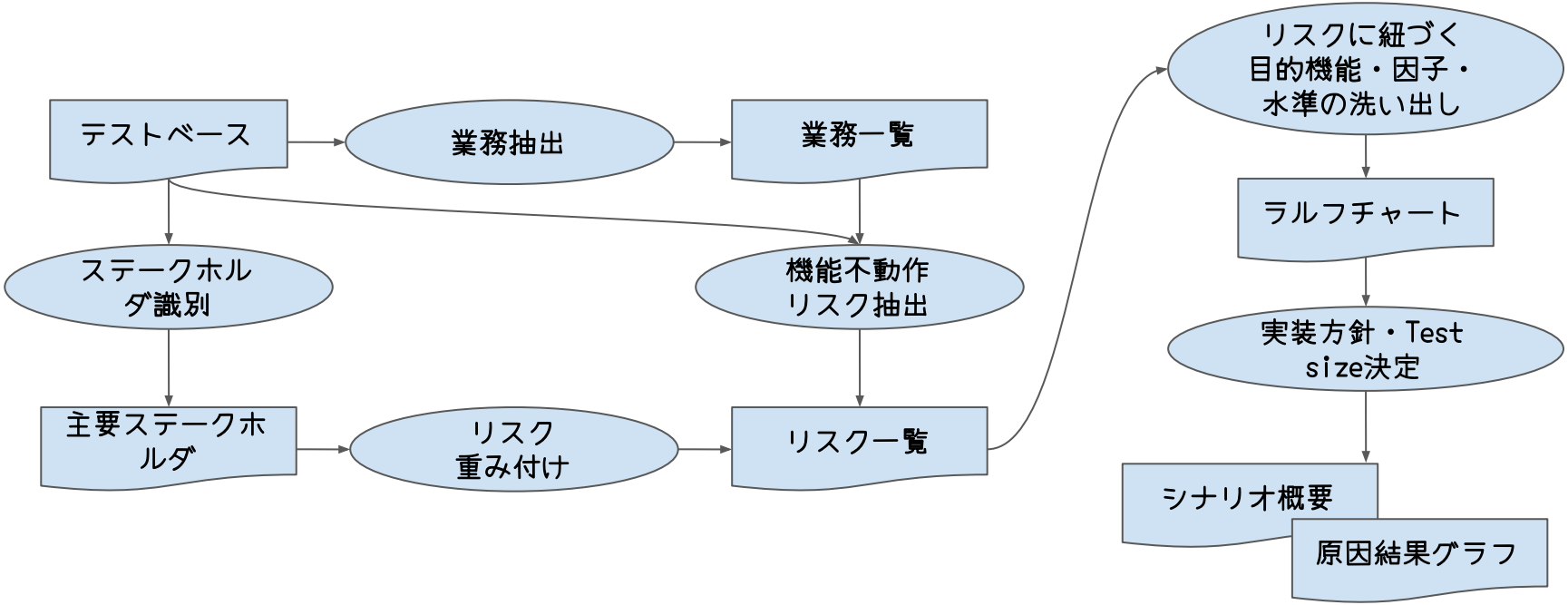
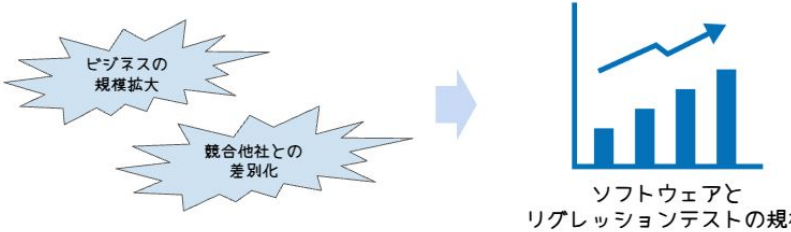
リグレッションテストの分析～設計の流れ



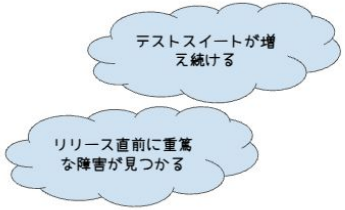
リグレッションテストとは

- 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。**



ソフトウェアと
リグレッションテストの規模

リグレッションテストのよくある課題



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

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		リグレッションテストにおけるリスクの捉え方
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	⇔	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が高い	⇔	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
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	⇔	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について

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

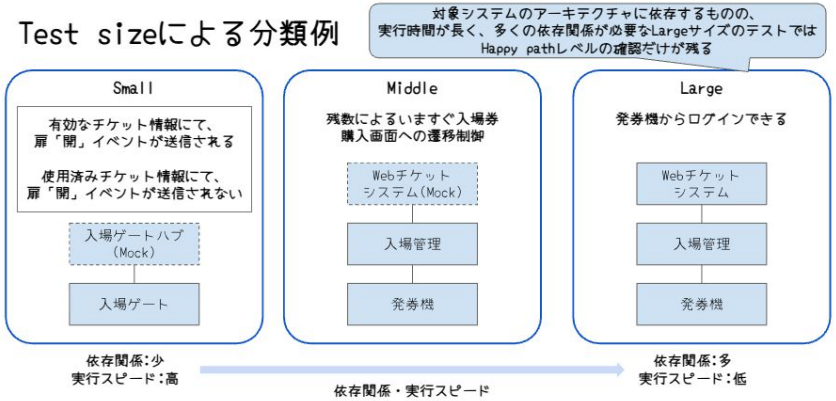
	Small	Medium	Large
Network access	No	Localhost only	Yes
Database	No	Yes	Yes
File system access	No(Mock)	Yes	Yes
Use external systems	No(Mock)	Discouraged	Yes
Multiple threads	No	Yes	Yes
Sleep statements	No	Yes	Yes
System properties	No	Partial Override	Yes
Time limit	60	300	1800+

予約管理DB、会員情報管理DBへのアクセス

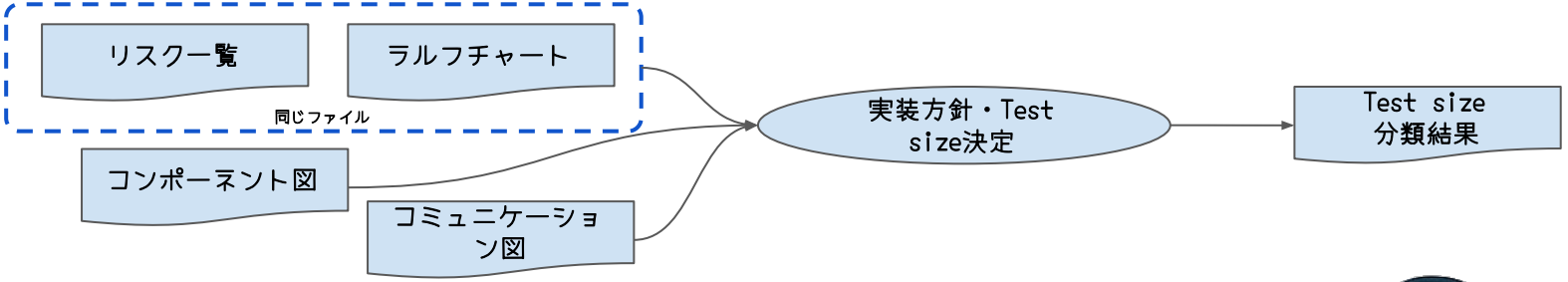
決済システム、運営管理システムへのアクセスはLargeのみ

00分/30分の時間枠の境界など

Test sizeによる分類例

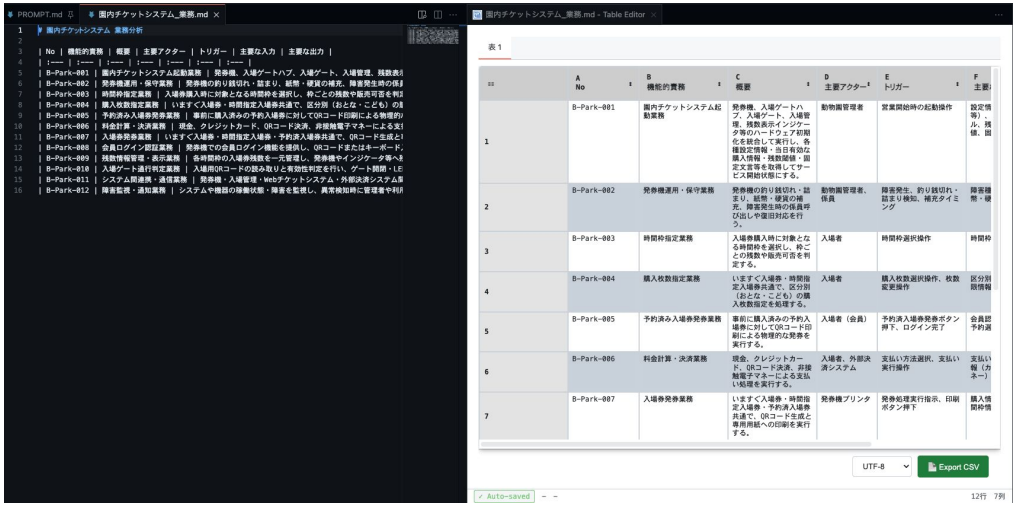


Test sizeを算出する際にLLMに対して与えているコンテキスト



*: Google Testing Blog. Test Sizes. 2010. https://testing.googleblog.com/2010/12/test-sizes.html

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



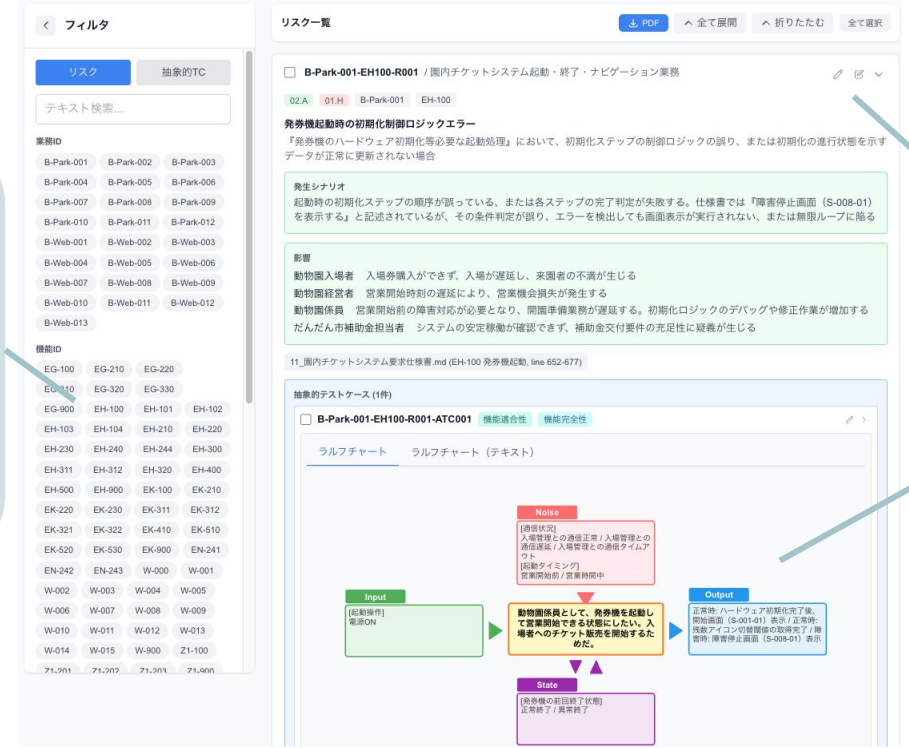
Excelやスプレッドシートにデータがある場合も多く、既存のデータからの変換が大変。
そこで、VSCode上でMarkdown形式のテーブルを直接編集したり、行・列の追加・削除が容易に行える拡張機能を開発[*]。
ExcelやスプレッドシートとMarkdown Table Editor上でコピー＆ペーストで相互にデータを貼り付けられたり、CSVインポート・エクスポート機能を兼ね備える。

*: 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



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

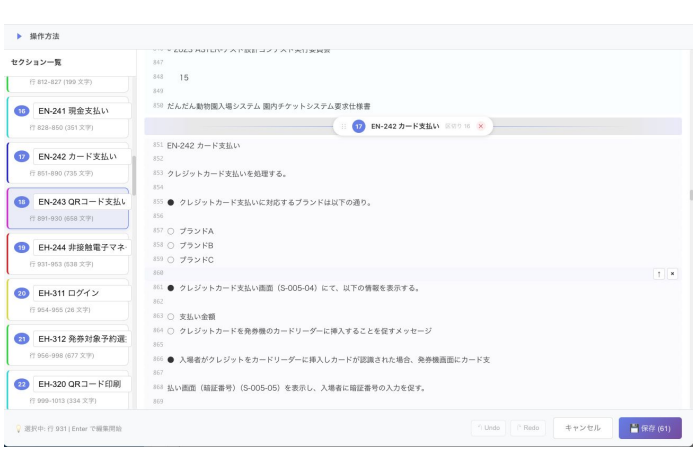
メモの内容を一括で修正指示することが可能に。

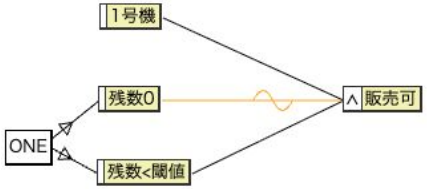
WebUI上で直接変更したり削除することも可能

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

*: Repository・実装は非公開です

原因結果グラフ(CEG)生成ツールの開発





デシジョンテーブル

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シ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シジョンテーブルに変換できます。

*: Repositoryは非公開です