JaSST'21 Tokyo

Autify を使った自動テストにおける属人化解消について

株式会社 チームスピリット　生井 龍聖

Agenda

1. はじめに
2. 解決したい課題と過去の失敗例
3. **Autify**を使ったアプローチ
   - NoCode
   - LowCode
   - 自動化の検討
4. まとめ

Section1

はじめに

Ryusei Namai

TeamSpirit Inc.
QA Team Manager
JSTQB Test Analyst & Test Manager

riririusei99

https://www.linkedin.com/in/ryusei-longsheng/

https://recruit.teamspirit.com/job/

開発体制について

チームスピリットでは2017年からスクラムでの開発を行っており、

現在、製品ごとの各サブシステムごとのスクラムチームで開発を行っている。

（チーム数は6。そのうち2つはシンガポール拠点で開発を行っている）

QAエンジニアは「開発チームの一員」として携わっており、

ストーリーテストの他にも、バグレポートの作成や自動テストの作成・メンテを行う。

専任の自動テストエンジニアは所属していない状態。

はじめに

発表について

自動テスト

運用の仕方

自動テストの導入についての事例発表です

はじめに

解決したいテーマ

属人化

なぜ起こるのかというのを考えました

Section2

解決したい課題と過去の失敗例

第一次 E2E自動テスト(Java)

START

2017 Nov
自動テストの検討・計画
ローカル環境での実施
スモークテストの実装

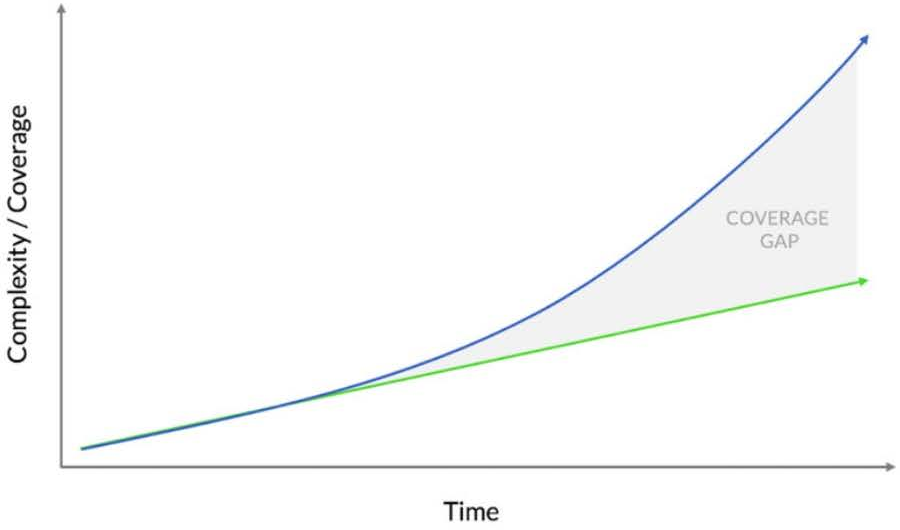
2018 Mar
開発サーバ構築
BEエンドエンジニアと協力

2018 Sep
BrowserStackでクロスブラウザ対応
パッケージ版に対してテストが可能に

2019 Aug
メンテナンスに課題を抱え始める

バックエンドエンジニアと二人で開発＆メンテナンス

プロダクトの成長



Features

Complexity increases exponentially as new features and states interact with existing features

Tests

Test coverage grows linearly because they can only be added one at a time

Source: AI for Software Testing, Jason Arbon, CEO, test.ai
https://www.linkedin.com/pulse/ai-software-testing-jason-arbon/

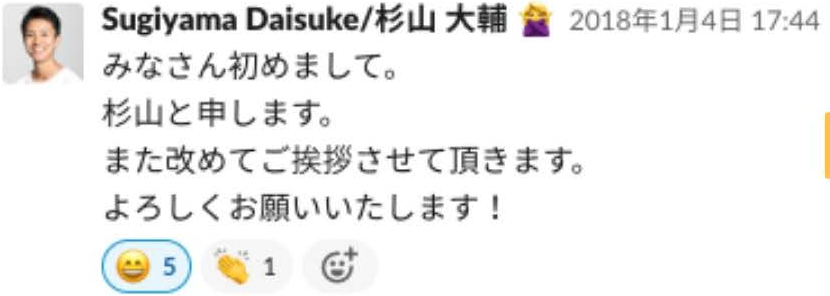
機能追加などのメンテナンス頻度が高まった

時間軸の変化で起こること

- メンバーの成長
  新しいことができるようになる
- 新しいスクラムチームの発足
  メンテナンスするにも修正内容のキャッチアップが必要
- メンバーの異動・昇進・退職など
  人事発令に対する適応力の低さが露見した

プロダクトの変化の他にも個人・チーム・ビジネス全体にも変化が起こる

時間の変化



時間の変化はチームにも変化を与える

原因を考える

二つの要素が属人化に影響していると考えた

Section3

Autifyを使ったアプローチ

Autifyとは

画面録画

シナリオ生成

回帰実行

解決策を考える

この二つを満たす手段としてAutifyの利用を開始した

Before

BE + QA

実装・メンテナンス

QA

自動化の検討

After

はじめて

NoCode

チームでつくる

経験者

LowCode

大事な部分に集中する

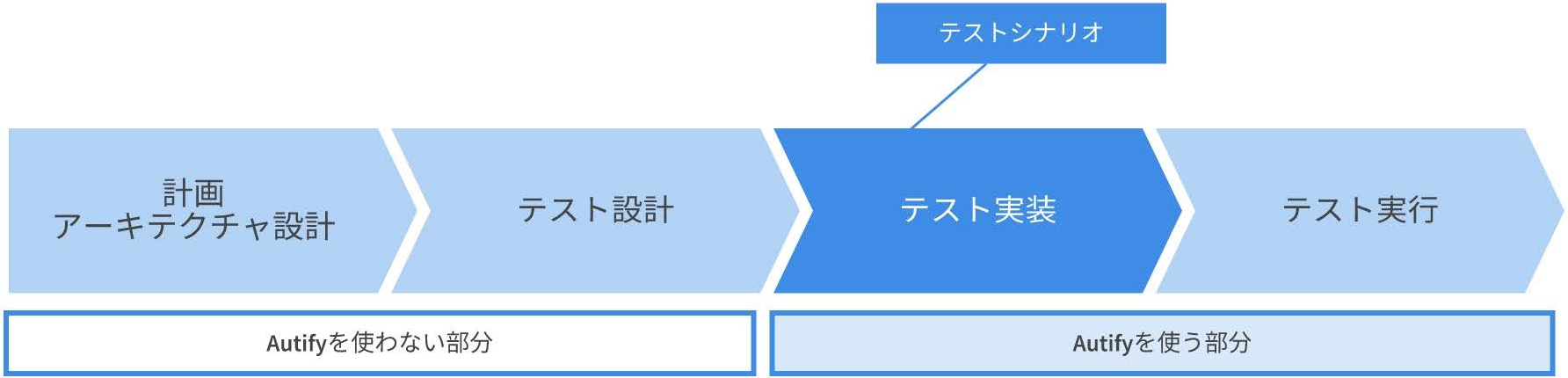
運営

自動化の検討

チームで取り組めるようにする

NoCode開発

Autifyをシンプルに使ってテストケースを増やす



NoCode開発

Autifyのシナリオ作成機能を使って自動テストを作成
必要なスキルのハードルを下げることで、チームでつくる

STEP #1
- Autifyのレクチャー
- チーム内ルールを読んでもらう

STEP #2
- チームのバックログに組み込む
- テストシナリオを準備しておく

STEP #3
- Autifyでシナリオ作成する
- 成功したらテストプランに組み込む

完成！
- 並列実行で失敗する場合は別タスク

NoCode開発の利点

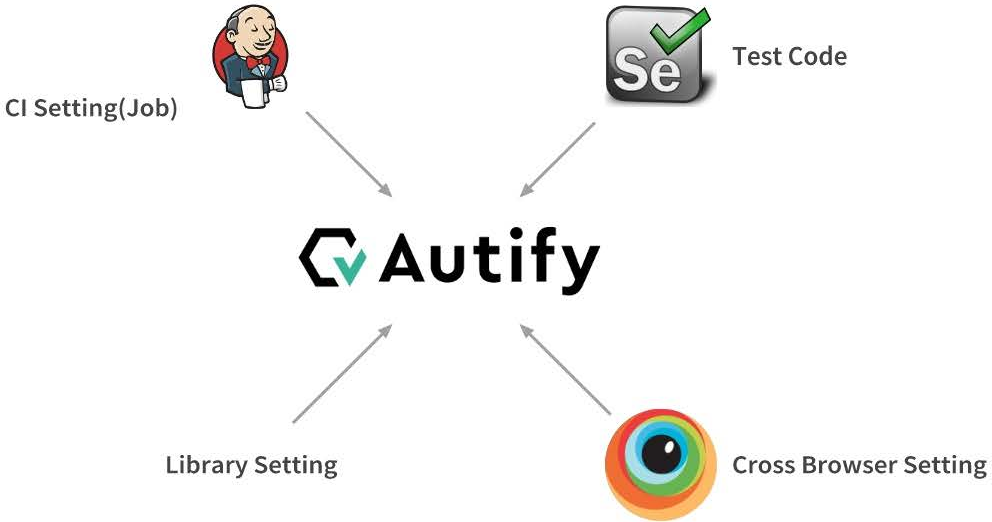
自動化のハードルを下げることで、自動テストに関わる人を大幅に増やすことができた

- 作業者が増えたことで開発できる総量が増えた
  NoCodeの作業だけで回帰実行できるテストシナリオも多い
- 環境構築の手間が不要
  失敗の原因を実行環境による違いなどから除外でき、
  Chromeプラグインをインストールすればすぐに着手できる
- 設定部分がAutifyに集約され理解しやすくなった

開発チーム20人にAutifyを利用してもらっている

NoCode開発の利点

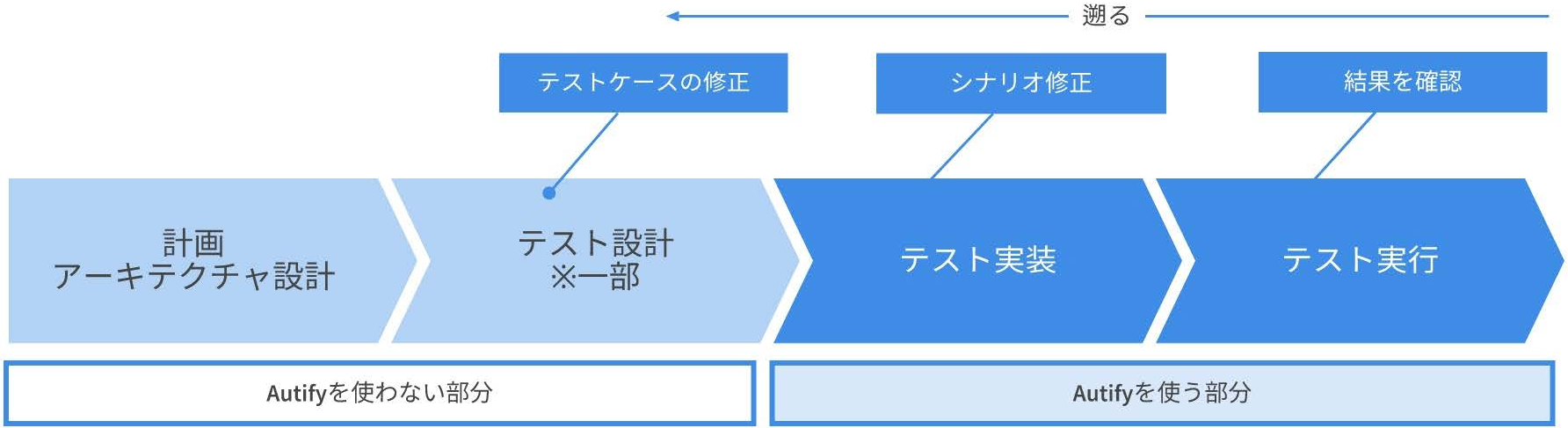
設定部分が集約され、理解しやすくなった

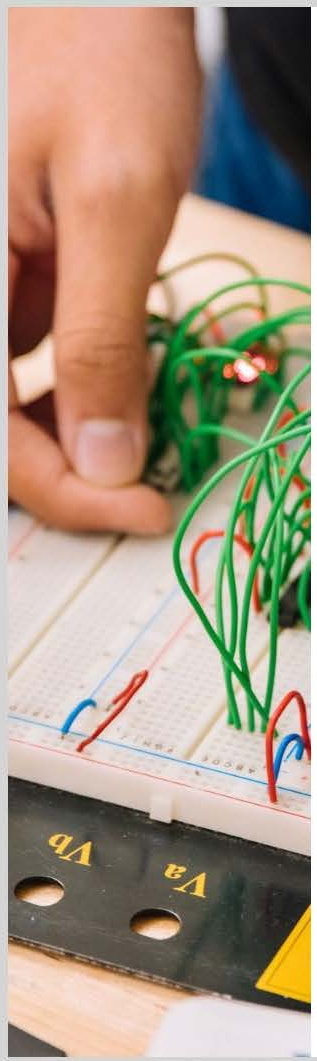


シンプルになり教えやすくなった

LowCode開発

安定した自動テストを作る





LowCode開発

自動化経験のあるメンバーを中心にNoCodeでできない部分をサポートすることで、
特定のメンバーしかできない仕事に注力できる。

STEP #1
- 回帰テストの結果を確認
- メンテナンスチケットを作成する

STEP #2
- 落ちている失敗しているシナリオを修正
- JSステップ・待機ステップ等を追加

STEP #3
- 成功したらテストプランに組み込み
- 関係するテストを全て実施する

完成！

LowCode開発の利点

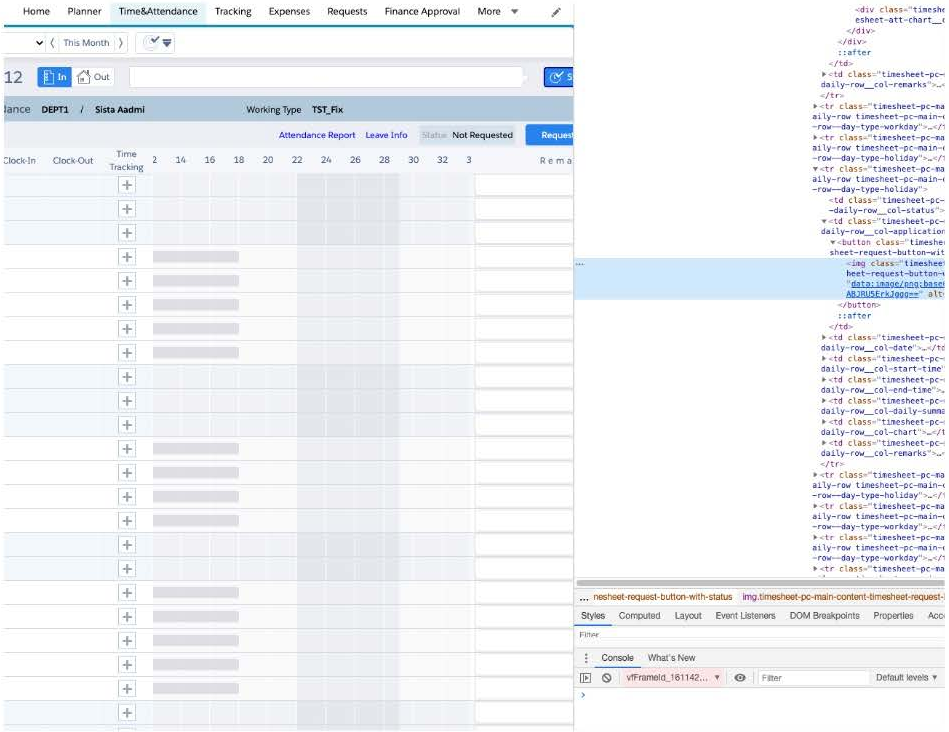
実装にまつわるカスタマイズの部分を機能として提供しているので、その部分の工数を肩代わりしてくれる。

やりたいこと	Autifyだと	自前でやると（例）
並列実行	プロジェクト設定から設定	フレームワークで並列設定
回帰実行	テストプラン作成時に “定期実行”から選択	CIに組み込む & 環境作成
クロスブラウザでの実行	テストプラン作成時に “実行環境”から選択	BrowserStackと連携 or 各Driverのメンテ
通知	Incoming WebHookを使って連携	ジョブを作成 & CI側で連携
レポーティング	“テスト結果”から閲覧	ライブラリ選定 & CIg側で設定
トラブルシューティング	サポートあり & snapshotが見れる	SnapShot取得処理を書く & ログをみる
独自のwait処理等の実装	なし	プロダクト毎に開発が必要
特殊なことをやろうとする	JSステップに書く	テストコードに書く

専任のいない環境にとっては十分な機能提供

LowCode開発の利点

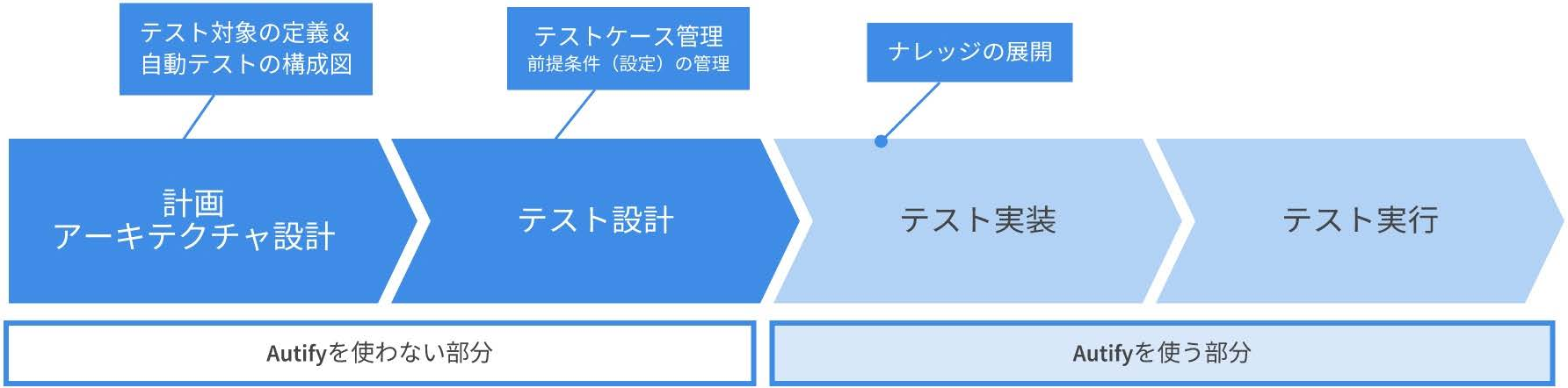
実装時の手間からの開放



- 要素の取得(指定もできる)
- 機能改修に伴うコードの修正
- AIによりUI変化を検知

自動化の検討

テストシナリオを作るまでの準備&ナレッジを蓄積させる



自動化の検討

導入時に準備しておいたこと

1. テストプラン(回帰実行の構成)の検討
2. テストケース設計と管理方法
3. 諸問題の解決 & ナレッジ共有手段
4. ルール・ポリシーの策定

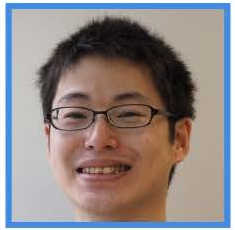
Section4

まとめ

成果

Test Manager

自動化の検討担当



QA Engineer

LowCode領域担当

QA Engineer

NoCode領域担当

導入初期からチームで取り組むことができた

成果

現在の自動テストは多くのメンバーが自動テストに携わっている

20人

NoCode

チームでつくる

5人

LowCode

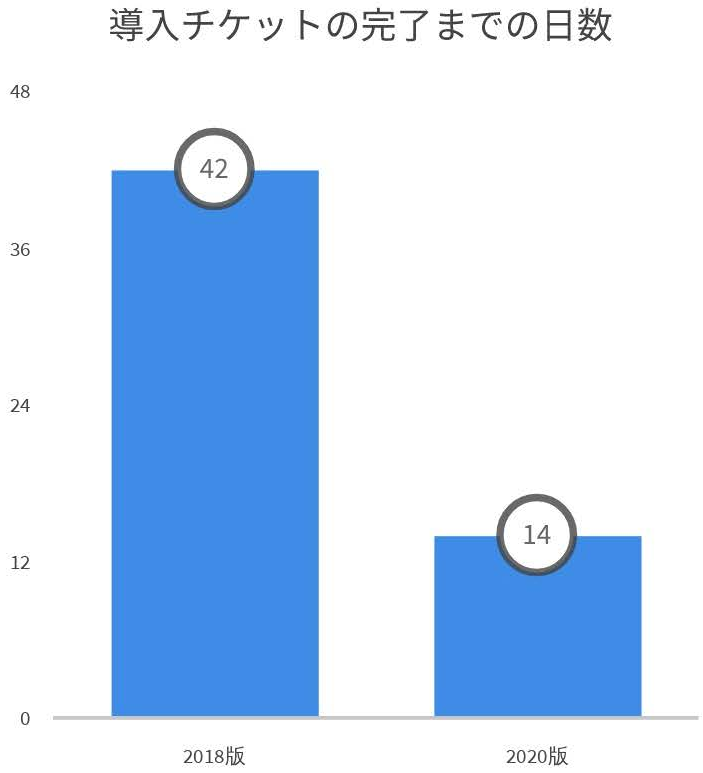
大事な部分に集中する

3人

自動化の検討

チームで取り組めるようにする

サイクルタイムの高速化

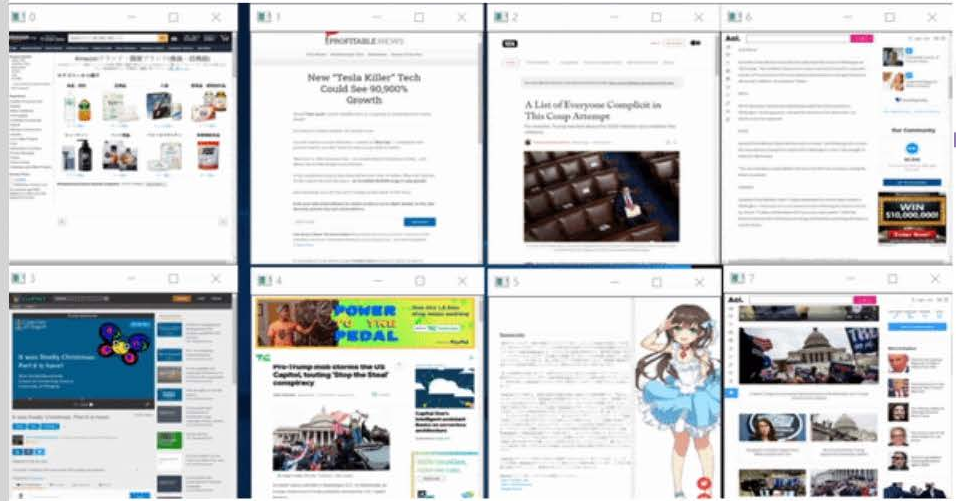


導入と初期のテスト実装チケットの完了までの日数を比較

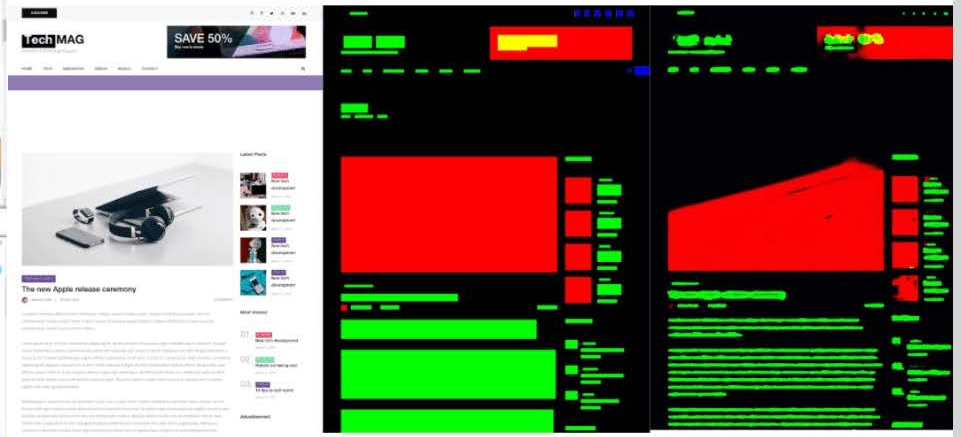
対応人数のボトルネックが解消され、サイクルタイムの高速化を実現。

自動テストにおけるスウォーミングが可能な体制になった。

今後の展望・期待



Intelligent Test Discovery



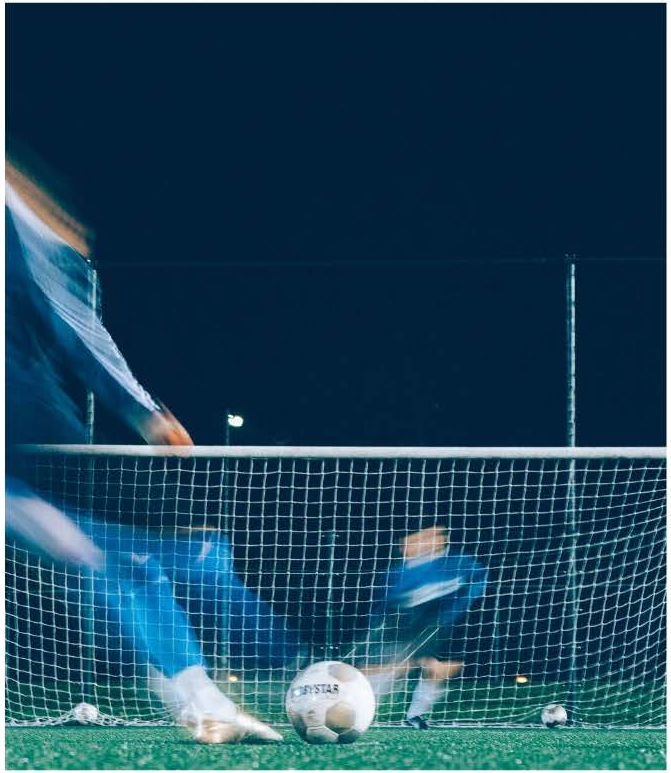
Visual Testing Engine

SaaSを使うことで高度なテストを使うことに注力できる

注意！

まとめ

- 作業領域を分けたことで、メンバーを巻き込むことができた
  初めての人にも手をつけられる作業になった
- **Autify**を導入することでメンテナンスコストを下げた
  間接作業を減らし、必要な仕事に集中できるようになった
- チームで使うための準備・仕組みづくりもまた重要
  Autifyを使わない部分でも成果物を意識する必要がある



Thank you for listening!