テスト設計コンテスト2018 決勝

テスト設計成果物の紹介

チーム

「紙印テスト倶楽部」

メンバー

小田部

1

チーム紹介

チーム名 「紙印テスト倶楽部」

コミケと技術書展にて**テスト設計本**を**頒布**

新たなネタを披露しに来ました

メンバー紹介

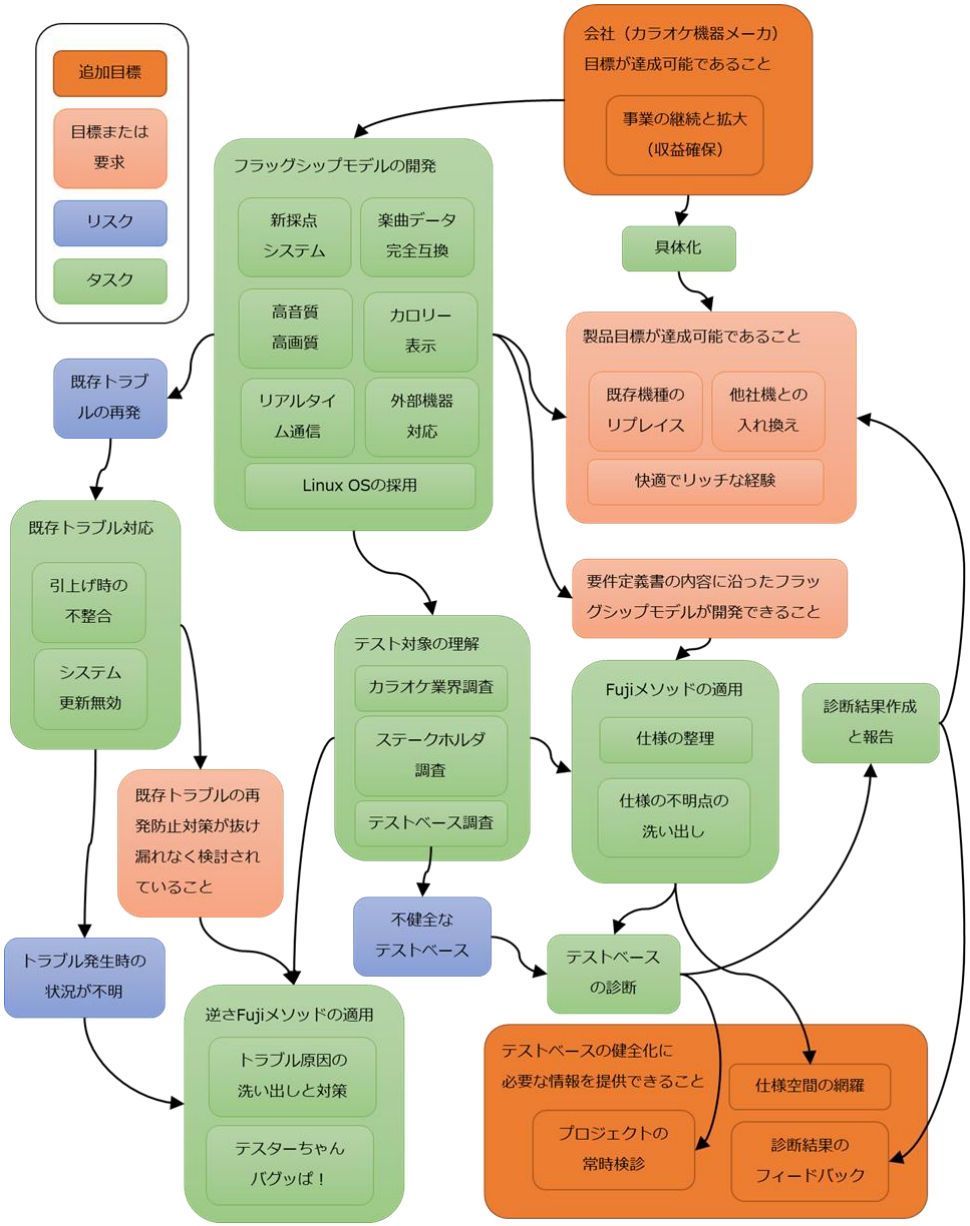
小田部

テスコン6年目にして

安定のボッチ

2

テスト設計全体像



テスト設計の目標

- **「テストベースの健全化」**が目標
  - 問題のあるテストベースは人を不幸にする
    - 開発されるのは**「問題のあるシステム」**
    - テスト開発で得られるのは**「問題のあるテストケース」**
    - テストベースの診断にはテスト設計成果物（≠テストケース）が最適である
      - テストベースとその関連情報を様々な観点でテスト設計成果物に変換する過程で、テストベースの問題点が浮き彫りになる
  - **テスト実施前に品質を確保**するのが紙印テスト倶楽部流のテスト設計
    - テストベースで定義された仕様空間をテスト設計成果物で診断し、開発されたシステムのテスト空間をテストケースを用いて網羅する

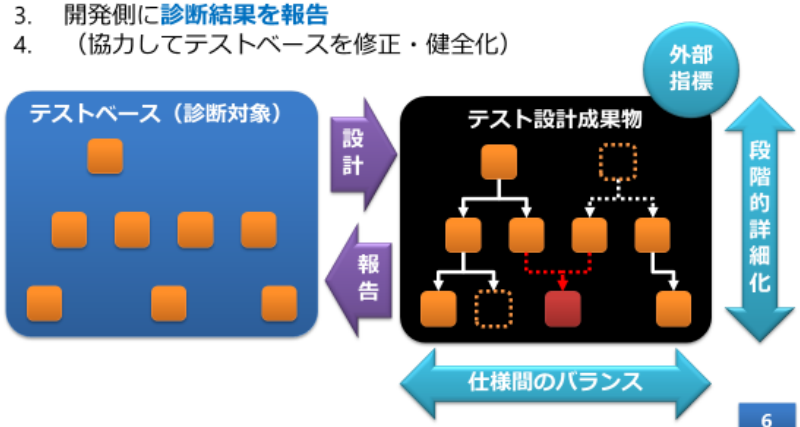
診断テスト（Diagnostic Test）

- 以下の方針でテストベースが健全か診断する
  - **仕様空間の網羅**
    - テスト設計を通じて仕様空間（テストベースで定義すべき仕様の集合）に不足や問題がないか網羅的に診断する
  - **診断方針の設定**
    - 仕様空間に対して適切な診断方針を設定する
  - **プロジェクトの常時診断**
    - 開発の進捗に合わせてリアルタイムでテストベースの診断結果を開発側にフィードバックする

- 上記のテスト手法を**「診断テスト」**と命名
  - 仕様空間を適切に診断し、その結果を速やかに開発側にフィードバックすることでテストベースの健全化をサポートする

診断テスト概要

1. テストベースを**テスト設計成果物**に変換
2. 垂直、水平、指標の**各診断の実施**
3. 開発側に**診断結果を報告**
4. （協力してテストベースを修正・健全化）



診断結果

- **垂直診断**
  - 自社の利益獲得にはカラオケ店オーナによるカラオケシステムの購入が必須条件だが、現状のテストベース通りのカラオケシステムが完成しても**オーナの購入動機に結び付かない**
  - 高音質・高画質なカラオケシステムは他社でも実現が容易なので、**製品の強みとなりにくい**
  - フラッグシップモデルの成功に欠かせない魅力的な**有料コンテンツの内容が不明**
  - ナイト店のユーザはカラオケ利用が目的でない場合も十分あり得るので、その状況から**カラオケ利用への導線を確立**することが重要課題

- **水平診断**
  - **古い外部機器（特にリモコン）との互換性**を確保すると、快適でリッチな体験から遠ざかる
  - 大容量通信の使用を前提としたフラッグシップモデルは**NBのナイト店に向いてない**

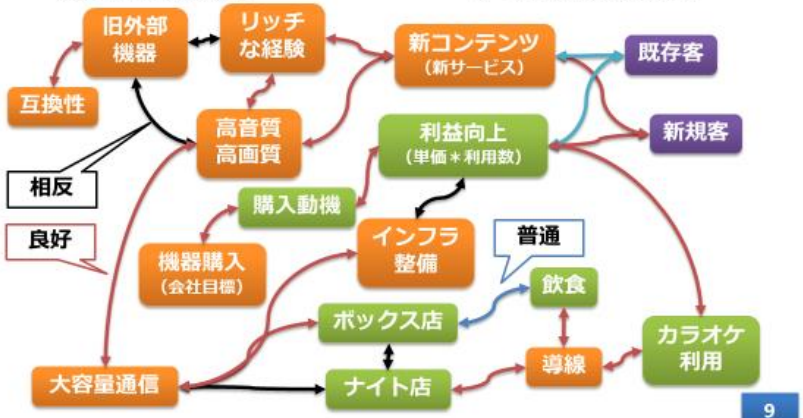
- **指標診断**
  - ユーザビリティを無視した、**全台再起動が必須**のオーナー設定
  - 運用性に影響する、**大容量通信のインフラ整備**とその維持

テスト設計サイクル

- テスト設計を**2種類のサイクル**で実施することで常時診断を実現
  - **職人型**のテスト設計サイクル
    - 仕様を分析し、**テスト観点および因子と水準を抽出**する
    - 高速かつ高精度なテスト設計が重要
    - 得られた知見の標準化や共有が重要
  - **探検型**のテスト設計サイクル
    - 「無いもの」や「矛盾」など、**「リスク」を可視化**する
    - これまでの診断結果から、次に実施するテスト設計の内容を決定する
      - 例：全体把握 → 資料の有無 → 記述の有無 → 記述粒度
    - 適切な**テスト設計内容の設定**が重要

要求同士の相関調査

- **機器購入実現の流れを想定**
  - 新コンテンツがカギだが現状不明
  - 新規客の獲得が重要
  - ボックス店とナイト店で要求が異なる
  - 旧機器との互換性確保は逆効果
  - ナイト店は導線確保が課題

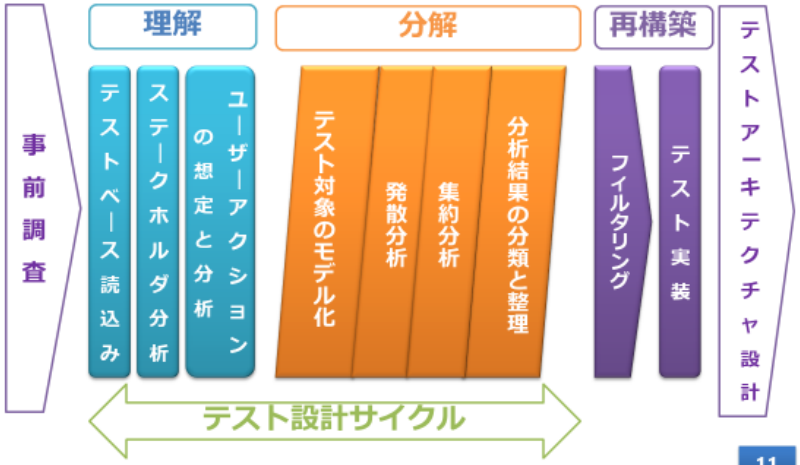


2種類のテスト設計手法

- **Fujiメソッド**
  - 通常のテスト設計に適用
- **逆さFujiメソッド**
  - 再発防止のテスト設計に適用

Fujiメソッド	逆さFujiメソッド
テストベースを基に設計を進める	バグの想定から始める
網羅的に設計を進める	ピンポイントかつ探索的に設計を進める
設計に必要なスキルが多様	バグを想定できれば設計を始められる
作業の量や種類が多い	作業の量や種類が発散しにくい
全プロセスの実施に多くの工数が必要	仮実装までなら1時間程度で設計可能
設計ノウハウが比較的形式化されている	非常に属人的

Fujiメソッド概要

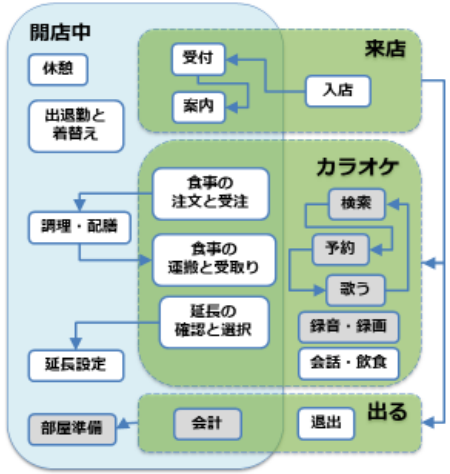


ステークホルダ分析

		当り前	魅力	反対
ボックス店	個人	曲の調整 練習環境	歌唱評価 全国比較	誤評価 曲が少ない
	団体	簡単操作 見やすい画面	曲が充実 演出盛上がり	使いにくい 曲が少ない
	ｵｰﾅｰ	一括設定 手間いらず	人気有料コンテンツ 専用の曲と映像	飽きられる
	新規	高音質 高画質	会員専用コンテンツ ライブ並みの臨場感	予約取れない 少ない専用ｺﾝﾃﾝﾂ
ナイト店	個人	課金明確 曲がある	周囲の注目 名音声ﾌｨﾙﾀ	課金トラブル 他の客が占有
	ｵｰﾅｰ	一括設定 手間いらず	人気有料コンテンツ ｵﾌﾗｲﾝで快適操作	課金トラブル 客同士トラブル
サプライヤ		機器の設定 しやすさ	自動接続 遠隔監視	不明な設定トラブル 解決しにくい

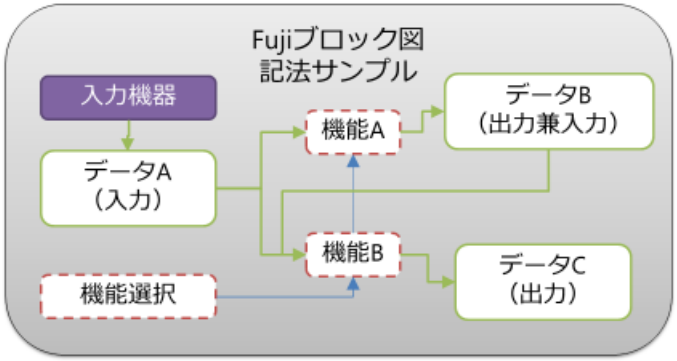
ユーザーアクションの分析

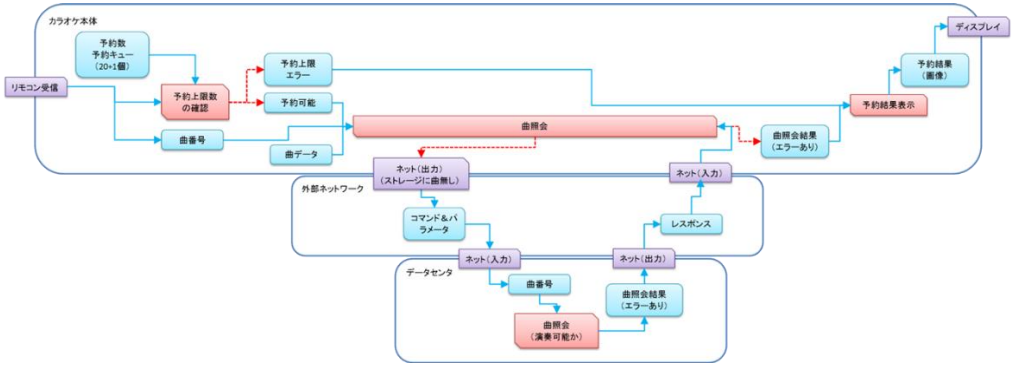
- テスト対象の利用を含む、**各ステークホルダの行動を想定**する
  - 開店中のカラオケ店を想定
- ステークホルダの行動にテスト対象の**システムが対応できるか**分析する
  - 「部屋準備」を早く終わらせるには、**一括標準設定機能**が欲しい
  - **録画中に店員の割込み**が入ったら嫌だろうな
  - 使い方を間違えたときに復帰出来るか？



モデル化（Fujiブロック図）

- テストベースを**個々の機能とその入出力でモデル化**
  - 個々の機能単位での**軽量・高速・柔軟なテスト設計**を可能にした





発散分析

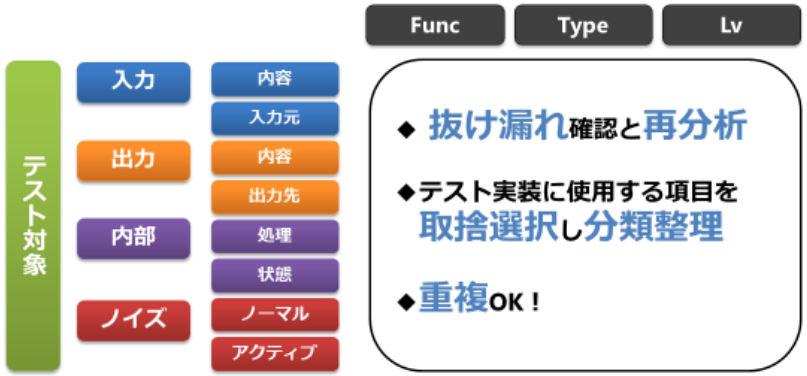
分類整理を前提としたマインドマップ

- ラルフチャートと論理的機能構造を参考に、メインブランチを設定

集約分析と分類整理

ツリー構造で分類整理

- 抜け漏れのチェックと再分析および「操作」「テストタイプ」「テストレベル」のタグ情報を追加し、テスト実装の内容を詳細化



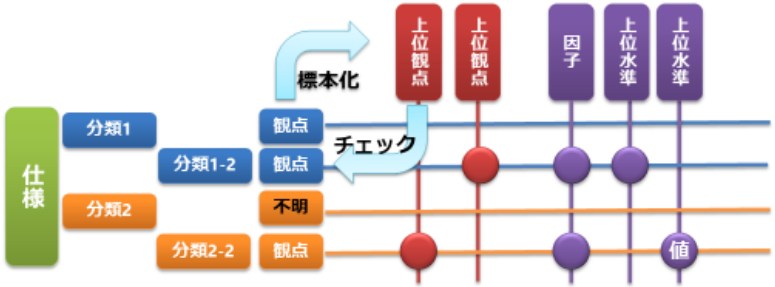
◆ 抜け漏れ確認と再分析

◆テスト実装に使用する項目を取捨選択し分類整理

◆重複OK！

観点および水準の標本化

- 各テスト設計サイクルでは以下の手順で分析漏れを軽減し設計精度を向上および平準化する
  - 分析結果から共通の観点および水準を標本化する
  - 分析時は上位観点および上位水準も用いて分析する
    - 仕様が曖昧なら「●」、詳細なら「値（水準）」を入れる
- 各項目を1つの表にまとめることで設計効率を向上させる



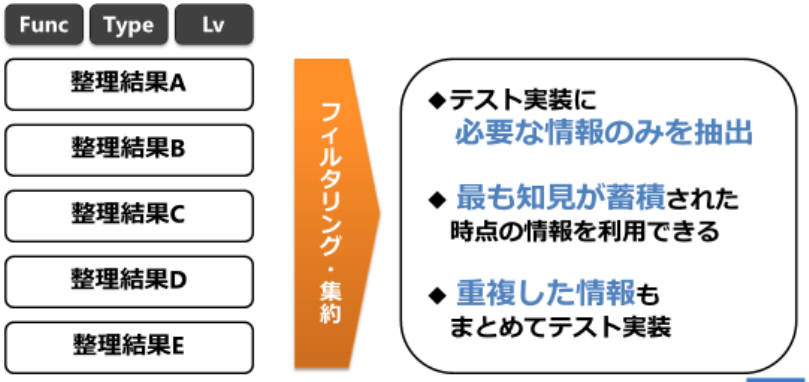
◆予約機能の集約分析（一部抜粋）

関連ID	観点概要（または指摘）	仕様の不足	仕様に問題	設定系					表示系						順序系			例外系			因子	水準	境界値・同値								検索系		データ属性					
				正常入力	未入力	異常入力	初期値・状態		表示内容	表示位置	表示エリア	表示数	干渉		時系列順	交換	人気順	エラー判定	有効／無効	限定条件		水準不要	指定値－	指定値＋	未入力	下限－	下限	中間	上限	上限＋	該当あり	該当なし	最古	最新	変更前	変更後	有効時	無効時
1-6																						●																
1-6-1 1-6-2	割込み予約可能か														●						予約曲数					0	1		20								
1-6-2	二重の割込み予約は無効化されるか																		●		割込み予約												未予	予約済み			
1-6-4	予約順で表示されるか														●						予約曲数						0		20								
1-6-4	予約順を変更可能か															●					予約曲数					1	2		20								
1-6-4	取り消し可能な場合に有効か																		●		予約曲数					0	1										
1-6-4	予約取り消しが可能か																		●		予約順						最初		最後								
1-6																						●																
1-6	全ての予約が表示されるか								●												予約曲数					0	1	20									
1-6	※割込み予約はどう表示されるのか？	●																				●																
1-6	※一画面中に表示する予約数は？	●																			予約曲数					●	●	●	●								
1-6	予約した曲名が表示されるか								●	●	●										曲名マスク設定									●							
1-6	オーナー設定から曲名マスクが可能か																		●		曲名マスク設定														●	●	
1-6	予約曲のアーティスト名が表示されるか																		●		アーティスト名									●							
1-6																						●																
1-6	割込み予約の結果が表示されるか			●					●									●			割込み結果														●	●	

フィルタリング と集約

全分析結果をフィルタリングで集約

- 全分析結果を「操作・作業」「テストタイプ」「テストレベル」でフィルタリングし、集約された情報を用いてテスト実装



逆さFujiメソッド

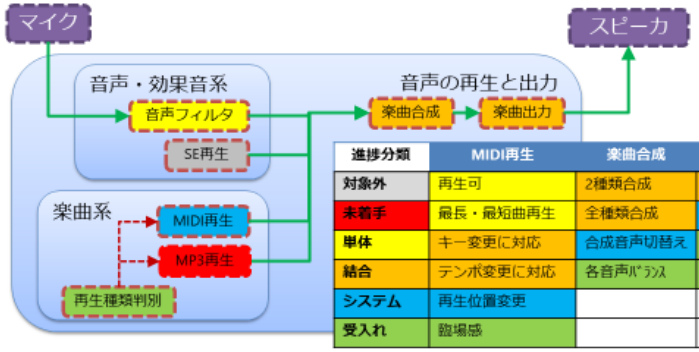
- 引き上げ操作不具合の想定原因
  - ネットワーク**エラー検知**機能の動作不良
  - 他の操作が**並行して実行**可能だった
  - **データセンター**側の不具合
  - テストーちゃん（サブライヤ）の**操作ミス&報告漏れ**
  - **手順書**の記述漏れや分かりにくさ等の不備
  - 引き上げ操作の**二重実行**が可能だった
  - 引き上げ操作中に**自動処理**が実行される
  - 引き上げ操作エラー時の**リカバリ**手段がない
  - カラオケシステムの**故障**
  - **ネットワーク設定**ミス

統合（相互補完）

- トップダウンとボトムアップの分析結果を**摺り合わせ**、各テストレベルのテスト観点を決定する
  - トップダウンは主に**最終的な品質レベル**を決定する
  - ボトムアップは主に**同時に確認すべき機能**を決定する

テストアーキテクチャ設計

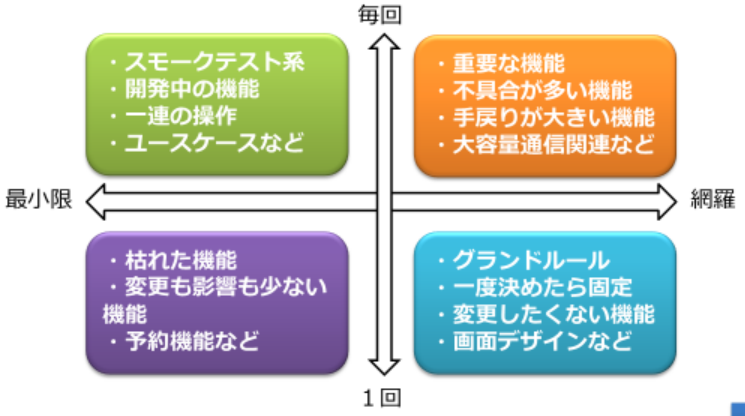
- 開発に合わせたテストアーキテクチャ
  - 開発とテストの**進捗から次に実施するテストを決定**
  - 各機能の**品質レベルも確認**



進捗分類	MIDI再生	楽曲合成	楽曲出力
対象外	再生可	2種類合成	音声入出力
未着手	最長・最短曲再生	全種類合成	音量調整
単体	キー変更に対応	合成音声切替え	臨場感
結合	テンポ変更に対応	各音声ﾊﾟﾗﾝｽ	ライブ並み
システム	再生位置変更		
受入れ	臨場感		

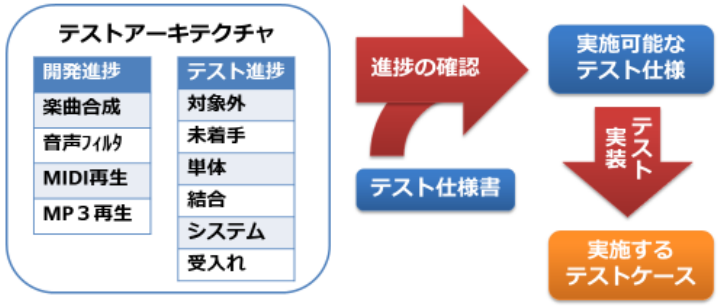
テスト実施の品質

- 実施内容を４カテゴリで分類
  - 不具合の発生状況に応じて所属するカテゴリを変更する



連携型テストケース

- **開発の進捗**に合わせて実施可能なテストケースを実装する
  - 開発済みの機能と実施済みのテストレベルから次に**実施可能なテストケースを判定**し、必要な情報を**テスト仕様書から抽出**してテスト実装する
  - テストアーキテクチャと連携する事で、開発状況に合わせたテストケースの実装と実施が可能になる



テスト設計まとめ

- **テストベースの健全化**
  - テスト実施前に品質を確保
  - 診断テストの実施
- **テスト設計サイクル**
  - 短いサイクルでテスト設計を繰り返し実施する
  - 知見の蓄積、標準化、水平展開
- **2種類のテスト設計手法と設計支援ツール**
  - 使い分けることによるテスト設計の完成度向上
  - テスターちゃん大人気♡

参考文献

- 唯野 奈津実著：「カラオケ上達100の裏ワザ」
- 湯本 剛氏　発表資料：「テストアーキテクチャの具体例」
  （http://jasst.jp/symposium/jasst12tokyo/pdf/A2-6.pdf）
- 小田部 健著：「テスト設計入門 2017年度版」

◆バグ連想ツール『バグッば！』 の遊び方と特徴

・本クリアファイルは3種類のワードを眺めながらバグをポンポン連想する（以降「バグッば」と呼称）ツールです
・ワードを3種類（カテゴリ：黄、ガイド：白＆紺＆赤、カタリスト：三角の赤）用意することで、バグッばを容易にします
・クリアファイルを傾けるとカテゴリワードとガイドワードが切り替わります。
・複数人で遊ぶとよりバグッばしやすくなります

◆バグッばの例

・表示機能 ＋ 「隠」 → メッセージウインドウ等が重なって、見たい情報が隠れる
・表示機能 ＋ 「混」 → 表示中のメッセージに異なるフォントが混ざっている
・メニュー操作 ＋ 「迷」 → 目的のメニューが探しても見つからない
・メニュー操作 ＋ 「散」 → 関連するメニュー同士がまとまっていない
・画面操作 ＋ 「戻」 → 「一つ前の画面に戻る」ボタンがない
・マウス操作 ＋ 「鈍」 → マウスを操作してもカーソルの反応が鈍い
・マウス操作 ＋ 「互」 → OSによって動かないマウスがある
・数値入力 ＋ 「限」 → 上限を超えて数値を入力できる
・数値入力 ＋ 「計」 → 計算結果が正しくない
・時刻表示 ＋ 「滞」 → 重たい処理を走らせると時刻表示が滞る
・ファイル読込み ＋ 「腐」 → 古いファイルが読み込めなくなる
・ファイル保存 ＋ 「消」 → 上書き保存に失敗するとファイルが消える
・ログイン ＋ 「合」 → PWが合ってなくてもログインできてしまう
・漢字変換 ＋ 「劣」 → IMEを更新したら漢字の変換精度が劣化した

◆ガイドワード：バグと関連がある連想用のワード

漢字	キーワード
差異	異常、違和感
抜ける	人、モノ、考えが抜ける、抜け道
内容	容量、許容、容姿、容器
要求	要件、要望、かなめ
知識	認識、識別
理由	理解、原理、論理、
再び	再生、再開、再帰、再現
終了	終始、終日
途中	最中、中間
戻る	後戻り、呼び戻し、先祖返り
休眠	無反応、待機
遊離	離散値、管理や制御から離れる
越える	Over、越権
過ぎる	Pass、過剰、過去
限度	限界値、限界、限定
同値	同じ、同時
判定	判断、評判、批判
無効	有効、効果
未満	未知、未熟
綻び	不完全さ、疲弊
延期	延長、予定変更
予測	予兆、予約、予備
見え	道具、具体化

漢字	キーワード
計画	計測、計る、計算
詰む	停止、致命的
滞る	遅延、ボトルネック
診察	診断、定期検診
順序	順番、順位
飽きる	飽和、陳腐化
慮る	配慮、遠慮、浅慮
手段	片手、使い方
操作	操縦、使いやすさ、遠隔操作
権利	権限、利権、越権
乱打	連打、打鍵、代打
誤る	誤解、誤差、誤植、紛らわしさ
陳腐	劣化、旬を逃す
障害	保障、目障り、故障
準備	備え、予備
歪み	正しくない、歪曲
代替	世代、交代、お代
派生	派遣、派手
影響	反響
欠点	欠損、欠員、欠乏
政治	悪政、政局
裸	弱点、脆弱
劣化	見劣る、優劣

漢字	キーワード
損失	損害、出来損ない
鈍い	鈍重、遅れ
流れ	フロー、二流、交流
単独	単純、単位
切断	判断、横断、遮断
経路	岐路、回路
飛躍	突飛、飛越す
待機	招待、期待、待避
割込み	割合、分割
書く	書式、書類
最初	初心者、初期値、初歩
消失	欠損、消去、解消、消失
漏れる	間接バグ、漏えい
混在	分類、玉石混合、混雑、混乱
合格	規格、適合、合格、合鍵
拡散	分類、胡散、解散、拡散
特別	特殊、特性
分類	分解、等分、差分
共有	共通、共感、共存
統一	統合、統制、系統
隠す	隠密、隠し味
欺く	詐欺、見た目正常
重複	重い、多重、重要

漢字	キーワード
縮小	圧縮、縮図
五感	感度、感情、感染
入力	出し入れ、受入れ
場面	現場、市場、場所
時刻	時機、時間
電波	電源、充電
国外	国語、国産
屋内	屋外、問屋、部屋
規制	規約、規格
協調	協議、協同、協力
独立	独自、独占、単独
例外	以外、想定外、外観
稀	稀少、稀有
満足	満たす、満点
禁止	禁則、解禁
耐久	耐性、耐用
遅れ	遅延、遅刻
全部	安全、全体、完全
連携	連絡、連続
反応	応答、応急、応用
間隔	時間、空間、中間
互換	相互、交互
資産	資料、資金、資格

◆カテゴリワード：バグのカテゴリを絞って連想しやすくする役割のワード

名称	説明
利用	利用者のカテゴリ。想定の斜め上の利用方法でバグになったり、飽きられて使われなくなったり、彼等の立場で考えてみましょう。
境界	テストの基本カテゴリ。境界値や同値はバグが潜みやすいのです。
運営	プロジェクト運営のカテゴリ。無理な計画や情報共有不足に人材不足など、運営がまずいとバグも入りやすくなります。
保守	保守や派生のカテゴリ。セキュリティ問題やシステムのレガシー化や環境の変化などが原因のバグが想定されます。
整理	情報整理のカテゴリ。仕様の抜け漏れや情報の共有不足に分類整理の不十分さなど、曖昧さにバグが潜みます。
事業	ビジネスのカテゴリ。例えバグゼロのシステムでも利用されない、利益が出ないならそれ自体が致命的なバグです。
ｼｽﾃﾑ	システムのカテゴリ。外部との連携や負荷に互換性など、システムの安定性を脅かすバグは何でしょう？
UI	ユーザーインターフェースのカテゴリ。使いにくい、分かりにくい、レスポンスが遅い等。利用者置き去りはバグの素。
実行	実行処理のカテゴリ。割込み、データの読み書き、初期値など、仕様書通りの動作になっていますか？
仕様	仕様や要件、要求のカテゴリ。その仕様は実現可能で抜け漏れなく要求を満足させられますか？
組合せ	組合せや独立動作のカテゴリ。その機能は他の機能に影響を及ぼさないか。あるいは複数機能が協調して動作するか。
環境	現実の動作環境のカテゴリ。異なる文化や様々な利用場面、順守すべき規約など、利用される環境の想定は十分か？
状態	システム状態のカテゴリ。各状態での動作や状態遷移の確認はテストで重要です。

◆カタリストワード：ガイドワードを絞って連想しやすくする役割のワード

名称	説明
説明	目の前のテスターちゃんが君が考えたバグを聞きたがっているぞ！テスターちゃんにバグを分かりやすく説明してみよう。
誘因	そのバグってどんな状況だと混入しやすいかな？　その状況だと他にどんなバグが混入しやすいかな？
合体	ワード同士を合体して新たなワードを生み出そう！　合体させる順番を変えてみるのもいいぞ。
基準	そのバグは別の人から見たらバグじゃないかもしれない。それぞれが持つ基準でバグを想定してみよう。
反転	ワードの持つ意味を反転させてみよう。それだけで発想の幅が２倍以上に広がるぞ！
ノイズ	割込み、電波障害、想定外の高負荷など、ワードの効果を十分発揮させなくするノイズは何だ？
加減	ワードの効果を、何度も、大きく、頻繁に、早く、大量に、長時間、同時に、またはそれらを反転させて発揮させてみる。
自他	そのバグは想定対象自体が原因で発現してるのか？ それとも周りの環境が原因で発現してるのか？

◆現場での活用方法

・設計時にバグッばすることでバグを予想し、整理した内容をチェックリストやテストで利用することでバグの予防や検出に役立ちます
・ベテランと初心者で一緒にバグッばすることで、バグの知識移転に役立ちます
・バグを具体的に説明することで、バグの原因や現象に発生リスクなどを明確に意識できるようになります
・電源不要で動作し、手続き不要でほとんどの現場に持ち込むことが出来ます