人間がレビューするには不向きのため、マインドマップツールで扱える形式に変換してレビューする


```
{
    "risk_id": "B-Park-004-R001",
    "risk": "購入枚数が残数制限を超えて指定される ",
    "abstract_testcases": [
      {
        "abstract_tc_id": "B-Park-004-R001-A001",
        "abstract_testcase": "指定枚数が時間枠残数を超過する場合に適切にエラーチェックが動作することを確認する ",
        "precondition": "",
        "view": "ビジネスロジックビュー ",
        "quality_characteristic": "機能適合性",
        "quality_sub_characteristic": "機能適合性 -機能正確性",
      }
    ],
    "reference": [
      "11_園内チケットシステム要求仕様書 .md",
      "シーケンス図 -いますぐ入場券購入 .md"
    ],
    "impact": "残数を超える枚数指定により売り越しが発生し、入場時
の重大な不満を招く ",
    "impact_level": "01.S",
    "frequency": "03.L",
    "risk_evaluation_reason": "入力検証の不具合は設計・実装
越しによる入場制限は入場者の重大な不満と動物園の社会的信用失墜を招
大"
}
```


インデント付きの
テキストに変換する
スクリプトを実行



リスクレベルによる分類だけでなく、
品質特性やビューごとにも表現できるらしいよ

リスクの評価

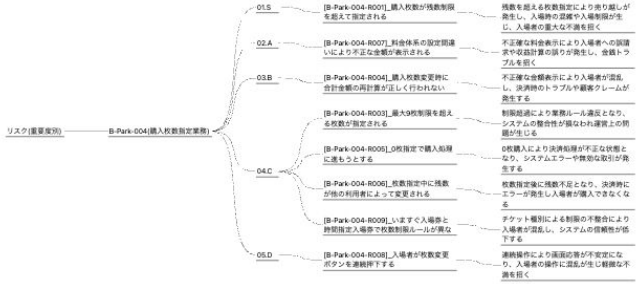
リスクの抽出・評価 → リスクのレビュー → リスクの見直し

レビュー結果の反映

不備があった項目をチャットで1つずつ修正していくのは辛い。
そこで、ツリー構造で変更した内容のdiffを一覧表にしてAI Agentに渡す


```
{
    "risk_id": "B-Park-004-R001",
    "risk": "購入枚数が残数制限を超えて指定される ",
    "abstract_testcases": [
      {
        "abstract_tc_id": "B-Park-004-R001-A001",
        "abstract_testcase": "指定枚数が時間枠残数を超過する場合に適切にエラーチェックが動作することを確認する ",
        "precondition": "",
        "view": "ビジネスロジックビュー ",
        "quality_characteristic": "機能適合性 ",
        "quality_sub_characteristic": "機能適合性 -機能正確性 ",
      }
    ],
    "reference": [
      "11_園内チケットシステム要求仕様書 .md",
      "シーケンス図 -いますぐ入場券購入 .md"
    ],
    "impact": "残数を超える枚数指定により売り越しが発生し、入場時の混雑や入場制限が生じ、入場者の重大な不満を招く ",
    "impact_level": "01.S",
    "frequency": "03.L",
    "risk_evaluation_reason": "入力検証の不具合は設計・実装品質に依存し発生頻度は低いが、売り越しによる入場制限は入場者の重大な不満と動物園の社会的信用失墜を招き事業継続に影響するため影響度は重大"
}
```


スクリプトで差分検知し、
修正内容を一覧表にする



リスクの評価

リスクの抽出・評価 → リスクのレビュー → リスクの見直し

テスト条件の具体化

抽象的テストケースで洗い出したざっくりとした「テストしたいこと」を
具体化し、リスク分析のjsonに追記していく

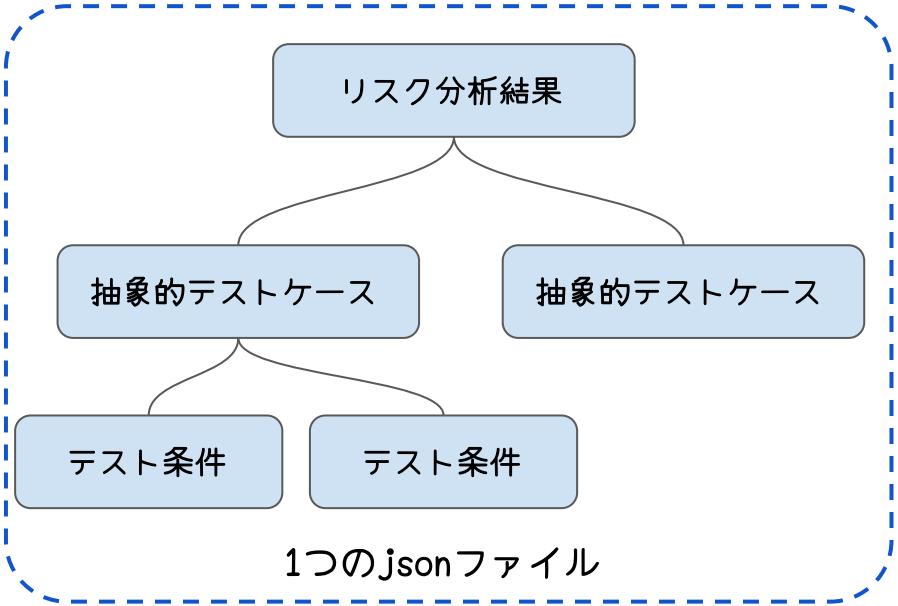
CIBFW[*]の考え方を入力として与える

- Condition: 条件
- Item: テスト対象
- Behaviour: 振る舞い
- Fault: 起きて欲しくないこと
- World: その他全部

Conditionに関しては、境界値分析の考え方もインプットし、適用可能な対象なら、境界値も出力

*: VSTePによるソフトウェアテストの開発. https://jasst.jp/symposium/jasst16tohoku/pdf/S1.pdf

特定の条件を満たすテストケースを抽出



一つのjsonファイルに

- リスク分析の結果
- 抽象的テストケース
- 具体的テストケース

の情報がまとまっているので、
影響度ランク「S」のものだけや、
製品品質特性「機能適合性」のものだけ
などメタ情報に含まれる値をもとに
ピックアップすることができる

ありがとうございました