JaSST ‘24 Hokkaido 基調講演

ソフトウェア品質のダンジョンマッピング

Aug. 23rd, 2024

鈴木 一裕

Image: Generated by GPT4o

この夏、いちばんのホラー

!?!?

「こちら千歳発のご予約のようですが、
お間違いないでしょうか‥？」@羽田

自己紹介

鈴木 一裕

Kazuhiro SUZUKI

株式会社 日立製作所
サービスプラットフォーム品質保証本部 担当部長

※今日は、会社の立場でのお話ではありません。

- ソフトウェアのQA業を20年ほど。
- テストを中心としたソフトウェア品質保証について考えることを趣味とし、マイナーなブログ『ソフトウェアの品質を学びまくる』を細々運営。ときどき社外イベントで発表などしてます。
- ISO/IEC JTC1/SC7/WG26（ソフトウェアテスト）に所属、テストの国際規格化にチョット貢献。
- ソフトウェア関連書籍の翻訳や執筆にチョット関与。
  - 『システムテスト自動化 標準ガイド』（翻訳・監訳）、『ソフトウェア品質知識体系ガイド(第3版)』（ごく一部執筆）、『実践ソフトウェアエンジニアリング 第9版』（2021年、一部翻訳）
- 打楽器のカホンと、キックボクシングを始めました。趣味の多い中年を目指しています。

自分が何が得意で、どんな発表ができるのか？

- **大前提: ソフトウェア品質の世界は広く、深い！**
  - それはまるで、剣と魔法の世界の、ダンジョンのごとし。
- **特に能力が高いわけでもなく、現場からも離れている、そんな鈴木の能力で、何ができるのか？**

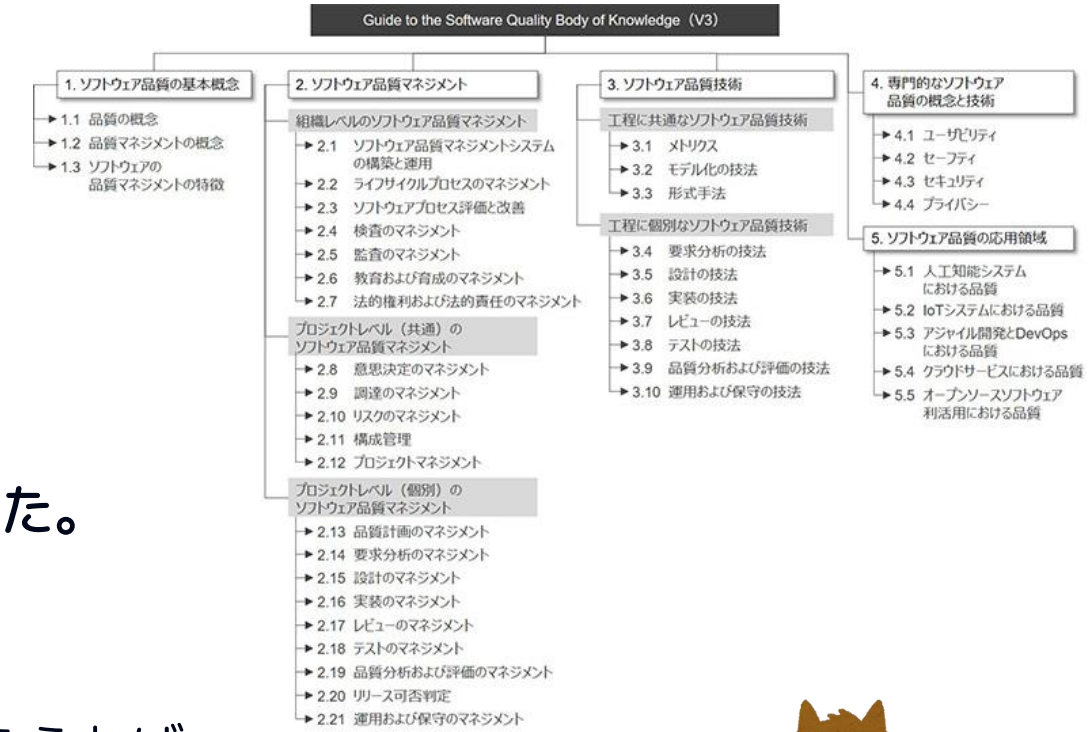
  QAエンジニアを、RPGにおける職業に雑に喩えてみると‥
  - **戦士**: 探索的テストを含めたテスト実行で力を発揮する！
  - **魔法使い**: テスト分析・テスト設計を駆使して、効果的・効率的なテストを導く！
  - **僧侶**: 設計段階でのレビューで、バグの作り込みを防ぐことを得意とする！
  - **勇者**: 開発全体を見通してテスト計画とモニタリング、改善を進める！
  - **賢者**: 新しい品質保証の技術を発見・普及する！
  - マッパー(*1): 深遠な品質ダンジョンを調べ、構造や仕組みを仲間に伝える。地味。

剣も魔法も使えんが、ダンジョンを調べるのが好き！ これだ！

(*1)「マッパー」という職業は、『フォーチュンクエスト』（深沢美潮・著）で知りました。RPGの世界で一般的なロールなのかは‥？

本日の発表について

- ダンジョンの進み方
  - 『ソフトウェア品質知識体系ガイド』（SQuBOKガイド）が樹形図(*1)を示している。
    つまり、ダンジョンの地図はもうある‥(糸冬)
  - が、今日はこういう体系化された進み方はせず
    ソフトウェア品質に関するキーワードを、
    数珠つなぎ的に追う旅をしましょう。



- Key Takeaways ← カッコいいので使ってみた。
  - 「こんなものがあるんだなー」
    「この辺、自分でも学んでみたい！」
    というリスキリングネタを1つでも見つけてもらえれば。
  - 眉唾で聴くくらいがちょうどいいです！
- TwitterとQ&Aについて

(*1) NTTデータ・町田 欣史さんの記事 品質技術のニューノーマルが分かるSQuBOKガイドの改訂 より引用。

唐突なエピソード

ドイツの友人が東京に来るというので、観光の計画を立てました。
その計画の過程でわたしたちは、
「品質と顧客価値」を見誤っていることに気づきました。
次のような検討の、何が問題だったのでしょうか？

友人の要望「Shibuya Crossing is a must-see for me.」

われわれの計画「じゃあ空いてる午前中、早めに渋谷に行こう」

最後あたりで、答え合わせをしましょう。

では、品質ダンジョンに入っていきましょう！

「ソフトウェア品質」という
ダンジョンにどこから踏み込むか。

多くの方はやはり、
「テストの実行」から
入ったのではないでしょうか？

想定経過時間 6:00

テスト実行

なぜあなたのテスト実行は退屈なのか？(*1)

(*1) よくある「ビジネス本」の煽りタイトル風ですんません。

テスト実行、退屈だなーという方 ✋

多い場合

→ 以降のスライドに進む

少ない場合

→ QA業界の強い人たちが
「テストは創造的な仕事である」と主張するので、
「テストはつまらない」と言いづらい空気なのだ、
と信じて以降のスライドに進む

テスト実行が退屈になる要因と対策

バージョンアップのたびに
まったく同じテストをやる
ことになってしんどい‥

テストケース一覧にあるから
やってるけど、やらなきゃ
いけない理由は誰も知らぬ‥

手順が細かく指定されていて、
それ以外のことをやることは
求められていない‥

「この機能、リリース前の
バグ出ししておいて〜」で
ぶん投げられる‥

テストしてもバグ出ないし、
自分は役に立っているのか
‥？

要因❶
単調な繰り返し

要因❷
テストの必要性不明

要因❸
厳密過ぎる手順

要因❹
無秩序

要因❺
バグが出ない苦しさ

自動化
正確・多数・高頻度が求められるテスト実行に効くかも

テスト分析・設計
「なぜ」必要なのかを明確にし、論理的で納得感のあるテストを作る

柔軟なテスト、探索的テスト
直感・経験・知識をフル活用し、自由度の高いテストでバグ摘出を目指す

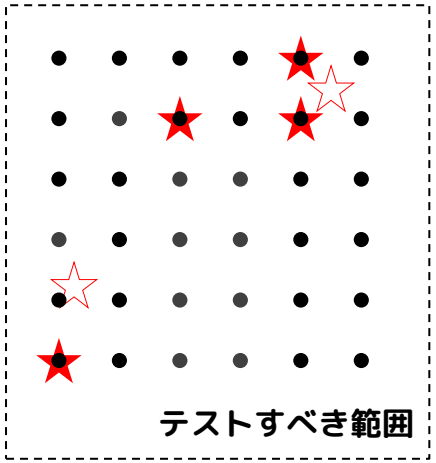
ユーザ価値の深い理解
仕様との合致だけでなく、ユーザによい結果をもたらせるかを考えてテストする

テストの目的の確認
保証のためのテストという概念を理解する

テストの目的と実行の柔軟性

● テスト実行
★ 見つかるバグ
☆ 見つからないバグ

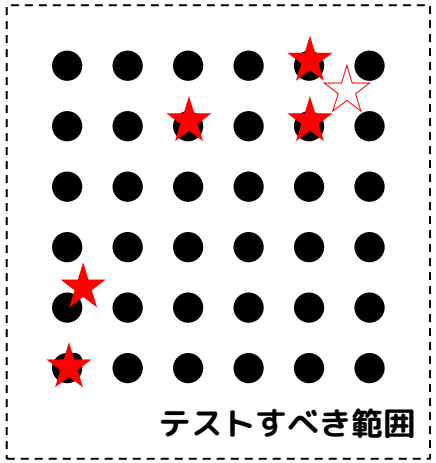
テストの目的



保証のためのテスト

- 全体をくまなく検証し、バグが見当たらないことを確認する。
- バグの摘出においては効果、つまりどれだけ見逃さずに済むかが大事。
- バグ検出効果 = 4/6 = 67%
  バグ摘出効率 = 4/36 = 11%

テストの柔軟性

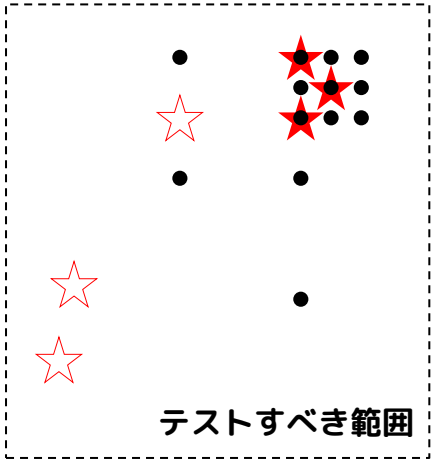


柔軟なテスト

- 気になった部分は再度実行したり、手順やパラメタを少し変えてみたりと周辺を探ってみる。
- 人間の気づき・違和感が効果を出しやすい方法。

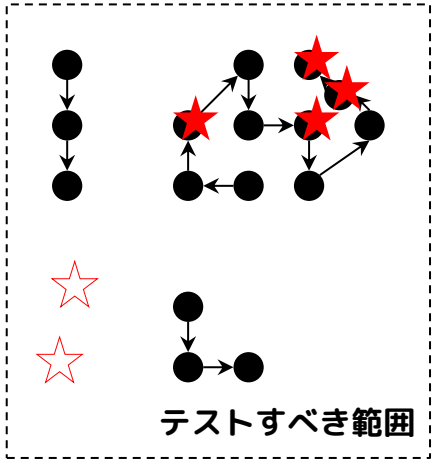
補完

両方のテストで補完し合い、かつ柔軟性をもたせるのが重要。



バグ出しのためのテスト

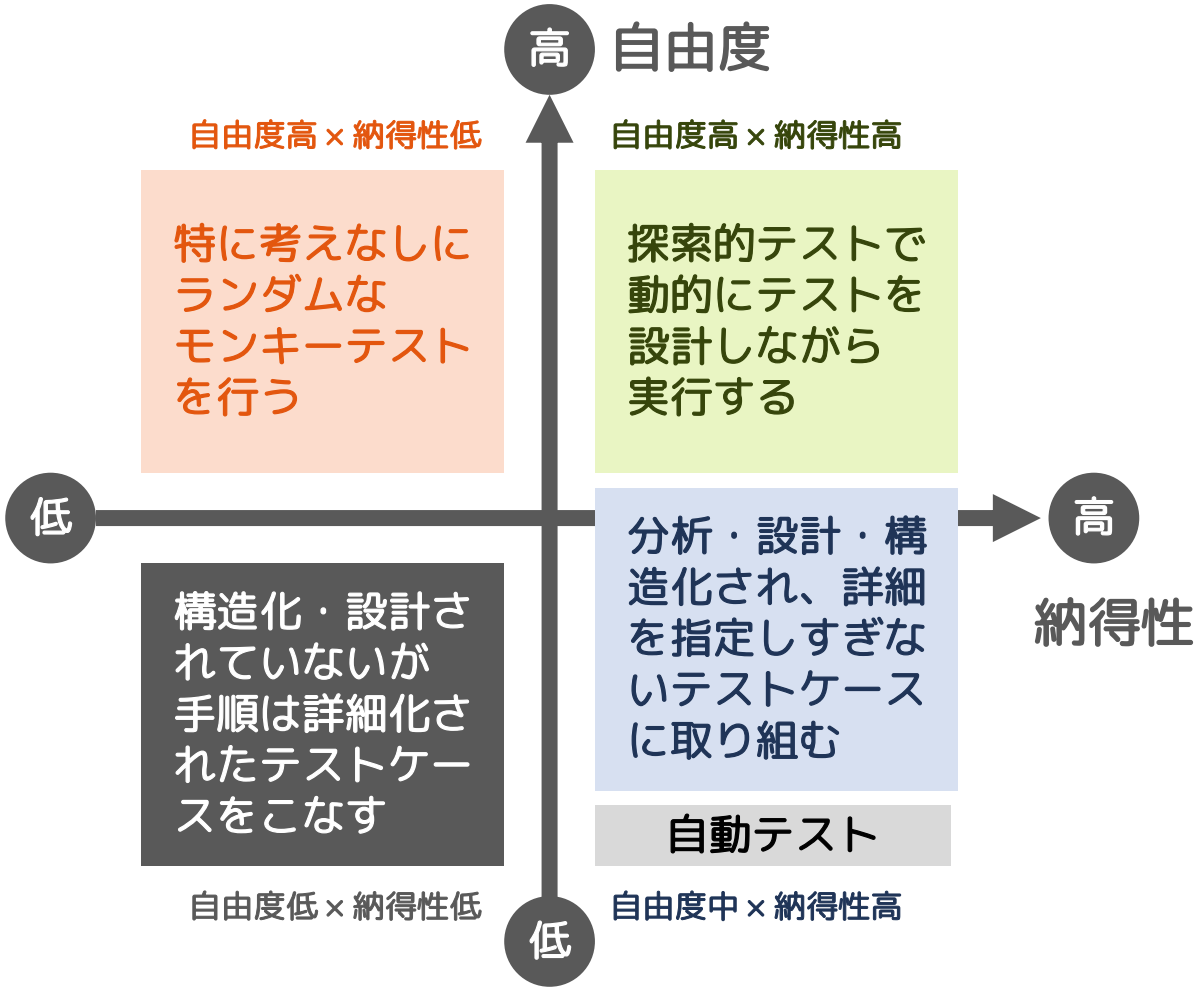
- 重要なバグが出そうな場所を集中的に確認する。
- バグの摘出においては効率、つまり少ない工数で多くのバグを見つけることが大事。
- バグ検出効果 = 3/6 = 50%
  バグ摘出効率 = 3/13 = 23%



探索的テスト

- 対象のプロダクトを触って得た反応をインプットとして、次に行うべきテストを決めながらバグの摘出を目指す。
- 人間の経験値・ドメイン知識が効果を出しやすい方法。

ご参考: 納得性と自由度の4象限



- モンキーテストを左上の
  自由度高×納得性低
  の「悪い位置」においたけれど、
  「自動化されたモンキーテスト」(*1)は
  網羅性が高くなることでむしろ右下、
  自由度低×納得性高
  に移動するかもしれない。
- 自動モンキーテストとファズテストは、
  物量で勝負をかけるという意味で、
  実は兄弟なのか？
  - モンキーテストは、操作・値のバリエーションを投入する。機能テスト寄り。
  - ファズテストは、脆弱性を露呈させやすい値を投入する。セキュリティテスト寄り。

右側上下での補完が筋よさげだが、テストの分析・設計が必須。

(*1) ベリサーブ社の記事が参考になります。『モンキーテストとは。ランダムテスト・アドホックテストとの違い、実行方法など解説』
(*2) ファジングについてはこちらの記事を参考にしました。『ファジングでIoT機器のセキュリティ対策 ～第3回　ソフトウェアテストの新常識：ファジング入門』

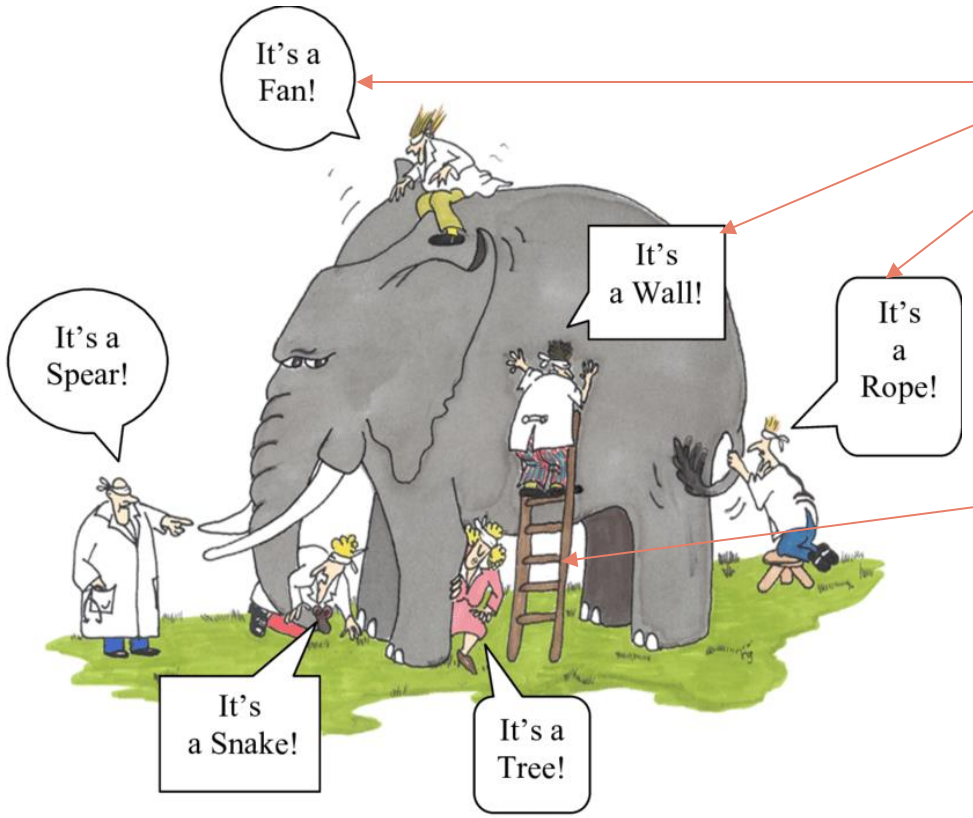
ただ「テスト」で済むものをなぜ、
「分析」「設計」「実行」
などといって切り刻むのか？

象のテストでテストプロセスを雑に説明する

デブサミ2023の資料を
一部修正、再掲

The Blind Men and the Elephant

「全体観がない」メタファーで使われがちだが‥



この人たちは「現実の象がいかなるものか」を、
いろんな観点できっちり見ている。

- 象に関する事前情報を得たうえで、象のどこをテストするかの観点を特定するのが、テスト分析。
  - 胴体、しっぽ、牙、‥、特徴ある箇所を確認。
  - ~~象のユーザビリティという非機能要件もカバーしている。~~
- 象のテスト観点を効果的・効率的に検証するテストケースを決めるのが、テスト設計。
- 設計したテストケースを実行する順番を決めたり、環境を整えたりするのが、テスト実装。
  - チェックのための足場を用意したりしている。
- 実際の象チェックを行うのが、テスト実行。
- 全体の計画やモニタリングが、テスト管理。

成果物も求められるスキルも異なるので、プロセスとして分割している。

The Blind Men and the Elephant: A Metaphor to Illuminate the Role of Researchers and Reviewers in Social Science - Scientific Figure on ResearchGate.
Available from: https://www.researchgate.net/figure/The-Blind-Men-and-the-Elephant_fig1_290127733 [accessed 14 Jan, 2023]

で、テスト実行は、退屈で低スキルな仕事なのか？

- 本来のテストの実行は、
  - ソフトウェアの知識・テストの知識・ドメインの知識
  - 顧客要求の理解力・想像力
  - 違和感を見逃さない観察眼と注意力
  - 次にすべきことを決める判断力と発想力
  - 問題を他者に伝えるコミュニケーション能力

  が求められる、難易度の高い仕事です。

プロダクトを理解と価値の判断は、
実物を触ることなしには難しい。
テスト実行しようぜ。

テスト実行といえば、バグでしょう。

想定経過時間 18:00

バグ管理とバグ分析

なぜQAエンジニアは
バグ管理に手間をかけたがるのか？

言葉の使い方の確認

- ややいい加減になりますが、
  - 本発表でいう「バグ」は、プログラム上の問題に限らず、ソフトウェアの欠陥一般を指すことにします。
  - 本発表でいう「バグ管理」は、テストを行う過程で挙げられる指摘を管理することを指すことにします。
    - よって「バグ管理」される対象が必ずしも「バグ」とは限りません。
    - たとえば、「テスト環境の問題であり、ソフトウェアには問題がなかった」という場合であっても、これを記録しているチケットは「バグ管理」の対象としています。
  - バグを管理するチケットのことを「バグチケット」と呼んでいます。
  - バグのことを「インシデント」「イシュー」「ハッピー」「ドラゴン」と呼称しているチームもありますが、ここでは雑に「バグ」という言葉を使っていきます。

バグを一元的に管理しておくことで、何が嬉しい(*1)か。

1 情報の集約

- 起票者と開発者の問題認識・調査結果・対策方針などの情報が一か所に集まる。
- 何が起きて、どういう根拠でどう対処したか、チケットを見れば振り返ることができる。

2 タスク管理・テスト管理

- 重要度や対策工数・難易度を横並びにして、対処のプライオリティが決められる。

3 バグ分析 ← QAエンジニアのみの興味になりがちかも

- バグの多い機能、よく起きている現象などの傾向から、プロダクトの品質がわかる。
- バグを作り込みやすい・見逃しやすい工程などの傾向から、プロセスの課題がわかる。

この後、バグチケ運用バトルのスライドを
4枚用意したのですが、時間短縮のために飛ばします！

(*1)「何が嬉しいかというと」は、極めて限られた分野の人にしか使われない言い回しらしい‥。

バグチケット運用バトル(*1)

その1: バグチケットの入力項目の多さで対立が起きる

- 目的1（情報集約）と目的2（タスク管理）だけが重要と考える人
  - 「バグ票の作成に時間をかけたくない。分析は役に立たないから、できるだけ少ない入力項目がいい」
    「開発効率を考えろよ！」

  ➡ きちんとしたバグ分析は、開発効率や品質にも寄与します！
- 目的3（バグ分析）を重視しすぎる人
  - 「いつどんな分析をしたくなるかわからないから、できるだけ多くの入力項目がある方がいい」
    「仕事なんだからやれよ！」
  - QAの発言力が変に強いと、異様に多い入力項目＆異様に細分化された選択肢という地獄が起きる。

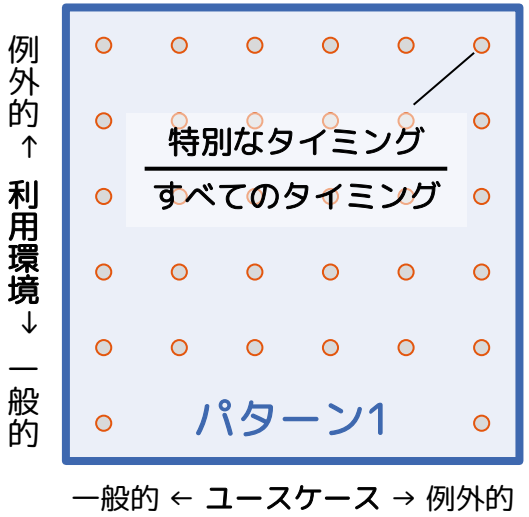
➡ やりもしない分析のための、使いもしない項目はなくそう。
入力項目は必要最低限に、できるだけ入力補助を。
何より、価値のある分析を提供しよう！

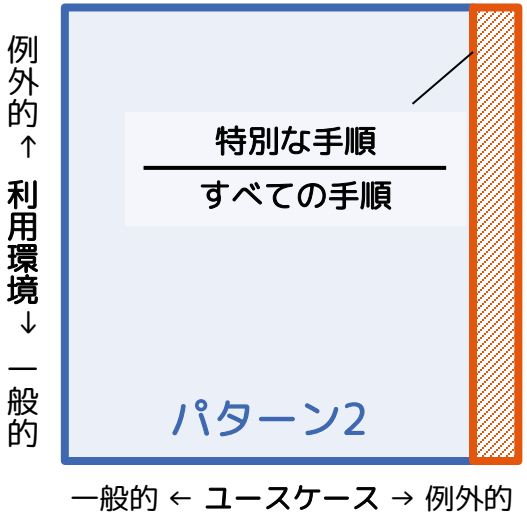
(*1) 2010年代半ばには「バグ票ワーストプラクティス改善プロジェクト」という活動があったが、そのサイトの霊圧は消えていました‥。

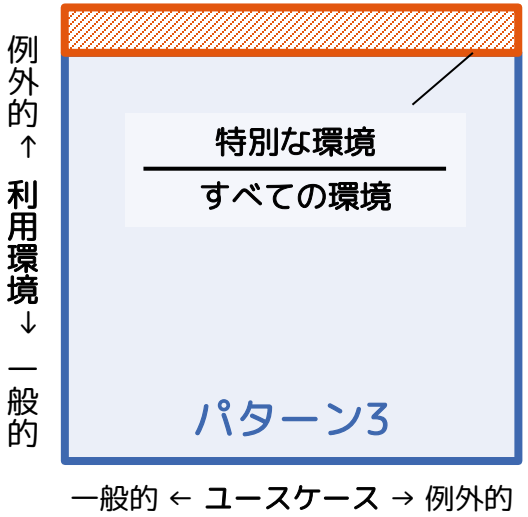
バグチケット運用バトル

その2: 選択肢の定義でモメる

- 「全部、優先度"高"じゃん」「全部、重要度"中"じゃん」
- 「発生頻度」の意味が人によって違う問題(*1)
  1. ロジック的に発生頻度が低い: 内部の特別な組み合わせや、レアなタイミングでのみ起こる
  2. ユースケース的に発生頻度が低い: 実行することが非常にまれな手順でのみ起こる
  3. 利用環境的に発生頻度が低い:ソフトウェアを利用する環境が特別な条件の時にのみ起こる



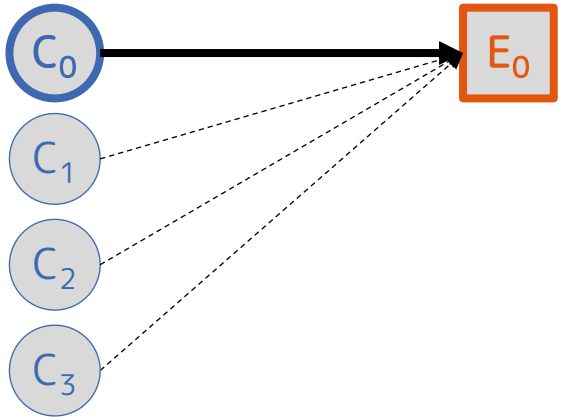




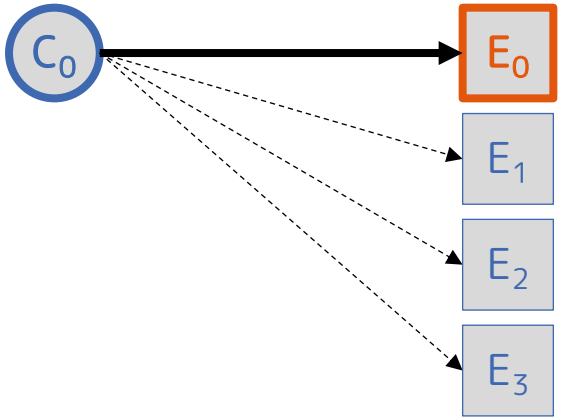
(*1) 詳しくは、拙ブログ記事・インシデントの「発生頻度」という言葉について をご覧ください。

ご参考: バグの原因と結果は1:1じゃない

- 「原因C_0により結果E_0が起きる」は、「原因C_0以外で結果E_0は起きない」を意味しない。
  - テストで行った手順が稀なものだとしても、もっと簡単な手順で同じ結果を起こせるかもしれない。
    よって「発生頻度: 低」とは限らない。
  - 発生頻度を判断するには、バグの発生条件を明確にする必要がある。



- 「原因C_0により結果E_0が起きる」は、「原因C_0で結果E_0以外は起きない」を意味しない。
  - テストで出たエラーが軽微なものだとしても、その原因はもっと重い現象を引き起こすかもしれない。
    よって「重要度: 低」とは限らない。
  - 重要度を判断するには、バグが起こしうる現象を想像する必要がある。



バグチケット運用バトル

- その3: ワークフローが複雑すぎて理解不能
  - 調査中 → 原因判明 → 対策検討中 → 対策レビュー中 → 対策内容承認済み → 対策中 → 修正中 → テスト中 → ‥ → 管理者承認待ち → ‥
  - 複雑なワークフローは、プロセスの不要な重さ、不適切な体制の臭い

  ➡ ワークフローはできるだけシンプルに、不要な人の関与はなくす。

バグの情報をきちんと入力してもらうためにも、
シンプル＆わかりやすいチケット設計にしよう！

何のためにバグ分析をするのか？

1. **プロダクトの品質を知り、リリース可否や品質改善要否を判断する**
   - 入力値の組み合わせに起因するバグが目立つ
     → 組み合わせテストのカバレッジの十分性を確認すべき？
   - 開発者が想定していない状態遷移でバグが出た
     → 設計時点で状態遷移図・表でのレビューができているか？
2. **プロセスの課題を特定し、プロセス改善につなげる**
   - バグ票の生存時間(*1)が長すぎる
     → 開発とテストの分断、不適切なトリアージなどがないか？
   - プログラミングでなく、仕様に起因するバグがたくさん出ている
     → 仕様を十分にレビューする仕掛けがないのではないか？
3. **バグのシャワーを浴びて、バグについての感覚を養う**

バグ分析は地味だが、きちんとやれば開発を良くできるスキル！

(*1) バグ票がオープンしてからクローズするまでの期間を指す。

何のためにバグ分析をするのか？

バグ分析、2つのジレンマ

1. **バグが多い時ほど分析の必要性が増す**
   ⇔ バグが多い時ほど分析は大変である。

   ➡ 量が多いなら、機械にやってもらえばいいじゃない。

2. **文章で書かれた部分には、分析コードよりも価値のある情報がある**
   ⇔ 文章で書かれた部分は人間でないと理解できず、分析が大変。

   ➡ 文章で書かれているなら、生成AIに読んでもらえばいいじゃない。

生成AIを使った横断的バグ分析

生成AIによるバグ分析、第一印象としては「いけそう」。ただし以下が重要。

- プロンプトの洗練
- 与える情報（今回の場合はバグ票）の充実
- 人間のプロによる「答え合わせ」

} 生成AIを使ううえで、全部当たり前のやつ

AIとソフトウェア開発

AIと品質保証の話をする前に‥これを切り分けておかねば。

- 横軸: AIを製品開発に用いるのか、AIを組み込んだ製品を開発するのか。
- 縦軸: ソフトウェア開発全体か、その中の品質保証か。

利用方法 / プロセス	AIを作業に用いる (AI4)	AIをプロダクトに組み込む (4AI)
ソフトウェア開発 (SE)	**AI4SE** ソフトウェアエンジニアリングのためのAI	**SE4AI** AIのためのソフトウェアエンジニアリング
品質保証 (QA)	**AI4QA** 品質保証のためのAI ★バグ分析の話はココ	**QA4AI** AIのための品質保証 

QAエンジニアも、各象限に関わりを持つことになる。

- **AI4SE**: Copilotや生成AI組込みツールでの開発など。とくさんが意見交換会をやってくれる！
- **SE4AI**: AIを組み込んだプロダクトを開発する。QA4AIとセットで考える。
- **AI4QA**: AIを品質保証に使う。バグ分析での生成AI利用はその例。
- **QA4AI**: AIの品質保証を行う。AIの特性にあったテスト技法の適用がその例。

そのバグはどのテストで起きた？
そのテストはどの仕様がベース？

そんな情報を管理してくれるのが、
テスト管理ツール！

想定経過時間 25:00

テスト管理・計画

なぜ鈴木はいつまでも テスト管理ツールの話を しているのか？

20230315_JaSST24Tokyo_テスト管理ツールの向かう先

kzsuzuki 1 200

JaSST'23 Tokyo / テスト管理ツール、利用者目線での課...

kzsuzuki 1 2.3k

テスト管理ツールとは

JSTQBの定義: テストマネジメントを支援するツール。 そのままかよ‥

- 現在のテスト管理ツールの多くは、「テスト管理」のすべてをカバーしてくれるわけではない。強いのは以下の分野。
  - **テストケース管理**
    - テストケースの一元管理、グルーピング、階層化
    - テストケースと仕様の関連付け
    - テスト自動化のステータス管理
  - **テスト実行管理**
    - ある期間（たとえばスプリント）に行うべきテストケースの選定
    - テストケース実行の順序付け
    - 各テストケースへの担当者アサイン
    - 計画されたテストケース実行の状況モニタリング
    - 実行が失敗した場合における、バグ票との関連付け

ご参考: Excel管理(*1)の比較

JaSST'23 Tokyoの資料を
一部修正、再掲

■ テスト管理ツールも万能ではないが、Excel管理の手間を省いてくれる。

分類	従来のExcel運用	一般的なテスト管理ツール
テスト計画・実装	・先にテスト期間があって、そこにテストケースを作成したり、過去のテストケースをコピーしてくるという発想。	・先にテストケース群があって、テスト期間に対して必要なテストケースを割り当てるという発想。 ・テストケースが、実行時間・リスク・関連づく欠陥などの情報を持っており、計画に役立てることができる。
テスト分析・設計	・仕様とテストケースのトレーサビリティは弱い。 ・図・表形式で設計したテストケースを、そのままテストケースとして使うことができる。ただし設計ごとに表現方法が異なるため、集計は困難になる。	・チケット管理システムとの連動で、仕様とのトレーサビリティを確保できる。 ・テスト設計のためのモデリングツールはあるが、テスト管理ツールとは連動させられない。モデルを変更してテストケースが変わっても、テスト管理ツールには反映できない。
テストケース管理	・テストケースが詳細も含めて一覧化されるので、全体感がわかりやすい。 ・一括置換・コピペ・行移動・フィルタ・ソートなど編集が容易。 ・過去のテストケースをコピペして寄せ集めにしがち。	・テストケースごとに1枚の画面があるため、全体感がわかりづらい(*2)。 ・複数のテストケースを一括で修正するのは難しい。 ・最新のテストケースを一元管理できる。
テスト実行	・「テストケースに対して実行は1回」という暗黙の前提があるため、複数回実行した場合の結果を適切に管理しづらい。	・1つのテストケースに対し、テスト実行が複数回あることが前提のデータモデル。 ・どのテストケース実行がどういうステータスにあるかが把握しやすい。
欠陥管理	・テスト実行に付随する情報として持たせることはできる。ただし、1つの実行に対して複数のバグ・・・といった状況に弱い。	・チケット管理システムとの連動で、バグとのトレーサビリティ管理ができる。 ・あるインシデントが、どのくらいのテストケース実行のブロックになっているか、といった情報が把握しやすい。
テストのモニタリング	・進捗やグラフは自分で作りこむ必要がある。Excel職人がいれば、必要なグラフを柔軟に作り込むことができるが、メンテが難しい。	・ビルトインで用意されたグラフや表はリアルタイムに集計され、すぐに使うことができる。
その他	ー	・専用ツールならではの機能がある。テストシナリオのパラメタライズや、テストケースと環境条件の組み合わせ管理など。

(*1) ここでいう「Excel管理」とは、テストケース自体から実行までが強力に結合した、モノリシックExcelシートを意味します。
より詳細な比較はこちらの記事を参照してください。テストケース管理ツールとスプレッドシートでは、テスト管理の何が違うのか
(*2) スプレッドシートのような2次元表形式でテストケースを管理するツールもある。

なぜ、ツールに言及し続けているのか

JaSST'24 Tokyoの資料を
一部修正、再掲

ソースコード管理ツールやチケット管理ツールは、
開発のプラクティスを根本的に変えている。

テスト管理ツールは今後、開発の何を変えるのか？

鈴木の夢

- テスト管理ツールが、ソースコード管理ツールやチケット管理ツールと連携すれば、
  仕様・ソースコード・テストケース・バグが関連付けられる。
- これをAIが学習すれば、「ある機能開発において、どのようなテストを追加し、
  既存のどのテストケースを実行すべきか」を提案することができるはず。
- つまりテスト管理ツールは、「過去に行ったことの記録」のインフラではなく、
  「未来に行うべきこと」を考えるためのインフラになっていけるのでは？

テスト管理ツールの進化も、注目すべき分野と思います。

「本当の」テスト管理

JSTQBでの定義

> テスト活動の計画、スケジューリング、見積り、モニタリング、レポーティング、コントロール、完了のプロセス。

- こうやってみると、テスト管理ツールでけっこうカバーできている。
- ただ、視点がミクロにも思える。
  - 計画:「このスプリントで、どのテストを実行対象にするか」は計画できる。
    ➡ 次のリリースまでに、どういうテストレベルを定義し、どのレベルにどのような範囲の品質を確保させるかといった、「高位のテスト計画」が埒外。
  - スケジューリング:「誰がいつどのテストを行うか」は計画できる。
    ➡ 他チーム・他組織のスケジュールや、環境・デバイスの制約などを踏まえた、「高位のテストスケジューリング」は対象外。
- そもそも、品質確保のための活動は、テストだけではない‥よね。
  狭義のテストだけでなく、レビューや静的解析ツールの適用なども含めた、「プロダクトの品質戦略全体」を扱う方法論(*1)と、そのツールがあったらいいな！

(*1) にしさんが、「アシュアランスパイプライン」と呼んでいたものが、それなのかもしれない。

必要なテストを選び、
実行計画を立てるというのは、
「テスト実装」の一部です。

テスト実装に行ってみましょう！

想定経過時間 29:00

テスト実装

テスト実装はなぜ
地味な響きを隠し切れないのか？

テスト実装とは

テスト設計とテスト実行に挟まれて、一見地味な存在だが‥

- **テスト実装の主な活動(*1)**
  - テスト手順(*2)を開発して優先度を割り当てる。
  - 自動化テストスクリプトを作成する。
  - 効率的にテスト実行ができるように、テスト実行スケジュール内でテストスイートを調整する。
  - テスト環境を構築する。
  - テストデータを準備し、テスト環境に適切に読み込ませてあることを確認する。
- **要するに‥**
  - テストの順序を決めたり、テスト環境を準備したり、テストを自動化したり、「テスト設計で準備したテストケースを実行するために必要なこと」を行う。
  - テストの効果・効率を高めるために重要な存在！

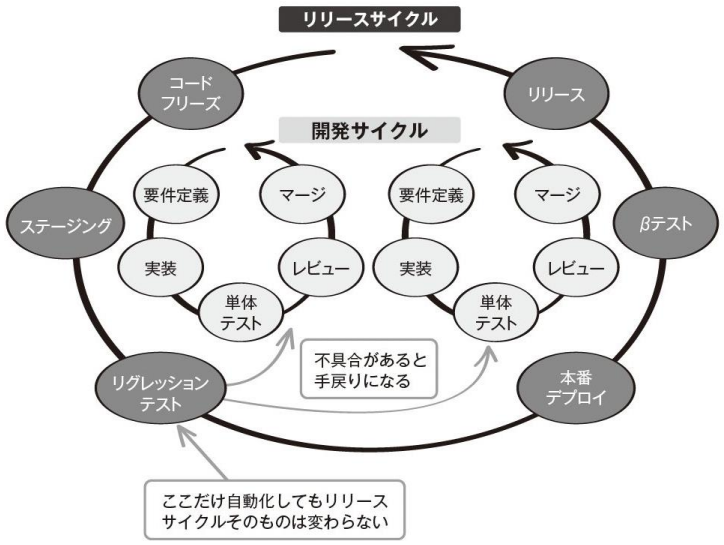
(*1) JSTQB Foundation Levelシラバスから抜粋。
(*2) ISTQBにおける「テスト手順」は、テストケースの実施手順ではなく、テストケース群の実行順番のことなので注意。

自動テストを作る

7月に発売された『テスト自動化実践ガイド』の第1部を読もう！
‥だとあんまりなので、本書の好きなところを紹介。

なぜ、手動テストを自動化するだけではダメなのか？

- 手動のE2Eテストがアジャイル開発のボトルネックになるのは、「実行時間が長いから」だけではなく、「最後にまとめてやるから」でもある。
- 自動化だけでは後者は解決できず、その結果「素朴なテスト自動化」に陥る。
手動テストのメリットを失ったのに、自動テストのメリットも享受できない。



項目	手動テスト	素朴な テスト自動化	開発を支える 自動テスト
実行サイクル	リリースサイクル	リリースサイクル	開発サイクル
頻度	少	少	多
実行手順	柔軟	厳密	厳密
検証方法	発見的	保証的	保証的

実装技術以前のあるべき姿を説明した良書！

(*1) 著者の末村さんご本人から図を提供いただきました。ありがとうございます。

テスト手順を作る

テスト所要時間についての真に驚くべき定理

プロダクトが成長していくとリグレッションテストが増加。毎回全数実行は困難になる。

- **テスト所要時間T = テストケース数n × 平均テスト実行時間e**
  - **テストケース数nを下げる**: テスト設計見直し、バグ検出に有効なテストケース選定、‥
  - **実行時間eを下げる**: 自動化、並列度を上げる、実行順序を最適化する、‥

テストケース選定・順序最適化（Test Case Prioritization）

- **リスク分析に基づく優先度付け**: テスト管理ツールを活用していれば、
  テストケースの各種属性や、機能とのトレーサビリティをベースに検討ができる。
- **アルゴリズムによる最適化**
  - プロダクトコードやテスト実行の各種属性から、バグを検出しそうなテストケースを選定する、
    Predictive Test Selection(*1)という技術が広まってきている。
  - 「忍者式テスト」(*2)では、効果の高いテストケースを抽出するアルゴリズムを使い、
    過去のテストを毎日少しずつこなすというアプローチを取っているという。

地味に思えるテスト実装も、迅速な開発サイクルに必須の技術。

(*1) 以下の記事がわかりやすい。ソフトウェアテストの実行を効率化するPredictive Test Selectionの衝撃。Launchableでは、プログラム変更時に実行するテストセットを、機械学習によって最適化しているとのこと。参考: 機械学習でテスト実行を効率化するLaunchable

(*2) 忍者式テストについては、以下の記事を参照。忍者式テストの秘密公開！驚きの20年の実績が明かされる！

これまで語ってきたテストケースは、
そもそもどこから出てくるのか？

根拠のある、納得感のある、価値ある
テストケース、それを導出するのが
テスト分析・テスト設計です。

想定経過時間 35:00

テスト分析・設計

テストのための方法論は、
どのように変化してきているのか？

テスト分析・設計とは

ざっくり言うと‥

- プロダクトに関する情報を調査したうえで、プロダクトのどこをチェックするかという観点を特定するのが、**テスト分析**。
- テスト分析で特定したそれぞれの観点を効果的・効率的にチェックするにはどうするかを決めるのが、**テスト設計**。
  - そのための技術が、テスト設計技法。

2000年代後半に学ばれていたテスト技法

- 2000年代後半、ソフトウェアテストの書籍が一般的になり、「テストクラスタ」の人たちが勉強会(*1)をするようになった。
- 当時よく学ばれていた代表的な技法は、以下(*2)。

ブラックボックステスト(*2)	ホワイトボックステスト	
• 同値分割 • 組み合わせテスト		テストケースを合理的に少なくする
• 境界値分析 • 探索的テスト		多くの欠陥が見つかるようにする
• デシジョンテーブル • 状態遷移テスト • シナリオテスト	• 制御フローテスト • データフローテスト	テスト対象を漏れなくテストする

(*1) 当時は企業主体のイベント・勉強会自体が少なく、場所の確保からして大変だったのです。もちろん配信などありません。
(*2) 主観です。赤枠の分類は、『ASTERセミナー標準テキスト』を参考にしています。
(*3) ブラックボックステストは、「コードを意識しない」というより、「コードよりも仕様をベースに設計したテスト」と考えられる。「グレーボックステスト」という言葉もある。

2000年代後半に学ばれていたテスト技法

- **同値分割**
  - 広大な「パラメータ」(*1)の空間を分割し、それぞれの領域を少数の値で代表させる。
  - 他のテスト技法のベースとなる技術である。
- **境界値分析**
  - バグを作り込みやすい、「端っこ」を特定する。
- **デシジョンテーブル**
  - 複数のパラメータの組み合わせに依存するふるまいを網羅的に検証する。
- **状態遷移テスト**
  - トリガーを契機に状態が変化していくソフトウェアのふるまいを検証する。
- **組み合わせテスト**
  - 複数のパラメータの組み合わせには依存しないはずのふるまいを、少ない組み合わせで検証する。
- **シナリオテスト（後述）**

(*1) ここでは「パラメータ」は広義のもの。たとえば「ユーザのペルソナ」のようなものもパラメータになりうる。

ご参考: テスト設計パターン(仮)を考える会

- テスト設計技法を習った瞬間
  - 「よーし、仕様書カモーン！ すべての仕様をテスト設計技法で捌くぜ！」
    →（数時間後）「テスト設計技法が当てはまる箇所なくない？」
- なぜこうなるのか？
  1. 練習問題と違って、仕様とテスト設計技法とは1:1でキレイに対応していない。
     仕様を解きほぐしていくことで、ある側面は技法A、別の側面は技法Bと対応づいたりする。
  2. 「よくある仕様に対する、よくあるテストのパターン」みたいなものがあるが、特別なテスト設計技法として明文化されていない。みんな各々の経験知で都度、テストケースを作っている。
     → これを「テスト設計パターン」みたいに整理できないものかね？
- 「考える会」をやってみませんか？
  - 抽象化・単純化された「ありがちな仕様」をお題として、テスト設計パターンを考えよう！
  - ものすごくミニマム化されたテスト設計コンテストみたいな感じです（テスコンに失礼）。

なお1個目の理由の方は、本日のワークショップ
「技法を探せ！」でカバーできそう！

テスト設計の広がり

テスト設計技法もいろんな方向に発展している。

1. 従来の技法の親戚的な技法
2. アジャイル開発とともに普及の進んだ技法
3. 機械学習・AIとともに普及の進んだ技法
4. 非機能要件のテスト、ドメイン固有のテスト

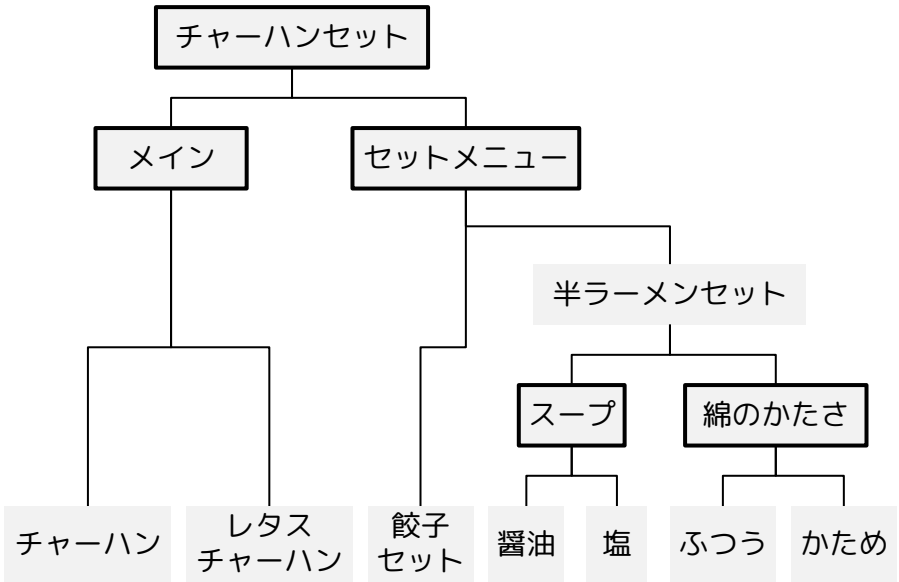
従来の技法の親戚的な技法

値の組み合わせに関するテスト設計技法に、新しい(?)仲間が！

■ クラシフィケーションツリー

- 組み合わせテストをどう作ればいいか分からない時に、テスト対象をクラシフィケーションツリーで分類して整理し、組み合わせを検討してテストケースを作成できる(*1)。
- 値と範囲と組み合わせを扱う技法を整理した(*2)のが以下。
  - ツリーを使って、パラメタや値の整理、分割を検討できる。

目的	単一パラ	複数パラ
パラメータの範囲を分割する	同値分割法	同値分割法 クラシフィケーションツリー
パラメータの境界の動作を評価する	境界値分析	ドメイン分析テスト
パラメータの値の組み合わせ論理（有則）を評価する	n/a	デシジョンテーブル 原因結果グラフ 原因流れ図
パラメータの値の組み合わせ論理（無則）を評価する	n/a	ペアワイズテスト 直交表テスト・HAYST法 クラシフィケーションツリー



(*1) JaSST'24東北のワークショップ資料より引用。右下のツリー図も同様。
(*2) 詳しくはこちらの記事を参照のこと。ドメイン分析テストとデシジョンテーブルの違い

アジャイル開発とともに普及の進んだ技法

以下の2つはアジャイルと親和性の高い技法として扱われている。

探索的テスト

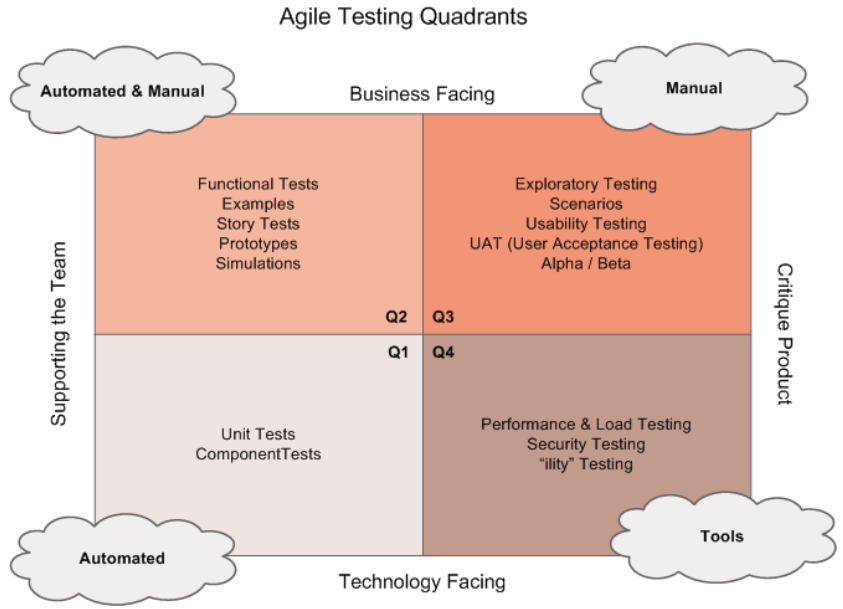
- テストアプローチのひとつ。テスト担当者がテストアイテムや以前のテストの結果の知識や調査情報を使用して、テストを動的に設計、および実行する。(*1)

シナリオテスト

- ブラックボックステスト技法の一つ。ユースケースの動作を実行するようにテストケースを設計する。

この2つに共通するのは、

- アジャイルテスト4象限(*2)における右上、ビジネス面×製品を批評するゾーンにある。
- テストをできるだけ自動化したうえで、人間の経験や直感を活かし、価値を確認する手動テストを重要視。



(*1) ISTQB Glossaryより引用。シナリオテストについては、同義語とされるユースケーステストの定義を引いている。
(*2) Lisa Crispin氏の『Using the Agile Testing Quadrants』より引用。

アジャイル開発とともに普及の進んだ技法

探索的テストとスクリプトテスト(*1)の比較(*2)

以下の赤字部分が、アジャイル開発における人気の理由？

項目	スクリプトテスト(*1)	探索的テスト	備考
アプローチ	網羅	ピンポイント	ー
バグの検出効率	中	高(*3)	スクリプトテストで「保証」を目指す場合、テストケースに対してバグ検出効率は相対的に低くなる。
説明可能性	高	低	テスト設計とその意図が残っていれば、説明可能性が高い。「いきなりテスト」型だと低い。
属人性	中	高	探索的テストの効果は、担当者の実力に強く依存する。実施すべきことが決まっているスクリプトテストでは属人性がそこまで高くないが、その前段階のテスト分析・設計はやはり人依存性が高くなる。
ドキュメンテーションコスト	高	低	スクリプトテストでは、テスト分析・設計・実行結果の記録のコストが高い。探索的テストはコストが低いが、その分「なぜこのテストが行われたか」を追跡するのが難しい。

(*1) Scripted Test。やるべきことが明文化されているという意味。Pre-designed Testとか言った方がかっこいい。
(*2) かなり感覚で書いていますので、鵜呑みにしないでください。
(*3) 『知識ゼロから学ぶソフトウェアテスト 第3版』で、探索的テストの効率についての言及がある。

アジャイル開発とともに普及の進んだ技法

『実践アジャイルテスト』におけるシナリオテスト

> 製品を批評するビジネスのテストをするときは、実際のユーザーがアプリケーションを使うように試行します。これは人手だけでできる手動テストです。(略)自分の感覚と頭、直感を使って、開発チームは顧客が求めるビジネス価値を提供しているか調べなければいけません。

- ストーリーテストとシナリオテスト(*1)は、別の象限で扱われている。
  - ストーリーテストは第2象限（ビジネス面×チームを支援する）で、機能テストに近い。

シナリオテスト設計の3類型(*2)

❶コネクション型	❷シチュエーション型	❸バリエーション型
複数のユースケース、ストーリーを結合して、業務に相当するシナリオを作る。この際、プロダクトではカバーできない「隙間」を見つけやすい。	プロダクトありきでなく、実際に起こりうる業務シナリオを想定し、その中でプロダクトをどう使うかを考える。5W1Hで、特に極端なケースも検討する。	ユースケースの中で、「操作ミスしたので少し戻る」「電話がきたので操作からいったん離れる」といった変化を付加することで、リアルな使用を模擬する。

探索的テスト・シナリオテストはより整理されていく分野のはず。

(*1)ユースケーステストというのもあるが、それぞれ人によって考え方が違う。
(*2) 鈴木による勝手な分類。これら3つのアプローチを生成AIにやらせたい。

機械学習・AIとともに普及の進んだ技法 (QA4AI)

機械学習におけるテスト設計の課題

1. 同値分割が困難である → 少ないテストケースで多くの範囲をカバーできない
2. テストオラクル(*1)に乏しい → テストを実行しても、適切な結果なのか判定が難しい

(*1) ISO/IEC TR 5469（人工知能－機能安全とAIシステム）では、「Non-existence of an a priori specification」（先験的仕様の欠如）と呼んでいるようだ。

機械学習・AIとともに普及の進んだ技法 (QA4AI)

以下は、機械学習のソフトウェアに適用可能なテストとして知られている。

メタモルフィックテスト

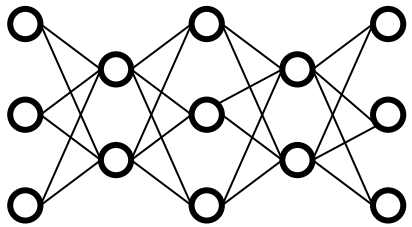
- 正しいとわかっている元テストケースから、別のテストケースをたくさん生成する。
  - 画像判定において、ある画像が「猿」という文字だとわかっていれば、その画像を90度回転させても猿である。
- 同値分割を適用しづらいという課題は、大量のテストケースの暴力で解決。
  元のテストケースの正しさをベースにして、テストオラクル(*1)問題を解決。

探索的テスト

- 仕様には必ずしも明示されない「人間の知識」で、テストオラクル問題を解決。
  - 生成AIによって生成された要約が妥当かどうかは、人間が判断できる（可能性がある）。

ニューロンカバレッジテスト

- 制御フローテストのアナロジーで、ニューラルネットワーク内の
  ニューロンをできるだけ活性化させるようにする(*1)。



29119の他、日本語で読める書籍(*2)もあり、実用が進んでいる。

(*1) 『知識ゼロから学ぶソフトウェアテスト 第3版』によると、「コードカバレッジほど指標としての強さはないが、指標の1つとして使えると思われる」。
(*2) 『AIソフトウェアのテスト』（佐藤 直人, 小川 秀人他・著）など。

生成AIの何をテストする？ (QA4AI)

テキスト系生成AIの品質特性

右は『QA4AIガイドライン』(*1)からの抜粋。

- 何をどう評価すればいいかわからんもの多数。
  - 創造性？ 多様性？ ？？？
  - 「事実性」と「創造性」って矛盾してない‥？
- ガイドラインでも、紹介されたベンチマークや評価手法を、その時点でのスナップショットとして紹介するにとどまっている。

これから研究の進む分野だが、正直想像もつかない‥

本章での用語	SQuaRE for AI
QC01：回答性能	Functional Correctness
QC01-1：自然言語処理における回答性能	
QC01-2：ツール活用に関する回答性能	
QC01-3：創造性・多様性に関する回答性能	
QC01-4：制御可能性	User Controllability
QC02：事実性・誠実性	Functional Correctness
QC02-1：一般的な知識に対する事実性・誠実性	
QC02-2：与えた知識に対する事実性・誠実性	
QC02-3：根拠の説明性・妥当性	
QC03：倫理性・アラインメント	Societal and Ethical Risk Mitigation
QC03-1：公平性	
QC03-2：安全性	
QC03-3：データガバナンス	
QC04：頑健性	Robustness
QC05：AI セキュリティ	Security

(*1) QA4AIガイドラインは、AIプロダクト品質保証コンソーシアムが発行している。ここで列挙されている品質特性は、試行的なもの。

生成AIとテスト設計 (AI4QA)

ソフトウェア開発プロセスの生成AIによる補助・代替が、各社で進んでいる。

■ テストケースの導出についての印象

- 仕様＋単純プロンプトだけでも答えは返ってくるが、一般的・大雑把な情報しか期待できない。
  ドメイン知識やプロダクトの知識を食わせない状態では、あまり役に立たない。
  - プロダクトの知識は、仕様書、マニュアル、ソースコード、テストケース、バグ票などから得られる？
- 仕様→テストケース ではなく、できれば
  仕様→テストモデル→テストケース と
  段階を踏むと、以下の利点がある。
  - 説明可能性が上がり、レビューもしやすい。
  - 生成AIの思考過程を段階的に確認できる。

■ ツールベンダーも対応中

- ベリサーブ社のGIHOZは、仕様からデシジョンテーブルの作成を行える（β版）。
- Autify社のAutify Genesisでは、仕様からテストシナリオの生成を行える。

ステップ	従来	モデリングツール	AIモデリングツール	近い未来
要件の記述	人類	人類	人類	人類？
仕様の記述	人類	人類	人類	ツール
モデルの記述	人類	人類	ツール	ツール
テストケースの導出	人類	ツール	ツール	ツール
テストの実行	人類	人類	人類	ツール

生成AIとテスト設計 (AI4QA)

仕様を適切に記述する、言語化能力があらためて求められる？

- **生成AIにうまく答えてもらうには‥**
  - 何のために答えがほしいのか、目的を明確にする。
  - 暗黙の前提をできるだけなくし、背景情報を明示的に伝える。
  - できるだけマギレのない、論理的な文章で伝える。

} 人に仕事をお願いするのと同じ、と言われたりする

- **今、あえての論理学＆ロジカルシンキング？**

記号論理学と自然言語にギャップがある。この辺の勉強も面白そう。

- 「明日雨なら、運動会は中止だ」
  → 雨が降らなかった場合には運動会が行われるとは、明には言っていない。
- 「わたしはカレーかラーメンを食べます」
  → 論理学では論理和かもしれないが、日常会話では排他的論理和である。

言語化能力を上げることで、テストのモデリングや仕様書レビューを、生成AIでブーストできそう。

さらに‥

「テスト設計」そのものではないけれど、

■ 非機能要件のテスト

定番のプロセスができたり、標準がまとまったりしている。たとえば

- **セキュリティ**: 静的解析＋動的解析＋コンポジション解析＋ペネトレーションテスト といったテストに加え、緊急度の高い脆弱性への対応プロセスの策定などがセットになっている。
- **性能**: ISTQBで「性能テスト担当者」のシラバスが発行され、性能テストの分析・設計・実行についての概念が整理された。日本語訳もあり。
  - ISTQBでは非機能テストのシラバスをたくさん出している！ 勉強会の季節かな。
- **アクセシビリティ**: JaSST'24 Niigataで、@ymrlさんによる素晴らしい発表があった！

■ ドメイン特有のテスト

ドメインが細分化され、基本的なテスト技術の上に乗る、ドメイン特化のテストタイプ・技術が発展している。

- Webアプリケーションに加えて、モバイルアプリ特有のテストも整理されてきているようだ(*1)。
- バイオメトリクスについてのテストがISO/IEC/IEEE 29119-13という形でTechnical Reportが出ている。
- ISTQBシラバスとしては、車載、ゲーム、さらにはギャンブルについてのテストまで発行されている。

(*1)『実践ソフトウェアエンジニアリング』でも、「移動体端末と特定ドメインに対するテスト」という1章が割かれている。

さらにさらに‥

- ミューテーションテスト でテストをテストする！
  - プロダクトではなく、テストの有効性を検証するためのテスト。
    プロダクトコードを故意に変異させ、その変異をテストスイートが検出できるかを見る。
  - テストの擬陰性を調べるものだが、変異の質が悪いと「failさせる必要のないの変異」を生成し、その結果ツールが「このテストの擬陰性を検出した」という擬陽性に陥る‥らしい(*1)。
- プロパティベースドテスト でテストケースを増やす！
  - テストケースの入出力について、具体的な入出力値（example）ではなく、満たされるべき性質（property）を指定することで、その性質を満たすようなテストケースを自動生成させる手法。
    - Lucy Mair氏の例(*2): 2つのリストを連結する関数 `c = concat(a, b)` を考える。
      たとえば、`concat([0, 1, 2], [3, 4]) = [0, 1, 2, 3, 4]` となる。この `concat` には、以下の性質がある。
      - `c` の長さは `a.length + b.length` となる。
      - `a` に含まれるすべての項目は `c` に含まれる。
      - `b` に含まれる項目はすべて `c` に含まれる。
      - 空のリストと連結すると、同じリストが返される `a = concat(a, []) = concat([], a)`。

テスト設計技術も進化が続いており、学んでいく必要がある。

(*1) 『Mutation Testingを活用して テスト品質を考える』参照のこと。
(*2) InfoQの記事より引用: 多くの入力パターンをテストできるProperty-based Testing

チョットダケ、テスト分析

テスト分析の必要性

- 「プロダクトの何をテストするのか」が決まってはじめて、
  「それをどうテストするか」を考えることができる。
- テストを通じてどのように品質保証するのかという全体像を描く技術。
  - 「何をテストするか」というテスト観点を洗い出し ← ゆもつよメソッドが得意そう
  - それらのテスト観点の関係を構造化し ← VSTePが得意そう
  - どのテスト観点を、どのテストレベルでカバーするのかの作戦を立てる
    ↑ VSTePが得意そう

**QAエンジニアが今後発展させ、身に付けていくべき
分野の1つがこのテスト分析ではないか？**

(*1) ベリサーブ社のGIHOZでは、VSTePによるテスト観点図の描画をサポートしている。
(*2) まこっちゃんさんがお試しで、ゆもつよメソッドのプロセスの一部をGPTsで実装している。

テストではバグ摘出までの
時間が長い。

できる限り早い段階で
バグ摘出したい！

想定経過時間 54:00

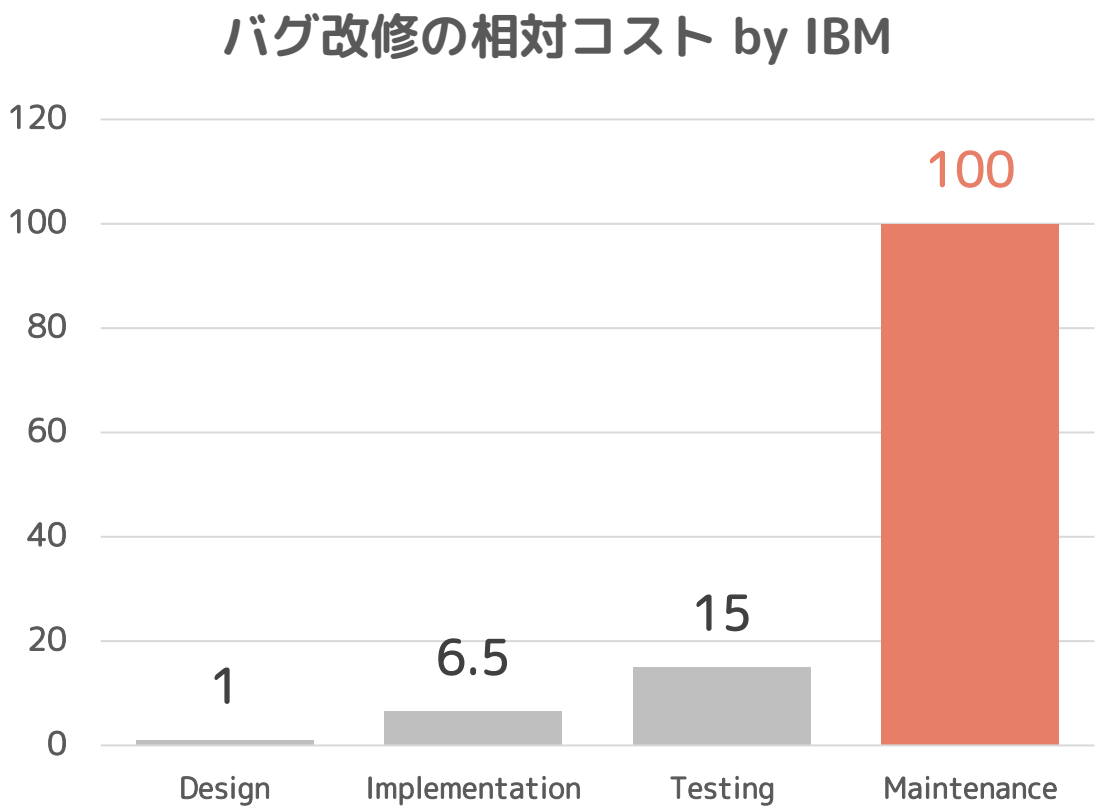
レビュー

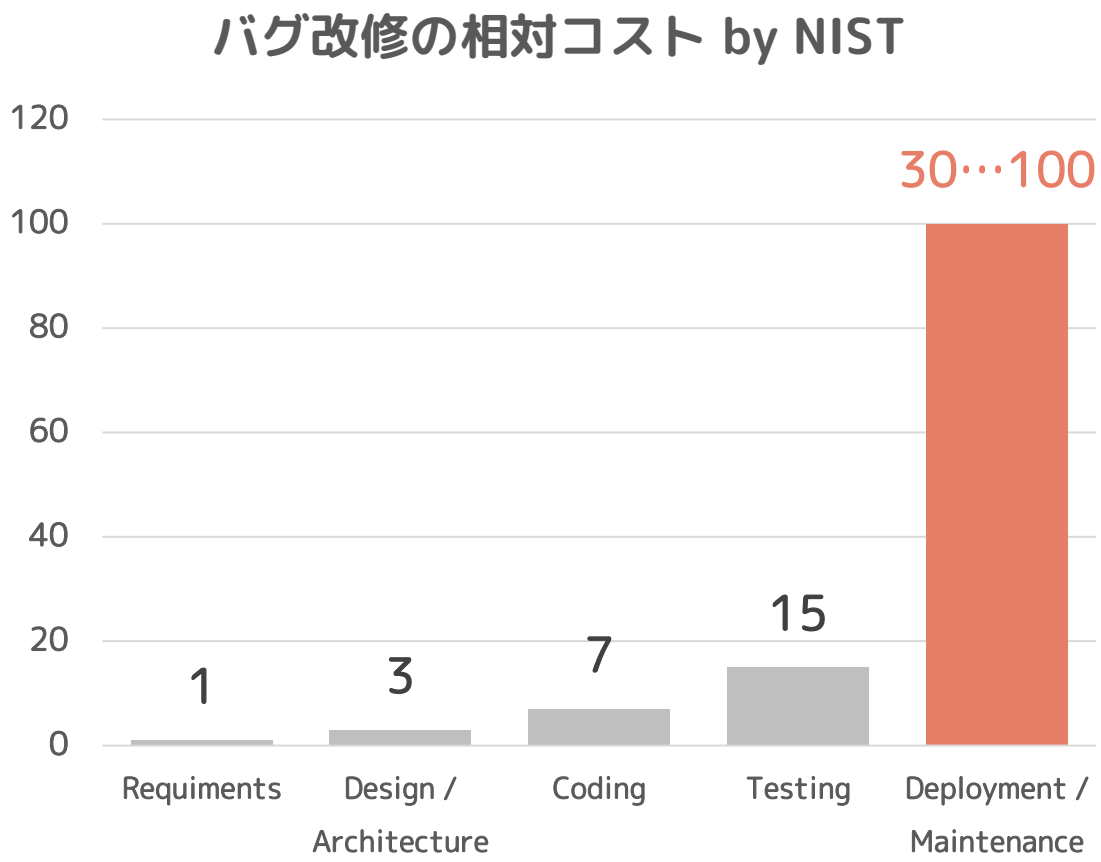
動くものを早く作って
テストした方がいい？

なぜレビューは必要なのか？

みんな大好き、あのグラフ

- バグは発見が遅れるほど、改修コストが激増していくのが本質。
  できるだけ早く、問題を見つけたい。→ シフトレフト？





(*1) IBMの2010年の論文から再構成。
(*2) NIST（National Institute of Standards and Technology）の2002年の論文から再構成。

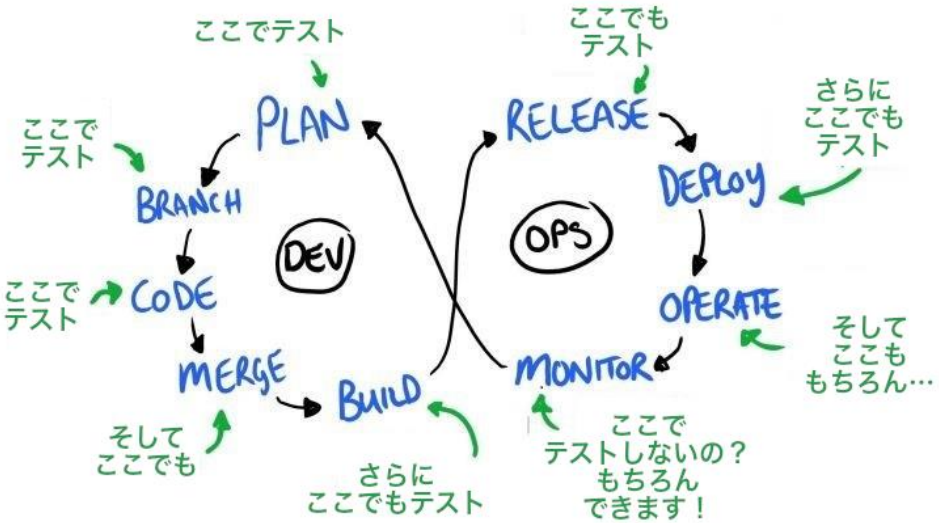
2つの「シフトレフトテスト」があるらしい(*1)

❶ テストを「最後に一発で行う巨大な塊」にしない

- テストの責務を分割し、それぞれのテストを最適なタイミングと頻度で行う。
- テスト自動化のところで「末村ダブルループ図」として言及したのもその一つだが、自動テストに限った話(*2)ではない。

❷ テスト(*3)に頼らずレビューでバグを見つける

- 「継続的テストモデル」(*4)の左上部分、「MERGE」より前段階での活動。
- シフトレフトという言葉によって、「作る前に問題を検出する」レビューの価値が、一周回って再認識されている？



レビューの技法を見ていきましょう。

(*1) もともとの意味は、❶に近いもののようです。参考: 今さら聞けないソフトウェア開発用語：「シフトレフト」は誰が言いはじめた？
(*2) 伊藤由貴氏・『シフトレフトの１歩目として"テスト"を分解＆可視化した』が参考になる。
(*3) レビューは「静的テスト」とされているので、「テストに頼らず」→「継続的テスト」という表現が矛盾していて申し訳ない。
(*4) Dan Ashby氏の記事・『Continuous Testing in DevOps…』の紹介された絵を、風間裕也氏が日本語訳したもの。

レビューの技法

- ISTQBシラバス(*1)で解説されているレビュー技法

レビュー技法	特徴
アドホック	レビューに関するタスクのガイダンスが、ほぼ提供されない。
チェックリストベース	チェックリスト（経験に基づいて導出された、起こり得る欠陥に基づく質問を列挙したもの）に基づきレビューを行う。
シナリオベース	期待する使い方のシナリオに基づいて、仮想的に動かしてみる（ドライラン）。
ロールベース	個々のステークホルダーの役割の観点から、作業成果物を評価する。
パースペクティブベース	ロールベースに加え、対象の成果物から、与えられた役割で導出する成果物を作成する。要件や技術的な作業成果物のレビューにおいて、最も効果的な技法とされる。

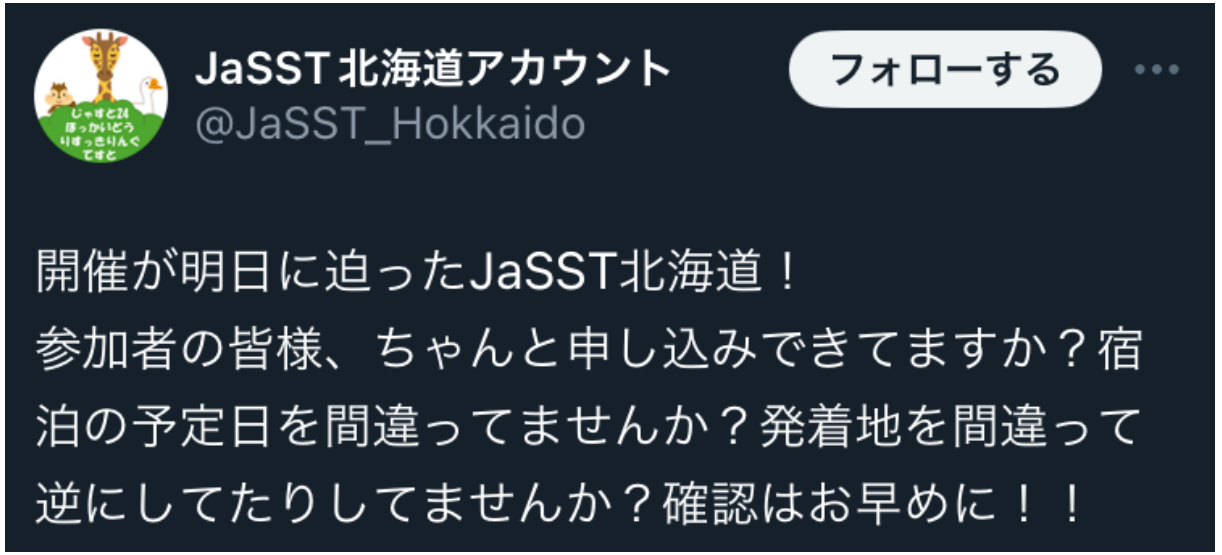
  - BDDのプラクティスとしての「実例マッピング」(*2)、具体的な状況を例にとって仕様の解像度を上げていく手法も、レビュー技法の一種と言えそう。

(*1) ISTQB Foundation Levelシラバス Ver 2018 V3.1日本語版より
(*2) 風間裕也氏の記事に詳しい。【翻訳記事+α】受け入れ基準の設定時などに役立つプラクティス「実例マッピング(Example Mapping)」

チェックリストベースのしんどさ

チェックリストの2つの志向

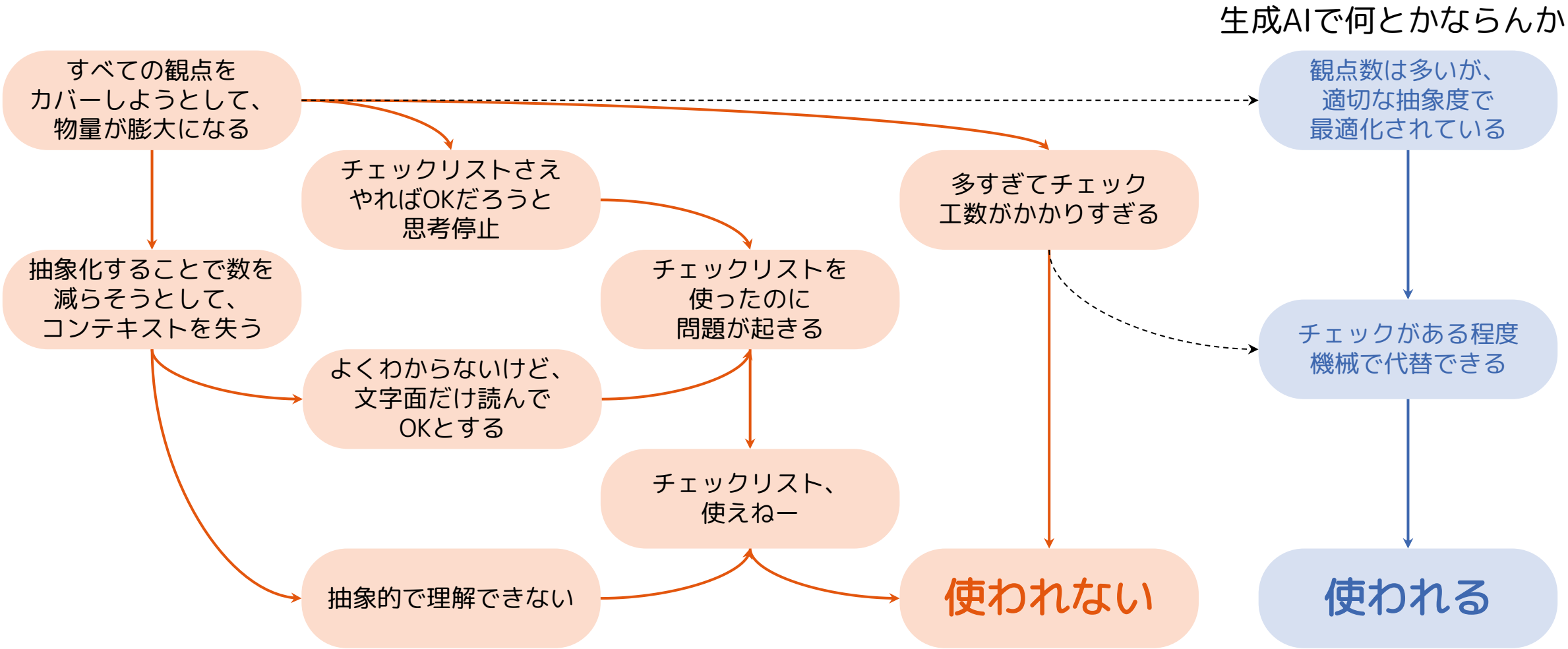
- **一撃必殺型**: 本当の急所だけを押さえた、筋肉質でソリッドなもの
  ➡ こういうやつです？



- **森羅万象型**: あらゆることをカバーしようとするもの
  ➡ ソフトウェア開発で出てきがち、嫌われがちなのはコチラ。

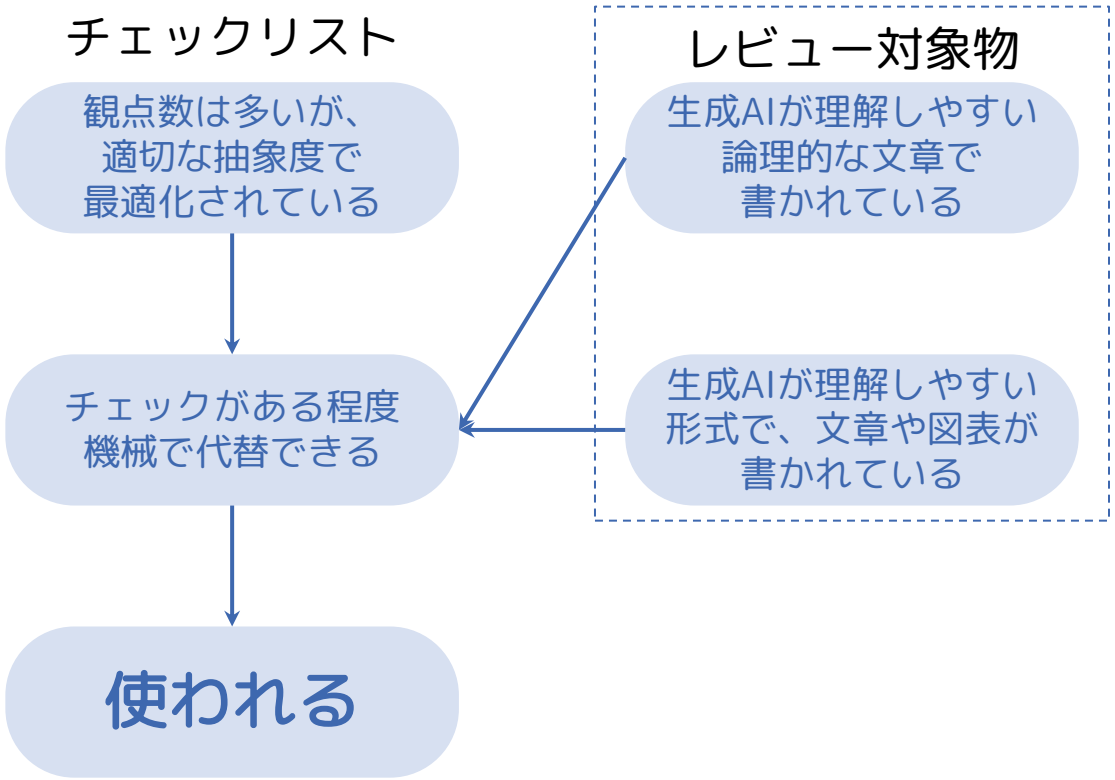
チェックリストベースのしんどさ

森羅万象型チェックリストの嫌われ方



チェックリストベースのしんどさ

チェックリストだけじゃなく、
レビュー対象物も改善が必要だろう。



1 以下のような点に配慮した記述

- 論理的、適切な接続詞の使用
- MECEを意識した記述
- 暗黙のコンテキストに頼らない
  - 生成AIに補完してもらうのは危険
- 主語や目的語の明示、主述のねじれの排除
  - 「DeepLにいい翻訳をしてもらえる日本語」
- 抽象的な記述に、適度な量の例示
- 用語の定義、曖昧さの軽減
  - 「サービス停止中は‥」

2 人と機械の両方に読みやすい形式

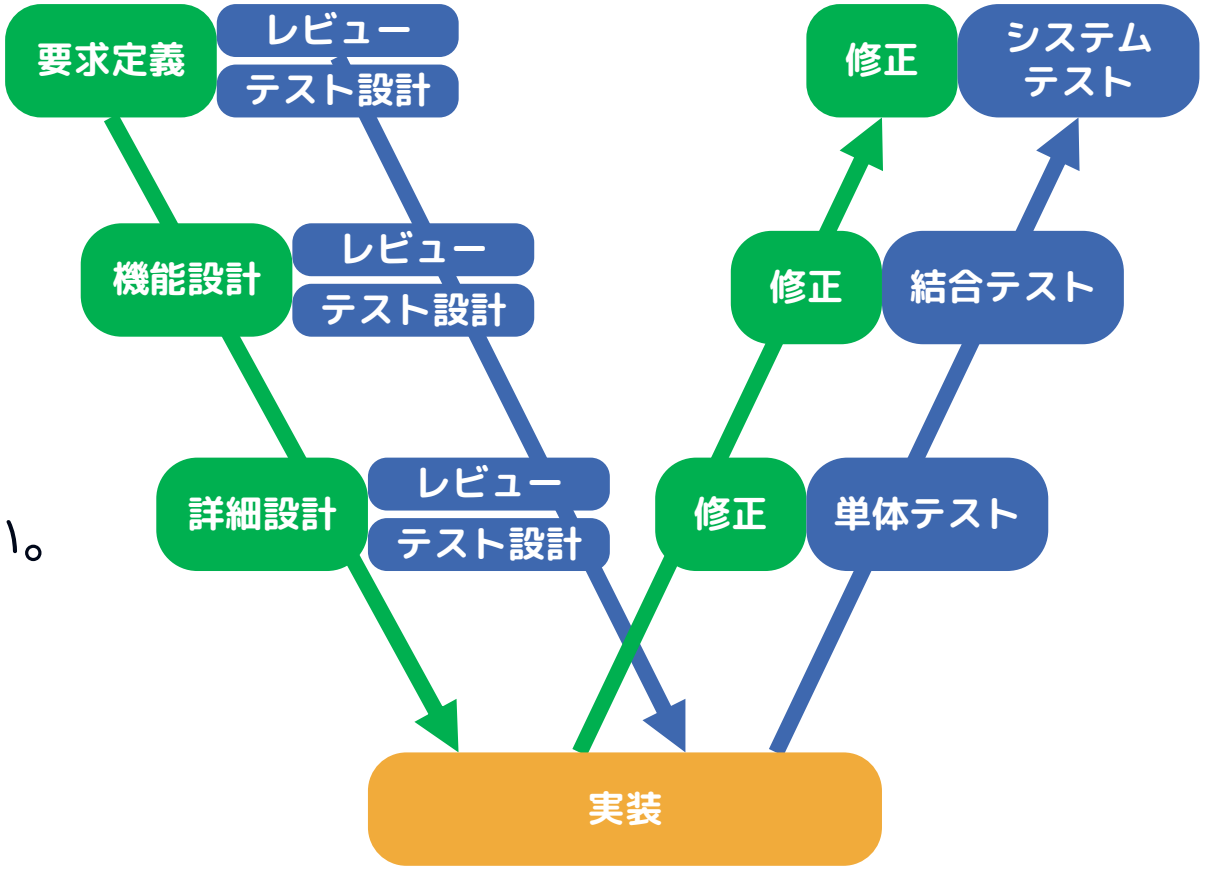
- 文章はMarkdown、図はMermaid?
  - 最近はPDFの複雑な表も認識してくれるっぽい

チェックリストベースのレビューが、AIの力で復活しそう。

パースペクティブベースドの良さ

W字モデル(*1)を知っていますか？

- 設計段階のバグをテスト工程まで持ちこさぬよう、レビューをしっかり行い、バグの早期摘出を目指す。
- パースペクティブベースドレビューの考えを踏まえると、この段階でテスト設計が行えると良い。「テスト設計ができない」＝「仕様の記述が十分でない」ことが多い。
- スクラム開発において、QAエンジニアがAcceptance Criteriaの策定に関わる(*2)というのも、その派生と考えられそう。



仕様の早期理解、設計品質向上、テスト設計の前倒しと、いいことづくめ。

(*1) W字モデルについて第三者検証会社のウェブサイトを調べたところ、意外なほど統一感がなかった。図は、VALTES社のサイトを元に、発表者が再構成したもの。
(*2) rinaさんの発表資料など参照。

シフトレフトの話が出たので‥シフトライトって何？

- 「なぁに、リリースしちまえばお客がバグ見つけてくれるよ」
  - 「品質の悪いものを出して、お客様にデバッグさせるな」という皮肉だった‥はず。
- シフトライトテスト(*1)

  リリース後も「テスト」を継続するのがシフトライトテスト。ざっくり2つに分かれる？

  1. 本番環境でないと試すことのできない、リアル＆複雑なテスト
     - **フォールトインジェクション、カオスエンジニアリング**: システムコンポーネントに故意に障害を起こし、システムの切換え・縮退・自動修復などのふるまいや、運用チームによる回復対応の妥当性を確認する。
  2. 実ユーザからのリアクションを受け取り、分析するテスト
     - **アルファテスト、ベータテスト**: 受入れテストの一種として、先行的な実ユーザなどに検証してもらう。
     - **フィーチャーフラグ、リングデプロイメント**: 機能の有効/無効の切換えを容易にしながら、限られた範囲でその機能をリリースし、問題の有無の確認やユーザの挙動の調査をする。
     - **A/Bテスト**: 多数のユーザ群に対し別々のモノをリリースし、どれがもっとも効果を発揮するかを判断する。
     - **ユーザの観察**: リリースしたものが実際に使われている現場で、ユーザビリティなどを調査する。

ユーザ価値・ゴールが曖昧で、かつ可変であることが前提。

(*1) それぞれのプラクティスについて、わたしは深い知識がなく、勉強も兼ねた整理になっています。
参考:シフト ライトで実稼働環境でテストする、シフトレフトとシフトライト

レビューもやった、
テストもやった、
品質がどうなったか知りたい！

想定経過時間 1:03:00

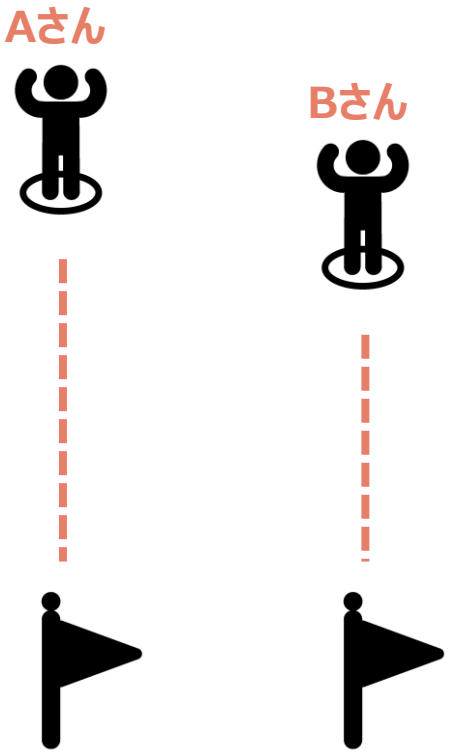
品質評価とメトリクス

プロダクトの品質は、
今どこにあるのか。

そもそも問うべきは
「どこ」だけなのか？

謎のアナロジー

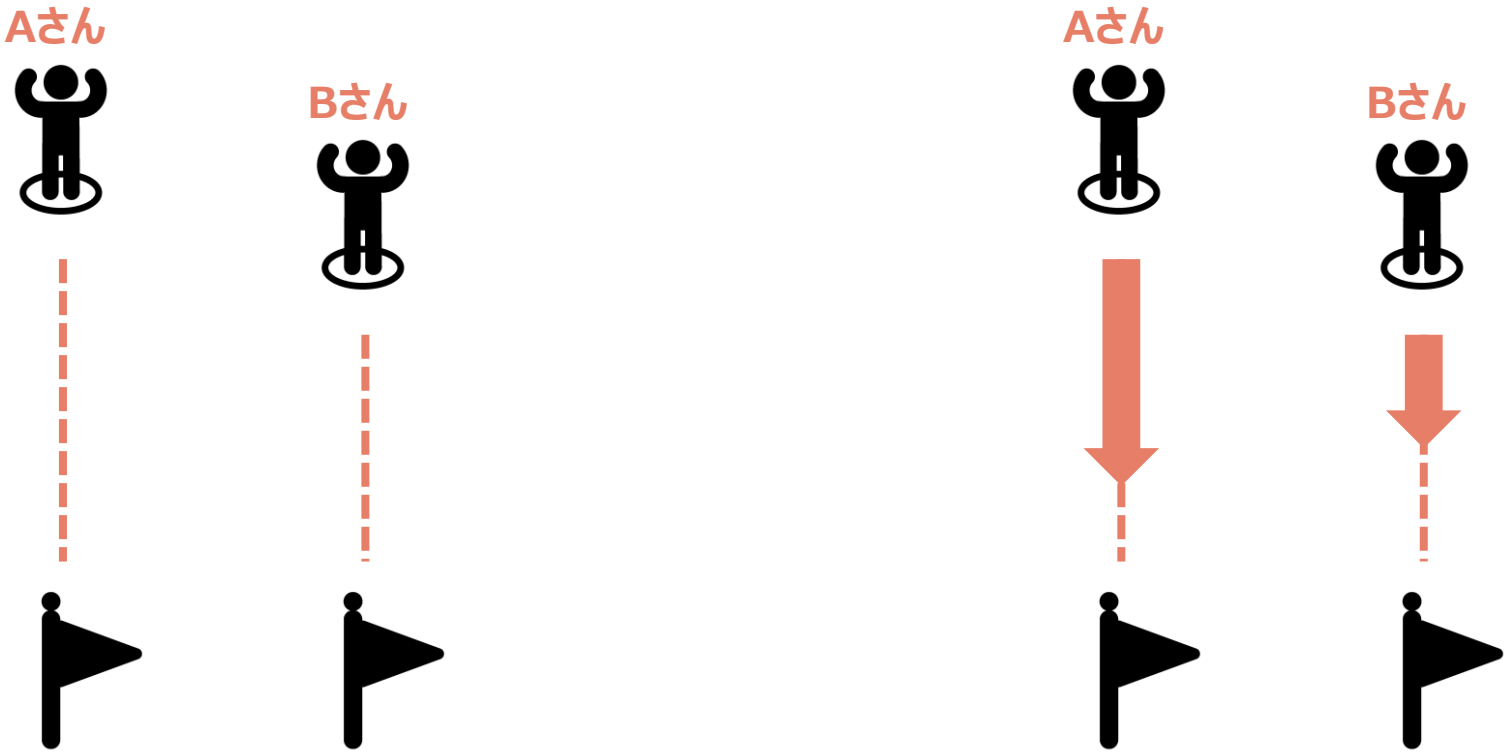
- AさんとBさん、ゴールに近いのはどちらでしょう？



位置を比べると、当然Bさんが近い。

謎のアナロジー

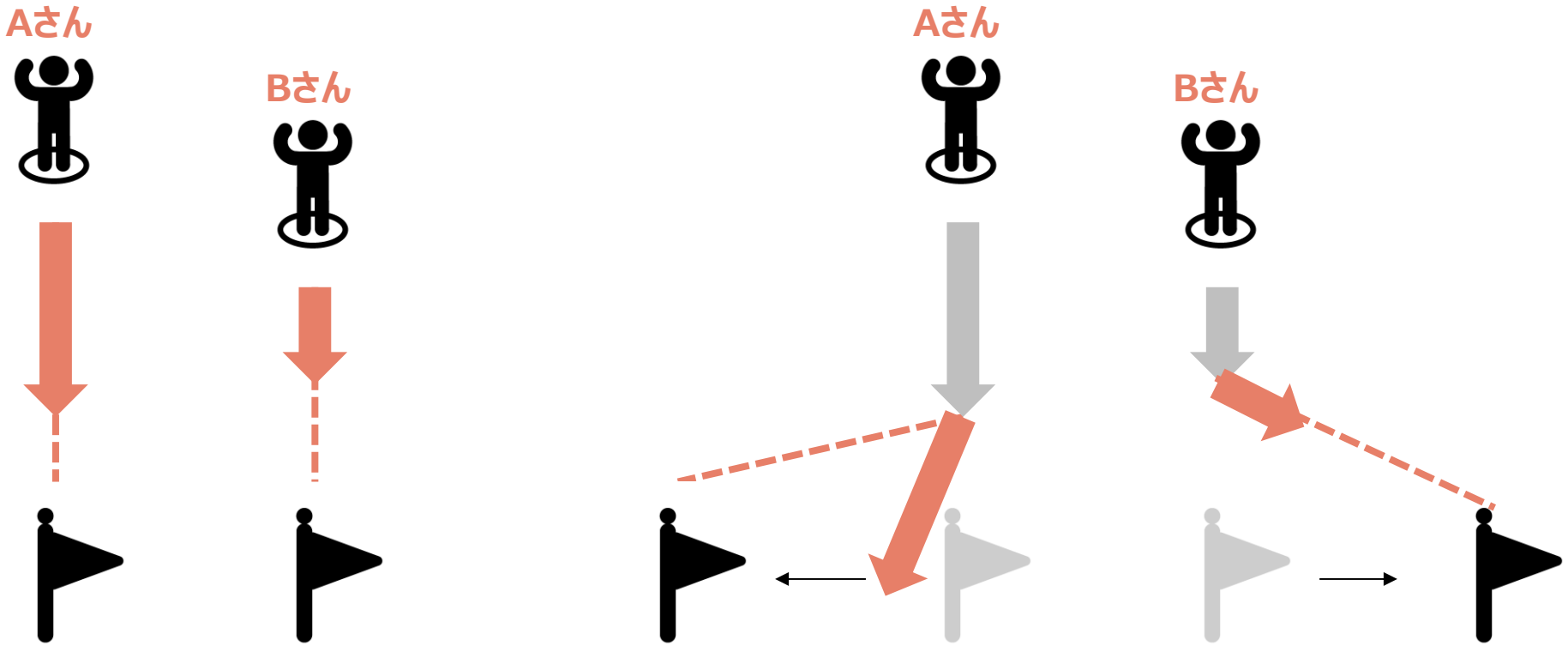
- AさんとBさん、先にゴールに着くのはどちらでしょう？



位置だけでは勝負は決まらない。
スタートが不利でも、速度で勝てるかも。

謎のアナロジー

- AさんとBさん、先にゴールに着くのはどちらでしょう？

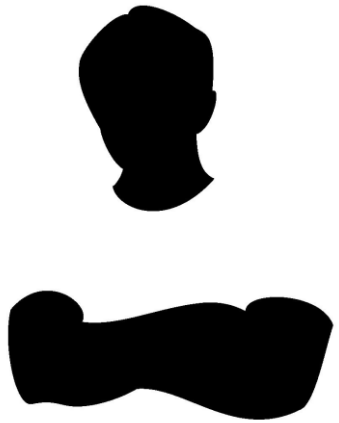


ゴールの位置が変わりうるとしたら‥？
速度に加えて、方向を合わせ直す加速度も重要である。

ゴールに早くたどり着くためのポイント

1. ゴールに近いほど有利
2. ゴールに近づく速度が速い方が有利
3. ゴールが変わっても追随できる方が有利

おまえは何を言っているんだ?



従来見てきた品質メトリクス

- 「今、目標に対してどの程度の品質を達成できているか」を問うてきた。
  先ほどのアナロジーでいうところの、「位置」的なメトリクスである。
  - 目標とする品質と現在の品質との距離を測ることで、リリースの可否を判断している。
- 代表的なメトリクスは、たとえば以下。
  - **テストケース密度**: 開発規模あたりのテストケース数
  - **欠陥密度**: 開発規模あたりの欠陥数
  - **信頼度成長曲線**: 時間を横軸に取ったときの欠陥摘出のトレンド
- 以下のような暗黙の前提がある。
  - 品質の良し悪しは、開発データから判断することができる（それしかない）。
  - リリース周期が長いので、一発で理想の品質を達成する必要がある。
  - ゴールの場所は変わらない。仕様変更は悪である。

これは今でも真なのか？

暗黙の前提が変わりつつある

アジャイル開発においては、先の前提が変わっている。

- 品質の良し悪しは、開発データから判断するしかない。
  → リリース後は運用データを取得し、これに基づいて品質も計測できる。
- リリース周期が長いので、一発で理想の品質を達成する必要がある。
  → リリース周期が短いので、品質が完璧でなくても(*)漸進的に近づける。
- ゴールの場所は変わらない。仕様変更は悪である。
  → ゴールの場所は変わるし、そもそも明白ではないので、追い続ける。

前提の変化は、品質メトリクスにどう影響するか。
新旧のメトリクスの本を調べてみた。

(*) そうは言っても、ユーザに迷惑をかけない程度に十分な品質は達成している必要がある。

モノの本で調べてみた結果(*1)

従来型の品質メトリクスの例

- **総欠陥数**: 開発中に摘出した欠陥を累積した数
- **残存欠陥数**: リリース時に残っている欠陥の数
- **欠陥率**: 開発中の時間あたりの発生欠陥数

➡ 「開発」にフォーカスしている。

アジャイルの品質メトリクス(*2)の例

- **平均修復時間 (MTTR)**: インシデント発生から修正・デプロイまでの平均時間
- **リードタイム**: 新しい機能が定義されてからユーザに届くまでの平均時間
- **コード変更量**: 機能追加やバグ改修のために変更するコードの量
- **バグ率**: 新機能の導入に伴い発生したバグの数

➡ 「開発」と「運用」は一体と捉えている。

品質メトリクスの性質の比較

項目	従来	アジャイル
品質の位置づけ	•リリース物に欠陥が少ないこと	•リリース物がユーザの要求に近いこと •近づける能力を組織が有すること
メトリクスの用途	•計画のために見積もる •開発中の品質状況を把握する	•品質の継続的な改善につなげる
データの取得タイミング	•主に開発中のデータ	•開発中と運用中のデータの両方
データの源泉	•コード管理システム •チケット管理システム	•左記＋その他の開発基盤・本番環境

➡ 「品質」の位置づけ自体が変化している。

(*1) 詳しくは、 従来とアジャイルで、品質メトリクスには本質的な違いがあるのではないか という記事をご覧ください。
(*2) 有名なFour Key Metrics は、デプロイ頻度、リードタイム、修復時間、バグ率の4つ。

アジャイル開発における品質メトリクス

- 推奨される品質メトリクスが、別モノ(*1)。
  ソフトウェアそのものだけでなく、それを作るチームや基盤が
  品質の指標になるという考え方である。
  - **従来型のメトリクス**: 達成した状態を図る ← 「位置」的
    - 「プロダクトは、リリース可能な品質をもっているか」
  - **アジャイルメトリクス**: 達成する能力も測る ← 「速度」「加速度」的
    - 「プロダクトは、ユーザ価値に早く近づくための性質をもっているか」
    - 「チームは、ユーザ価値に早く近づくための能力や仕掛けをもっているか」
- モノの良し悪しを高頻度で確認できるから、「いま品質がどうか」に加え、
  「品質が悪いときにそれをよくできるか」を知りたい。
  - 「よくできない」となったら、プロダクトやプロセスの改善につなげていく。

わたしのようなウォーターフォール出身者は、
パラダイムの変化に対応しなければならない。

(*1)「メトリクスは正しく使われなくなる」性質については、従来もアジャイルも変わらない。
メトリクスの弊害については、『測りすぎ』（ジェリー・Z・ミュラー・著）が詳しい。
ブログ記事としては以下が秀逸。開発生産性指標を向上させるためにやってはいけないアンチパターン

品質が評価できたら、
次はその「改善」ですよ。

想定経過時間 1:09:00

プロセス改善

プロセス改善、
地味で退屈そうっていう
印象ありますよね？

(*)「継続的インプルーブメント」とかいえば流行るのでしょうか？

プロセス改善(*1)のアプローチ

大きく2つのアプローチがある。

1 既存の改善モデルに乗る

- プロセス改善の専門家によって練られたTo Beが整理されている。
  - テストでいうと、TMMiやTPIなど。
- 事前にモデルの学習が必要である。

2 自分たちが直面した問題からスタートする

- 問題が目の前に認識しやすいので、手がつけやすい。
- 短期的な問題にばかりフォーカスしてしまうことがある。

今年のJSTQBカンファレンスでは、Erik Van Veenendaal氏がテストプロセス改善についての基調講演を行う予定！

(*1) 本来は「プロセス改善」の前に「プロセス策定」があるはず。が、わたしが「ゼロからのプロセス立ち上げ」を経験していないため、何も書けず‥。

直面した問題からのプロセス改善

- 品質分析から
  - 根本原因分析(*1)でプロセス上の改善点を特定し、改善を検討する。
    - Ex.) 仕様の記述が不十分で、実装漏れが多い
      → 仕様の記述の十分性を確保するために、パースペクティブベースドなレビューを行う
  - 同じ失敗の再発を防ぐための施策が出ればなおいい。
    - 「周知徹底する」「意識を醸成する」など、聞こえがいい「施策」に注意。
- ふりかえりから(*2)
  - 高頻度で自分たちの活動を見つめられるのがアジャイルのいいところ。
- インシデントから
  - 問題を解決して終わりではなく、そこから教訓を得て、改善につなげる。
    - 「ポストモーテム」(*3)という概念も知られている。

改善を積み重ねることで、チームとして強くなれる。

(*1) RCA、Root Cause Analysis。何を「根本」というかはチームに依ったりする。ここでは「問題を作り込んだプロセス」のこととする。
(*2) テストのプロセスをどんどん改善していく素敵な記事がコチラ。エンジニアとQAEの壁が崩れていくのを眺めていた
(*3) そのままの書名で『ポストモーテム みずほ銀行システム障害 事後検証報告』（日経コンピュータ・著）という本があり、当時の恐ろしい内情を伝えている。

最悪のシフトライト、インシデント

テストをどうシフトしても本番環境で起きてしまう、それがインシデント。

- シフトライトテストとインシデントの大きな違い
  - シフトライトには「満足か不満かの仮説検証」の側面があるが、インシデントは「不満」一択。
  - シフトライトテストは計画されたものなので「戻し」がしやすいが、インシデントは想定外で、「戻し」の方法があるとは限らない。その中で、できる限り短期間で解決する必要がある。
- インシデントコマンダーという概念の普及
  - 従来、知識と経験とリーダーシップを持った人間の職人技であった「障害対応」だが、ノウハウが書籍で整理され、「インシデントコマンダー」というロールも知られ始めている(*1)。
    - 弊社では、障害解決とまとめやお客様対応を、QAエンジニアが担うことが多い。

「失敗」を解決するためのインシデント対応も、QAが修めるべき分野の一つ。

(*1) 『システム障害対応の教科書』（木村誠明・著）に詳しい。

失敗とは何なのか？

- 「失敗学」でいう失敗(*1)
  - 許される失敗: 人間や組織が新しいことに遭遇して、ジタバタしながら何とか解決していくときに起こす失敗。
  - 許されない失敗: 自分の行ったことから何も学ばず、同じ愚を繰り返す失敗。
    - 許されない失敗を起こしたとしても、そこで失ったものより、得られた知識で次に起こりうる同種の失敗を未然に防ぐことで全体として失敗の総量を減らすことができたときに、はじめの失敗は「許される失敗」に消化できる。
- アジャイル開発では、毎日「失敗」している
  - 「イテレーションの本質は、私たちが誤りを犯し、犯した誤りを正すことを認めることであり、学びを発展させ、広げていくことを認めることです」(*2)
  - 失敗学でいう「許される失敗」を小さく繰り返すことで、成功に近づこうとするアプローチ。「前向きな失敗」といえそう。

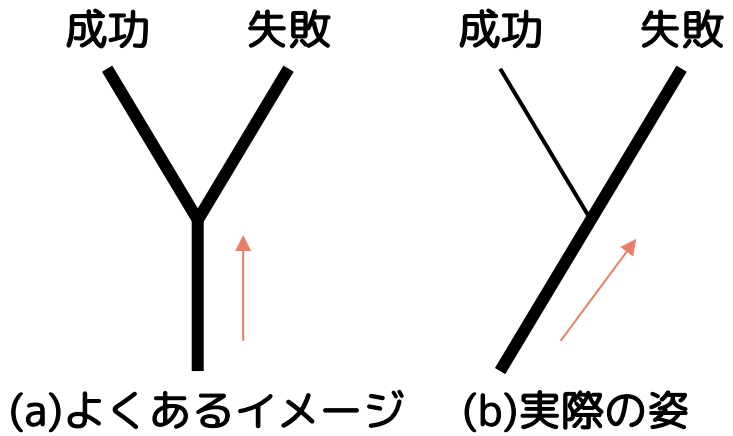
では、許されない失敗はどうする？

(*1) 失敗学は、畑村洋太郎氏が提唱した学問。以下の文章は、同氏の『技術の創造と設計』より。
(*2) 『継続的デリバリーのソフトウェア工学』（David Farley・著）の第4章より。

許されない失敗をどう軽減していくのか

失敗のイメージ(*1)

- 右図(a)は、「成功か失敗か」の分岐点が明確。
  人はそこで立ち止まるので、間違えづらい。
- 右図(b)が、実際に現れる失敗の姿。自然に進んでいく先に
  失敗があり、失敗だったことは後になってわかる。



失敗の軽減

- 仕組みでの対策: 失敗を起こさせないフールプルーフ、失敗が起きても大事に至らない
  フェイルセーフなど、仕組みを設けて人間のミスを予防・軽減する。
- チェックリストでの対策: レビューの項で言及したような森羅万象型ではなく、
  「絶対に犯してはいけないミス」を防ぐための、一撃必殺型チェックリスト(*2)。
  - 飛行機のエンジンが停止した時のチェックリスト、1つ目の項目は、「飛行機を飛ばせ」。

**ITスキルだけでなく、失敗学・レジリエンス工学あたりも
学んでいく必要性を感じる。**

(*1) 『失敗学 実践編』（濱口哲也・平山貴之・著）より引用、加筆。
(*2) 『アナタはなぜチェックリストを使わないのか？』（アトゥール ガワンデ・著）に詳しい。

改めて、品質とは？

想定経過時間 1:15:00

品質と価値

あんたは‥
さっきから価値と言っている。
‥価値って、何かね？(*1)

(*1) 中年以上にしか伝わらない、ドラマ「北の国から ‘92巣立ち」より。

「品質」「価値」とは何なのか

- 21億回は引用されている、Weinbergの定義とその派生
  - “Quality is value to some person.”

    ——— Gerald M. Weinberg
  - “Quality is value to some person, at some time, who matters.”

    ——— James Bach, Michael Bolton
- 「価値を届ける」って何だろう‥
  - アジャイルの普及に伴い、「価値」という言葉が大人気になった印象。

品質と価値の関係

- **品質特性(*1): あるモノに対し、ある時点で一意に決まる客観的な特性**
  - モノ自体が固有にもつもので、誰が見ても同じになる。
    - エイジングビーフの熟成年数やレスポンスタイムは、ある時点で一意に決まる。
- **価値: あるモノに対し、ある時点である利用者が受ける主観的な特性**
  - 同じ品質特性でも、利用者によって感じる価値は異なる。
    - 30か月エイジングビーフの熟成を美味と感じる人と、新鮮な肉を求める人がいて、異なる価値を受け取っている。Webページのレスポンスタイム3秒が早いか遅いかは人による。
  - 同じ品質特性、同じ利用者でも、時間の経過によって感じる価値は異なる。
- **品質と価値**
  - 価値 `V` は、品質特性の関数として、`F(Q1, Q2, …,Qn)` と表現できる。
  - 「顧客の立場に立つ」というのは、関数 `F` を正確に想像しようとする営み。
    完全には理解できないし、変化するし、顧客自身も説明できないかもしれない。

(*1) ソフトウェアにおいては、広義の「品質」に「スコープ」を含むものとする。ISO/IEC 25010でいう品質特性には「機能適合性」が含まれる。

ご参考:「過剰品質」とは何なのか

「過剰品質」という時、2つのケースがありそう。

1 品質も価値も上がっていない「品質向上」

- たとえば、リスクの高くない部分に対して、組み合わせテストのカバレッジを高める。
  テストケースは増え、時間も工数もかかるが、品質は上がっていない。
  顧客にとっての価値も増えていない状況。

2 品質が上がっているが価値は上がっていない「品質向上」

- たとえば、すでに十分な応答性能をさらに高める。
- 狩野モデルでいうところの「当たり前品質」。
  その品質は上がっているかもしれないが、顧客にとっての価値は増えていない。

渋谷交差点の価値

友人の要望「Shibuya Crossing is a must-see for me.」
われわれの計画「じゃあ空いてる午前中、早めに渋谷に行こう」

何が間違っているのか？

わたしたちは、「空いている」という品質特性に価値があると考えた。
一方「観光地としての渋谷交差点の価値」は、「混んでいる」ことだった。

価値があると自分達で考えている品質特性は
ユーザにとって価値があるのか、考えないといけない。

発表時間90分、スライド120枚だから価値があるわけではない。

「品質より価値だ！」となったわけではない

価値が品質の関数ならば、品質特性を知っておく必要性は変わらない。

SQuAREで定義する従来型プロダクトの品質特性

開発視点とユーザ視点を分けている。

- ISO/IEC 25010: 製品品質モデル、利用時の品質モデル
  - 利用時の品質特性には、「実用性」「快感性」といった、主観的なもの(*1)も含まれる。
  - 簡単に測定できず、自動化も難しく、仕様書にも現れないこの部分が、QAエンジニアの能力を活かしたり、シフトライトテストに頼れる部分かもしれない。
- ISO/IEC 25012: データ品質モデル

機械学習に関係するプロダクトの品質特性

従来の製品品質にはない特性が追加されている。倫理・道徳・正義といった徳目も‥

- ISO/IEC 25059: AIシステムの製品品質モデル（次スライド）
- AIプロダクト品質保証ガイドライン: AIプロダクトの品質保証において考慮すべき軸を5つ定義し、それぞれに対し、チェックリストの形で考慮すべき項目を述べている。
  - たとえば「Data Integrity」という軸に対し、「学習データの量の十分性」といった項目。

(*1) さっき、品質特性は客観的で一意に決まるって言ったばかりなのに‥

ご参考: AIシステムの製品品質モデル

- ISO/IEC 25059:2023(*1)
  - ISO/IEC 25010:2011の製品品質モデルに対し、品質副特性の1つが修正、5個が追加されている。

AI System Product Quality

Functional Suitability
- Functional Completeness
- Functional Correctness
- Functional Appropriateness
- Functional Adaptability

Performance Efficiency
- Time Behaviour
- Resource Utilisation
- Capacity

Portability
- Installability
- Replaceability
- Adaptability

Usability
- Appropriateness Recognizability
- Learnability
- Operability
- User Error Protection
- User Interface Aesthetics
- Accessibility
- User Controllability
- Transparency

Security
- Confidentiality
- Integrity
- Non-repudation
- Accountability
- Authenticity
- Intervenability

Reliability
- Maturity
- Availability
- Fault Tolerance
- Recoverability
- Robustness

Maintenability
- Modularity
- Reusability
- Analysability
- Modifiability
- Testability

Compatibility
- Co-existence
- Interoperability

(*1) iTeh, Incのサイトで、プレビューとして公開されているものから引用・加筆。

チーム内で「品質」の意識がズレた場合に‥

開発チーム内でのギャップも生まれうる。

■ グローバルな開発での学び

品質に対する価値観に、顕著な差が出る。大事なのは以下の3点。

1. その時点の目的に合った「あるべき品質」を定める。
   - PoCでも本格稼働でも、同じ「ピカピカ品質」を目指そうとしていないか？
2. 決めたことをできるだけ具体的に書き下し、伝える。
   - よく言われる「日本人はハイコンテキスト」(*1)を自覚する。
   - 「言わなくても普通やるでしょ」という感覚は捨てる。

   } 生成AIの話に似てくる‥
3. それでも現れるギャップについて、要因を話し合い、改善する。
   - 「QAエンジニア」のロール定義自体も違ったりする。
   - 「自分たちが正しい」と思いあがらない。

それでも、「品質は大事だよね」という共通認識を持つには？

(*1) 出身国だけで人の特性を決めつけるのは危険だが、国民性の類型については『CULTURE MAP』がわかりやすい。

品質文化

突然の自社自慢

- 弊社は、Quality > Delivery > Cost をマントラにように唱えている。
  判断の分岐点では、品質に基づいて判断することが体に染みついた風土。
  - 「基本と正道」「損得より善悪」という価値観も、ひたすらに叩きこまれる。
  - 「品質第一」は退屈な標語だが、繰り返すことが大事なのだと今ならわかる。
- 品質は、関係者全員で作り込むことが当たり前だとみんなが思っている。

「品質第一」の二つの側面

- **ポジティブ**: 品質の高いプロダクトを提供できれば、お客様に喜んでいただけ、また利用していただくこともできる。社会にとっても良い影響を与えられる。
- **ネガティブ**: 品質の低いプロダクトを提供してしまうと、インシデントの対応にコストがかかるだけでなく、自社・お客様・社会にとって取返しのつかない損害をもたらしかねない。

品質文化

品質を大切にする文化をどう作っていくのか

❶ マントラ化

- トップはもちろん、ミドルマネジメントもひたすら繰り返す。
- 精神論っぽいのと、若干の「洗脳」感、思考停止につながるリスクもあるけれど、この種の原則は、継続的に伝えることが重要。九九のごとし。

❷ ふりかえりと蓄積

- 品質における失敗をふりかえり、その失敗を起こしてしまった要因を突き止め、次にそれができるだけ起こらないようにプロセスを改善する。
  - チーム単位から事業部単位まで様々な単位で行うことができる。
  - 「ふりかえりミーティングの開催」自体が目的ではない。
- 失敗の知識を蓄積し、教育や勉強会を通じて、組織横断的に伝えていく。

❸ 損失の可視化

- たとえば「障害対応にいくらかかったのか」と、金額の形で理解する。
- ユーザが困難に直面している姿を目の当たりにする。

品質第一‥なのか？

- Q > D > Cからの変化
  - 品質が、納期やコストに優先するという順番は変わらない。
    ただ、QとDとCは、あくまでも「ソフトウェア開発」に閉じた話になっている。
    これらにさらに優先されるものとして、SafetyやComplianceを挙げている。
  - Safety: 品質の良いソフトウェアを、予算内・期限内に納めた。
    しかしそれが、過剰なサービス残業と健康被害の上に成り立っているものだったら？
    ➡ 人々の健康や安全を優先する。
  - Compliance: 素晴らしい精度のAI組込みシステムを完成させた。
    しかしそれが、不当に取得したデータを学習させたものだったら？
    ➡ 社会に対する誠実な対応を優先する。
- 倫理とか道徳とかインテグリティとか(*1)
  - テキスト系生成AIの品質特性として、倫理・誠実といった言葉が現れた。
    しかしこれは、技術者の持つべき特性としては決して新しいものではない。

(*1) 失敗と企業倫理については、『軌道 ――福知山線脱線事故 JR西日本を変えた闘い』を読んでいただきたい。

参考: ISTQBにおける倫理規範

- **倫理規範 (Code of Ethics)(*1)**
  - 公共 - 認定ソフトウェアテスターは、公の利益に一貫して行動しなければなりません。
  - クライアントおよび雇用者 - 認定ソフトウェアテスターは、公の利益と一致する形で、クライアントおよび雇用者の最善の利益にかなう行動をとらなければなりません。
  - 製品 - 認定ソフトウェアテスターは、自身が提供する成果物（テストする製品およびシステム）が可能な限り最高の専門的基準を満たしていることを確認しなければなりません。
  - 判断 - 認定ソフトウェアテスターは、専門的判断において誠実さと独立性を維持しなければなりません。
  - 管理 - 認定ソフトウェアテストマネージャーおよびリーダーは、ソフトウェアテストの管理において倫理的なアプローチを支持し、促進しなければなりません。
  - 職業 - 認定ソフトウェアテスターは、公の利益と一致する形で職業の誠実さと評判を向上させなければなりません。
  - 同僚 - 認定ソフトウェアテスターは、同僚に対して公平で協力的であり、ソフトウェア開発者との協力を促進しなければなりません。
  - 自己 - 認定ソフトウェアテスターは、自身の職業に関する生涯学習に参加し、職業の実践において倫理的なアプローチを促進しなければなりません。

ビジネスの前に、職業人としての倫理の順守が、業界を問わない大前提。

(*1) ISTQBのサイトに掲載されたものを、ChatGPTの翻訳をベースに引用。

品質ダンジョンを徘徊して、品質の話まで辿りつきました‥

最後突然、説教くさい話になったし、
カバーできていない分野まだまだあるけど‥

QAエンジニアの仕事

QAエンジニアは、
何をリスキリングしていこう？

QAエンジニアとして学んでいきたいこと

- QAエンジニアの役割 わたしの場合(*1)

技術面

- レビューやテスト実行、テスト自動化の推進による、直接的な品質確保
- テスト分析・設計といった方法論の適用と普及

マネジメント面

- プロダクトの品質確保のための品質戦略・計画立案
- 品質メトリクスやバグ分析に基づく品質向上・プロセス改善の推進
- ユーザの反応を知るチャネル作り

文化・ナレッジ面

- 品質と倫理をビジネスのベースにおく風土造り、人材育成
- 品質や失敗の経験のナレッジ化・展開・維持

こんな役割を果たすために、何を学んでいくといいだろう？

(*1) 慌ててイイワケしておくと、「QAはこれだけやればいい」という意味でもなければ、「QAだけがこれをやればいい」という意味でもありません。

QAエンジニアとして学んでいきたいこと

わたしはこんな感じで「リスキリング」しようと思っています。

1 品質を学ぶ

- 品質戦略、テスト分析といったテストプロセスの「上流」をもっと知りたい。
- レビューやバグ分析など、自然言語が壁になっていた活動に、生成AIブーストかけたい。
- あえて、手動テストの方法論を整理したい。
- 温故知新として、品質管理を学び直したい。

2 人を知る

- 行動経済学などから、人間の心理や行動について学びたい。
- 年齢・国民性・宗教など多様な価値観と文化を理解したい。

3 文化を作る

- 過去の人為的事故や失敗学から、失敗と学習について理解を深めたい。
- 大学で学んだはずの、倫理・公正・正義といったこそばゆいテーマを学び直したい。

みなさんにとってのリスキリングネタが、
今日の話の中に1つでも転がっていれば幸いです。

ソフトウェア品質というダンジョン、
90分ではとても歩き切れませんでした。
まさに迷宮入りですね。（ここで爆笑）
プロダクトの品質と組織の文化を
リードするQAエンジニアとして、
ダンジョンを切り開いていきましょう！


ご清聴ありがとうございました！

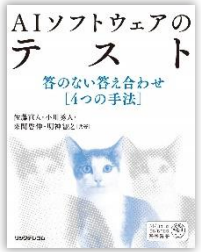
Happy Quality!!

QAエンジニアとしてのわたしを育ててくださった
会社の同僚、社外コミュニティのみなさん、
とりわけ故・にしさんに、この拙い発表を捧げます。

紹介書籍一覧(*1)

ソフトウェア品質知識体系ガイド(第3版)

AIソフトウェアのテスト



アジャイルメトリクス

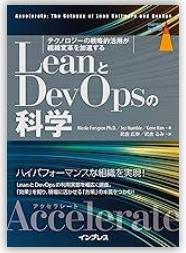
継続的デリバリーのソフトウェア工学

測りすぎ——なぜパフォーマンス評価は失敗するのか?

テスト自動化実践ガイド

システム障害対応の教科書

LeanとDevOpsの科学



失敗学 実践編

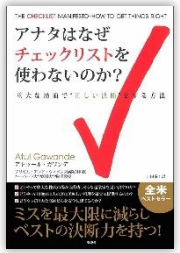
異文化理解力

知識ゼロから学ぶソフトウェアテスト 第3版

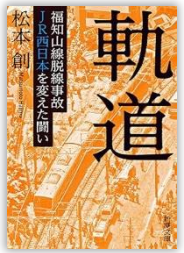
ポストモーテム みずほ銀行システム障害 事後検証報告

LeanとDevOpsの科学

アナタはなぜチェックリストを使わないのか？



軌道 ——福知山線脱線事故 JR西日本を変えた闘い



(*1) アフィリエイトリンクです（堂々）。チャリンしたお金でまた本を買って紹介します！

Q & A