フレイキーテストにどう立ち向かうか
How to deal with the flaky test

一般財団法人日本科学技術連盟
第40年度(2024年度)ソフトウェア品質管理研究会　成果発表会
研究コース4　アジャイルと品質　自動テストチーム
2025年3月7日(金)

研 究 員：山田　高史 (ブラザー工業株式会社)
今村　大輔 (株式会社オージス総研)
主査 ：永田　敦（サイボウズ株式会社）
副主査：荻野　恒太郎（株式会社カカクコム）
アドバイザー：山口　鉄平（株式会社LayerX）

自己紹介

- 研究コース4 アジャイルと品質 自動テストチーム

山田高史 (ブラザー工業株式会社)

- プリンタ制御ソフトウェアの品質保証・テストエンジニア
  - ここ最近はシステムテストの自動化がメイン
  - フレイキーテストに悩まされる
- 偏愛
  - 旅ラン(旅先でのジョギング、全国マラソン大会に出ること)

今村大輔 (株式会社オージス総研)

- 認証基盤パッケージソフト プロダクトオーナー 兼 ITアーキテクト
  - 実態は何でも屋、ドメインモデリングからインフラ構築・運用オペレーションまで
- WEBマガジン「オブジェクトの広場」で「マイクロサービスアーキテクチャに効く!テスト技法」を不定期連載中
  - https://www.ogis-ri.co.jp/otc/hiroba/technical/microservices-test/
- 偏愛
  - ~~泡盛、アイリッシュウイスキー~~ <= 尿酸値閾値超えにて禁酒中

Goal & Agenda

- Goal
  - テスト自動化に奮闘するエンジニアの皆様に、フレイキーテスト対応方針の「考え方」を考えてもらう
    - 我々から対応方針を判断するフレームワークを提案します
    - 皆様の現場にフレームワークを当てはめるとどうなるか？フィードバックをいただきたい
- Agenda
  - イントロダクション
  - フレイキーテストとは
  - 対策方針を判断するフレームワークの提案
  - 現場の実情と対策実験
    - Case 1: 制御系の現場
    - Case 2: WEB系の現場
  - 実験結果考察・今後の課題
  - まとめ

イントロダクション

- 研究目的
  - フレイキーテストの対策を通じて、テストを高度化し、プロダクトの品質を向上する

- コンテキスト
  - アジャイル開発に伴いテスト自動化を進めた結果、無視できない頻度でフレイキーテストが発生するようになった
  - フレイキーテスト自体も問題であるが、それに起因して開発業務の運営に大小様々な支障をきたしている
  - 研究員のドメイン領域は制御系とWEB系であるが、ドメイン領域を問わないフレイキーテストの傾向と対策を探索する

フレイキーテストとは

- テストの実行結果が実行するたびに異なるテスト
  - テストの条件は変わっていないのに、passしたりfailしたり
- 不具合の誤検知や見落としにつながり、テストの信頼性を低下させる
  - 単に「もう一回実行したら動きました！」だと、信用し難い
  - 偶発的な事象として無視すると、不具合を見落とす場合もある
- 原因はさまざま
  - テスト対象の状態
  - テストの実行順序
  - ・・・
- 先行研究ではGoogleのテストエンジニアによる記事が有名
  - Googleのテスト実行において、1.5%の割合でフレイキーネスが発生している
  - Googleのテストケースの16%がフレイキーテストである

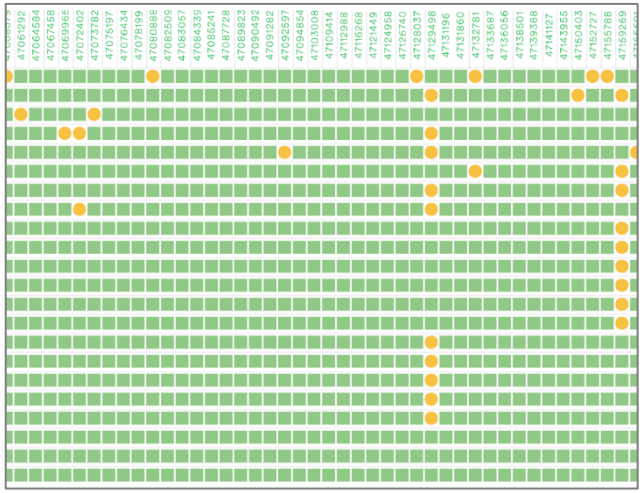
https://testing.googleblog.com/2016/05/flaky-tests-at-google-and-how-we.html
Flaky Tests at Google and How We Mitigate Them

フレイキーテストとは

- 先進企業ではどのように対処しているか？
  - 統計データをもとに、フレイキーテストを特定することを重視
  - フレイキーネス自体の修正にはあまり力を入れていない

- 事例
  - Google
    - フレイキーネスの発生を監視し、フレイキーネスが高い場合は自動的にテストを実行対象から外す
    - 「フレイキーネスを減らすため」の不具合として管理する
  - Spotify
    - フレイキーテストを可視化して明らかにする
      - 横軸でビルド（CI環境でのテスト）毎のpass or fail
      - 縦軸でテストケース

https://testing.googleblog.com/2016/05/flaky-tests-at-google-and-how-we.html
Flaky Tests at Google and How We Mitigate Them



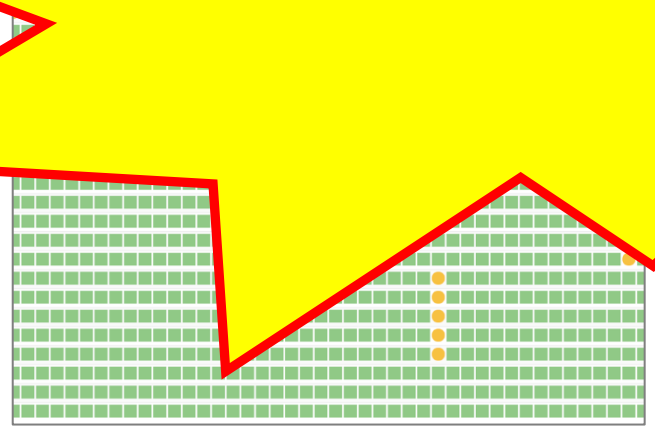
https://engineering.atspotify.com/2019/11/test-flakiness-methods-for-identifying-and-dealing-with-flaky-tests/
Test Flakiness – Methods for identifying and dealing with flaky tests

フレイキーテストとは

- 一般的にはどのように対処しているか？
  - 統計データをもとに、フレイキーテストを特定することを重視
  - フレイキーネス自体の修正にはあまり力を入れ[illegible]ない
- 事例
  - Google
    - フレイキーネスの発生
  - Spotify
    - フレイキ[illegible]

対策を真似できない！

統計を取るにも、ツールを作るにもリソース（資金・人員・設備）が足りない！



[illegible]g.com/20[illegible]-tests-at-google-and-how-we.html
[illegible]est[illegible]ow We Mitigate Them

[illegible]ps://engineering.atspot[illegible].com/2019/11/test-flakiness-methods-for-identifying-and-dealing-with-flaky-tests/
Test Flakiness – Methods for identifying and dealing with flaky tests

対策方針を判断するフレームワークの提案

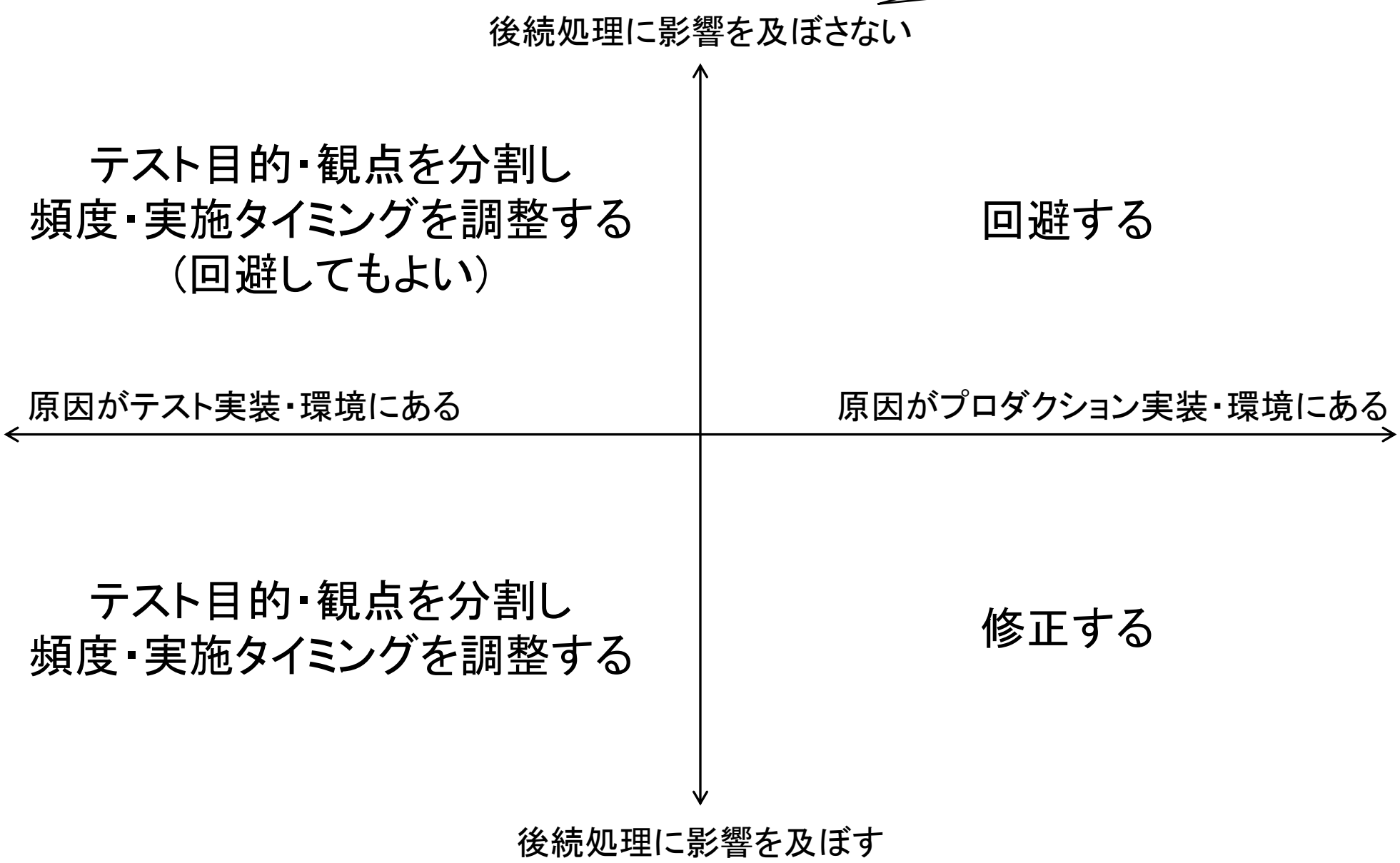
そこで！

リソースが慢性的に不足している現場において、
おおまかにフレイキーテストの当たりをつけたら
シンプルに「原因」と「影響」で対応方針を判断できないか？

対策方針を判断するフレームワークの提案

- 「原因」と「影響」のマトリクス

ここで言う「後続処理」の例：
- テスト対象機能の出力を別機能や連携先システムが入力として扱うケース
- テスト実行後の開発工程やアクティビティに影響を及ぼすケース



対策方針を判断するフレームワークの提案

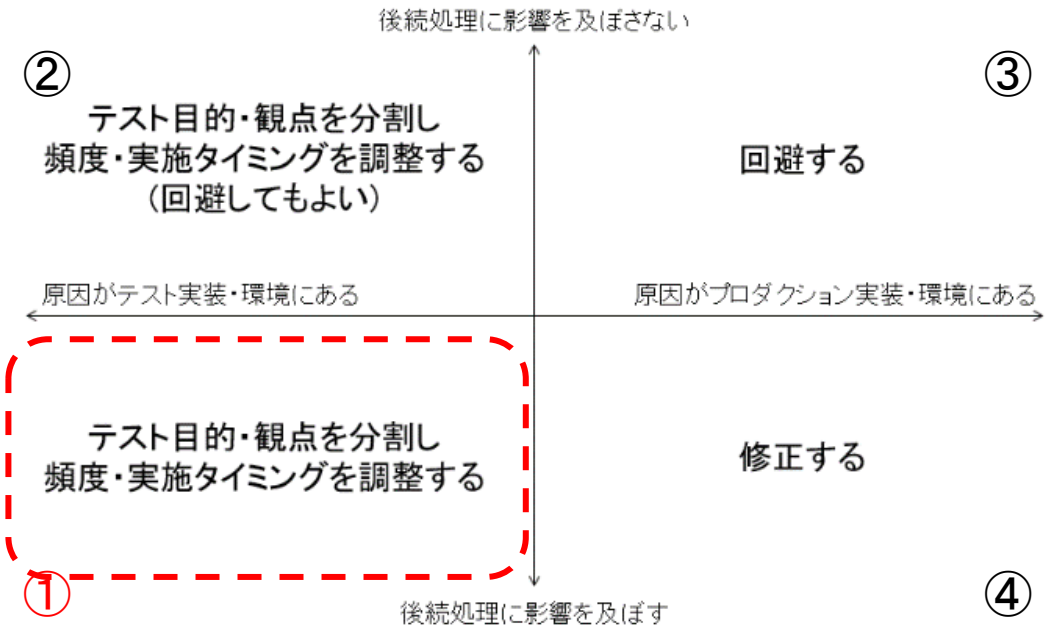
- テスト目的・観点を分割し頻度・実施タイミングを調整する

  - テスト側の問題
  - 顧客満足には影響しない
  - 修正で便益を得るのは開発者

  積極的に対応する必然性は薄い

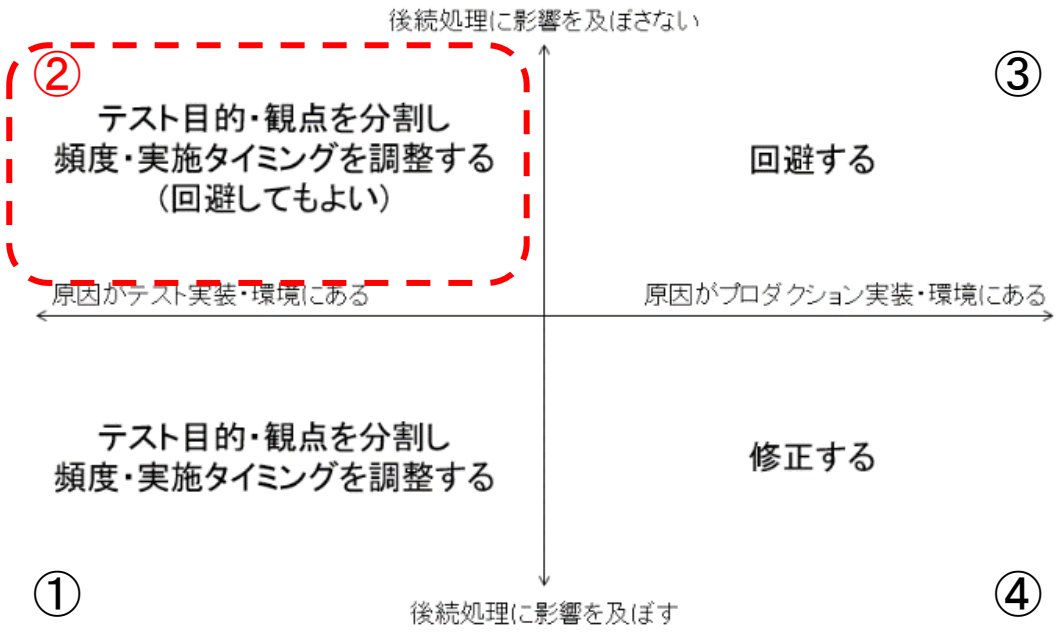
  - 後続処理に影響が及ばないようになっていればよい
    - テスト目的・観点の分割
    - 実行頻度・実施タイミングの調整

  - 高頻度実施のデベロッパーテストはフレイキーネスを無視する
  - 低頻度実施のQAテストはフレイキーネスを無視しない



対策方針を判断するフレームワークの提案

- テスト目的・観点を分割し頻度・実施タイミングを調整する（回避してもよい）
  - 第一象限とほぼ同じ
  - テストの問題、且つ、後続処理へ影響しないため、割り切って回避してもよい



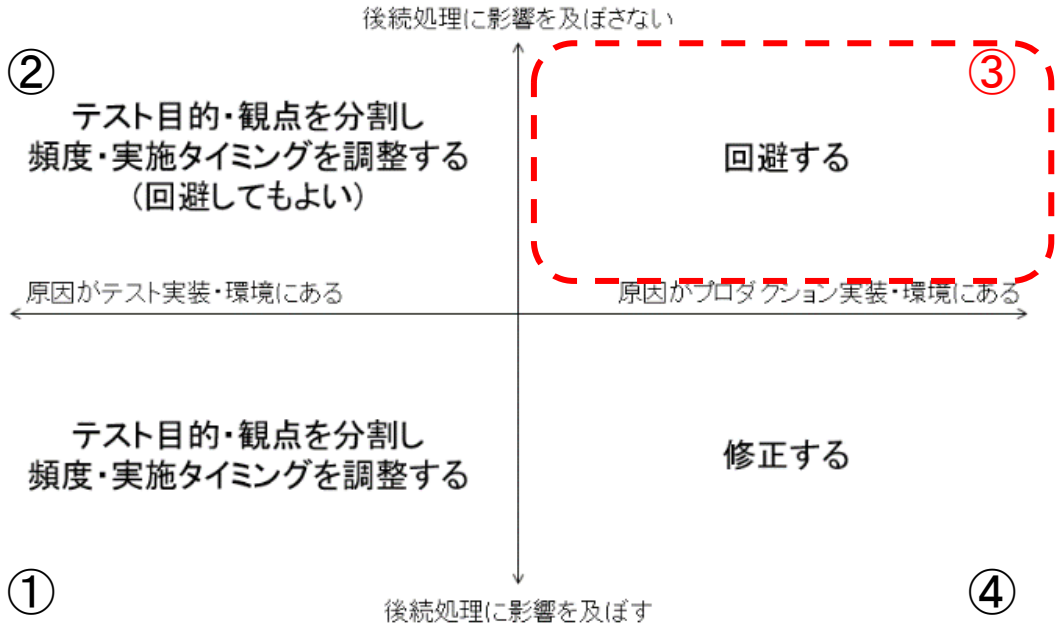
対策方針を判断するフレームワークの提案

- 回避する

  - テスト対象の不具合
  - フレイキーネスの影響がテスト対象に閉じている

  - 後続処理で予期しないトラブルに発展することはない
  - 単純なリトライ等で回避できる

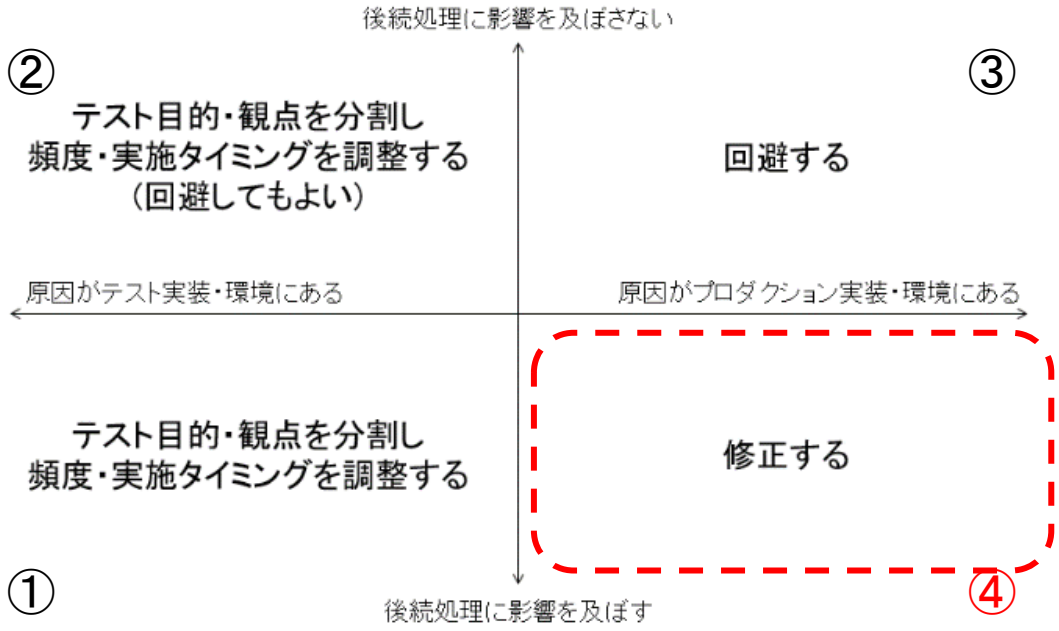
  - 他の優先すべき課題へ開発リソースを振り向けるとよい



対策方針を判断するフレームワークの提案

- 修正する
  - テスト対象の不具合
  - フレイキーネスの影響がテスト対象に閉じていない

- 後続処理で予期せぬトラブルに発展する可能性
- 低くはない優先順位で修正すべき



対策方針を判断するフレームワークの提案

- コンセプト
  - 優先順位付けを通じて、テスト戦略としてリソースを効率に活用する
  - 弱者の戦略として、ユーザニーズの本質を捉えた箇所にリソースを集中する
  - YAGNI & KISS

現場の実情と対策実験

- 実施事項
  - 先行研究の枠組みに当てはめて定量的に現状把握
  - フレイキーテストで何が問題なのか、本質を考えて課題を設定する
  - 「原因」と「影響」のマトリクスを当てはめて対策方針を導出
  - 試行して効果を確認

- 対象
  - Case 1: 制御系の現場
    - プリンタの制御ソフト
    - 自作ツールでテスト対象のGUIをステップ実行し、ステップ毎に期待値比較を行う
    - QAの立場で、出荷前の最終的な品質保証目的で自動テストを運用
  - Case 2: WEB系の現場
    - 多数のWeb APIで構成されるパッケージソフト
    - mocha（Node.js向けのxUnit）でテスト対象へHTTP要求し、HTTP応答と出力されたログに期待値比較を行う
    - 開発の立場で、日々の回帰結合テスト目的で自動テストを運用

現場の実情と対策実験 Case 1: 制御系の現場

- 定量的に現状把握

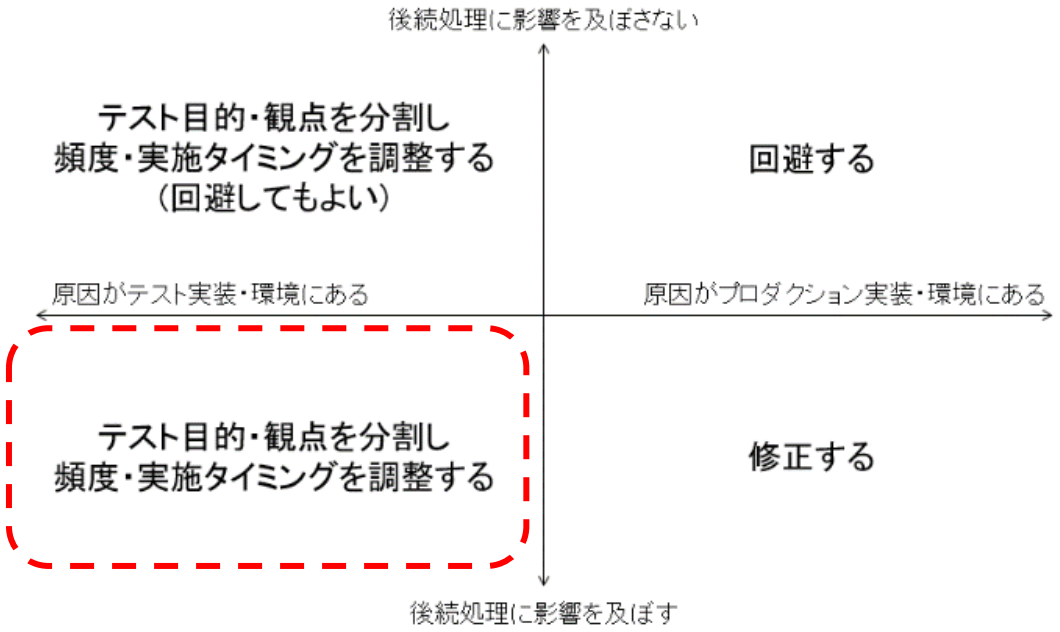
現場	総実行回数	テスト失敗回数	テスト失敗確率	フレイキーテストケース数	総テストケース数	フレイキーテストケース割合
制御系	387回	20回	5.2%	275ケース	816ケース	33.7%
Google（参考/比較用）	N/A	N/A	1.5%	N/A	N/A	16%

  - Googleの統計に対し、失敗確率・ケース割合共に大きい
  - フレイキーネスが発生した環境として、以下の特徴があった
    - 無線LANでプリンタを制御している
    - PCに搭載しているメモリが少ない
- フレイキーテストで何が問題なのか、課題を設定する
  - フレイキーネスでfailすると、テストを最初からやり直す必要があり、約40時間かかる
  - 出荷前・締切直前でのフレイキーネス発生は市場投入スケジュールを乱してしまう

  - 課題は「リリーススケジュールを乱さない」
    - 必ずしもフレイキーネスの根絶ではない

現場の実情と対策実験 Case 1: 制御系の現場

- 「原因」と「影響」のマトリクスを当てはめて対策方針を導出
  - 原因がテスト環境にある
  - 後続の処理に影響を及ぼす
    ↓ ↓ ↓
  - テスト目的・観点を分割し頻度・実施タイミングを調整する
    ↓ ↓ ↓
  - 出荷前のスモークテストとして「基本的な機能が正常に動作すること」を確実に保証する
  - 悪条件テスト環境での実施は別スケジュールに切り出し，クリティカルパス外で不具合検出に注力する
    - ネットワークが不安定
    - メモリ不足



現場の実情と対策実験 Case 1: 制御系の現場

- 試行して効果を確認
  - 安定したテスト環境でスモークテストを実行することで，フレイキーネスが発生せず，予測した実行時間でテストが完了すること
    - メモリを多く搭載したPC
    - 有線LANでのネットワーク接続

試行検証ポイント	結果	評価	コメント
フレイキーネス発生確率が変化するか	• 5.2% → 1.4% に削減	○	• 根絶には至っていないが、スケジュールを乱すリスクは下がっている
予測した実行時間でテストが完了するか	• 予測範囲の40時間で完了	○	• スケジュールを乱すリスクにはならない

  - 悪条件のテスト環境で実行することで，フレイキーネスを意図的に発生させ，詳細な調査ができること
    - メモリの少ないPC
    - 無線LANでのネットワーク接続

試行検証ポイント	結果	評価	コメント
フレイキーネス発生確率が変化するか	• 5.2% → 7.7% に向上	○	• 意図的にフレイキーネスを発生させやすくなった（再現性が高まった）
詳細な調査ができるか	• 付帯スクリプトや動画記録を並列実行できるようになった	○	• トラブル解析がしやすくなった

現場の実情と対策実験 Case 2: WEB系の現場

- 定量的に現状把握

現場	総実行回数	テスト失敗回数	テスト失敗確率	フレイキーテストケース数	総テストケース数	フレイキーテストケース割合
WEB系	77回	42回	54.5%	46ケース	約10,000ケース	0.46%
Google（参考/比較用）	N/A	N/A	1.5%	N/A	N/A	16%

- ほとんどのテストケースで総実行回数のうち1度だけfailが発生している
  - 今後、潜在的に発生し得るケースはまだまだ多いと思われる
- 例外である1つのテストケースのみ高頻度でfailが発生している

  ↓ ↓ ↓
- 自動テストの方式設計に問題がある
  - フレイキーネスが発生した場合、ログ出力内容の期待値比較でfailしている
  - ログは非同期で出力されている
  - テストはログが同期出力されている前提で実装されている

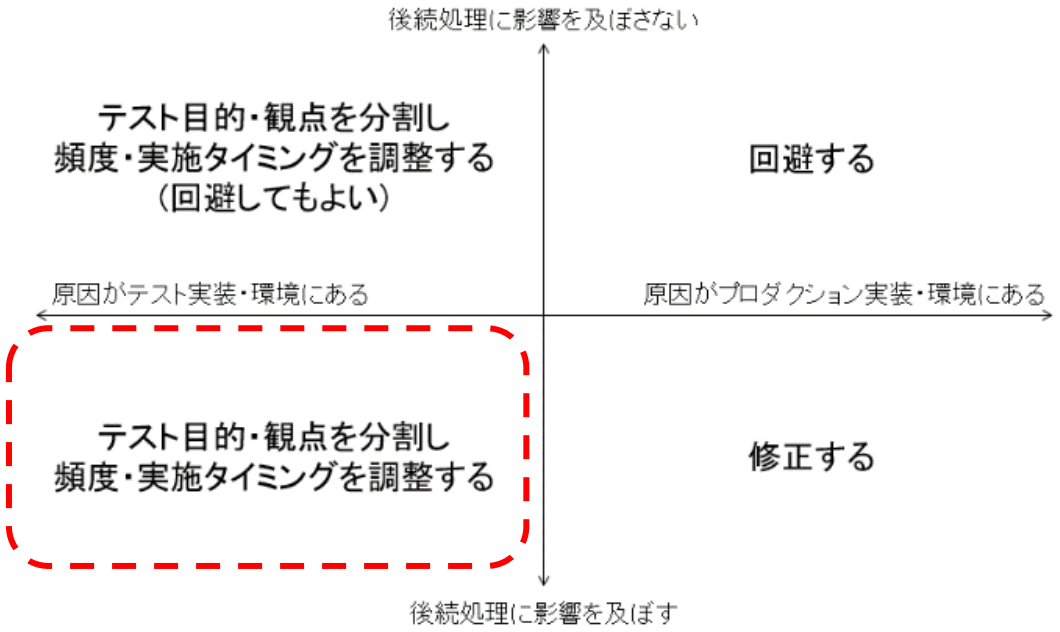
現場の実情と対策実験 Case 2: WEB系の現場

- フレイキーテストで何が問題なのか、課題を設定する
  - Pull Requestを起票する際、全ケースpass（デグレしていないこと）を確認するため0〜3回ほどリトライする必要がある
  - クラウドサービスを利用してテストを実行しているため、1回あたり$0.3ほど課金される
  - 月ごとのクラウドサービス利用料に制限がかかっている

  - 課題は「テスト再実行に伴う無駄なキャッシュアウトを抑制する」

現場の実情と対策実験 Case 2: WEB系の現場

- 「原因」と「影響」のマトリクスを当てはめて対策方針を導出
  - 原因がテスト環境にある
  - 後続の処理に影響を及ぼす
    ↓ ↓ ↓
  - テスト目的・観点を分割し頻度・実施タイミングを調整する
    ↓ ↓ ↓
  - 日々の回帰結合テストとしてはログ出力の期待値比較を実施しない
  - リリース前のシステムテストとしてログ出力内容のテストを実施する



現場の実情と対策実験 Case 2: WEB系の現場

- 試行して効果を確認
  - 日々の回帰結合テストでフレイキーネスによるテストのfailが発生しないこと
    - ログの期待値比較は実施しない

試行検証ポイント	結果	評価	コメント
フレイキーネスが発生しないか	• 5.2% → 0% に削減	○	• フレイキーネスを排除できている
Pull Request起票にあたり、回帰結合テストのクラウドサービス利用料を削減できているか	• 最大$0.9程度 → 0.3$ 程度に削減	○	• 無意味なリトライをせずに済んでいる • 結果として、キャッシュアウトを抑制できている

  - リリース前に適切なログが欠損なく出力されていることを保証できること
    - ログの期待値比較を実施する

試行検証ポイント	結果	評価	コメント
ログの期待値比較ができていること	• 何度かのリトライは必要だが、期待値比較できている	○	• リリース前のみであるため、リトライしてもコスト上の大きな影響はない
複数回テストを実行し、同内容のログが出力されること	• フレイキーネスが発生した場合でも、同内容のログが出力されている	○	• フレイキーネスの原因はログ出力が非同期であるにもかかわらず、テストドライバが同期しているものとして出力内容を検証していたことと確定できた

現場の実情と対策実験 Case 2: WEB系の現場

- 試行して効果を確認
  - 副次的効果
    - フレイキーネスによるfailか意図しないデグレードかを調査する手間が省けている
    - フレイキーネスという本質的ではない事由によるテスト再実行とその完了を待つストレスから解放されている
    - 開発中バージョンの主ブランチや本番稼働中バージョンのリリースブランチに対する日次回帰テストでデグレードを見逃すリスクを排除できている

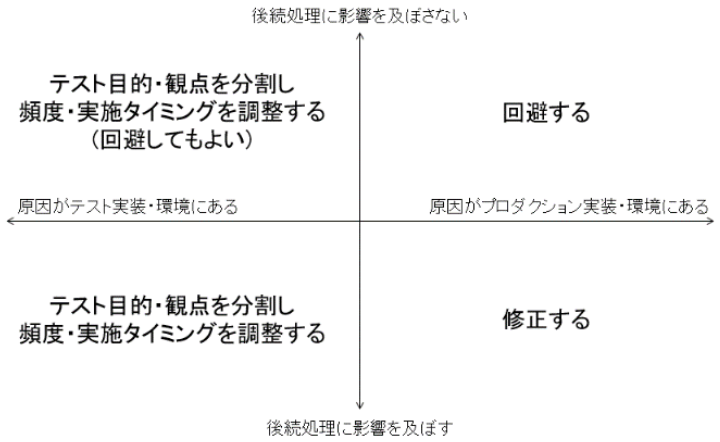
実験結果考察・今後の課題

- 実験結果の考察
  - ドメインを問わず、フレイキーネス発生以上に開発業務の円滑な進行を妨げるのが問題
  - 視野を広げて解決したいことを明らかにし、「原因」と「影響」のマトリクスに当てはめると対策方針が導出できる

- 今後の課題
  - サンプリング数を増やし、より的確な表現へのブラッシュアップ
  - 対策のみならず、より簡単にフレイキーテストを特定する方法論の探索

まとめ

- フレイキーテストとは・・・
  - テストの実行結果が実行するたびに異なるテスト
  - 先進企業の対策としては、統計データをもとにフレイキーテストを特定することを重視
- 対策方針を判断するフレームワークの提案・・・
  - 我々は先進企業のように潤沢なリソースを保持しておらず、先進企業の対策を真似できない
  - 先進企業との競争にあたり、テスト戦略とリソース効率を考えて対策する必要がある
  - そのために、「原因」と「影響」のマトリクスはいかがでしょうか？



- 研究員の現場にフレームワークを当てはめて、テストの目的・観点を分割し、実施頻度を調整することで・・・
  - （制御系）出荷前のスモークテストでテスト失敗確率が5.2%→1.4%に下がり、スケジュールが乱れるリスクを軽減
  - （WEB系）回帰結合テストの無意味なリトライを根絶し、テスト1回あたりのクラウドサービス利用料を$0.9→$0.3程度に削減