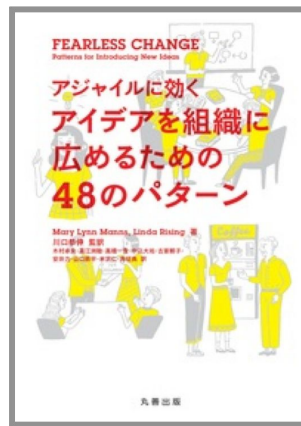


プログラミング未経験者を含むQA組織で、 コードベースE2E自動テストをどう始め、 どう続けるか

Teppei YAMAGUCHI @ JaSST'26 Niigata

自己紹介 Teppei YAMAGUCHI

- 株式会社LayerX 所属 / 個人事業主
 - テスター / コーチ / コンサルタント / プログラマー
 - **現場でテストコードのレビューをし続けています**
- 翻訳
 - 『ソフトウェアテストをカイゼンする50のアイデア』
 - 『Fearless Change』
- コミュニティワーク
 - Regional Scrum Gathering Tokyo 実行委員
 - SQiP 研究会 研究コース4「アジャイルと品質」アドバイザー
 - テスト自動化研究会 お世話係



「みんなで自動テストを書こう」では、
普及できませんでした

今日お話しすること

- 「みんなで自動テストを書こう」をやめて、まず最初の一人が型を作りきる理由
- 型ができてからの巻き込み方と、レビューを最初の一人に集約する進め方

コードベースE2E自動テスト普及の課題

【以前の現場】 「1本でいいから書いてよ」と頼んだ

- テストフレームワークは用意した。あとは書くだけのつもりだった
- 声をかけたのは開発チームの3～4名。「やります」と言ってもらえた
- 2ヶ月働きかけて、増えたテストは2～3本だった
- 続かなかったし、広がりもしなかった

【以前の現場】当時の私は「みんな忙しいんだろう」と思っていた

- 書かない側に理由がある、と思っていた
 - 実際、忙しかったのは本当だった
- 2ヶ月かけて私が変わえようとしたのは、相手のやる気だった
- いま振り返ると、変えるべきだったのは、そこではなかった

課題① テストを1本書くまでの判断が多い

テストを1本書くまでに必要な判断の一部

- このテストは、どのファイルに置く？
- テストケースの名前は？ メソッドの名前は？
- この処理は共通化する？ しない？
- 待ち方は？ ロケータはどれを選ぶ？
- アサーションはどの粒度で書く？

テストは書けても、判断の基準は一人で決めにくい

- プログラミングが分かる人ならテストを書けるのか？
 - テストは書ける。ただし、前に例示した判断には一つ一つ自分で答えを出すことになる
- 「みんなが使うテストの判断を、自分が決めていいのか」が分からない

課題② テストケース固有の知識がない

テストケースは読みやすく、とメンバーに伝えても、返ってくる問いは…

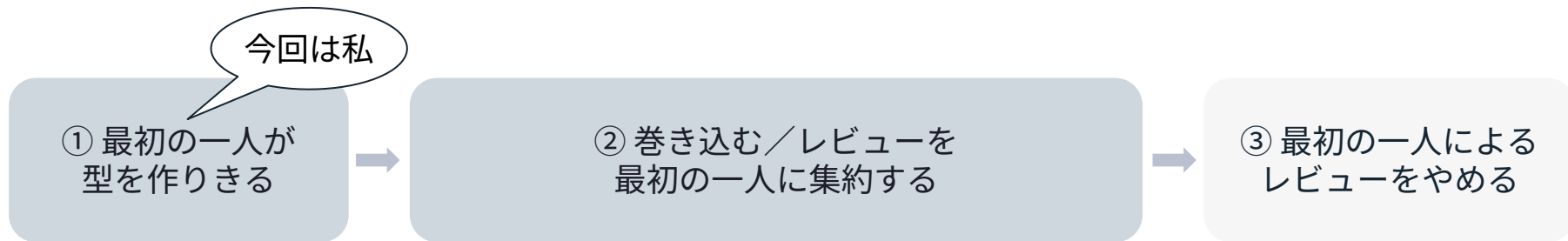
- テストケースとして、どうなっていたらいいのか？
- そもそもファイルのはじめから何を書いたら良いのか？
- テストケース内でのログインはどう書いたら良いのか？

AIでテストを書けるだけでは、数年続く自動テストにはならない

- 数年使い続けると、パッと作ったテストは、直す箇所が分からなくなる
- AIがテストを書けても、テストのための判断基準がないことは変わらない
- むしろ、書ける量が増えるぶん、直せなくなるのが早い

普及させるためには工夫が必要

全体の流れ



型ができたら

レビュー指摘がなくなったら

- ① 判断回数を減らすガイドやコーディングルール、サンプルコードで構成する型を最初の一人が作りきる
- ② 巻き込みつつ、人により判断がぶれないように最初の一人にレビューを集約する
- ③ レビュー指摘がなくなったら、最初の一人によるレビューをやめる

最初の一人が型を作りきる、とは何をするのか

次に書く人が、コピペ&モディファイでテストケースを書けるようにする

- 自動テスト作成ガイドとコーディングルールを整備する
- 主要なフローを含むテストケースを、サンプルコードとして一通り、最初の一人が作りきる
- 完璧を目指さず、書く人が手を動かし始められるところまで作る

例：「保存」ボタンをクリックしたい

- 「保存」ボタン要素を、どの種類のロケーターで定義しますか？
 - A. data-testid
 - B. 表示されている「保存」の文字
 - C. role とアクセシブルネーム
 - D. CSS セレクタ (.btn-primary)
 - E. XPath (フォーム内の3番目のボタン)
- どれでも動く
 - ただし、UI の変更には強い種類と、どの要素でも使える種類は異なる
 - 要素に応じて選ぶ

ガイドやコーディングルールとして決めること (1)

要素は、極力、上位の種類のロケーターで定義する

- A. テスト用に割り振ったID (data-testid、getByTestId)
- B. 表示されているテキスト (getByText)
- C. role とアクセシブルネーム (getByRole)
- D. class / id / name、タグ名との組み合わせ (.btn-primary など)
- E. XPath による何番目や列指定などの構造情報

※ Playwright 公式は role を先に勧める。

A を最優先にしたのは、自分たちで管理でき、文言やスタイルの変更では壊れないから

例：申請プロダクトの requests/new 画面の要素定義をしたい

- この画面の要素定義、どこに置いて何と呼びますか？
 - A pages/requestNew.ts に RequestNewPage クラス
 - B pages/workflow/requests/new.ts に WorkflowRequestsNew クラス
 - C pages/申請/新規作成.ts に CreateRequest クラス
 - D specs/ のテストケースと同じファイルの中に書く
- どれでも動く
 - ただし、次に書く人とAIが、同じ答えにたどり着けるか

ガイドやコーディングルールとして決めること (2)

可能な限り、名前は「決める」のではなく「**決まる**」ようにする

- クラス名とファイル名は、プロダクト名かドメイン名と URL の階層で決まる
 - **URL** {申請プロダクトドメイン}/requests/new
 - **ファイル** pages/workflow/requests/new.ts
 - **クラス** WorkflowRequestsNew

※「workflow」は「申請」プロダクトを指す
- 要素名は、見た目にあわせて名付ける。input タグでも、ボタンなら ~Button

ガイドやコーディングルールとして決めること (3)

ファイルの保存場所や依存関係も、先に決めておく

- ディレクトリ構成

```
root/  
  pages/                // 画面の要素定義と、1画面内の操作 (Page Object Model)  
  shared_components/    // サイドバーなど、プロダクトごとに共通の部品  
  steps/                // ページをまたぐ一連の処理  
  specs/                // E2Eテストケース  
  (ほか testdata/ utils/ download/)
```

- steps と specs は、直下がプロダクト名 (doc-issue / id / payer ...)
- pages は、直下がプロダクト名かドメイン名。その下は URL の階層

- 参照の向きを固定する specs → steps → pages / specs → pages

ルールを全部守ると、こうなる

```
test("パスワードが未入力の場合ログインに失敗すること ", async ({ page }) => {  
  // 期待値：パスワードが未入力だとログインできない  
  const idIndex = new IdIndex(page);  
  const idAuthSignIn = new IdAuthSignIn(page);  
  
  await idAuthSignIn.load();  
  // 固定IDで失敗が続くとロック時のメッセージになるため、ランダムな IDで試みる  
  await idAuthSignIn.usernamePlaceholder.fill(`hoge+${utilsGenerateTimestampedUniqueId()}@XXXXXX`);  
  await idAuthSignIn.passwordPlaceholder.fill("");  
  await idAuthSignIn.loginButton.click();  
  
  await expect(idAuthSignIn.alertToast).toContainText("ユーザー名とパスワードを入力して下さい");  
  await expect(idIndex.globalNavigationBar.productDropdown).toBeHidden();  
});
```

IdAuthSignIn：プロダクト名 (id) と URL (/auth/sign_in) の階層から決まる
loginButton：ボタンだから ~Button expect ×2：アサーションは spec に残す

最初の一人が作りきるテストケースに含めていること

ファイルの書きはじめや、基本的な操作が書かれたファイルを用意する
例)

- ファイル内で実装しているテストケースのテスト設計書へのリンク
- テストアカウントの定義
- テストケース内でのログイン方法
- テストケース内での主要フローの実装・利用
- テストケース内でのテストにおける期待値の表現

サンプルコードの上部

```
import { IdAuthSignIn } from "@pages/id/auth/sign_in";
import { IdIndex } from "@pages/id/index";
import { test, expect } from "@playwright/test";
import { utilsGenerateTimestampedUniqueId } from "@utils/generateTimestampedUniqueId";

// テストケース表：「[ID] テストケース表」の「【ID】認証」シート
// パスワードが未入力の場合ログインに失敗すること
// https://docs.google.com/spreadsheets/d/XXX/edit?gid=YYY#gid=ZZZ&range=xx:xx

test("パスワードが未入力の場合ログインに失敗すること ", async ({ page }) => {

  【以下略】
```

型ができてから、
QA組織のメンバーをどう巻き込んだか

ペアプロは、セットアップから一緒にやる

- ガイドを書いても、実行環境のセットアップで止まる人がいる
- テストケース1本を一緒に書いて、ガイドの使い方を見せる
- 分かりにくかったガイドは、その場で直す

やってみせる会で、考えていることを実況する

- 「なぜこの順番で書くか」を、声に出しながら書く
- 「こうしておくと後が楽になる」を、その場で共有する
- 同じところでハマっている人を、まとめて助けられる
- エディタの使い方なども共有できる

テストケース群の品質と一貫性を維持する

- テストケースが少ないうちは、書き方のばらつきが発生しやすい
- 書き方がばらつくと、次にテストを書く人の判断が増える
 - そのため、レビューは最初の一人に集約し、ばらつきを減らすようにする
- レビューアーの負荷は上がる
 - この期間は引き受ける、と最初の一人が先に決めておく

レビュー指摘を、AI向けのガードレールに書く

- 当初狙ってはいなかったが、AIに問題を繰り返させないために、レビュー指摘をAI向けのガードレールに書いている
- ガードレールとして書いていること（一部）
 - 既存のテストと同じ形で書き、新しい書き方を持ち込まない
 - ロケーターはUI上の意味で狭く定義する
 - アサーションはspecに残す。意味のないアサーションを作らない
 - タイムアウト、ループ、リトライを雑に書かない
 - シナリオは、共通化（DRY）より業務の読みやすさを優先する
- これらはレビューをしながら増やしている

レビュー指摘がなくなったら、最初の一人によるレビューをやめる

- 同じ指摘が2～3回出たら、ガイドやガードレールに書く
 - 書いた分、次のレビューで言うことは減る
- 最初の一人によるレビューをやめる流れ
 1. 直接、最初の一人がレビューする
 2. 慣れてきたら、まず書く人同士でレビューし、最初の一人は最後に見る
 3. 2. で書いた人同士でも、最初の一人によるレビューでも、レビュー指摘がなくなり安定したら、最初の一人によるレビューをやめる
- 書く人同士のレビューは、レビューの練習でもある
 - 指摘された人と指摘した人の両方が学ぶ

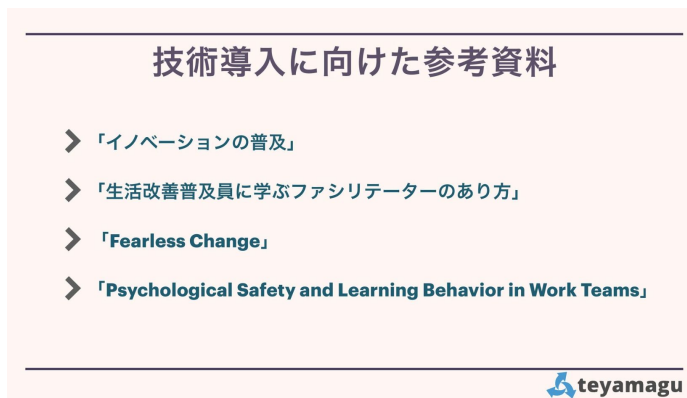
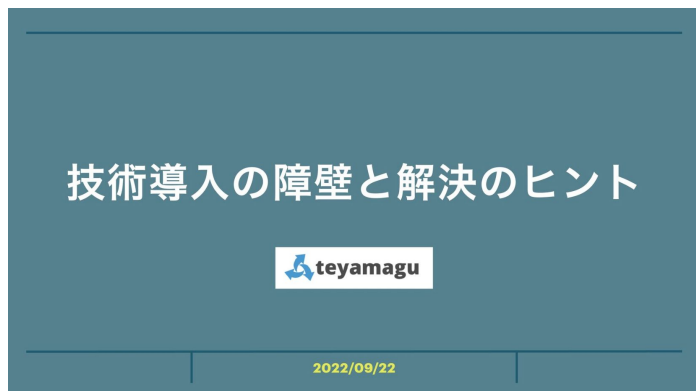
状況に応じて、最初の一人によるレビューを再開する

- 自動テスト作成経験のある人から順に、半年ほどで最初の一人による直接的なレビューをやめた
- ところが最近、AIで書く量が増え、ガードレールが追いつかない場面が出てきたため、最初の一人によるレビューを再開している

(おまけ) 巻き込みや普及拡大に関しては戦略的な視点もある

- ここで紹介したものは、自動テストに特化した施策レベルのもの
- より大きな範囲での巻き込みや普及を考えると戦略的なものも必要になる
- 参考：「技術導入の障壁と解決のヒント」

<https://speakerdeck.com/teyamagu/ideas-for-introducing-new-methods>



巻き込んだ結果、何が変わったか

自動テストが、みんなのものになった

型を作りきってから巻き込んで得られた効能

- **効能①** 書きはじめるまでの迷いが減った
- **効能②** プログラミング未経験のメンバーも、サンプルコードとガイドを見ながら書けるようになった
- **効能③** レビュー指摘が、人向けのガイドやAI向けのガードレールとして残るようになった

とはいえ、まだ途中で

- 書いた人は増えた。ただし、月ごとに書く人には波がある
- 自動テストでカバーしたい範囲は、プロダクトの成長や増加とともに広がり続ける
- レビューを完全になくせる段階には、まだ至っていない



まとめ

どう始め、どう続けるか

- 始め方
 - 「みんなで自動テストを書こう」をやめ、まず最初の一人が型を作りきる
 - 型は、判断回数を減らすガイドやコーディングルール、サンプルコード
- 続け方
 - 巻き込むのは型ができてから。レビューは最初の一人に集約する
 - レビュー指摘がなくなったら、最初の一人によるレビューをやめる
 - ただし、状況に応じて再開することもある

明日からできる第一歩

- 自分がやっているテストの書き方を、1枚のドキュメントとして作成する
- 他者にとって参考になるテストケースを、最初の一人として1本書く
- 誰かとペアで、テストケースを1本書く