

テスト設計コンテスト'25 決勝

# だんだん動物園入場管理システム 更新に伴うテスト設計のご提案

考えるアシカbeta  
(Takafumi Yanagawa)

2026/1/24



- 01 はじめに
- 02 背景とご提案の概要
- 03 継続的に運用可能なリグレッションテストのご提案
- 04 リスク分析を進める上での考え方
- 05 AI Coding Agentを用いたテスト設計
- 06 まとめ



# 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



# テスト設計のコンセプト



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？



- 01 はじめに
- 02 背景とご提案の概要
- 03 継続的に運用可能なリグレッションテストのご提案
- 04 リスク分析を進める上での考え方
- 05 AI Coding Agentを用いたテスト設計
- 06 まとめ



# 背景と今回のご提案の概要

だんだん動物園様では、人気動物による入場者数に伴い、機器ならびにシステムの更新を予定されております

そこで・・・

『入場ゲートの増設、ならびに、入場ゲートハブ新設に伴うテスト』

『継続的に運用可能なリグレーションテスト』

をご提案いたします！

そのまんまじゃないの…



# 今回のプレゼンテーションのスコープ

だんだん動物園様では、人気動物による入場者数に伴い、機器ならびにシステムの更新を予定されております

そこで・・・

『入場ゲートの増設、ならびに、入場ゲートハブ新設に伴うテスト』

『継続的に運用可能なリグレッションテスト』

をご提案いたします！

こちらにFocus



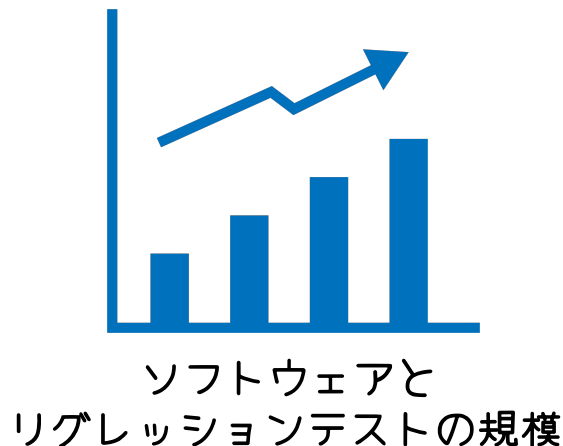
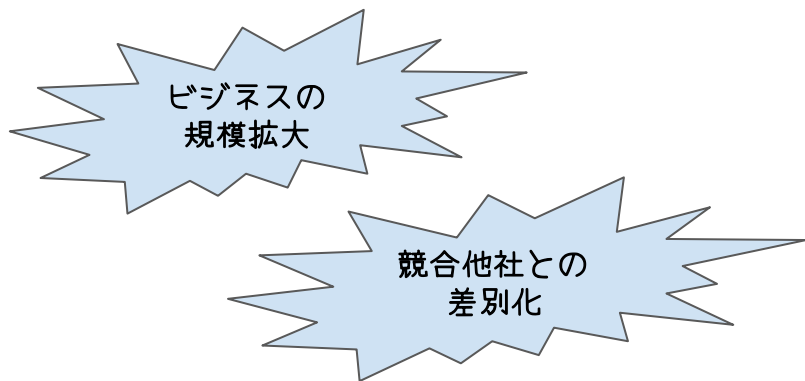
- 01 はじめに
- 02 背景とご提案の概要
- 03 継続的に運用可能なリグレッションテストのご提案
- 04 リスク分析を進める上での考え方
- 05 AI Coding Agentを用いたテスト設計
- 06 まとめ





# リグレッションテストとは

- **リグレッション**： ソフトウェアに変更を加えた際、変更されていない既存の機能に意図せず干渉し、期待するふるまいが得られなくなる事象
- **リグレッションテスト**： リグレッションが発生リスクを低減させるためのテストのこと。変更部分に対するテストはスコープに含めない。



# リグレッションテストのよくある課題

テストスイートが増え続ける

リリース直前に重篤な障害が見つかる



なぜこのような状況に陥ってしまうのか

- 新しい機能を実装した時、変更を加えた際のリグレッションテストの追加基準が不明瞭
- テストの目的が不明瞭なため、テストを削除することが困難

リグレッションテストはソフトウェアのライフサイクルとともに継続的に使われるものだから、途中で担当者が変わったりすると、テストの意図が引き継がれずに失われてしまうのが一因なのかな～



# リグレッションテストを継続的に運用するために

## リグレッションの 「リスク」を 基にしたテスト

リスク分析では、ある機能が動作しなかった場合に、誰にどのような影響を及ぼすのかを評価。  
そのテストをやらなかった時にどうなるかを明示する

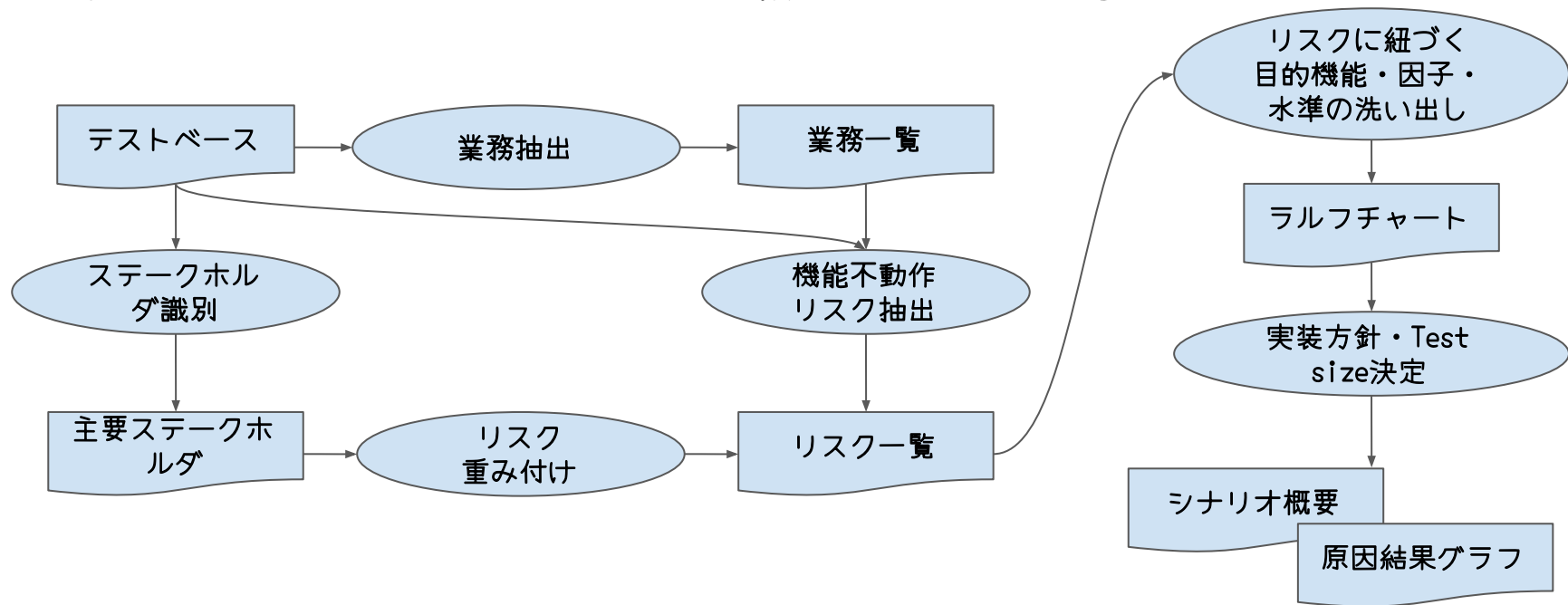
## Test sizeに基づく 適切なテストレベル への分類

異なるテストレベルで同じテスト目的を持ったテストが実装されてしまったり、あるテストレベルで実施すべきテストが他のテストレベルに偏る問題を解決。

- 01 はじめに
- 02 背景とご提案の概要
- 03 継続的に運用可能なリグレッションテストのご提案
- 04 リスク分析を進める上での考え方
- 05 AI Coding Agentを用いたテスト設計
- 06 まとめ



# リグレッションテストの分析～設計の流れ



# 業務一覧の作成

リスクの特定にあたって、こ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。

リグレーションテストのベースであるリスク管理表は継続的に使うから、機能的責務毎にまとめるほうが保守しやすいと思っただけよ。

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！

システム化前に行っていた人間の仕事の単位で分割

※便宜上業務と呼称

時間指定入場券購入  
ユースケース

いますぐ入場券購入  
ユースケース

何人ですか？

大人2人、子供1人

購入枚数指定業務

1000円です

ジャスPayで

料金計算・決済業務



# リスクレベルの定義

リスクの重篤度や発生頻度を定量的に導き出すのは困難。（未来予知でもできない限り）

しかし、リスクベースドテストを選択する限り、避けては通れない問題。

「考慮していなかった」を避けるために、複数の判断軸でリスクレベルを決定。

重篤度	直接的な影響度、関連機能への波及影響度、短期的な金銭的影響度、長期的な金銭的影響度から最も深刻な影響を基準 → S, A, B に割り当て
発生頻度	対象機能の利用頻度 × 障害の発生しやすさ係数[*] で算出 → High, Medium, Low に割り当て

それでもなお、感覚的だよね…





# リグレッションリスクの考え方

## リグレッションテスト **以外**での リスクの捉え方

- 変更箇所に対するテストであるため、テストをすることでリリース可能である品質であることの確信を持ちたい
- 新しい欠陥が見つかる可能性が高い
- リスクベースドテストをする場合、リスクの考慮漏れがあると、すなわち欠陥を見逃すことに繋がるため、できる限りさまざまなリスクを抽出する必要がある



## リグレッションテストにおける リスクの捉え方

- 変更していない(影響を及ぼしていないはずの)箇所に対するテストなので、既存のふるまいに影響がないことを確認したい
- 新しい欠陥が見つかる可能性は低く、コストもかけたくない
- 主に仕様に定義されることになる論理的な関係が満たされていることの確認にフォーカスしたい  
→ **機能不動作**が起きた時に、誰にどのような影響があるかをベースに機能ごとに分析していく





# Test sizeの定義

Mediumでは入場ゲート  
ハブ<->入場管理など  
園内localで実施

固定文言ファイルの読  
み込み、一時的な印刷  
スプールなど

入場ゲート複数台での  
処理など

号機番号、残数切替閾  
値など

	Small	Medium	Large
Network access	No	Localhost only	Yes
Database	No	Yes	Yes
File system access	No(Mock)	Yes	Yes
Use external systems	No(Mock)	Discouraged	Yes
Multiple threads	No	Yes	Yes
Sleep statements	No	Yes	Yes
System properties	No	Partial Override	Yes
Time limit	60	300	1800+

予約管理DB、会員情報  
管理DBへのアクセス

決済システム、運営管  
理システムへのアクセ  
スはLargeのみ

00分/30分の時間枠の境  
界など



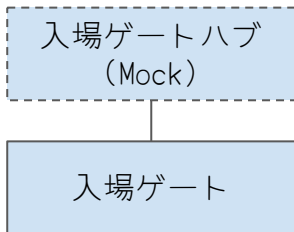
# Test sizeによる分類例

対象システムのアーキテクチャに依存するものの、  
実行時間が長く、多くの依存関係が必要なLargeサイズのテストではHappy pathレベルの確認だけが残る

## Small

有効なチケット情報にて、  
扉「開」イベントが送信される

使用済みチケット情報にて、  
扉「開」イベントが送信されない



依存関係:少  
実行スピード:高

## Middle

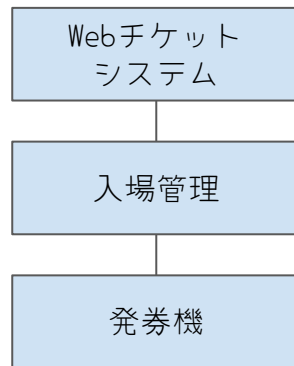
残数によるいますぐ入場券  
購入画面への遷移制御



依存関係・実行スピード

## Large

発券機からログインできる



依存関係:多  
実行スピード:低



- 01 はじめに
- 02 背景とご提案の概要
- 03 継続的に運用可能なリグレッションテストのご提案
- 04 リスク分析を進める上での考え方
- 05 AI Coding Agentを用いたテスト設計
- 06 まとめ



# AIは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



# AIによる出力結果にありがちな結果

> ○○機能についてのテストケースを生成してください

- 通信失敗時に適切なエラーハンドリングが行われ、デフォルト値で起動すること
- 「発券機と入場管理が通信失敗したとき○○となる」「入場ゲートハブが故障したとき～○○となる」
- 仕様では、通信失敗時は障害停止画面を表示して係員に通知するはず【期待結果の誤り】
- 対象のコンポーネントは何か？と、そのコンポーネントがどうなるか？という2つの視点が混ざっていて、十分性が判断しにくい【説明根拠の不足】



# 機能不動作リスク抽出

構造化データ(json)で出力し、出力項目を厳格に定めた上で各項目の内容はできるだけ小さく出力させる。そして、GUIで視覚的に確認・編集・削除可能。

- リスクID
- リスクの概要・詳細・失敗シナリオ例
- 影響
  - ステークホルダ毎（入場者、動物園係員、動物園経営者、だんだん市補助金担当）
- 重篤度
  - 直接影響, 波及影響, 短期的金銭影響, 長期的金銭影響をそれぞれ5段階の数値で表現
- 発生頻度
  - 機能の利用頻度, 故障のしやすさをそれぞれ5段階の数値で表現
- 根拠となったテストベース名・機能ID

The screenshot displays a web application for risk management. On the left, a 'フィルタ' (Filter) sidebar allows users to narrow down results by '影響度' (Impact) with checkboxes for 02.A, 03.B, 01.H, 02.M, and 03.L. It also includes filters for '実装テストサイズ' (Implementation Test Size) with buttons for Large, Medium, and Small, and '品質特性' (Quality Characteristics) with checkboxes for セキュリティ (Security), 互換性 (Compatibility), 使用性 (Usability), 信頼性 (Reliability), 性能効率性 (Performance Efficiency), and 機能適合性 (Functionality). Below these are '品質前特性' (Quality Pre-Characteristics) with checkboxes for ユーザーエラー防止性 (User Error Prevention), 共存性 (Coexistence), 可用性 (Availability), 拡張性 (Extensibility), 成熟性 (Maturity), 時間効率性 (Time Efficiency), 機密完全性 (Confidentiality), 機能正確性 (Functional Accuracy), 相互運用性 (Interoperability), 真正性 (Authenticity), and 運用操作性 (Operational Usability). At the bottom of the sidebar are checkboxes for '表示設定' (Display Settings): 発生シナリオ (Occurrence Scenario), 影響 (Impact), and 抽象的テストケース (Abstract Test Cases). The main area, titled 'リスク一覧' (Risk List), shows a table of risks. The first risk is 'B-Park-001-EH100-R001' with a description '国内チケットシステム起動・終了・ナビゲーション業務' and a status of '発生シナリオ' (Occurrence Scenario). The second risk is 'B-Park-001-EH100-R002' with a description '国内チケットシステム起動・終了・ナビゲーション業務' and a status of '発生シナリオ' (Occurrence Scenario). The third risk is 'B-Park-001-EH100-R003' with a description '国内チケットシステム起動・終了・ナビゲーション業務' and a status of '発生シナリオ' (Occurrence Scenario).



# リスクに関連する目的機能・因子・水準の洗い出し

モデル(ラルフチャート [\*]、ISO/IEC 25010の品質モデル)に沿って出力することで、レビュー容易性を上げる

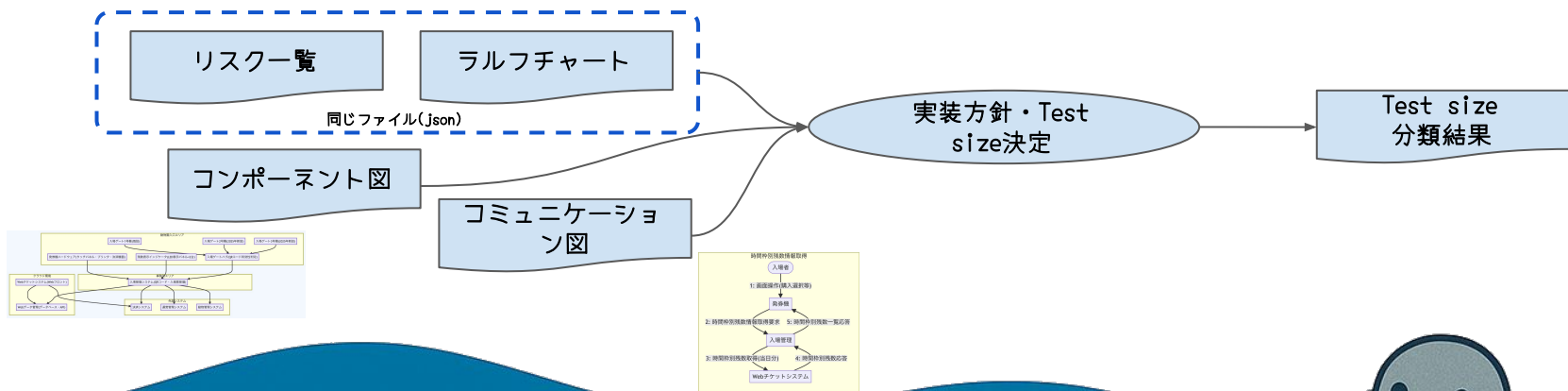
- 目的機能（ユースタories）
  - <利用者の役割>として <ゴール> を達成したい。<理由>のためだ。の形式で出力
- inputs(信号因子), noise(誤差因子), state(状態因子)と水準の洗い出し
- 主にどの品質特性に基づくテストなのかを分類
  - 特定の品質特性に不足がないかをチェックする目的で記載



# 実装方針とTest sizeの決定

- Largeサイズのテストは原則として シナリオテスト にて実装
- Mediumサイズ以下のテストは、入力条件による出力のパターンなどを見るケースがあるので、原因結果グラフ を作成(仕様書から自動生成)

Test sizeを算出する際にLLMに対して与えているコンテキスト

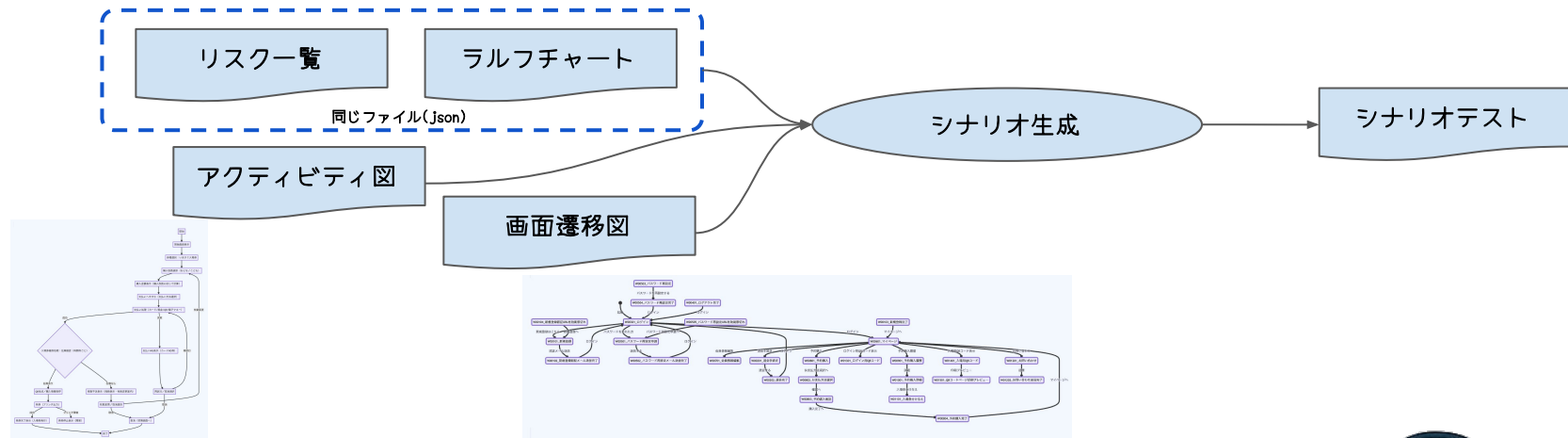




# シナリオテストの自動生成

画面遷移図や、アクティビティ図をもとに、  
実際に操作可能な手順の組み合わせでシナリオテストを生成

シナリオテストを生成する際にLLMに対して与えているコンテキスト



# 原因結果グラフの自動生成

仕様書の論理関係の記述を読み取り、AIが原因結果グラフを自動生成。

モデル化することで、文章では読み取りづらい論理関係や曖昧さを検知！

テストベースを読み込ませ、意味のある段落でセクションに分割  
→MCPサーバを提供しており、AI Agentsからセクション分割を指示することができる



当該セクションの仕様に対して、mermaid形式で擬似的に原因結果グラフを作成。  
(本来の原因結果グラフの表現方法とは異なる)  
原因結果グラフからデシジョンテーブルを生成することも可能であるが、本ツールとしては未対応。  
CEGTestを用いる。



- 01 はじめに
- 02 背景とご提案の概要
- 03 継続的に運用可能なリグレッションテストのご提案
- 04 リスク分析を進める上での考え方
- 05 AI Coding Agentを用いたテスト設計
- 06 まとめ



# まとめ

- リグレッションを前提としたリスクの捉え方を再定義。
- リスクxTest sizeを軸に、リグレッションテストの追加・削減の基準を明確化
- 構造化データ(json)でキーワードレベルで小さく出力したり、情報のソースを示すことで、納得感のある出力結果を得た上で、GUIで出力結果のレビューができる仕組みの構築

担当者が変わっても「なぜこのテストをすべきなのか」を途切れさせない  
継続的に運用可能なリグレッションテスト を実現！



# ありがとうございました

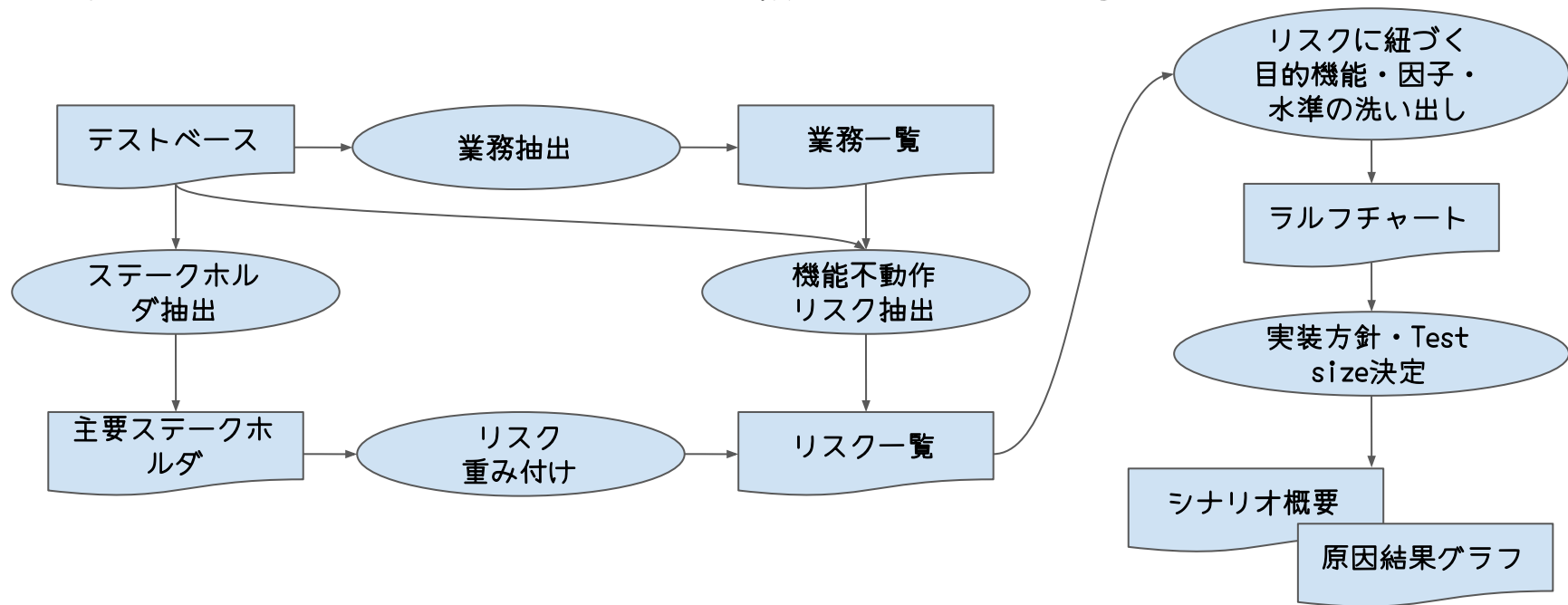
おしまい！今年は時間内に話せた？



# Backup (フロア展示資料)

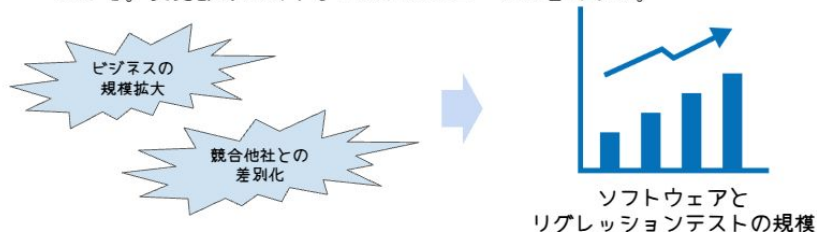


# リグレッションテストの分析～設計の流れ



## リグレッションテストとは

- **リグレッション**: ソフトウェアに変更を加えた際、変更されていない既存の機能に意図せず干渉し、期待するふるまいが得られなくなる事象
- **リグレッションテスト**: リグレッションが発生リスクを低減させるためのテストのこと。変更部分に対するテストはスコープに含めない。



## リスクレベルの定義

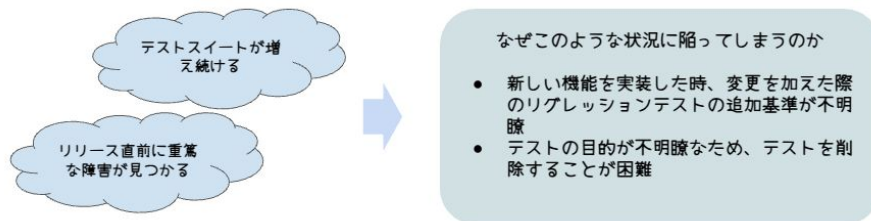
リスクの重篤度や発生頻度を定量的に導き出すのは困難。(未来予測でもできない限り)

しかし、リスクベースドテストを選択する限り、避けては通れない問題。

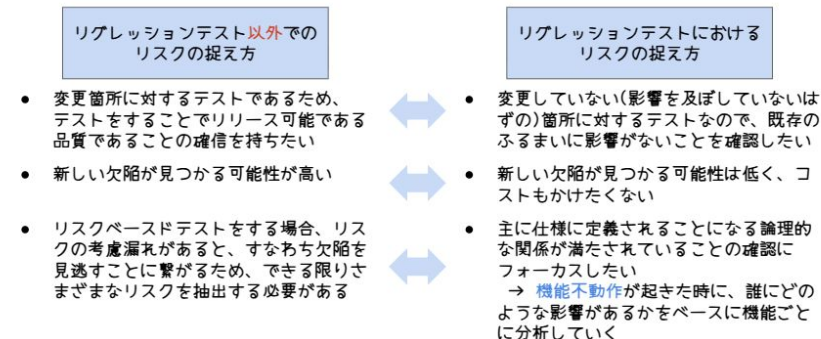
「考慮していなかった」を避けるために、複数の判断軸でリスクレベルを決定。

重篤度	直接的な影響度、関連機能への波及影響度、短期的な金銭的影響度、長期的な金銭的影響度から最も深刻な影響を基準 → S, A, B に割り当て
発生頻度	対象機能の利用頻度 × 障害の発生しやすさ係数[*] で算出 → High, Medium, Low に割り当て

## リグレッションテストのよくある課題



## リグレッションリスクの考え方





# Test sizeについて

## Test sizeの定義

	Small	Medium	Large
Mediumでは入場ゲートハブ<->入場管理など 園内localで実施	No	Localhost only	Yes
固定文言ファイルの読み込み、一時的な印刷スプールなど	No	Yes	Yes
入場ゲート複数台での処理など	No(Mock)	Yes	Yes
号機番号、残数切替閾値など	No(Mock)	Discouraged	Yes
Multiple threads	No	Yes	Yes
Sleep statements	No	Yes	Yes
System properties	No	Partial Override	Yes
Time limit	60	300	1800+

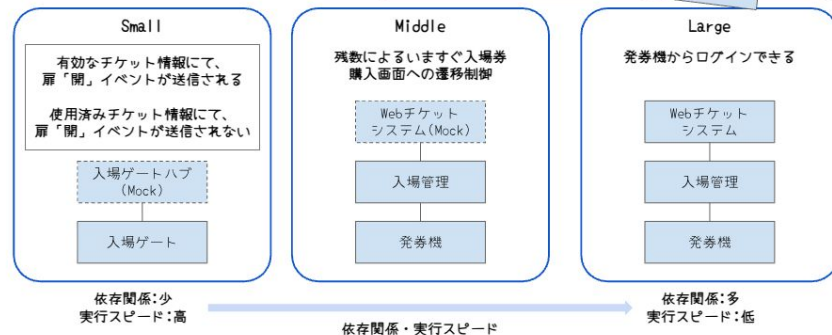
予約管理DB、会員情報管理DBへのアクセス

決済システム、運営管理システムへのアクセスはLargeのみ

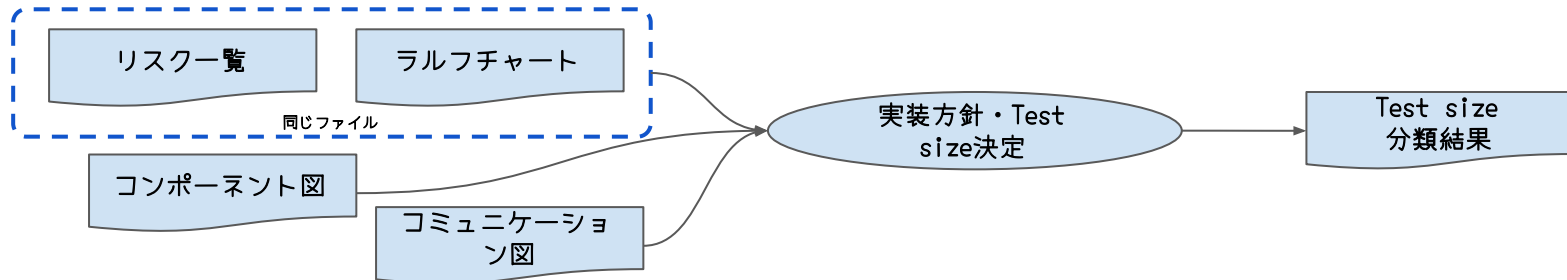
00分/30分の時間枠の境界など

## Test sizeによる分類例

対象システムのアーキテクチャに依存するものの、実行時間が長く、多くの依存関係が必要なLargeサイズのテストではHappy pathレベルの検証だけが残る



## Test sizeを算出する際にLLMに対して与えているコンテキスト



# Markdown形式のテーブルを扱いやすくするための拡張機能の開発

No	機能的機能	概要	主要アクター	トリガー	主要出力
1	B-Park-001	国内子システム起動業務	発着機、入場ゲートハブ、入場ゲート、入場管理、乗客管理	乗客発生時の乗客発生、乗客発生時の乗客発生、乗客発生時の乗客発生	乗客発生時の乗客発生、乗客発生時の乗客発生、乗客発生時の乗客発生
2	B-Park-002	乗客発生時の乗客発生	乗客発生時の乗客発生、乗客発生時の乗客発生、乗客発生時の乗客発生	乗客発生時の乗客発生、乗客発生時の乗客発生、乗客発生時の乗客発生	乗客発生時の乗客発生、乗客発生時の乗客発生、乗客発生時の乗客発生
3	B-Park-003	乗客発生時の乗客発生	乗客発生時の乗客発生、乗客発生時の乗客発生、乗客発生時の乗客発生	乗客発生時の乗客発生、乗客発生時の乗客発生、乗客発生時の乗客発生	乗客発生時の乗客発生、乗客発生時の乗客発生、乗客発生時の乗客発生
4	B-Park-004	乗客発生時の乗客発生	乗客発生時の乗客発生、乗客発生時の乗客発生、乗客発生時の乗客発生	乗客発生時の乗客発生、乗客発生時の乗客発生、乗客発生時の乗客発生	乗客発生時の乗客発生、乗客発生時の乗客発生、乗客発生時の乗客発生
5	B-Park-005	乗客発生時の乗客発生	乗客発生時の乗客発生、乗客発生時の乗客発生、乗客発生時の乗客発生	乗客発生時の乗客発生、乗客発生時の乗客発生、乗客発生時の乗客発生	乗客発生時の乗客発生、乗客発生時の乗客発生、乗客発生時の乗客発生
6	B-Park-006	乗客発生時の乗客発生	乗客発生時の乗客発生、乗客発生時の乗客発生、乗客発生時の乗客発生	乗客発生時の乗客発生、乗客発生時の乗客発生、乗客発生時の乗客発生	乗客発生時の乗客発生、乗客発生時の乗客発生、乗客発生時の乗客発生
7	B-Park-007	乗客発生時の乗客発生	乗客発生時の乗客発生、乗客発生時の乗客発生、乗客発生時の乗客発生	乗客発生時の乗客発生、乗客発生時の乗客発生、乗客発生時の乗客発生	乗客発生時の乗客発生、乗客発生時の乗客発生、乗客発生時の乗客発生

Excelやスプレッドシートにデータがある場合も多く、既存のデータからの変換が大変。

そこで、VSCode上でMarkdown形式のテーブルを直接編集したり、行・列の追加・削除が容易に行える拡張機能を開発[\*]。

ExcelやスプレッドシートとMarkdown Table Editor上でコピー＆ペーストで相互にデータを貼り付けられたり、CSVインポート・エクスポート機能を兼ね備える。



\*: Interactive Markdown Table Editorという名称でVisual Studio Marketplaceで公開中(Free)。  
2025年12月21日現在、ダウンロードが可能ですが、今後予告なくアクセスできなくなる可能性があります。

# json形式で出力されたリスク一覧、ラルフチャートを視覚的に表示できるWebUIの実装

jsonに含まれるさまざまなメタ情報を用いて絞り込むことができるので、気になった視点でレビューをすることが容易になります

ex) 特にリスクレベルが高いのはどれ？リスクレベルは妥当？

- 業務ID
- 機能ID
- 品質特性
- テストサイズ
- リスクレベル

フィルタ

リスク 抽象的TC

テキスト検索...

業務ID

B-Park-001 B-Park-002 B-Park-003  
B-Park-004 B-Park-005 B-Park-006  
B-Park-007 B-Park-008 B-Park-009  
B-Park-010 B-Park-011 B-Park-012  
B-Web-001 B-Web-002 B-Web-003  
B-Web-004 B-Web-005 B-Web-006  
B-Web-007 B-Web-008 B-Web-009  
B-Web-010 B-Web-011 B-Web-012  
B-Web-013

機能ID

EG-100 EG-210 EG-220  
EG-310 EG-320 EG-330  
EG-900 EH-100 EH-101 EH-102  
EH-103 EH-104 EH-210 EH-220  
EH-230 EH-240 EH-244 EH-300  
EH-311 EH-312 EH-320 EH-400  
EH-500 EH-900 EK-100 EK-210  
EK-220 EK-230 EK-311 EK-312  
EK-321 EK-322 EK-410 EK-510  
EK-520 EK-530 EK-900 EN-241  
EN-242 EN-243 W-000 W-001  
W-002 W-003 W-004 W-005  
W-006 W-007 W-008 W-009  
W-010 W-011 W-012 W-013  
W-014 W-015 W-900 Z1-100  
Z1-201 Z1-202 Z1-203 Z1-900

リスク一覧

PDF 全て展開 折りたたむ 全て選択

☐ B-Park-001-EH100-R001 / 園内チケットシステム起動・終了・ナビゲーション業務

02.A 01.H B-Park-001 EH-100

発券機起動時の初期化制御ロジックエラー

「発券機のハードウェア初期化等必要な起動処理」において、初期化ステップの制御ロジックの誤り、または初期化の進行状態を示すデータが正常に更新されない場合

発生シナリオ

起動時の初期化ステップの順序が誤っている、または各ステップの完了判定が失敗する。仕様書では「障害停止画面 (S-008-01) を表示する」と記述されているが、その条件判定が誤り、エラーを検出しても画面表示が実行されない、または無限ループに陥る

影響

動物園入場者 入場券購入ができず、入場が遅延し、来園者の不満が生じる  
動物園経営者 営業開始時刻の遅延により、営業機会損失が発生する  
動物園係員 営業開始前の障害対応が必要となり、園内準備業務が遅延する。初期化ロジックのデバッグや修正作業が増加する  
だんだん市補助金担当者 システムの安定稼働が確認できず、補助金交付要件の充足性に疑義が生じる

11\_園内チケットシステム要求仕様書.md (EH-100 発券機起動, line 652-677)

抽象的テストケース (1件)

☐ B-Park-001-EH100-R001-ATC001 機能適合性 機能完全性

ラルフチャート ラルフチャート (テキスト)

Input (起動操作) 電源ON

Notice

【警告状況】  
入場管理との通信障害 / 入場管理との通信切断、入場管理との通信タイムアウト  
【起動タイミング】  
営業開始前 / 営業時間中

動物園係員として、発券機を起動して営業開始できる状態にしたい。入場者へのチケット販売を開始するため。

Output

正常時: ハードウェア初期化完了後、開始画面 (S-001-01) 表示 / 正常時: 複数アイコン初期化確認の取得完了 / 異常時: 障害停止画面 (S-008-01) 表示

State

【発券機の前回終了状態】  
正常終了 / 異常終了

都度LLMに指示を出して修正するのは大変なので、メモ機能を実装

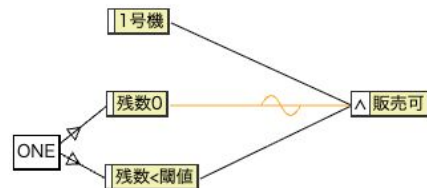
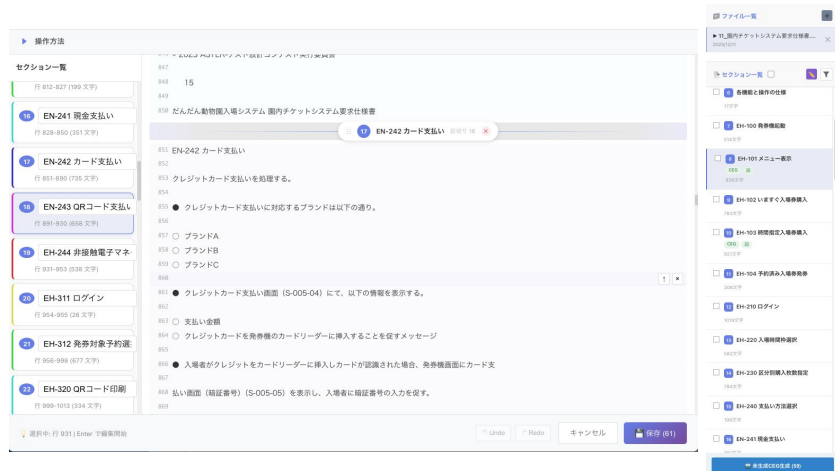
メモの内容を一括で修正指示することが可能に。

WebUI上で直接変更したり削除することも可能

モデルに当てはめて表現することで、条件に不足がないかを容易にチェックすることができる



# 原因結果グラフ(CEG)生成ツールの開発



デジジョンテーブル				
ノード名		#1	#2	#3
原因:	1号機	T	F	t
	残数0	F	F	t
	残数<閾値	T	T	f
結果:	[obs] 販売可	T	F	f

## [STEP1]

テストベースを読み込ませ、意味のある段落でセクションに分割

手動でのセクション区切りはもちろん、MCPサーバを提供しており、AI Agentsからセクション分割を指示することもできます

## [STEP2]

分割したセクションに含まれる論理関係をLLMに分析させ、原因結果グラフをmermaid形式で出力。[\*] mermaid形式では原因結果グラフの表記をそのまま表現することはできないため擬似的なモデルで表現。

## [STEP3]

原因結果グラフからデジジョンテーブルを生成できますが、本ツールでは未対応。既存のCEGTestなどを用いることでデジジョンテーブルに変換できます。

