テスト基本のキ　ーテストをする前にー

koyaman / JaSST’19 Tohoku

freee 株式会社

この発表のゴール

テスト技法を使う前に認識すべき
大切なことを
皆さんが認識すること。

本日の流れ

※当たり前のことしか話しません。

- おっさん誰？
- テストの目的と本質の話
- リスクの識別の話
- テスト技法の必要性の話
- まとめ

Section

00

おっさん誰？

QAアニキ：コヤマン

小山 竜治

Ryuji Koyama
@koyaman2
https://github.com/koyaman2

CoreEngine/申告freee/AI月次監査などのテストを担当

ソフトウエア品質屋 / テストマネージャー / 自動化エンジニア

元海上J官(面影ナシ..)→ブラック派遣企業(転職・転籍)
→富士ゼロックス子会社の契約社員(転職)→社員(転職)
→カスペルスキー(転職)→freee(転職)。

アジャイルチームに入ってのテストや品質戦略などなど、
「何もないところからいいカンジにテストする仕組みを作る」
ことを生業としている。

組み込み系システム（複合機、医療機器）からクラサバシステム、
B2Bセキュリティソリューションなどを経て
Web系クラウド会計のfreeeでアジャイルテストを実践。
日本科学技術連盟にてソフトウエアテスト資格試験公認トレーニングコース講師、テスト自動化研究会やSeleniumConfTokyoなどの活動。
ASTER（ソフトウエアテスト技術振興協会）正会員。

犬好きのおっさんです。

色々なテストプロジェクト
をやってきた
何もないところに入って
テストのやり方を構築するマン

クラウド会計のfreee株式会社について

freee 株式会社

スモールビジネスを、世界の主役に。

アイデアやパッションやスキルがあればだれでも、
ビジネスを強くスマートに育てられるプラットフォーム

資本金

161億603万円 (資本準備金等含む)

設立年月日

2012年7月

従業員数

465名（2018年6月末時点）

事業内容

クラウド型バックオフィスサービスの開発・販売

PRODUCTS

freee 会計フリー

freee 申告フリー

freee 人事労務フリー

freee マイナンバー管理フリー

freee 会社設立フリー

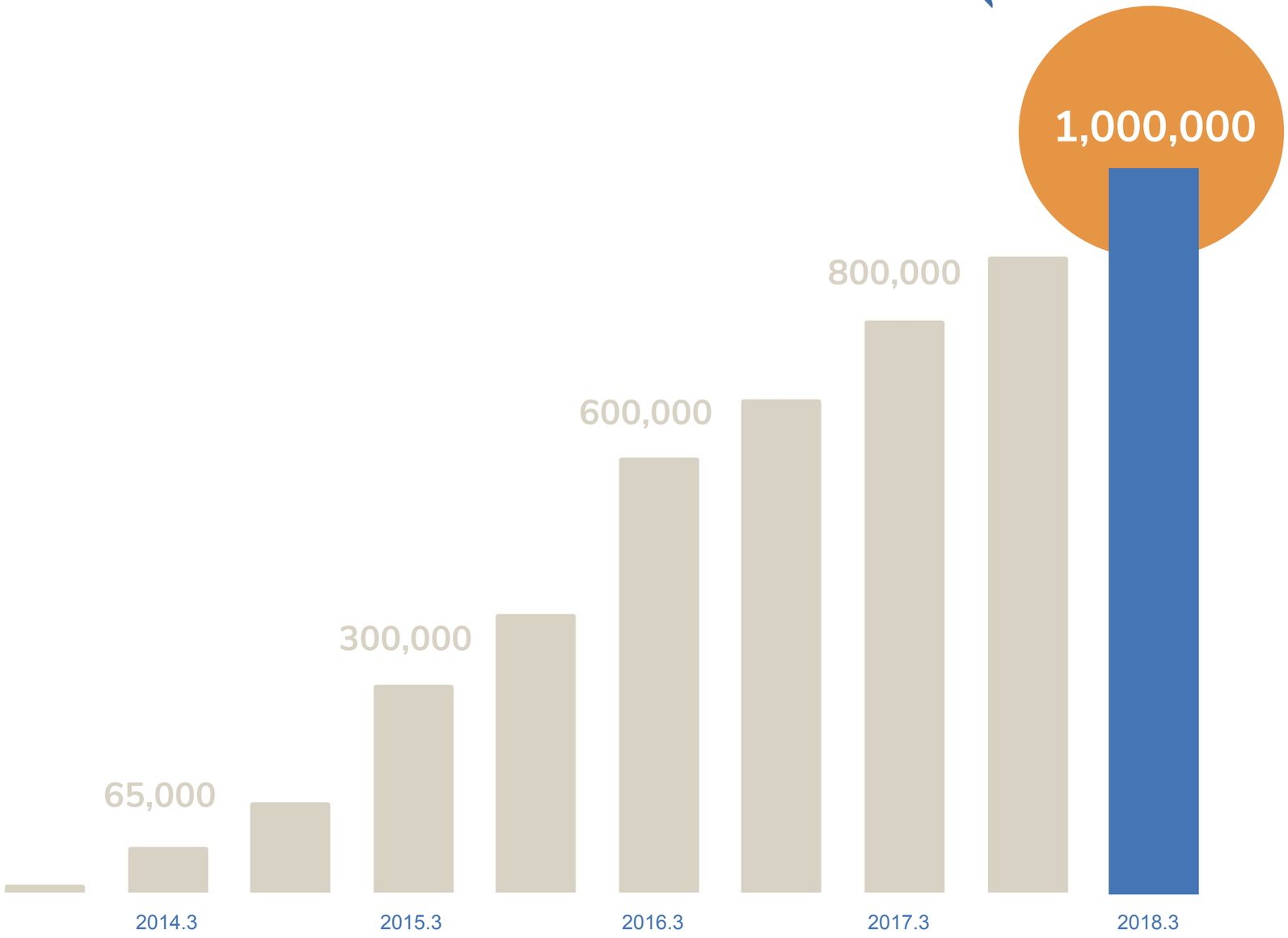
freee フリーカード

freee 開業フリー

税理士検索、税理士登録、民泊、顧問先管理
など、ここに書いていないものもあります。

利用事業所数累計

100万



最新技術でリアルタイムデータ連携

あらゆる業務がクラウド上で完結し連携できる時代に

Standard of Value

価値基準

freeeの価値基準は、仕事をする上で実際に使われている判断基準です

本質的(マジ)で価値ある

ユーザーにとって本質的な価値があると自信を持って言えることをする

理想ドリブン

理想から考える。現在のリソースやスキルにとらわれず挑戦しつづける

アウトプット→思考

まず、アウトプットする。そして考え、改善する

Hack Everything

取り組んでいることや持っているリソースの性質を深く理解する。その上で枠を超えて発想する

あえて、共有する

人とチームを知る。知られるように共有する。オープンにフィードバックしあうことで一緒に成長する

本日の流れ(再掲)

※当たり前のことしか話しません。

- おっさん誰？
- テストの目的と本質の話（←ココ）
- リスクの識別の話
- テスト技法の必要性の話
- まとめ

Section

01

テストの目的と本質の話
ーそもそも
「なぜテストするんですか？」

どんな現場でも使えるテストの目的の定義

色々な現場に入ってきましたが、どこでも使える
JSTQB(旧シラバス)の「テストの目的」の定義

- **欠陥を摘出する**
- **対象ソフトウェアの品質レベルが十分であることを確認する**
- **意志決定のための情報を示す**
- **欠陥の作りこみを防ぐ**

※新シラバス(2018版)ではこれをもう少し細かく書いていますが、
抽象的にいうと上の4つで間違いないです。

(補足)ISTQBシラバス(2018)の「テスト目的」引用

1.1.1 テストに共通する目的

すべてのプロジェクトで、テストの目的は以下を含む

- **要件、ユーザーストーリー、設計、およびコードなどの作業成果物を評価する**
- **明確にしたすべての要件を満たしていることを検証する**
- **テスト対象が完成し、ユーザーやその他ステークホルダーの期待通りの動作内容であることの妥当性確認をする**
- **テスト対象の品質に対する信頼を積み重ねて、所定のレベルにあることを確証する**
- **欠陥の作りこみを防ぐ**
- **故障や欠陥を発見する**
- **ステークホルダーが意志決定できる、特にテスト対象の品質レベルについての十分な情報を提供する。(以前に検出されなかった故障が運用環境で発生するなどの)不適切なソフトウェア品質のリスクレベルを低減する**
- **契約上、法律上、または規制上の要件または標準を遵守する、そして/またはテスト対象がそのような要件または標準に準拠していることを検証する**

テストの本質

知ること。
そして知ったことを伝える（＝意思決定のための情報を示す）こと。

逆に言うと「知ること」しかできません。

知ったことを伝えて利用することでさまざまな活用ができますが
- 問題や欠陥があることを欠陥レポートで伝え、それを修正・是正する
- 不明点を明確にして伝え、それを改善したりする

テスト自体はあくまで「知ること」しかできない。

カメラでの撮影とテストは似ている

個人的によくテストをカメラ撮影に例えます。

フイルムカメラ：現像するまでどんな写真が撮れたのかわからない
デジタルカメラ：撮影直後にどんな写真が撮れたのかわかる
→すぐに「知ること」ができると嬉しい

手動テスト(テスト管理ツールなし)：報告するまでわからない
自動テスト(テスト管理ツールあり)：結果が自動で通知あるいは反映される
→すぐに「知ること」ができると嬉しい

他の共通点については後述します。

さて。では何を知りたいんですか？

ゲームだと王様や村人が
教えてくれる。
「敵は魔王〇〇〇〇じゃ！」
「向こうの塔が宝があるよ！」

テストにおいては何を知りたいのか誰か
が教えてくれるとは限らない。

要求仕様があれば根拠がある？

一体いつから――要求仕様が正しいと錯覚していた？

「ソフトウェアテスト技法」故ボーリス・バイザー本にこうあります
MannaとWaldinger(MANN78)は、テストの完全性に対する理論上の障壁をつぎのようにいっている

「要求仕様が正しいという保証はどこにもない」
「どんな検証システムでも、プログラムが正しいことは証明できない」
「検証システム自体が正しいという保証はない」

簡単に言うと、人間はミスする不完全な生き物なので
「要求仕様が正しい」も「プログラムが正しい」も無理です。
※ミスらないんだったらそもそもバグ出ないですもんね。

でもオレが作ったもんは正しい？

「ソフトウェアテスト技法」故ボーリス・バイザー本にこうあります
プログラムの正しさを証明するには、
プログラムの構造のみに着目したテスト
機能のみに着目したテストおよび
正当性の形式的証明の3とおりがある。
しかし、どんな方法を導入しても、バグの数が0であるという意味での完全なテストは
理論的に不可能であるし、もちろん実際問題としても不可能である

利用しているライブラリやgemやツールにバグ無いって言いきれます？
クラウドだとしたらAWS/GCP/Azureにバグ無いって言いきれます？
KubernetesやDockerにバグ無いって言いきれます？
作ったその機能、誰からもクレーム来ないって言いきれます？
人間が使う、人間が作ったモノなんで無理筋じゃないですか？

JSTQBのテストの7原則

- テストは欠陥があることは示せるが、欠陥がないことは示せない
  - 悪魔の証明だぜ
- 全数テストは不可能
  - 「全部テスト」は非現実的だぜ
- 初期テストで時間とコストを節約
  - 早めからやろうぜ
- 欠陥の偏在
  - 偏るんだぜ
- 殺虫剤のパラドックスにご用心
  - 同じ写真じゃ新しいところに気付けないぜ
- テストは状況次第
  - モノや状況によるんだぜ
- 「バグゼロ」の落とし穴
  - 大事なのはそこじゃあないぜ

SWEBOKのソフトウェアテスティングの定義

SWEBOK―SoftWare Engineering Body Of Knowledge―
ソフトウェアエンジニアリング基礎知識体系より引用

第5章　ソフトウェアテスティング

ソフトウェアテスティングとは、通常は無限に大きいと考えられる
「プログラムの振る舞いの実行領域」から
最適だと考えられる有限な「テストケースの集合」を選定し
所期の通りかどうかを実際に動かして検証すること

無限から
最適だと考えられる有限
を選定する

でも何が最適かなんて
わからないんです。
どうします？

まず自分で考えて
決断する必要があるんです

おおまかな目的ごとに変わる「知りたいこと」

- 欠陥を摘出する
  - ➢ 欠陥がまだありそうか？出なそうか？
- 対象ソフトウェアの品質レベルが十分であることを確認する
  - ➢ 想定通り／予定通りに動くか？
- 意志決定のための情報を示す
  - ➢ 何を知ったのか？どんなことが起こったのか？
- 欠陥の作りこみを防ぐ
  - ➢ ヤバそうなところ、ヤバそうなことは何か？

状況や対象によっても目的や知りたいことが変わる

- テストは欠陥があることは示せるが、欠陥がないことは示せない
  - 悪魔の証明だぜ
- 全数テストは不可能
  - 「全部テスト」は非現実的だぜ
- 初期テストで時間とコストを節約
  - 早めからやろうぜ
- 欠陥の偏在
  - 偏るんだぜ
- 殺虫剤のパラドックスにご用心
  - 同じ写真じゃ新しいところに気付けないぜ
- テストは状況次第
  - モノや状況によるんだぜ
- 「バグゼロ」の落とし穴
  - 大事なのはそこじゃあないぜ

皆さんの関わりはどういう状況ですか？

- 皆さんの現場の開発プロセスは？
  - ウォーターフォールですか？
  - イテレーティブもしくはインクリメンタルですか？
  - よくわからんけどアジャイルですか？
  - リーンですか？

皆さんの関わりはどういう状況ですか？

- 皆さんはどこで絡んでいますか？
  - 要件定義ですか？
  - 開発ですか？
  - テストですか？
  - 運用・保守ですか？

皆さんの関わりはどういう状況ですか？

- 皆さんがテストするテスト対象のジャンルは？
  - 医療系など、人命に関わるもの？
  - 金融系など、経済的インパクトが甚大なもの？
  - 人工衛星など、社会的意義が高いもの？
  - 便利アプリ的なもの？

自分自身の立ち位置・背景を認識できましたか？

SDLCによっても目的が変わる

- 簡易SDLC(Software Development Life Cycle)別主目的
  - **企画：欠陥の作りこみを防ぐことが主目的**
  - **開発：欠陥の摘出が主目的**
  - **受入：品質レベルの確認が主目的**
  - **運用：品質レベルの確認が主目的**

※詳細なSDLCはIPA SECが発行している「共通フレーム」をご参考に。

動的テストと静的テストとでも目的が変わる

- **動的テスト:テスト対象を動かすテスト**
  - ➢ **欠陥の摘出、品質レベルが十分であることを確認する**

- **静的テスト:テスト対象を動かさないレビューや静的解析**
  - ➢ **欠陥の摘出、欠陥の作りこみを防ぐ**

※JSTQBではレビューもテストのうち、としている

テスト対象の特性によって徹底度が変わる

- 1万回に1回落ちる
  - **便利アプリなら？→まぁ許容できる**
  - **人命にかかわるシステムなら？**
    →1万人に1人が死亡する可能性アリ
  - **経済的インパクトが大きいものなら？**
    →1万回に1回訴訟問題
  - **ユーザーが100万人いる人命に関わるシステムなら？**
    →即100人に影響
  - **ユーザーが1000万人いる経済的インパクト大きいものなら？**
    →即1000回訴訟

テスト対象の特性によって許容できるかどうか変わりますよね。
なので**テスト対象と今の状況をまず知る必要があります。**

まず現状を知りましょう。
そして考えましょう。

まず現状を知る＝テスト対象の分析をしましょう

JSTQBのテストプロセス

最近、個人的には以下のイメージ（あくまでコヤマンの自己流です）

テスト分析のやり方もいろいろあります

- **HAYST法のラルフチャート/FV表**
- **VSTePのNGT**
- **ゆもつよメソッドの論理的機能構造**
- **マインドマップを使って整理・分析**
- **5W1Hや6W2Hでのまとめ**

などなど。

自分たちの背景・状況にあったものを使いましょう。
大雑把に言うと「理解」して「整理」できればOK。

JSTQBのテスト分析の説明
「何をテストするか」を決定する

あなたがテストをすることで
知りたいことは何ですか？

テストはアウトプット→思考

まず、自分が知りたいことを考えてみることが必要
誰かが答えを持っているわけじゃない。

あなたが理解した「状況」と「現時点のテスト対象」において、さらに「このテスト対象について自分たちのフェーズで知りたいと思うこと」は何ですか？

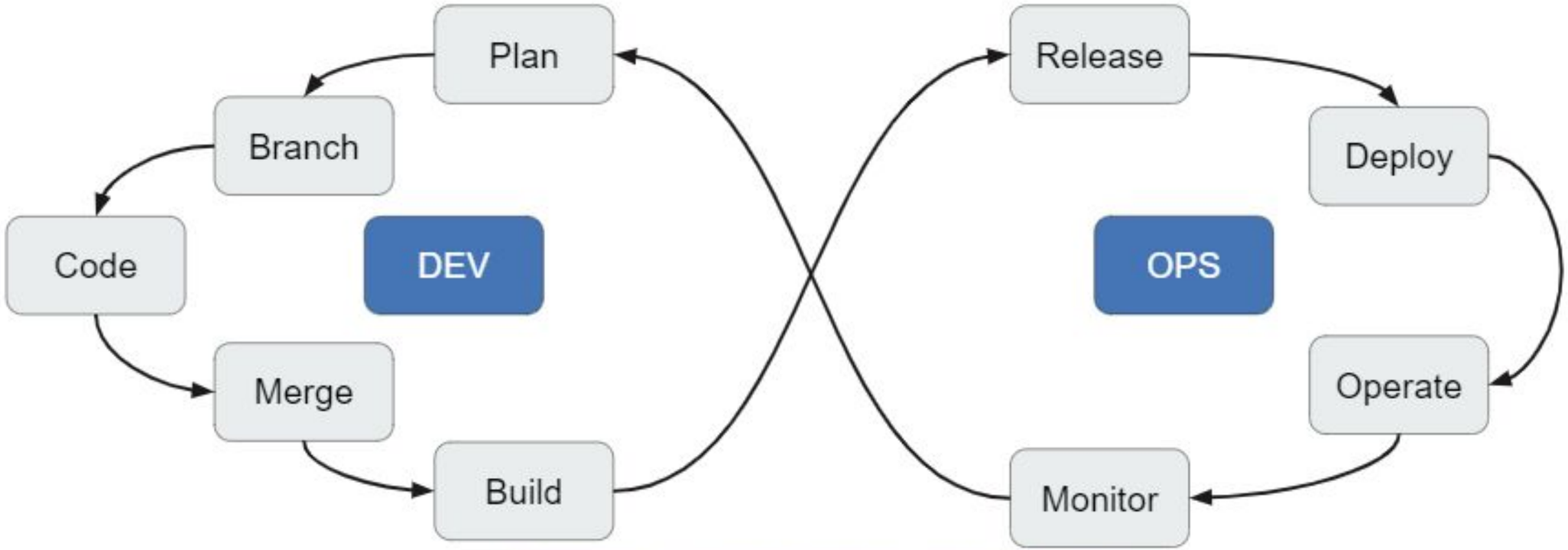
まず、自分で考えてたたき台をアウトプットしましょう。
それをみんなで叩いてブラッシュアップして磨きをかける方がリーズナブル

誰かが答えを持っている
わけじゃないんです。
あなたがテストをすることで
「知りたいこと」は何ですか？

ケーススタディ：freeeの状況

freeeはDevOps,アジャイルサイクル

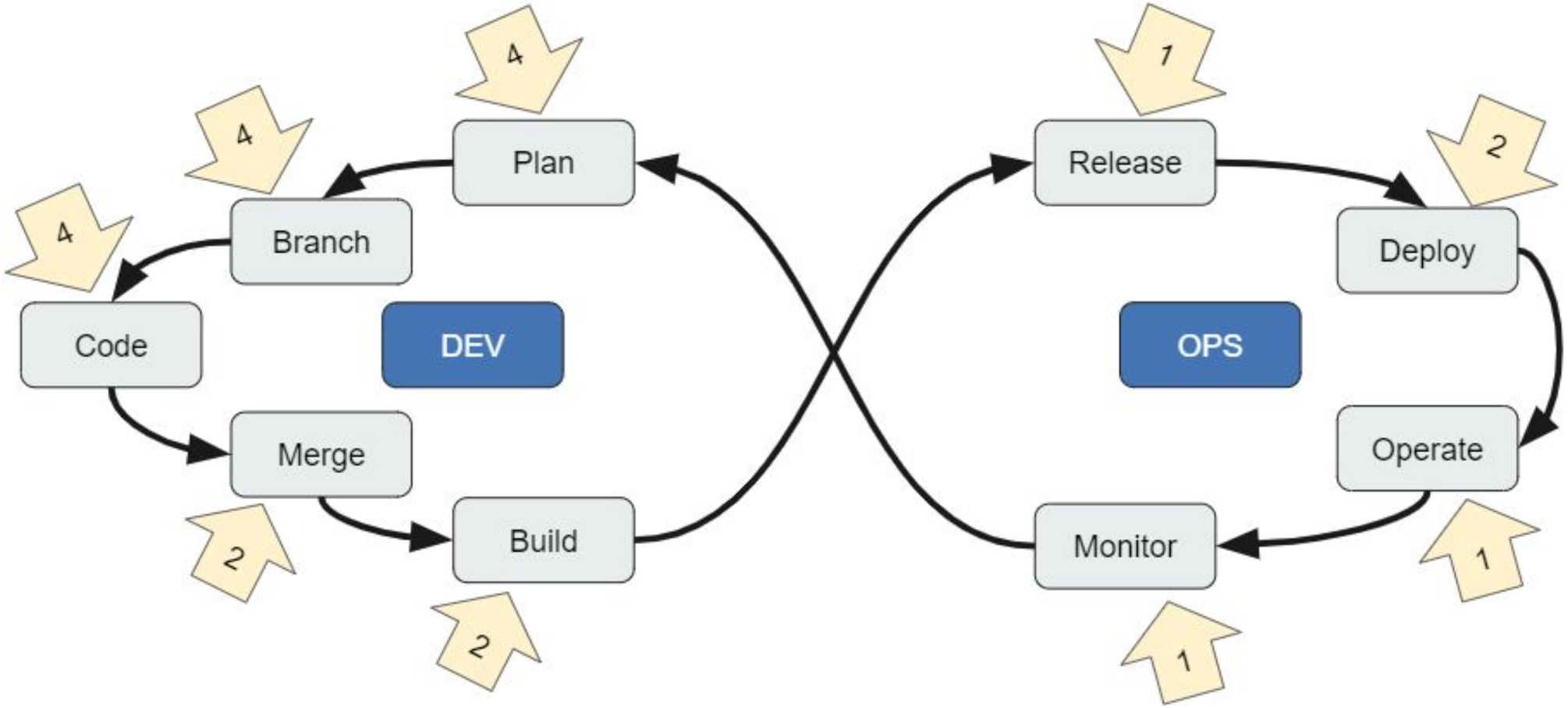
DevOps：プロダクトオーナー、開発、QA、IT運用、情報セキュリティが互いに助け合うだけでなく、組織全体の成功を目指して協力し開発・運用するソフトウェア/システムによってビジネスの価値をより高めるだけでなく、そのビジネスの価値をより確実に迅速にユーザーに届け続ける」という概念
freeeはPM,UX,Eng,QA,SRE,Security,Support,Salesが協力する世界



©dan ashby 「Continuois Testing in DevOps」 https://danashby.co.uk/2016/10/19/continuous-testing-in-devops/

DevOpsサイクルとテストの主目的

1.欠陥を摘出する
2.対象ソフトウェアの品質レベルが十分であることを確認する
3.意志決定のための情報を示す
4.欠陥の作りこみを防ぐ



QAエンジニアの主な関わり

主にDevOpsのうち、以下のフェーズでかかわる

- **Plan：リスク洗い出しで未然防止**
  - ➢ **未然防止したい**
- **Code：テストし、マージ前に欠陥を見つけて未然防止**
  - ➢ **欠陥を見つけて未然防止したい**
- **Merge：E2E for PullRequest環境でビルド前に品質レベルが十分か確認**
  - ➢ **自動テストで問題を検出したい**
- **Build：自動でE2Eを回し、Release前に品質レベルが十分か確認**
  - ➢ **コードをマージした後、意図しない影響が出ていないか検出したい**

それぞれのフェーズで知りたいことが違います。それぞれに工夫が必要

品質にこだわる理由

そもそも**取り扱うドメインが複雑かつ確実性が求められる**領域

- 会計freee：会計という膨大な情報と複雑な計算
- 人事労務freee：労務・給与計算という膨大な情報
- 申告freee：税務という膨大な情報と複雑な計算をビジネスユーザーが使う

かつ、**漏れるとヤバイ情報がたくさんある**のです。

- 会計freee：会計情報
- 人事労務freee：給与情報
- マイナンバー管理freee：マイナンバー
- 申告freee：税務情報

それを**100人以上のエンジニアが毎日更新する**
それを**クラウドで企業に利用されているため、止められない**システム

freeeで実践しているテスト分析

- 主に「会話」と「まとめ」＋あれば「ドキュメント」と「チケット」
  - アジャイルテストのキモはコミュニケーションだと感じている。
- できる限りメンバーの時間を割かないように集中して理解→ドキュメントにアウトプット→mtgで整理しなおす。という大まかな流れ。
  - 具体的には「ヒアリングシート」というものをチケット毎に作ったり
  - 単純にテスト分析した結果をまとめておいて、質問表を作ったりする
    - 手段はプロジェクトの対象・メンバー・状況によって変える
      - ようは理解してまとめられればいい。
    - あまりに情報が無い／時間が無い場合は探索的テストをすることとして、テストを実行しながらまとめることもある。
      - テストのやり方は柔軟性を持って変えている。

※モノによっては理解に時間がかかるものもあるのでスケジュールも頭に入れておく

freeeの中で自己流をどうしているかというと

個人的なイメージでいうと...

一番左側の4つを高速にぐるぐるしてからチャチャッと実行まで実施する感じ。

※自分たちがやるべきでない（自分たちがやると遅くなる/理解に時間がかかりすぎると判断した）場合は、カジュアルに他チームにお願いしたりして、全体をテストできるように動いたりもする

本日の流れ(再掲)

※だいたい折り返し
地点です。

- おっさん誰？
- テストの目的と本質の話
- リスクの識別の話（←ココ）
- テスト技法の必要性の話
- まとめ

Section

02

リスクの識別の話
ー不安（ヤバイ）と向き合う

「知りたいことは？」
というと色々考えてしまうので
「何が起きたらヤバイのか」
を考えてみましょう

やると便利なリスク識別

- 早めにフィードバックできる
- 早めにヤバイところ（不安なところ）が共有できる
- ヤバイところがわかるのでテストの方針・計画・戦略が立てやすくなる

やらなきゃ損です。
経験的にどんなプロセス・組織だろうとやった方がいい。
致命的な問題を減らすことができるのにとても有効です。
また「どこまでなら許容できるのか」という点も共有しやすくなります。

フィードバック早いとリーズナブル

- テストは欠陥があることは示せるが、欠陥がないことは示せない
  - 悪魔の証明だぜ
- 全数テストは不可能
  - 「全部テスト」は非現実的だぜ
- 初期テストで時間とコストを節約
  - 早めからやろうぜ
- 欠陥の偏在
  - 偏るんだぜ
- 殺虫剤のパラドックスにご用心
  - 同じ写真じゃ新しいところに気付けないぜ
- テストは状況次第
  - モノや状況によるんだぜ
- 「バグゼロ」の落とし穴
  - 大事なのはそこじゃあないぜ

フィードバックは早いほど良い

欠陥が埋め込まれてから摘出されるまでの期間が早ければ早いほどよい。

モブプロやペアプロの場合、コードを書いているそばから「タイポしているよ」などのフィードバックがもらえれば、秒で直せます。
※フィードバック速度を突き詰めるとTDDやコードを書く前のデザインレビューになる。
また実際に工夫して動かすべきか、動かさずに問題を検知すべきかも重要

レビューでの検出なら数秒～数分
ユニットテストでの検出も数分～数十分
システムテスト後の検出だと、モノによっては1日～1週間はかかる

Watts S. Humphrey「ソフトウェアでビジネスに勝つ」でも言及されてます

リスクの種類

- ❖ プロジェクトリスク
  - ➢ プロジェクトの遂行する際に影響を与えるリスク
- ❖ プロダクトリスク（品質リスク）
  - ➢ ソフトウェアやシステムで失敗する可能性のある領域

テスト設計のインプットになるのはプロダクトリスク。
プロダクトリスクを識別、分析する。

※テスト管理を行ううえではプロジェクトリスクも重要です。

リスクの分析

R-Map法の考え方が基本

発生頻度							
5	頻発する	C	B3	A1	A2	A3
4	しばしば発生する	C	B2	B3	A1	A2
3	時々発生する	C	B1	B2	B3	A1
2	起こりそうにない	C	C	B1	B2	B3
1	まず起こりえない	C	C	C	B1	B2
0	考えられない	C	C	C	C	C
		無傷	軽微	中程度	重大	致命的
		0	Ⅰ	Ⅱ	Ⅲ	Ⅳ

影響度

リスクの度合い＝発生可能性（頻度）×影響度

影響度をプロダクトごとに定義する

リスクを識別・分析するにはテスト対象分析が必要

「このテスト対象がどうなってしまうとヤバイのか」
を考えるためには、やはり**まずテスト対象の現状を知る必要がある**

- **どのような構造か**
  - ➢ **構造上ヤバイことになりそうなところを知る**
- **どのような機能を持つのか**
  - ➢ **機能上ヤバイことになりそうなところを知る**
- **どのようなユーザーが使うのか**
  - ➢ **ユーザーにとってヤバイことになりそうなところを知る**
- **どのような動きをするのか**
  - ➢ **どう動いてしまうとヤバイのかを知る**

などなど

そのテスト対象は
何が起きたらヤバイのか？
を考えるために
まず現状を知ることが必要

例:実際にfreeeで使ったリスク洗い出しシート

起こりうる問題・リスクのチェックリスト（ざっくりでOK）	ヤバさ	発生可能性予測
インポートの誤取り込みが発生する	小	中
エンドユーザー向け公開を見据える必要があるが、それを考慮出来ておらず、直前で慌ててしまう	中	
既存の機能が動作しなくなる	大	小
既存の機能の動作が変化する	大	小
勘定科目推測で財務データを活用していることについて市場からセキュリティ上の批判が来る	大	小
検知がループして終わらなくなる	中	小

そのテスト対象は
何が起きたらヤバイのか？
を考えると
「それが起こらないことを
知りたい」が出てくる

故ボーリス・バイザーのテスト道

「ソフトウェアテスト技法」より。テストにはいくつかフェーズがあり、
フェーズ3では「リスクを許容範囲にまで減らすこと」と定義されている。

フェーズ	内容
フェーズ4	テストは行動ではない。 大げさなテストをすることなく 品質の高いソフトウェアを作るための精神的な訓練である
フェーズ3	**テストの目的は、何かを証明することではなく、 プログラムが動かないことによって発生する 危険性をある許容範囲にまで減らすことである**
フェーズ2	テストの目的は、 ソフトウェアが動かないということを示すことにある
フェーズ1	テストの目的は、 ソフトウェアが動くことを示すことである
フェーズ0	テストとデバッグには何の差もない。 デバッグ以外にはテストには特別な目的はない

Section

03

テスト技法の必要性の話
ー技法って何がうれしいの？

「ヤバイことが起こらない
ことを知りたい」
→どう工夫してこれを解決する？

リスクはヤバイところから潰す

- **ヤバイところは早めに潰して安心したい**
- ヤバイところは往々にして複雑だったり中心だったりするので、早めに状況を知っておきたい
- ヤバイところは早い段階からたくさんテストして知っておきたい
- **テスト終了間際にリスク高い問題見つけたらプロジェクトが遅延しやすいので早めにやっておきたい**

※直接的な関連性は薄いので細かい説明を省略するが、
うまく工夫するためにはリソースを把握している必要があるため、
テスト要求分析（テストプロジェクトに関する要求の分析）をし
プロジェクトの制約条件なども識別し、建付けを検討しておく必要がある
※開発チーム・QAなどがテストで確認すべきか？他の対応はできないか？
を考えることも重要です。

リスクに基づくテスト設計

ヤバイからしっかりテストしておきたい

→しっかりって？全部？

→全部はムリ（テストの原則2：全数テストは不可能）

→納期もせまっているからホント少ししかできない

→一番ヤバイところからやろう

→数もいい感じにサンプリングしよう

（工夫＝設計しよう）

→さて、どうすればいいのか？

（再掲）
無限から
最適だと考えられる有限
を選定する

有限を選定するために使うテクニック

テスト技法はテクニック、ツールです
「いかに有意義なサンプリングをするか」
を効率的に行うために使います。

無限から
最適だと考えられる有限
を導き出すテクニック

どう工夫して、
知りたいことを知るのか

カメラでの撮影とテストは似ている(2回目)

カメラの撮影：被写体を見てコンセプトを考え、切り取る部分を考え、構図や絞り、などのテクニックを使う

いかに有意義な撮影（＝サンプリング）をするか
どういうアングルか、どういう光量で撮影するのか、それで何を引き出したいのか、
どう切り取るとイケてるのかなど、
テクニックを知らないとうまく撮れない（＝知れない）

テストで知る：テスト対象を知りコンセプトを考え、戦略を考え、テスト技法などのテクニックを使う

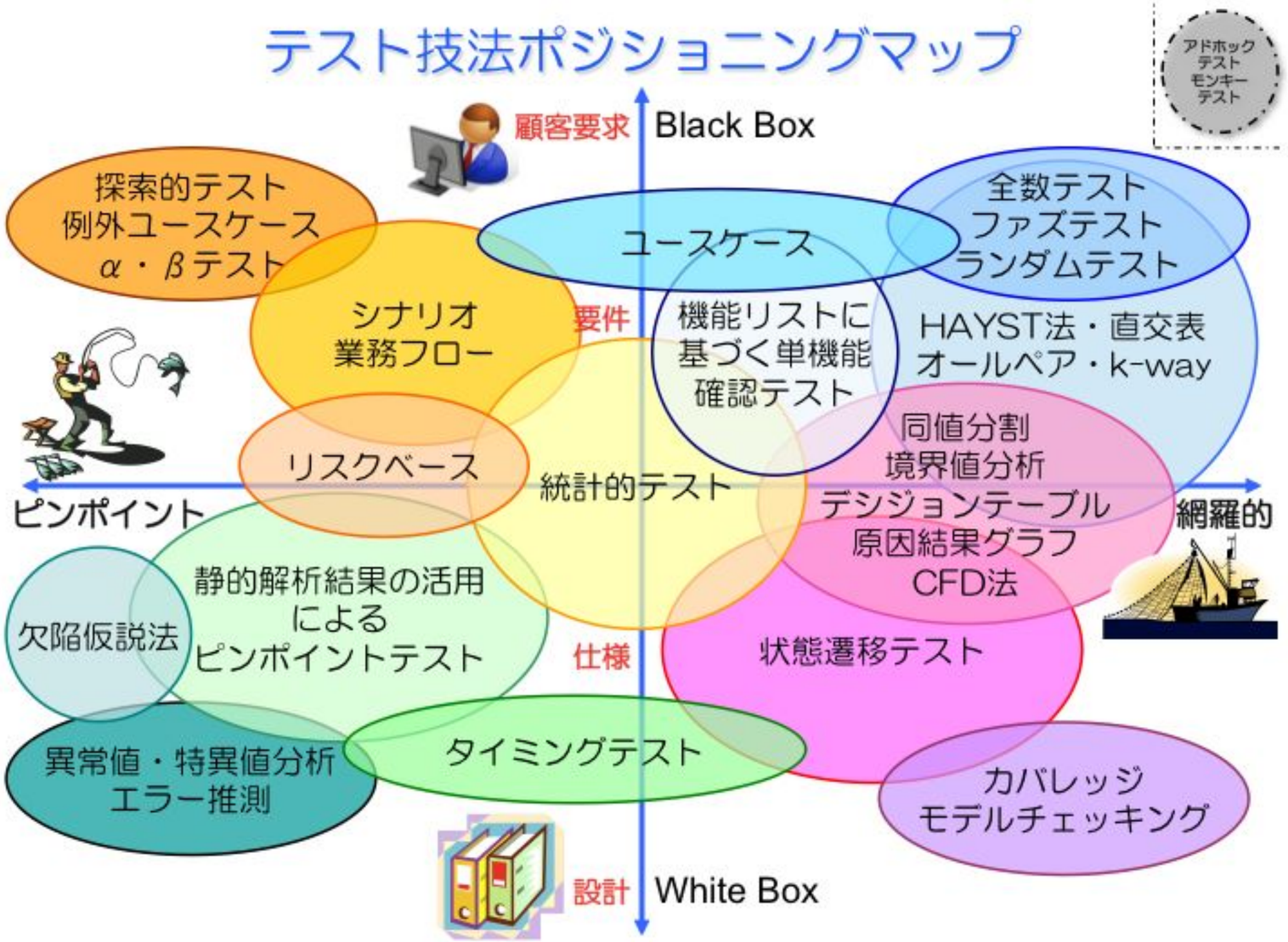
いかに有意義なサンプリングをするか
何を知るためにどのようなテスト戦略を考え、何を知りたいのか、どう工夫するとリーズナブルなのかなど、
テクニックを知らないとうまくテストできない

では、どういった
テクニックがあるのか？

テスト技法の種類

テスト技法ポジショニングマップ

いろいろテクニックあります※©秋山さん(@akiyama924)作成



テスト対象を知るために、どういう側面を狙うのか

- 網羅的にみるのかピンポイントに見るのか
  - 条件を整理して一通り見るのか？
  - ピンポイントに怪しいポイントを見るのか？
- 構造に着目するのか、仕様に着目するのか
  - 構造上問題が起こりやすそうなところを見るのか？
  - 仕様どおり動くことに重点を置いて見るのか？
- シナリオに着目するのか、単機能に着目するのか
  - シナリオ通り動かして不都合が無いかを見るのか？
  - 単機能が正しく動作しているか見るのか？

など。

皆さんの関わり方やSDLCのどこのテストなのか？を考えて、
狙いを決めましょう。

業界別、コヤマンがよく使った技法

- 組み込み系（統合～システムテスト）テストチームのマネージャー：
  - 同値分割／境界値分析／状態遷移テスト／タイミングテスト
    - ハードウェアを動かしたうえでの非同期処理が多いため、状態遷移やタイミングに着目することが多い
  - HAYST法（直交表を使った組み合わせテスト）
    - 単機能での動作確認は単体テスト時点でされていることが多かったため、統計により2機能間条件網羅を確認するために直交表を使った組み合わせテストをした。
      - 富士ゼロックス在籍当時はMatrix Testerというツールが使えたので、単機能の品質が十分な場合やパラメータをある程度網羅したい際には大変お世話になりました。

業界別、コヤマンがよく使った技法

- Web系（主にシステムテスト）開発チームのテスト担当：
  - 同値分割／境界値分析／デシジョンテーブル
    - 単体テストでは見切れない設定の組み合わせやバリデーションを見るのに有効
    - デシジョンテーブルにまとめる際に組み合わせ条件などを深く掘り下げることができるので、想定漏れなどに気付きやすい。
  - 探索的テスト
    - スピード重視をしたいときにヤバそうなところだけザッと見れる
    - エビデンスを残すほどでもない軽い変更のとき、ザッと見れる
    - 少し不安が残っているときに掘り下げるのに有効
  - シナリオ業務フロー／ユースケーステスト
    - 単体テスト等では気づかない操作時の不満に気付く
    - UXデザイナーが想定していない操作や違和感に気付く

基本的な考え方（同値分割法）

ズームイン・ズームアウト

抽象的に知りたいことを考えた（ズームアウト）のち、
具体的にどこにフォーカスすべきかを考える（ズームイン）

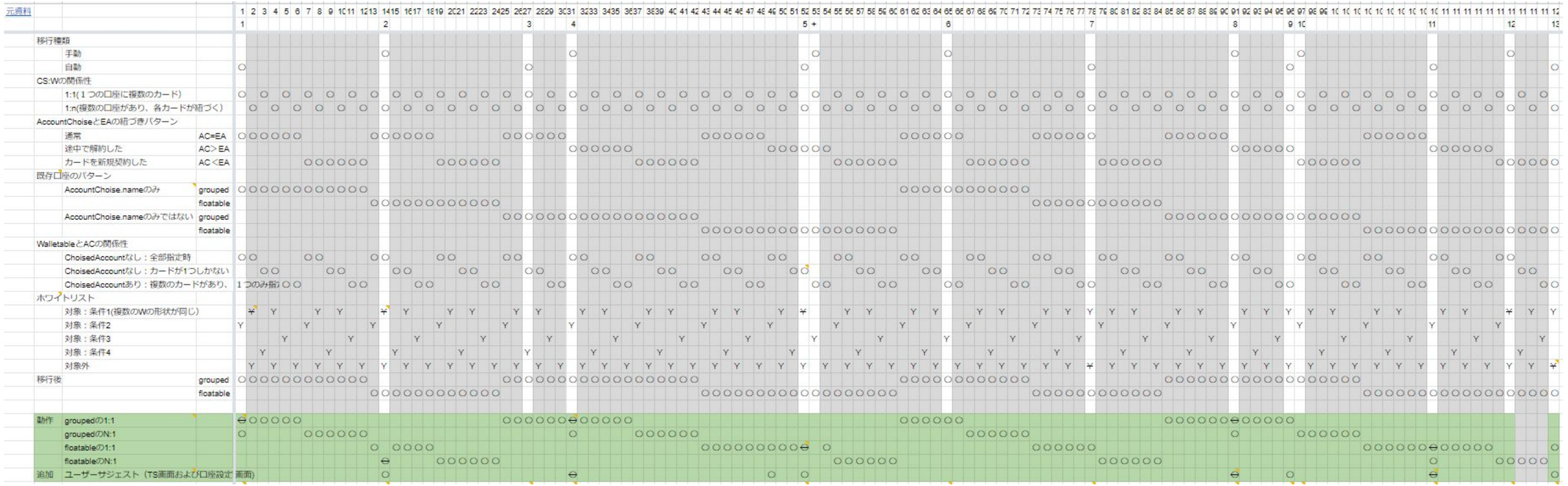
〇〇が問題ない/イケてるかどうか（ズームアウト）
→何をもって問題ない/イケてると判断するのかを定義する必要がある

- **ヒゲが整っている（ズームアウト）**
  - **抽象的で判断基準が人によって変わる。※これで良い場合もある。**
- **ヒゲの長さがこめかみから下顎まで4mmで統一されている（ズームイン）**
  - **具体的で判断基準は人によって変わらない。**
  - **剃っていたら整っていないのか？といった基準の妥当性に注意が必要**

ケーススタディ：freeeの場合

例:旧口座を新口座の仕組みに移行するテスト

リスクが高いと判断したので、ちゃんと条件出しをしたうえで、
デシジョンテーブル（拡張書式）で整理し、想定漏れを検出。
全部のパターンをやるのは条件が複雑かつ困難なので、ある程度論理的に絞って
実行（グレー部分は実施しないことにした）



例:計算が複雑すぎるので、ダブルチェック

とある税目のとある項目の税額計算の帳票プロジェクトにおいて、事前に洗い出したリスク
「計算が複雑すぎて、計算を間違いが発生する」（かなりヤバイ）
に対処するため、念のためダブルチェックをしましょう、とEng/PM/QAで合意。

条件を整理して検算をspreadsheet上に実装し、以下の問題を検出。

- 仕様漏れによる計算間違いを2件
- ユニットテストの条件漏れによる計算間違いを1件

実際の仕様と検算用の計算式

```
=IF(A2="確定申告A",
IF(A5="政令指令都市該当",
(IF(('申告書A'!D62+'申告書A'!D63+'申告書A'!D65)>=T1,T1,'申告書A'!D62+'申告書A'!D63+'申告書A'!D65)-2000)*0.08
+ROUNDUP((IF(('申告書A'!D62+'申告書A'!D63+'申告書A'!D65)=0,"",IF(('申告書A'!D62-2000)*R1)>=T5,T5,('申告書
A'!D62-2000)*R1))*4/5))),
(IF(('申告書A'!D62+'申告書A'!D63+'申告書A'!D65)>=T1,T1,'申告書A'!D62+'申告書A'!D63+'申告書A'!D65)-2000)*0.06
+ROUNDUP((IF(('申告書A'!D62+'申告書A'!D63+'申告書A'!D65)=0,"",IF(('申告書A'!D62-2000)*R1)>=T5,T5,('申告書
A'!D62-2000)*R1))*3/5)))),
IF(A5="政令指令都市該当",
(IF(('申告書B'!D89+'申告書B'!D90+'申告書B'!D92)>=IF(A3="第四表作成する",T3,T2),IF(A3="第四表作成する",T3,T2),'申
告書B'!D89+'申告書B'!D90+'申告書B'!D92)-2000)*0.08
+ROUNDUP((IF(('申告書B'!D89+'申告書B'!D90+'申告書B'!D92)=0,"",IF(('申告書B'!D89-2000)*R1)>=T5,T5,('申告書
B'!D89-2000)*R1))*4/5))),
(IF(('申告書B'!D89+'申告書B'!D90+'申告書B'!D92)>=IF(A3="第四表作成する",T3,T2),IF(A3="第四表作成する",T3,T2),'申
告書B'!D89+'申告書B'!D90+'申告書B'!D92)-2000)*0.06
+ROUNDUP((IF(('申告書B'!D89+'申告書B'!D90+'申告書B'!D92)=0,"",IF(('申告書B'!D89-2000)*R1)>=T5,T5,('申告書
B'!D89-2000)*R1))*3/5)))))
```

リスクを認識しておくと
設計ややり方、方針も決まります

Section

04

まとめ

この発表のゴール（再々掲）

テスト技法を使う前に認識すべき
大切なことを
皆さんが認識すること。

テストにおいては何を知りたいのか誰か
が教えてくれるとは限らない。

まず現状を知りましょう。
そして考えましょう。

誰かが答えを持っている
わけじゃないんです。
あなたがテストをすることで
「知りたいこと」は何ですか？

そのテスト対象は
何が起きたらヤバイのか？
を考えると
「それが起こらないことを
知りたい」が出てくる

どう工夫して、
知りたいことを知るのか

そもそも実証する必要があるのか？
についても考えましょう

テスト技法は、無限から最適だと考えられる有限を導き出すテクニック


スモールビジネスを、
世界の主役に。
freee

Section

05

Q&A

事前アンケートより 1

期限やリソースが限られている中、どのようにやるべきテストを抽出し計画するか

コヤマン回答

状況によるので一概に言えないのですが、コヤマンは以下のように動きます。

1.まずそもそも期限がヤバくならないように事前に調整します。

2.リソースは最近明らかに品質が悪くて、期限優先にしたところで市場トラブルが爆発的に発生しそうな場合、その説明をして期限延長交渉します。

3.とにかく短い期間で効率的なテストをしようとする場合は、洗い出したリスクの最も危険なものにフォーカスして探索的テストします。
※探索的テストにする理由はできる限りテスト実行を軽量化することで、修正するための期間を確保するためです（万が一欠陥を検出した場合、修正しないと安心してリリースできないため）。

事前アンケートより 2

AIを使用したテスト自動化に取り組んでいるが、テスト設計(テストシナリオ作成)にどうしても工数がかかってしまっている。
HMI仕様/状態遷移から項目(条件)等をグルーピング化しながらテスト項目の作成を行っているが、もう少し効率が良いテスト設計技法がないか聞いてみたい。

コヤマン回答 1/2

ちょっとよくわからなかったのですが、時間がかかっているのは「自動化するためのテスト設計・テストシナリオ」でしょうか？

また、その自動化しているテストって、ユニットテストのレベルですか？
それとも統合テスト/サービステストのレベル(APIのレベル)ですか？
それともシステムテスト(E2Eテスト)のレベルですか？

「HMI仕様や状態遷移から項目の作成をしている」ということは、システムテスト(E2Eテスト)、ですかね？と仮定してコメントします。

E2Eを流すときは、まず一番リスクが高い部分、あるいは最も基本的なシナリオのみにしています。※拡張はしていますが

コヤマン回答 2/2

なぜ自動テストをやっておきたいんでしょう？
常に問題が無い状態で安心してリリースしたいから、ではないかと思っているのですが、その「安心」をするためにコストが高く実行も遅いシステムテスト(E2E)を使うのは、少し非効率なのかな？と認識しています。
常に状態遷移の1スイッチカバレッジを全て確認したいものですか？
基本的な遷移シナリオ＋ある程度ヤバそうな遷移シナリオの2シナリオくらいでよくないですかね？
ちなみにユニットテストであれば、ある程度同値分割と境界値分析をした状態でテストを準備するのが良いかな、と思います。
統合テストの場合は、他のモジュールとの結合部分においてのパターン網羅はある程度必要と考えていますので、組み合わせテストでざっと条件を出すことをオススメします。直交表の利用は4,8,16,32,64,といったある程度のボリュームで落ち着く＋２因子間網羅ができるのでオススメです。

事前アンケートより 3

テスト技法の導入効果を
第三者に示すことができるデータがあったら紹介して欲しい
(テスト技法を導入することで
バグが何%減った等)

コヤマン回答

弊社では「テスト技法の導入効果」を第三者に示す必要があまりないのでデータが無いです。すみません。

過去に使った経験があるのは以下ですが、実データは手元には無いです。
・テスト設計充当工数とDDP(テスト中の不具合検出率)の相関
→テスト設計に時間かければ不具合検出率が上がるのでほぼ正の相関ができる

・前テスト設計充当工数：現テスト設計充当工数
・前テスト密度　　：現テスト密度（規模あたりのテスト数）
・前バグ密度　　：現バグ密度（規模あたりのバグ密度）
→テスト設計したときとしないときなどで、テストの数が変わらないのにバグの検出率が上がることが見える化できる

事前アンケートより 4

テストについて、
自組織内での啓蒙活動は
実施されていますか？
具体的な内容がございましたら
ご教示ください。

コヤマン回答

啓蒙活動というと「啓蒙」しているかどうかはわからないのですが、
私のロールが「QAアニキ」なので基本的にはお悩み相談に乗ったりするんですが、以下の活動は担当のテストプロジェクトと別に実施しています。

- 1on1やweeklyにおけるQAメンバーへのナレッジシェア
- QA実施基準の制定
- 簡単なルールの策定
- アドベントカレンダーの執筆
- 社内勉強会での発表
- 品質特性のワークショップ開催
- 品質関連の新卒研修の実施
- ゲリラ的meeting参加
- テスト関連情報の社内発信
- 成果発表会での発表

事前アンケートより 5

テスト技法を他メンバー(Dev,Opt)に伝えればいいか？
またどのように
興味を持たせればいいか？

コヤマン回答

どのようにDevやOpsに伝えればいいか？ってことですか？

であれば「少ない手数で効果を得るため」というお話になりますかねぇ。。

時々網羅至上主義みたいな人がいるんですが「ホントにヤバイところだけの芽を摘むのにちゃんと脳みそ使うっていう方がクールじゃないです？」っていうことを私はやんわりと伝えます。（リーズナブルです！というフレーズ便利）

興味を持ってもらうには、どうすればいいか？というと結構難しいですね。
個人的にテスト設計というものはほとんどツールと同じような感覚なので、
いいツールあるよ！こんな使い方できるよ！ってアウトプットするくらいですかね。
要望が出てくれば勉強会開く、とかできればなおよいかと！

事前アンケートより 6

freeeのテスト自動化に関する
戦略について、
少し紹介して頂けると嬉しいです。
ユニットテストとインテグレーションテスト
の割合やそれをいつ誰がやるのか

コヤマン回答

今のところ、ドドーン！と戦略があるわけではなく、これから作っていくというか、見直ししていく、というフェーズになっています。

今はほとんどServiceTest/IntegrationTestの自動化はしておらず、
ココの強化はいくらか必要、という認識を持っています。

但しテストピラミッドにこだわりがあるわけでもなく、
アイスクリームコーンにならなければいいよね、くらいです。
基本的に「常に安心したいクリティカルなもの」をUnitTestでEngが中心となって安心を担保し、
「基本的な操作」や「退行したらマズイポイント」をE2E TestでQA/SETが中心となって安心を担保する形にしています。
ちなみにQAもSETもE2Eをメンテしたり追加したりします。

事前アンケートより 7

「テストが楽になる」ってどういう状況をイメージされていますか？

コヤマン回答 1/2

個人的には毎回「テストって大変だなぁ」と感じてしまうので、楽に感じたことは無いのですが、いくつかのプロジェクトを並行していたとしても
今日お話しした「テストの目的」「テストの本質」「今のテスト対象の状況」「何を知りたいのか」「何がヤバイのか」に加えて「実際に適用するテスト設計技法」を知っていると、以下のようなメリットがあると考えています。

・闇雲にではなく、ある程度の根拠を持ってテストを減らすことができる
・時間が無い場合でも、無限から最適だと考えられるテストをある程度の根拠をもってチョイスできるようになるため、迷いが無くなる

またカメラでの撮影に例えるのですが、サンセットやマジックアワーなど時間がシビアな時に「良い一枚」を撮影するには、ある程度の戦略とテクニックが必要ですよね。

コヤマン回答 2/2

良いテストは
「早く・安く・巧い」
実行結果が早くわかり
低コストで実施できて
かつ効果的なもの