UX/マイクロサービスアーキテクチャー時代の自動テスト戦略

2019年３月28日

株式会社ＳＨＩＦＴ
山下 裕晃 / 柏井 香里

本日のテーマ

【1】背景

【2】第一部（バックエンドのテスト戦略）

【3】第二部（フロントエンドのテスト戦略）

【4】本セッションのまとめ

著しい市場の変化

- GAFAなどのテックジャイアントによる
  IT以外の既存市場のディスラプション
- 継続・都度利用を前提とした
  サブスクリプションなどの新しい売り方の普及
- シェアリングなどのユーザーの消費スタイルの変化
- AI/IoT/5Gなどの新技術の登場

ビジネス側から開発側への要求が変化

① 高頻度な仮説検証の実施

② 多様なチャネルでの展開

③ 使えるから楽しいへ

要求に対応出来る様に開発が変化

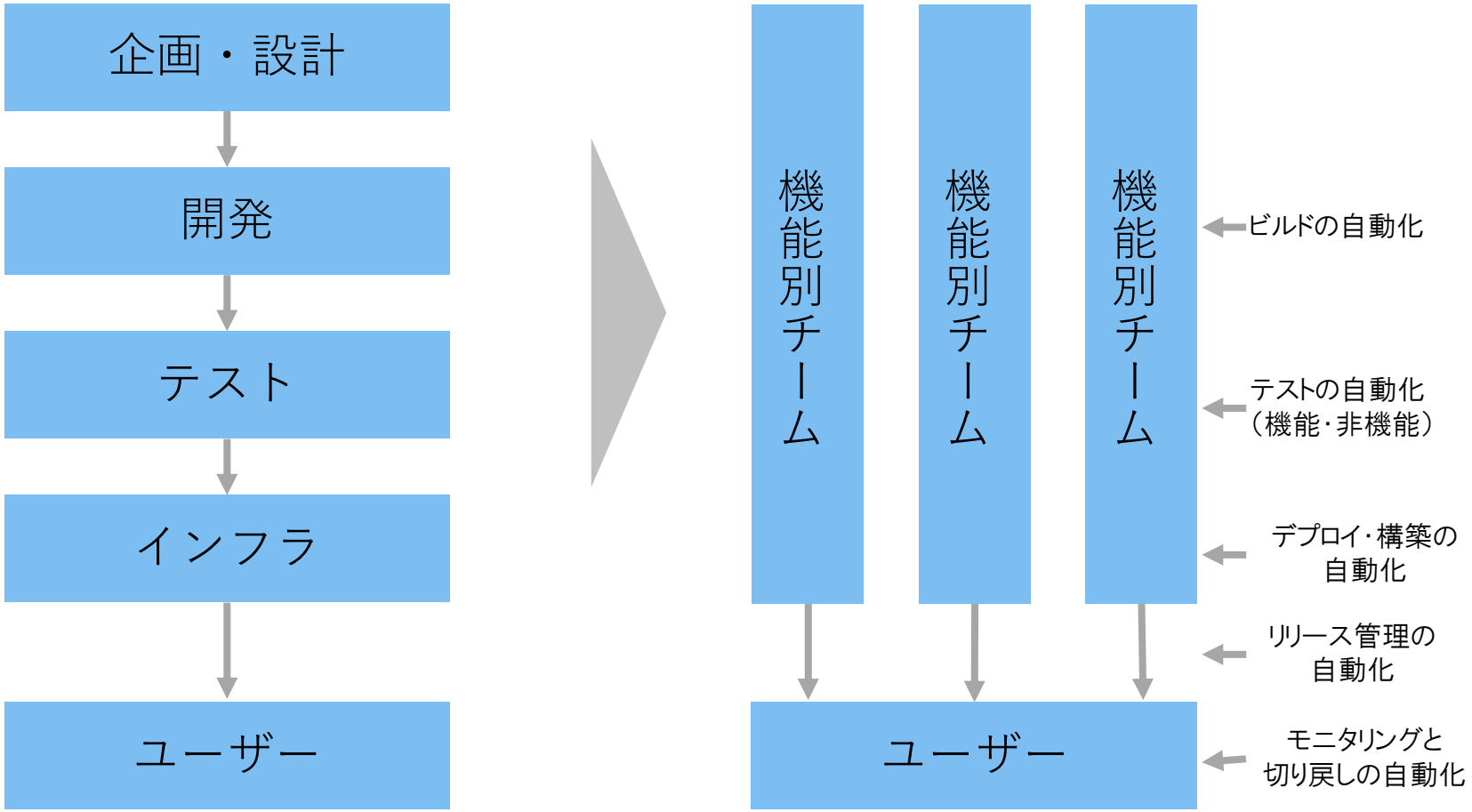
① 開発組織・手法の変化

② アーキテクチャーの変化

③ フロントエンド技術の変化

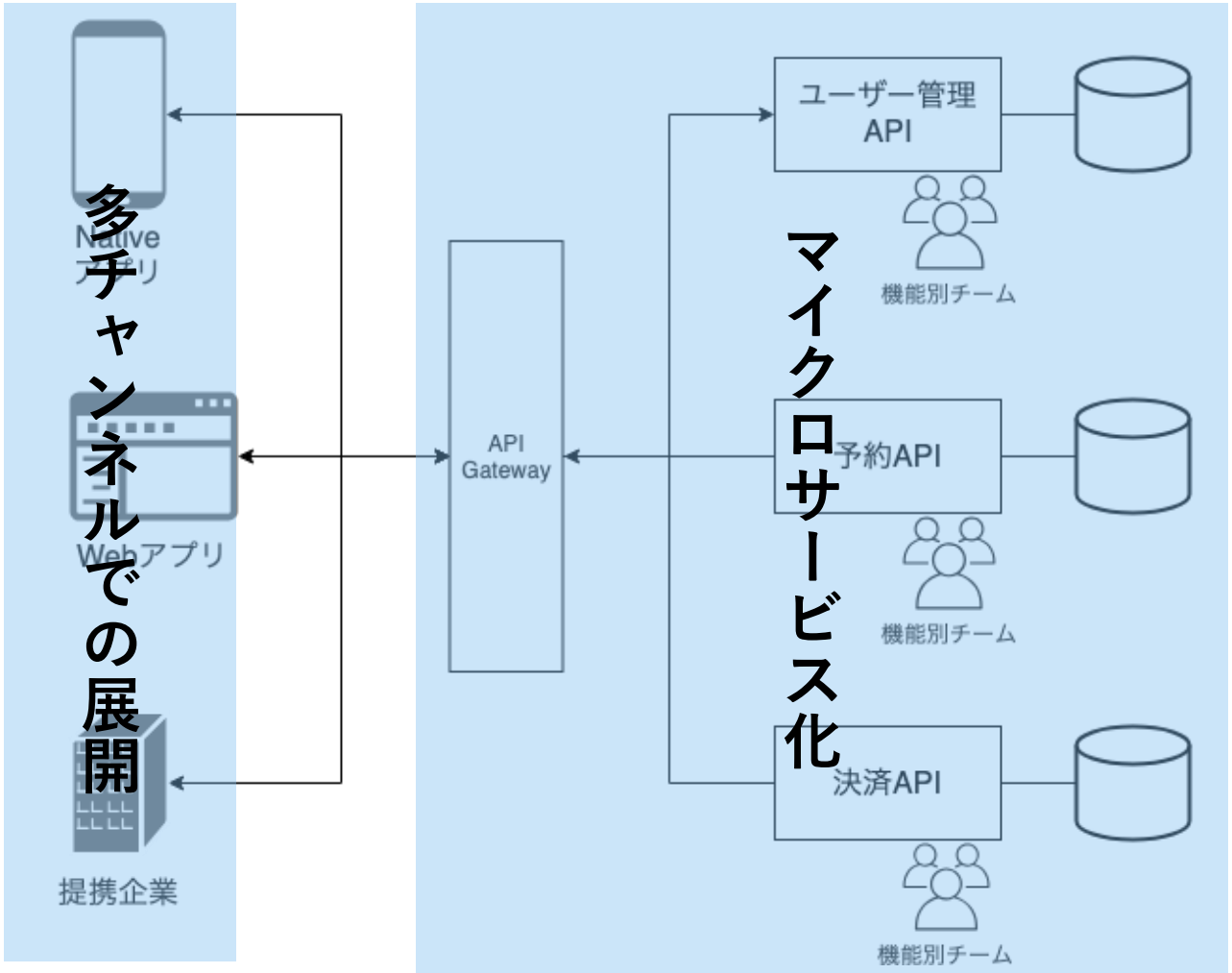
① 開発組織・手法の変化

アメーバ組織と一気通貫の自動化



② アーキテクチャーの変化

新組織構造と多チャネルに対応



③ フロントエンド技術の変化

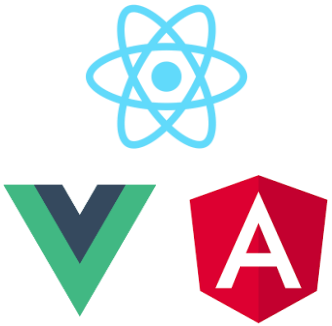
高い要求水準に合わせて技術が変化

UIのリッチ化 ＋ 操作感の追求（SPA）

↓

開発が複雑化

↓



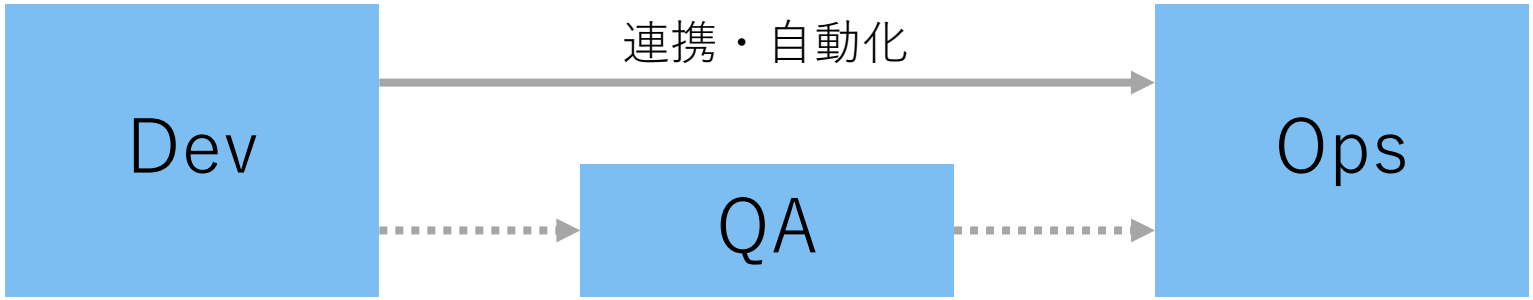
新しいデザイン・開発手法が普及
（Atomic DesignとJSフレームワーク）

テスト側への要求の変化

- 品質とスピードの両立
- 開発・運用との密な連携
- 新構成・技術に適したテスト手法の導入

QAも変化する時が来ている

現状だとQAがボトルネックに？



原因はQAの変化の停滞？

- アーキテクチャー・技術は変化
- QA側は依然として
  ブラックボックス＆GUIテストベース
- スピードを犠牲にするか、
  品質を犠牲にするか

勿論QA側も変化してきている

① 探索的テスト

② GUIテストの自動化

それだけで十分？

探索的テストだけだと？

① 属人的

② 再現性がない

③ 影響範囲の増加に伴い工数爆発

なら自動化しよう

GUIテストは自動でも課題あり

脆い

- 仕様変更に弱い
- SPA/JSフレームワークの登場により更に不安定に

狭い

- 作成や維持に掛かるコスト・時間が大きい
- それにより限定的なテストしか自動化出来ない

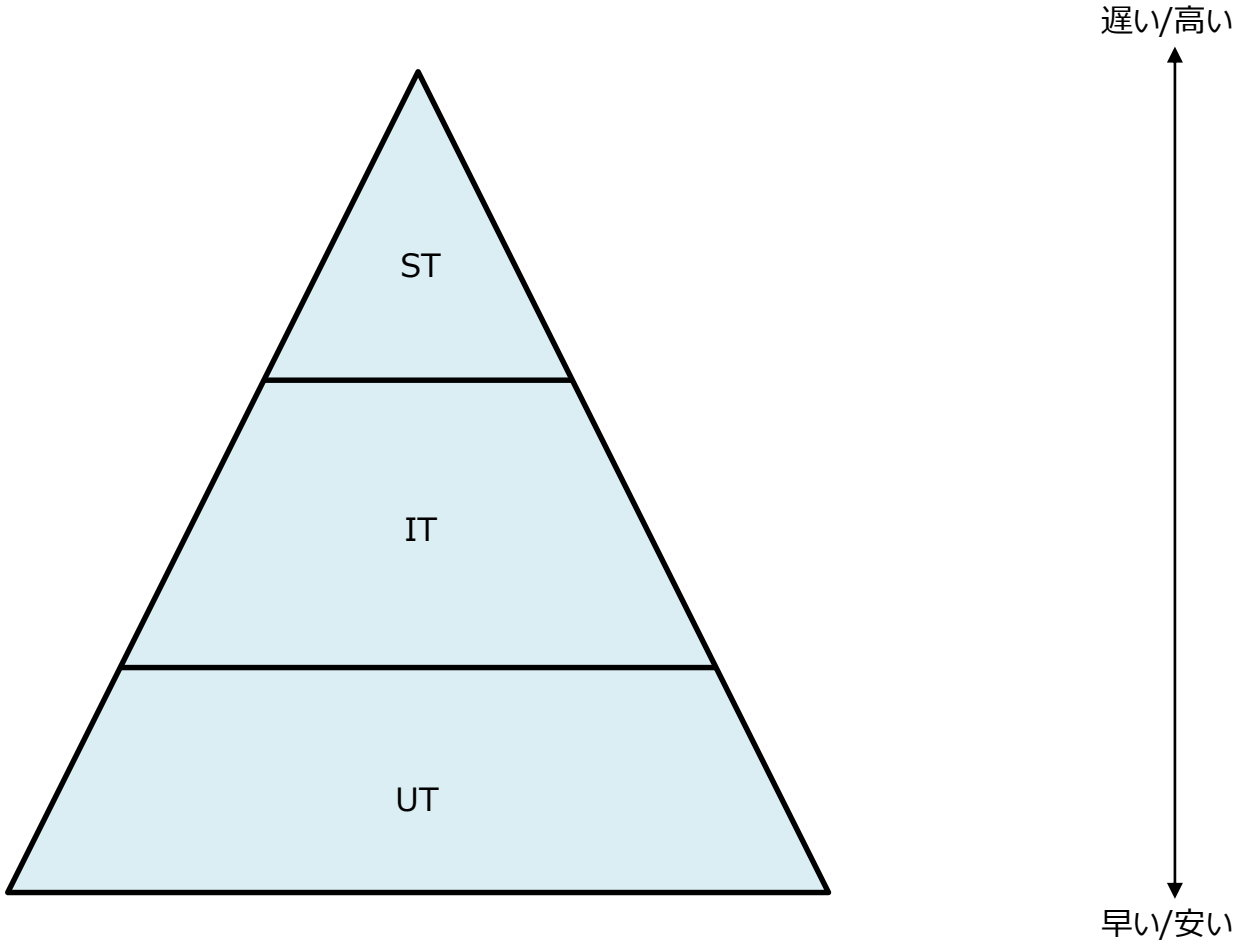
遅い

- 他のテスト手法に比べて実行時間が長い
- 開発サイクルに組み込む際の負担に

新しい自動テスト戦略と手法が必要

前からあったよね？

今までの自動テストピラミッドの弱点



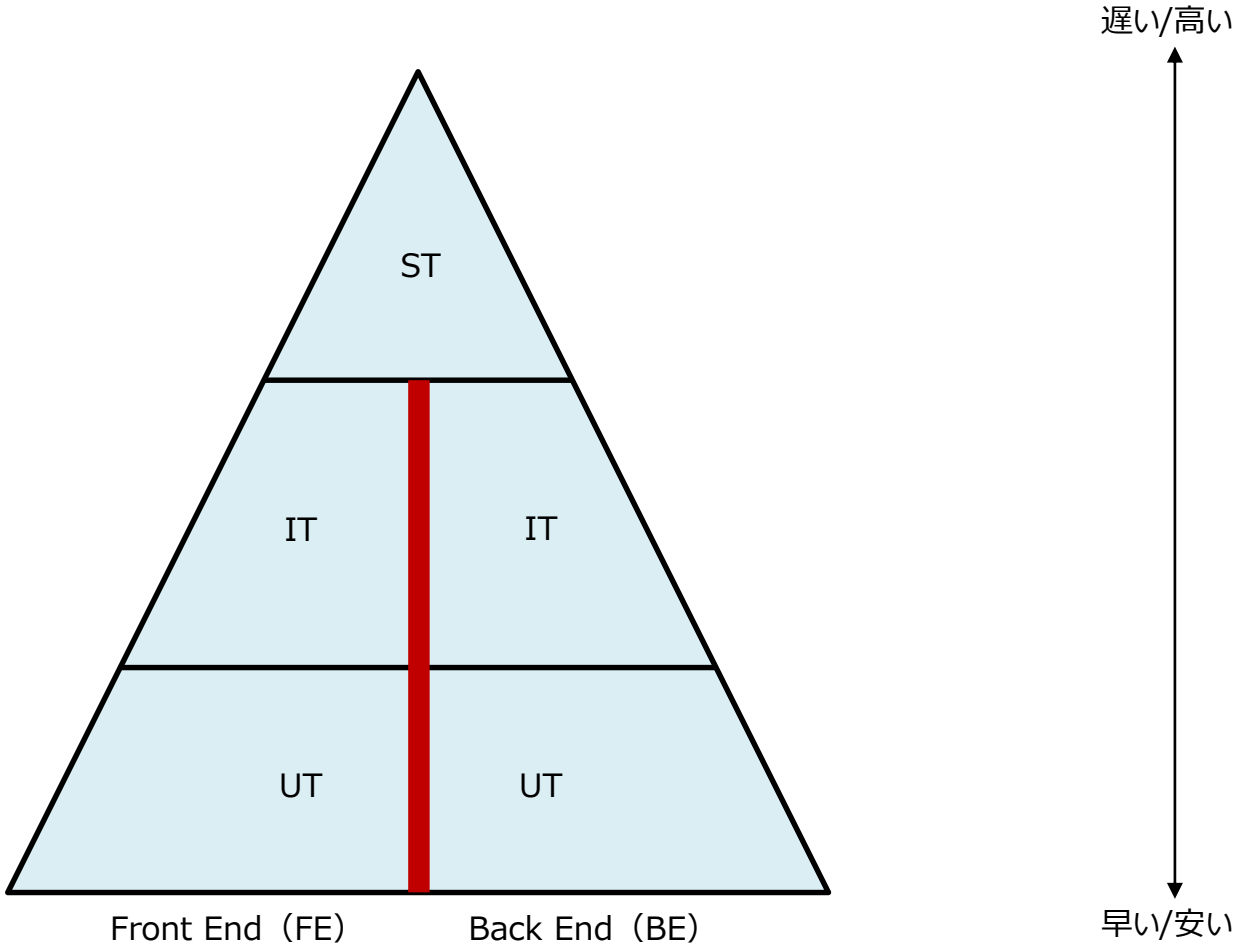
今までの自動テストピラミッドの弱点

- 各工程では具体的に何をやる？
- 複雑化したフロント側への対策は？
- 新しい構造・技術への対応は？

じゃあ、どうするの？

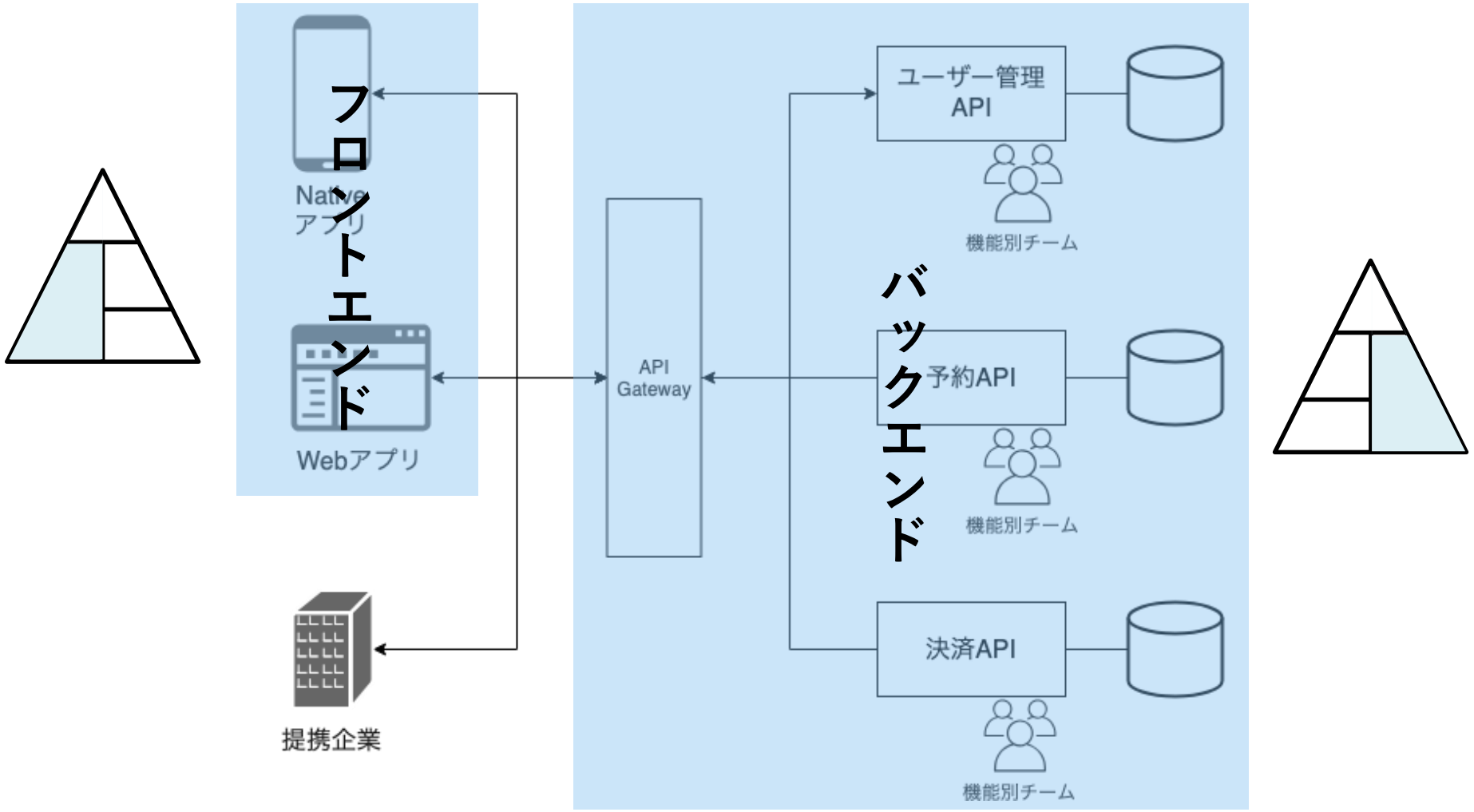
分割して管理せよ

FEとBEでテストを分割する



新・自動テストピラミッド

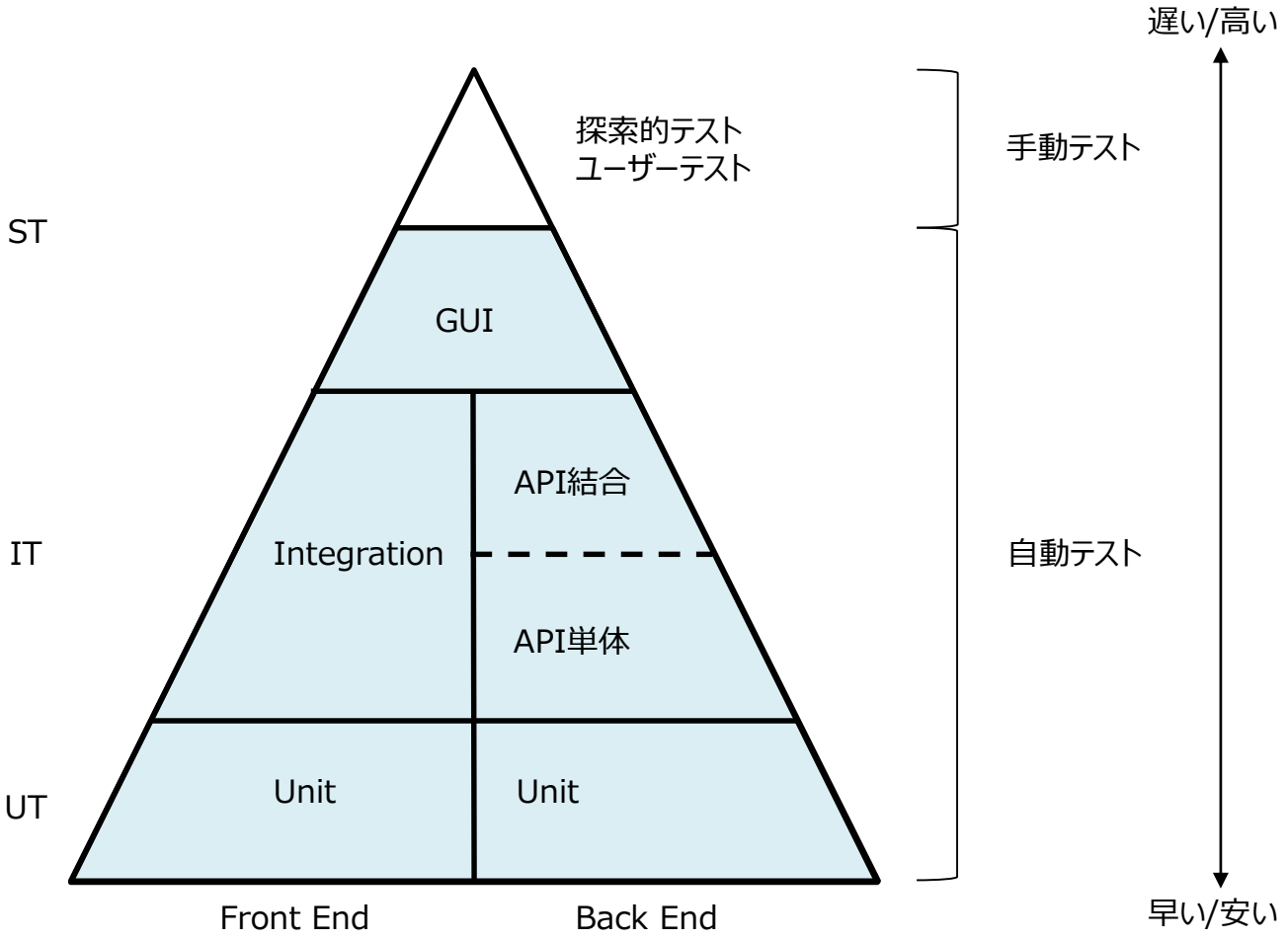
FEとBEは構造的に分離している



もっと分割出来るのでは？

新・自動テストピラミッド

分割して品質を下から積み上げる



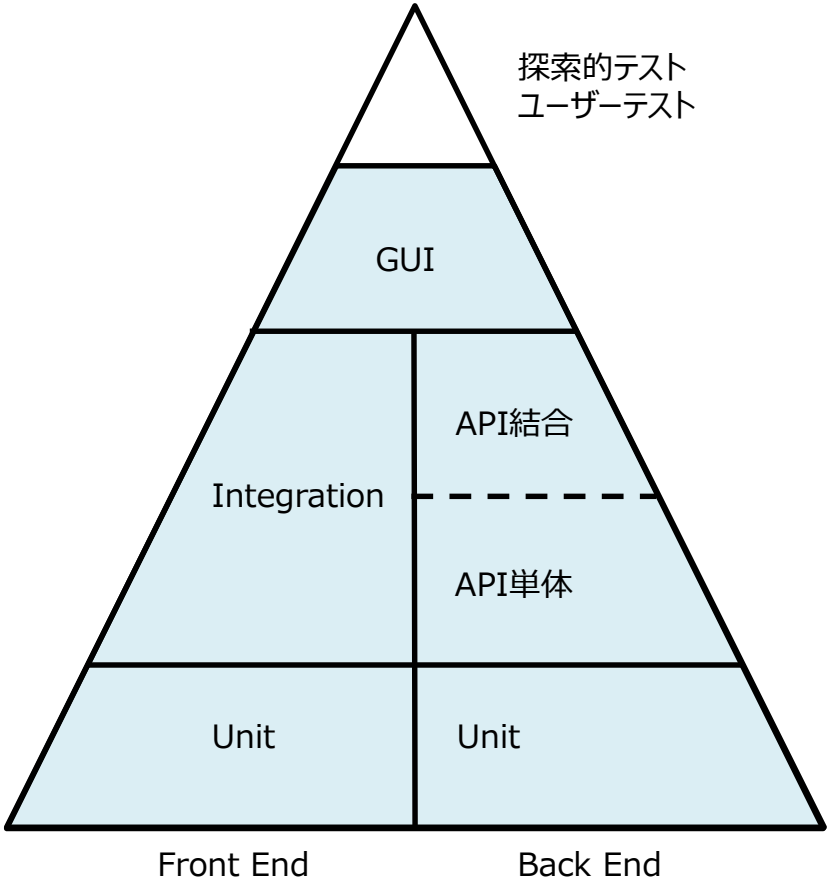
新・自動テストピラミッド戦略の特徴

1. 新構造に沿ってFEとBEを分割
2. 新技術を前提としている
3. IT以下についてもフォーカス

新・自動テストピラミッド

（以降は二部に分けて説明）

第2部

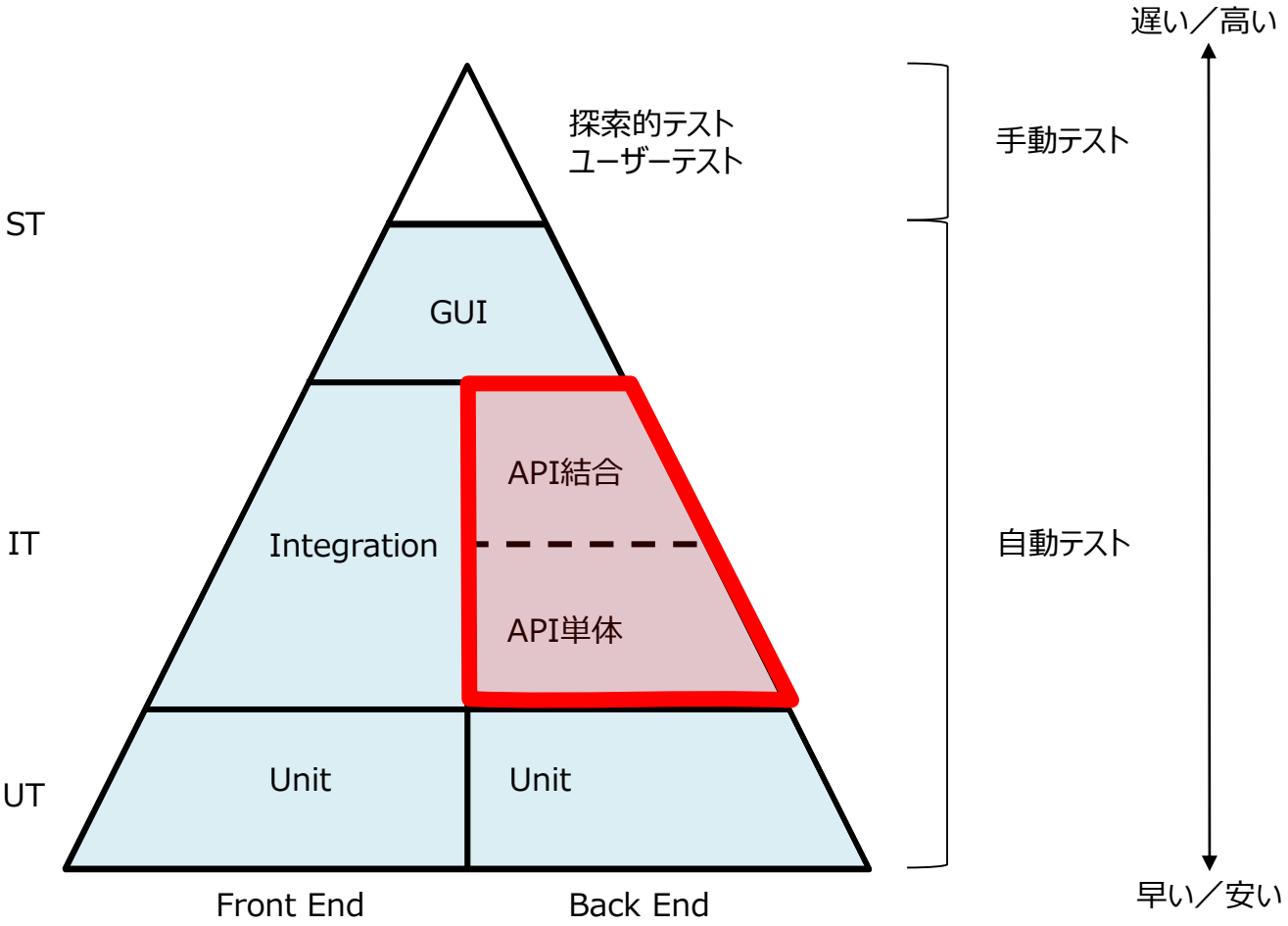


第１部

第1部
バックエンドのテスト戦略
～APIテスト自動化を行ってみて～

株式会社SHIFT
柏井 香里

概要


遅い／高い
探索的テスト
ユーザーテスト
手動テスト
ST
GUI
API結合
IT
Integration
自動テスト
API単体
UT
Unit
Unit
Front End
Back End
早い／安い

目次

目次

テスト自動化については全くの初心者からスタート

柏井 香里（Kaori Kashiwai）

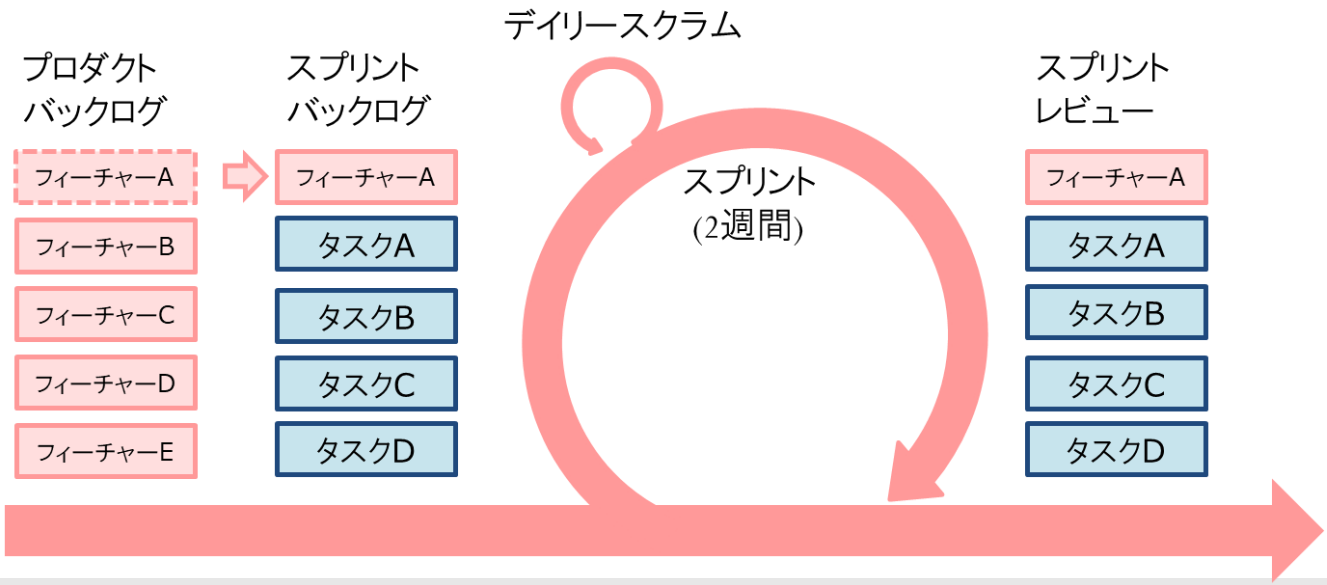
- 所属
  - ビジネストランスフォーメーション事業本部
    品質・技術統轄部 技術推進部 技術推進グループ
- 入社
  - 2018年4月（新卒入社）
  - 同年11月から現在の案件に自動テストスクリプターとして参加
- 経験
  - ちょっとしたプログラミング（C,Java,Ocaml,Python など）
  - SHIFT入社後、テスト実行、設計、管理などを学ぶ

テスト自動化については
全くの初心者からスタート

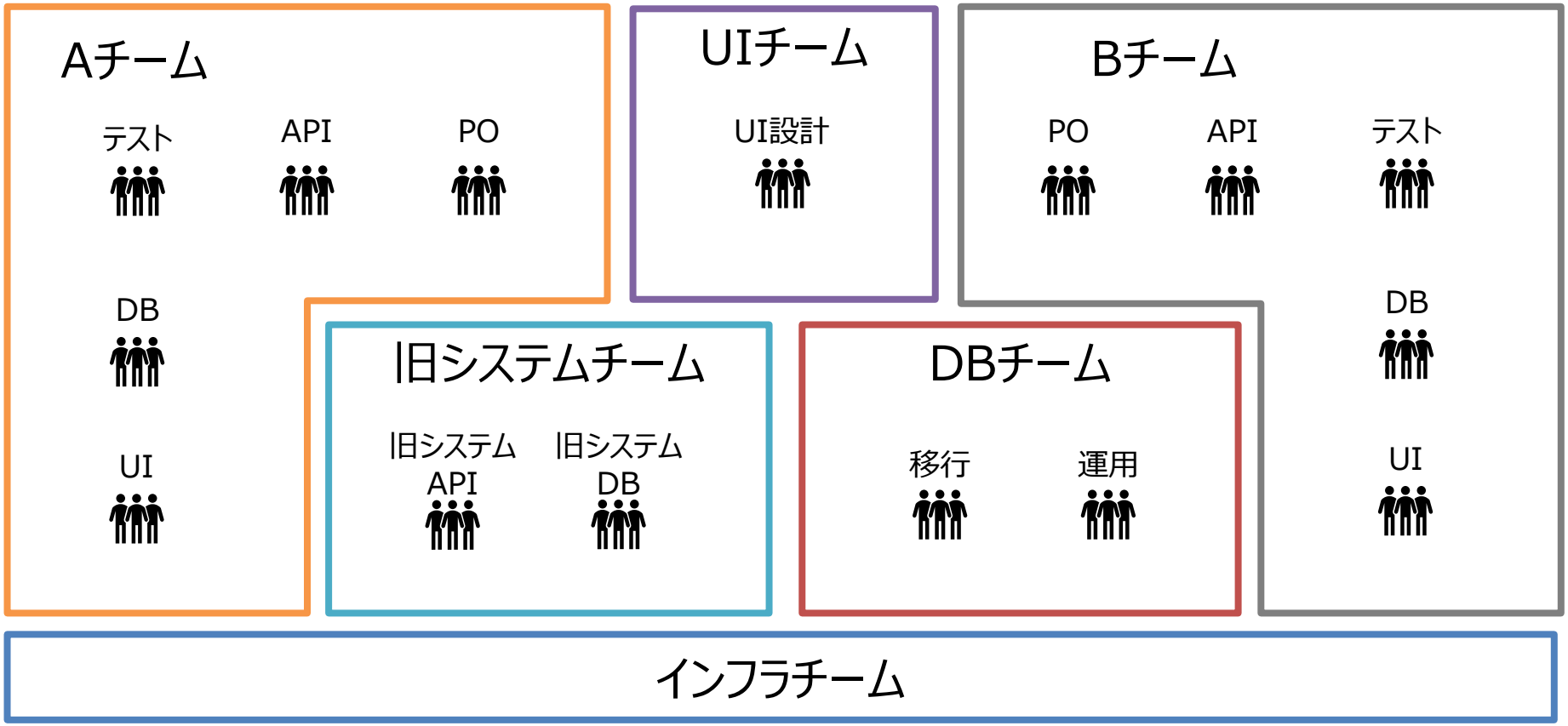
目次

概要

- 基幹システムの刷新
- Agile開発
  - ２チームあり、それぞれのチームが別システムを開発
  - 各機能リリース後も改修、リファクタリングがある

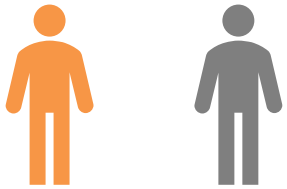


案件全体の体制図


Aチーム
テスト
API
PO
DB
UI
UIチーム
UI設計
Bチーム
PO
API
テスト
DB
UI
旧システムチーム
旧システム API
旧システム DB
DBチーム
移行
運用
インフラチーム

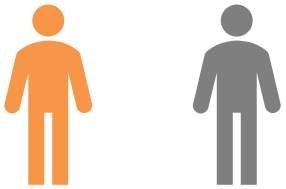
SHIFTテストチームの体制図

自動テストチーム

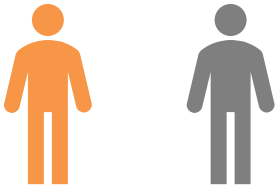


テスト設計、
スクリプト作成

手動テストチーム



テスト設計



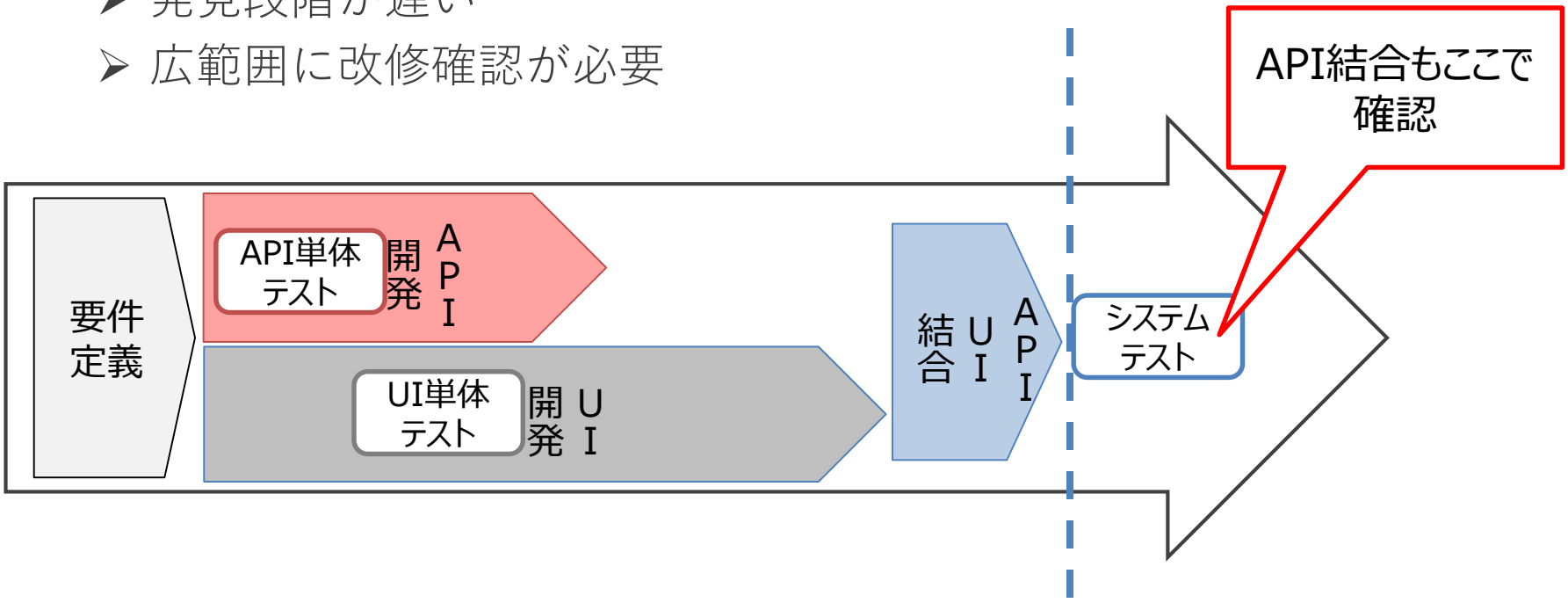
テスト実行

Aチーム　Bチーム

課題①結合テストを行っていない

ＡＰＩ間の確認もシステムテストに含めていた

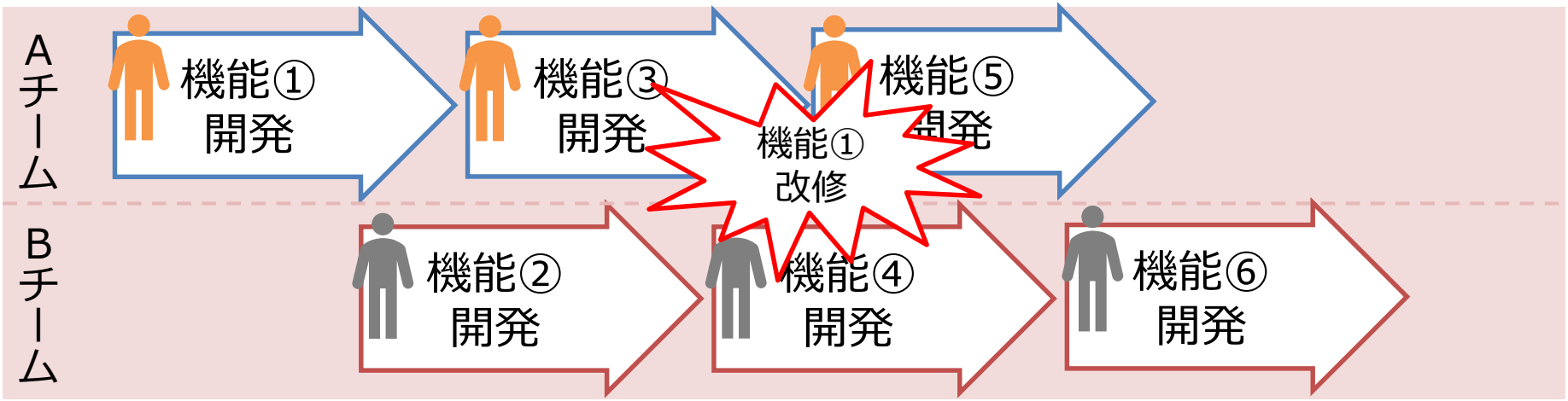
- 発見段階が遅い
- 広範囲に改修確認が必要



課題②改修が頻繁

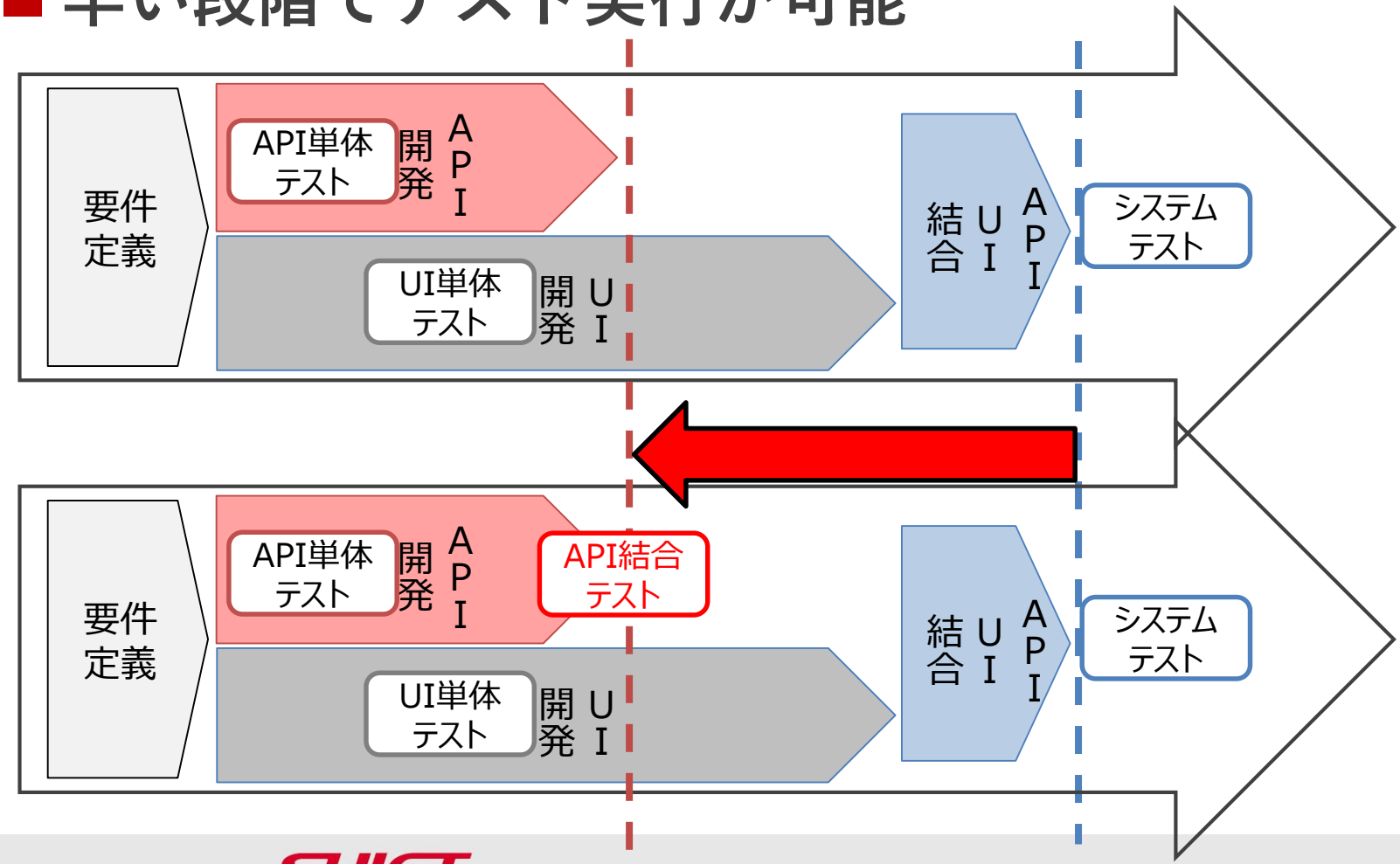
■ 機能リリース後の改修が頻繁に行われる

- 改修されるたび、毎回同様のテストを実施する必要がある
- 他スプリントで別のシステムを開発しているため、リファクタリングのテストにテスト要員をあまり割けない



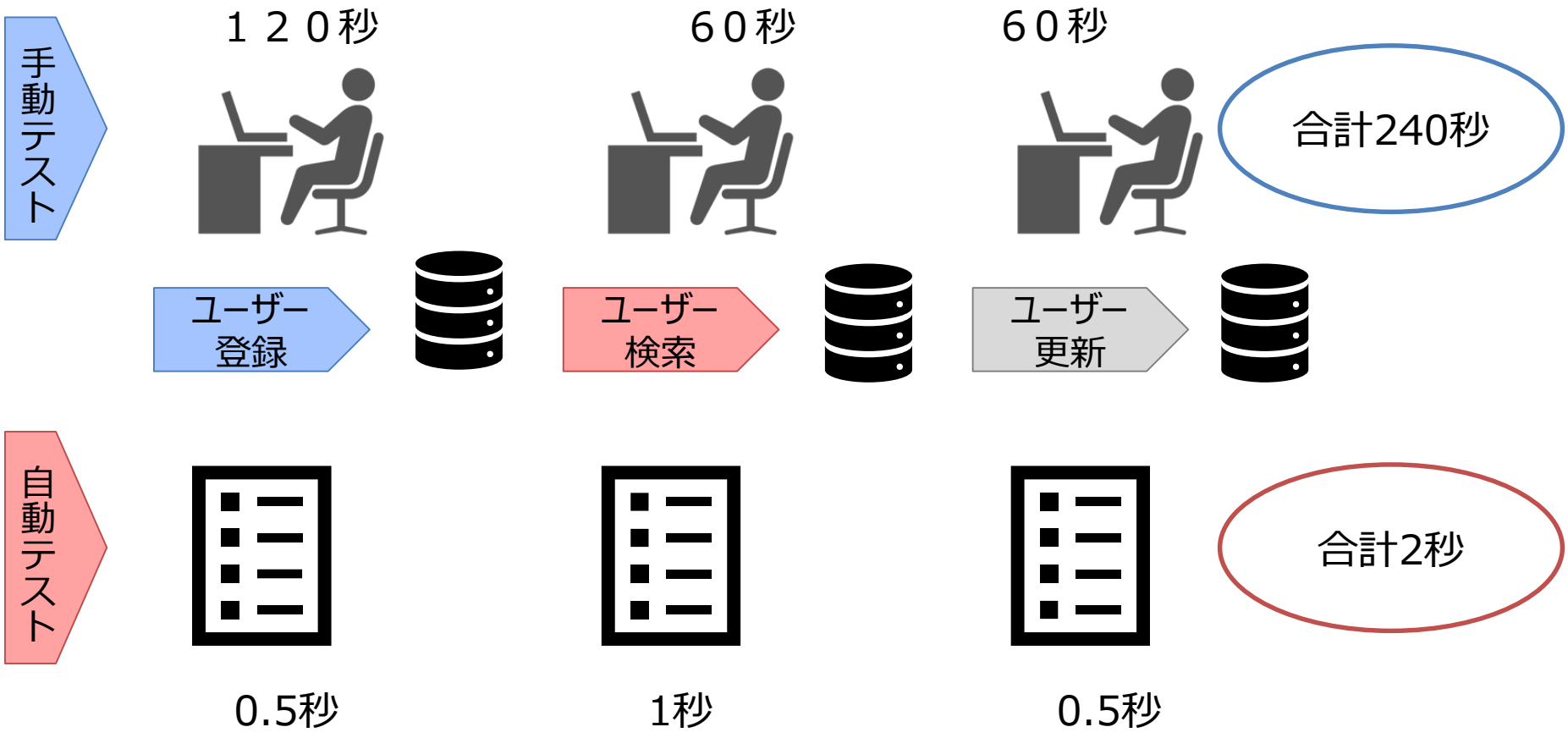
API結合テスト自動化の利点①

■ 早い段階でテスト実行が可能



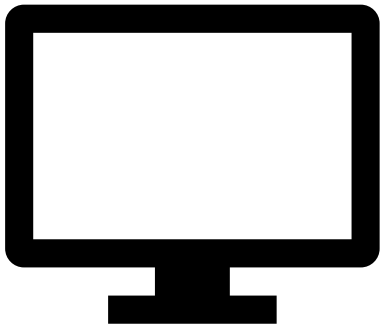
API結合テスト自動化の利点②

■ 実行時間がより短い



とはいえ…

- **ただし、APIテストをやったからGUIテストをやらなくてよい、というわけではない**
  - API結合テストだけでは確認できない部分があるので、両方必要



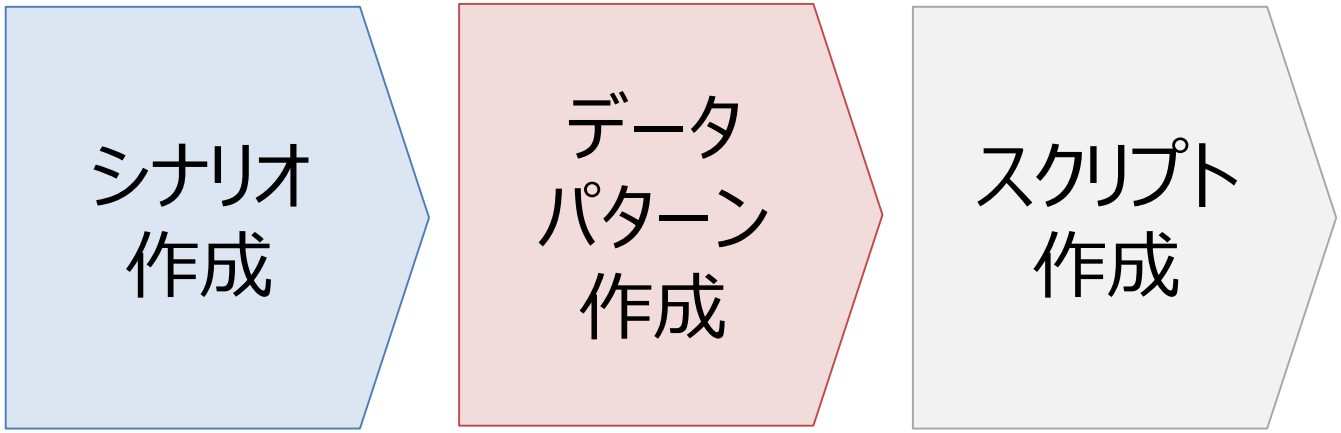
- オブジェクトの表示
- UIとAPIの結合

など

目次

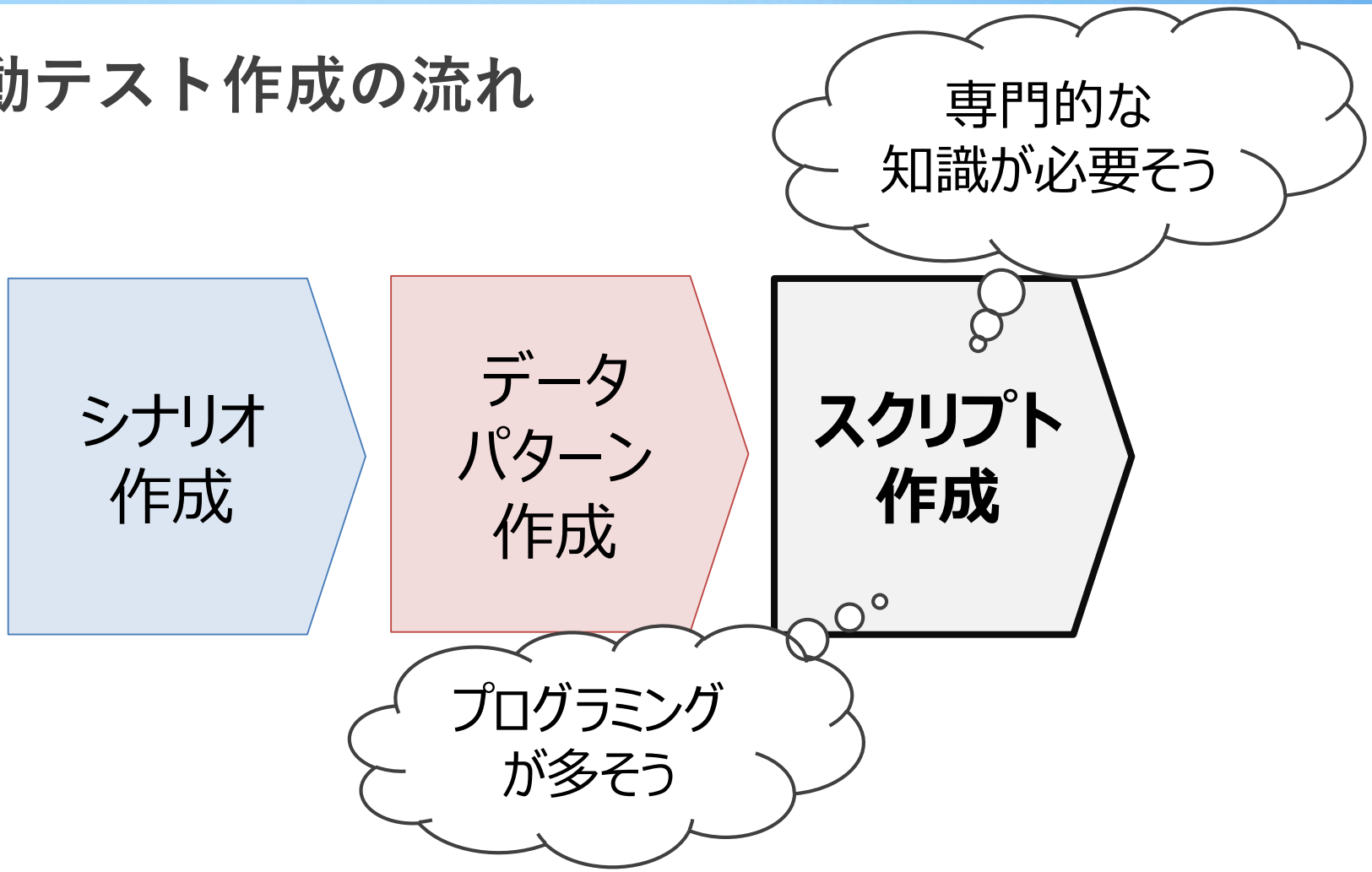
3．ぶつかった壁

API結合自動テスト作成の流れ



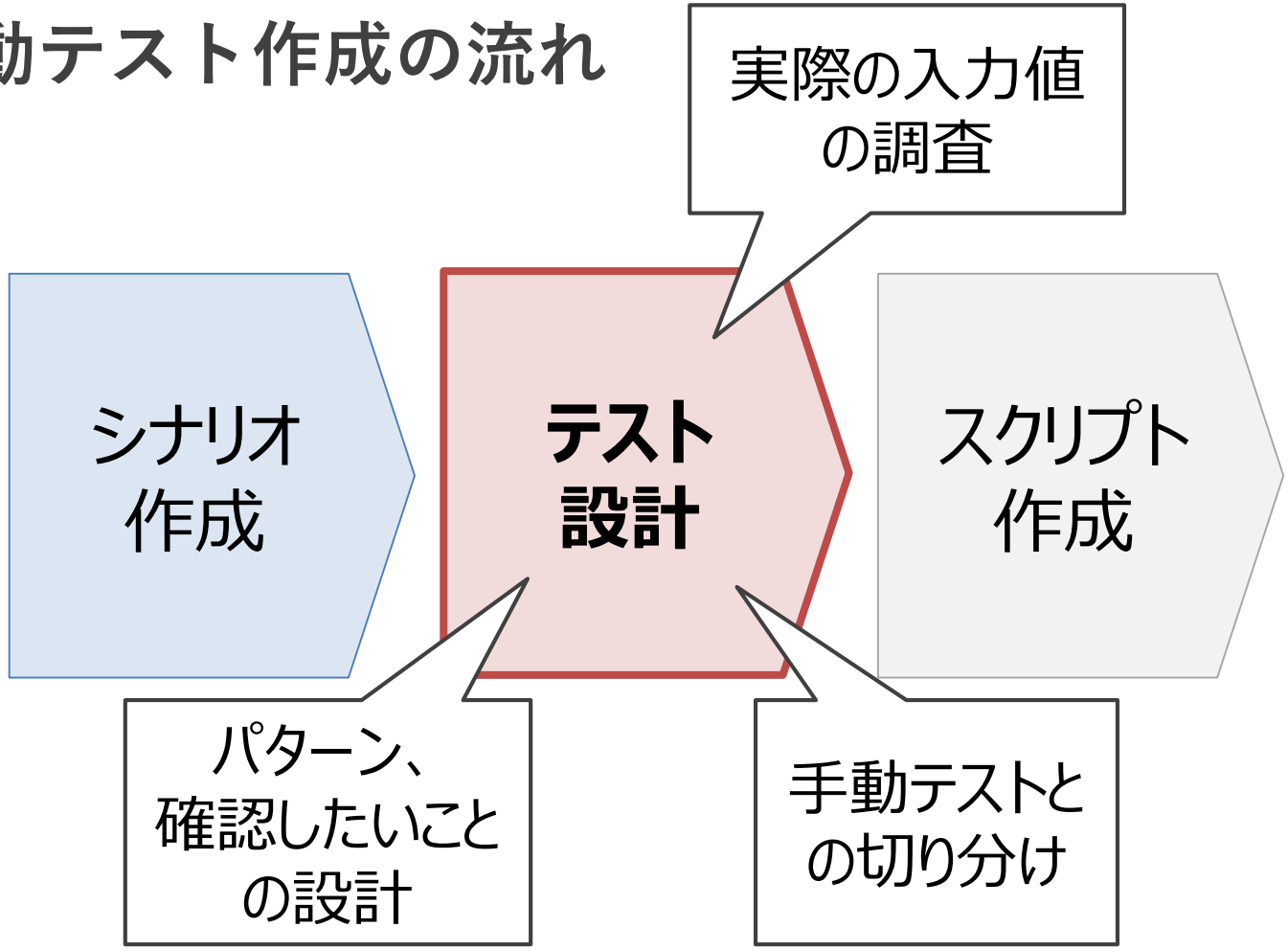
苦労すると予想していた箇所

■ 自動テスト作成の流れ



実際に苦労した箇所

■ 自動テスト作成の流れ



テスト設計のやり方

やり方① テスト観点を決める

ユーザー情報を
更新するための
ユーザー作成

新規で作成した
ユーザー情報を更
新できること

情報を更新した
ユーザー検索
できること

更新した情報が
反映されていること

テスト設計のやり方

やり方② パターンを考える

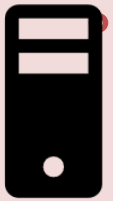
情報を
何も変えず
更新できるか

登録時あった
情報を
変更できるか

登録時なかった
情報を足せるか

新規ユーザー
登録API

ユーザー情報を
更新するための
ユーザー作成



ユーザー情報
更新API

新規で作成した
ユーザー情報を
更新できること

ユーザー検索
API

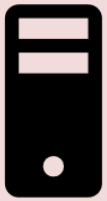
情報を更新した
ユーザーが検索
できること

ユーザ情報取得
API

更新した情報が
反映されていること

テスト設計のやり方

やり方③ 必要なパラメータと値を書きだす

入力値	名前：SHIFT太郎 性別：男 生年月日：1990/1/1	ユーザーID:新規登録時のユーザーID 名前：SHIFT次郎 性別：男 生年月日：1990/1/1	名前：SHIFT次郎	ユーザーID：登録時に返ってきたユーザーID
	 新規ユーザ登録 API	 ユーザ情報更新 API	 ユーザ検索 API	 ユーザ情報取得 API
期待値		ユーザーID：新規登録時のユーザーID	名前：SHIFT次郎 性別：男 生年月日：1990/1/1	ユーザーID：登録時に返ってきたユーザーID 名前：SHIFT太郎 性別：男 生年月日：1990/1/1

ぶつかった壁

① 準備時間が必要

② 手動テストと重複してしまう

壁①準備時間が必要

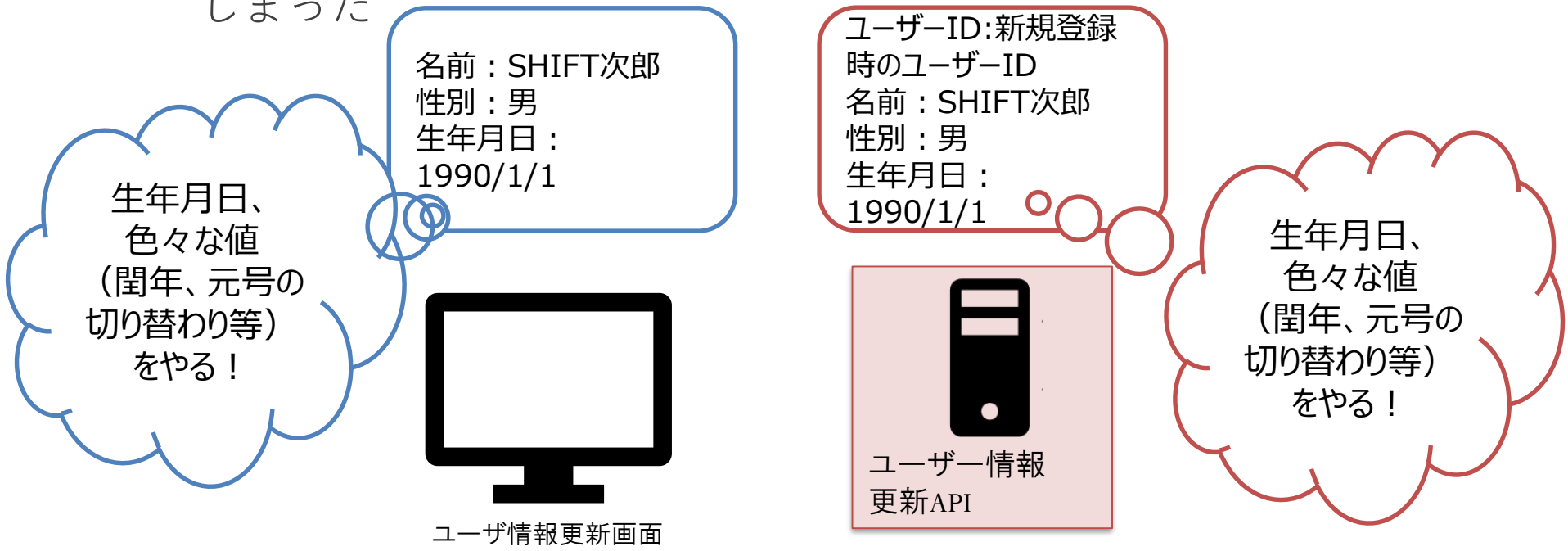
■ APIテストの自動化は、準備に時間を要する

- Agile開発では短い期間でリリースするため、自動テストの準備が間に合わず手動と自動で切り分けたものを結局全部手動テストで行うことになった場合もあった
- シナリオやテスト設計をする際、システムの中身の理解が必要
- テスト設計にかかる時間は手動とあまり変わらない（設計項目が手動とあまり変わらないため）

壁②手動テストと重複してしまう

■ 手動GUIテストが同じテストを行ってしまう

- 既に自動テストで確認できていることを手動テストの方でも行ってしまった



目次

ぶつかった壁（再掲）

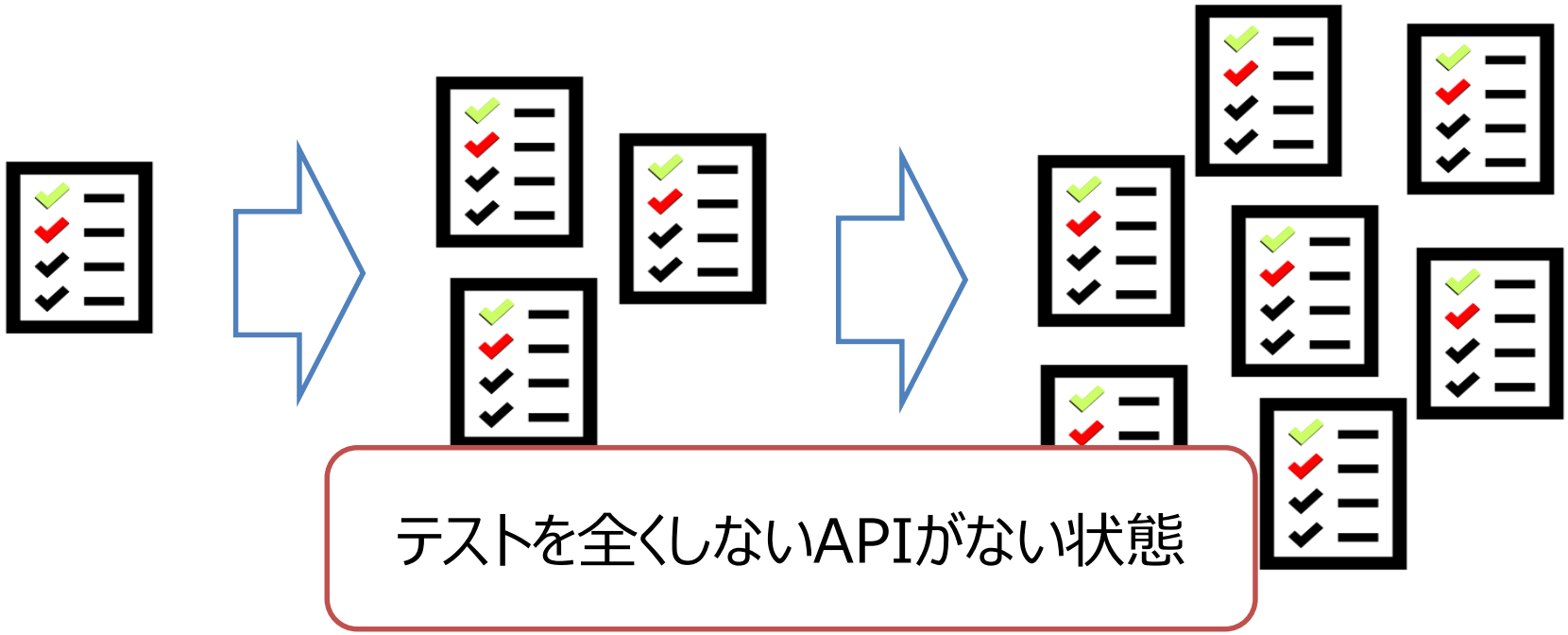
① 準備時間が必要

② 手動テストと重複してしまう

解決策①時間短縮のための工夫

■ テストパターンの粒度を段階的に細かくする

- 疎通確認（1パターンだけ）→パターンを少し増やす→パターンをたくさん増やす、といったように、粒度を段階的に細かくして自動テストを作成する



解決策②手動テストとの分担

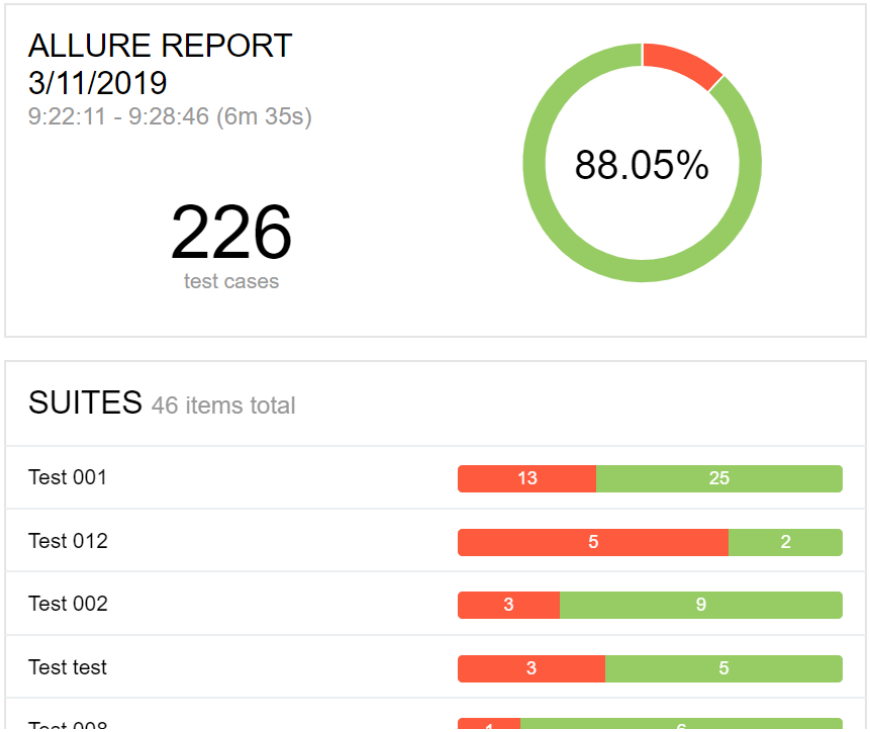
■ 方針を共有し、レビューを行う

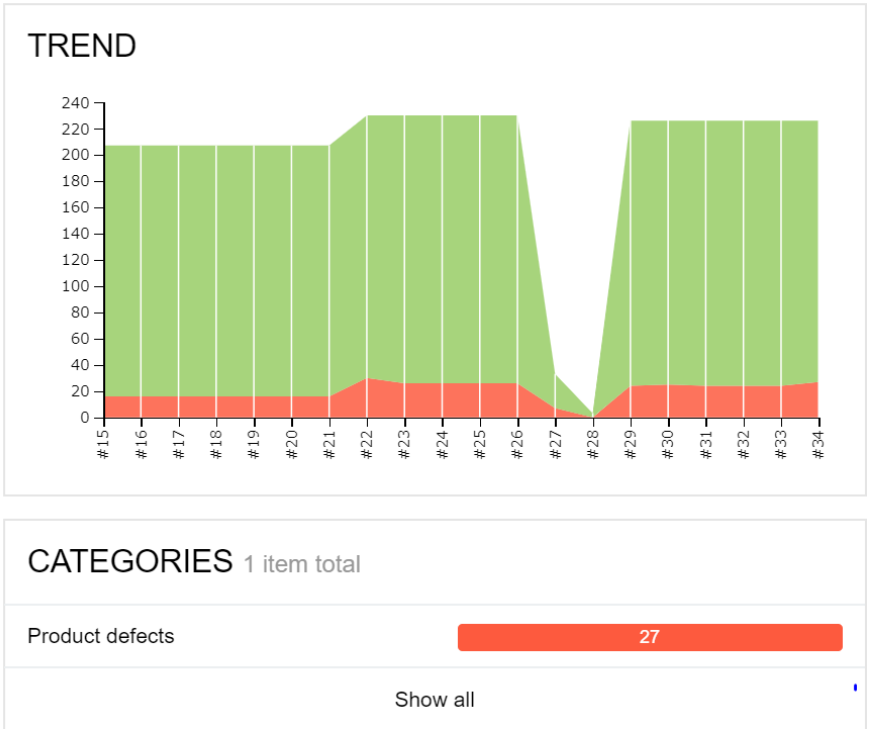
① 手動と自動で同じことを確認しないようにしたい
② 取りこぼしの無いようにしたい
③ 手動テストの負担を減らしたい

- テスト設計前に打ち合わせをし、方針を決める
- GUIテスト、APIテストそれぞれでしか確認できないことを明確化
- 手動GUIテストのレビューを行い、API自動テストで確認できるケースは自動テストに吸収

結果

- CIツールを用いて毎日決まった時間にAPI結合テストを実行





目次

発見

- API間の障害発見時期が早くなった
  - 手動GUIテストを実施する前に発見できるので、手動テストの手戻りが少なくなった
- 実行結果がすぐわかる
  - 実行時間が短い為（２００ケースで５分程度）、テストを実行したいときにすぐできる
  - 登録＞更新＞検索など、流れを確認したい場合手動に比べて短い時間で実行できる

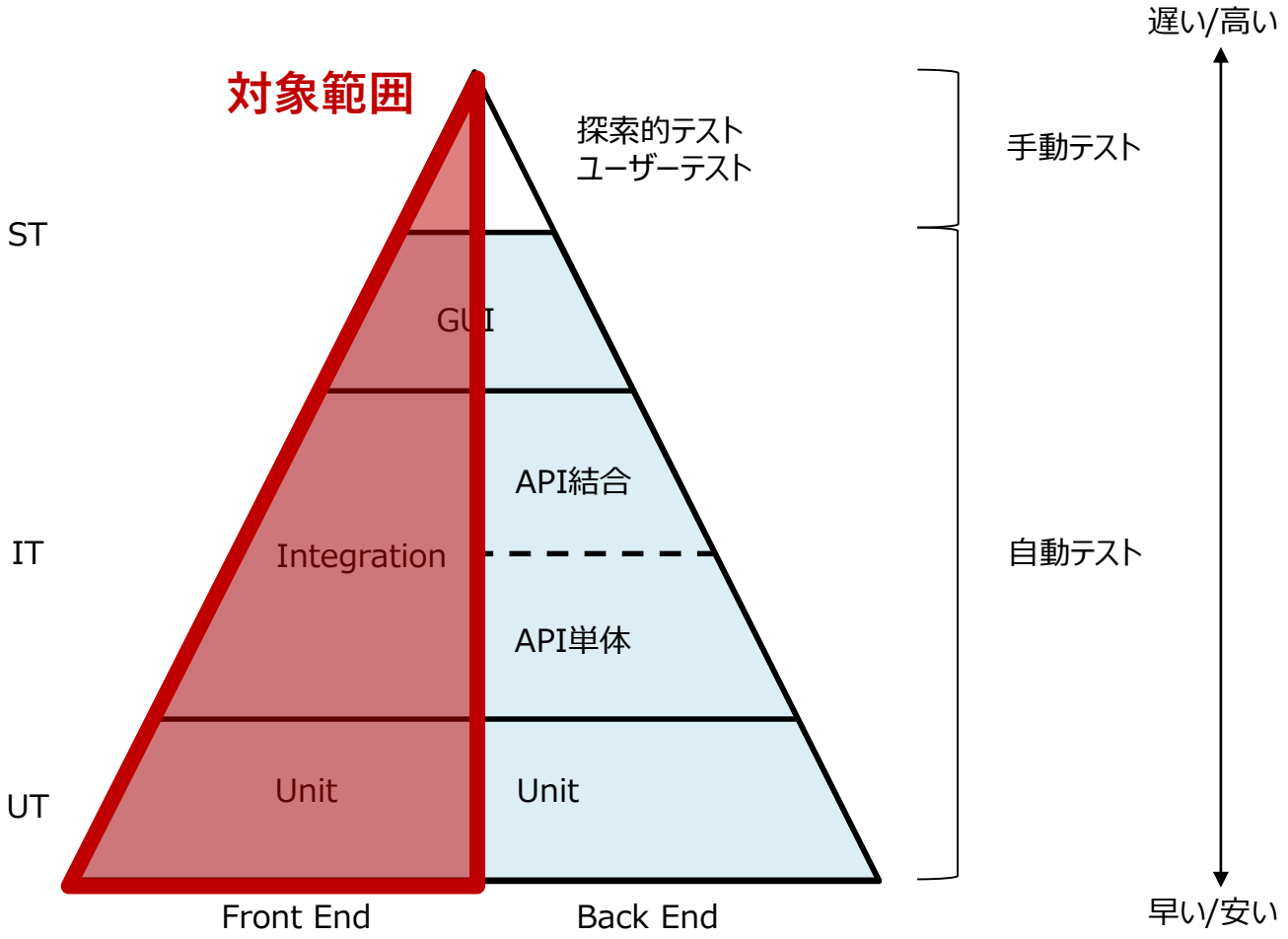
今後取り組みたいこと

- テスト範囲を広げていく
  - まだ実装途中のため、テストするAPIや確認することを増やしたい
  - テストパターンの粒度を上げたい（第２段階のものを第３段階へ）
- 自動API結合テストと連携した自動GUIテスト
  - 今後GUIテストも自動化するにあたって、効率よくAPI結合テストと重複しないように設計したい

第2部
フロントエンドのテスト戦略

株式会社ＳＨＩＦＴ
山下 裕晃

概要


遅い/高い
対象範囲
探索的テスト
ユーザーテスト
手動テスト
ST
GUI
API結合
IT
Integration
自動テスト
API単体
UT
Unit
Unit
Front End
Back End
早い/安い

目次

目次

自動テストからDevOpsの世界に

- 名前：山下 裕晃
- 所属：品質・技術プラットフォーム事業統轄部
  技術推進部
- 経歴：
  - 自動テストエンジニアからキャリアスタート
  - その後、DevOps支援業務に従事
    - 詳しくは去年の講演をご参照下さい
  - 現在は、DevOps・新テスト戦略に適した
    API自動テストツールの企画・設計・開発

目次

お客様の概要

- 外資系ブランドのECサイトを開発・運営
- フロントエンドの開発だけで10名程度のチーム
  - 運用（コンテンツ・インフラ）を合わせると20人弱に
- 新技術・手法を積極的に導入
  - Atomic Design + Storybook
  - Next.js(React) + MobX + TypeScript
- 社長が**かなり厳つい**

最初に聞いていた話

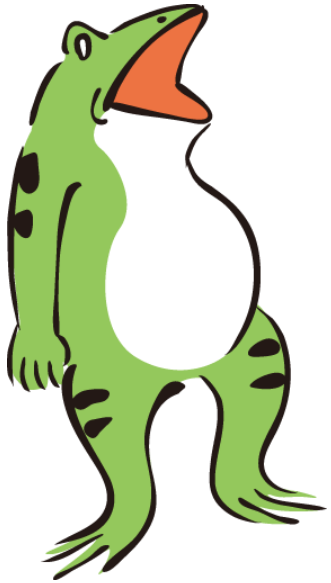
以前の経験を生かしてCI/CDとか
API自動テストとか導入してきて

よし、DevOpsの推進をして
品質もスピードも上げてみせる！

ドヤ顔で提案

- 今の時代、DevOpsですよ
- 機能テストはGUIではなく
  APIテストでカバーしましょう
- インフラ周りも合わせて
  整備していきましょう

思わぬ返答

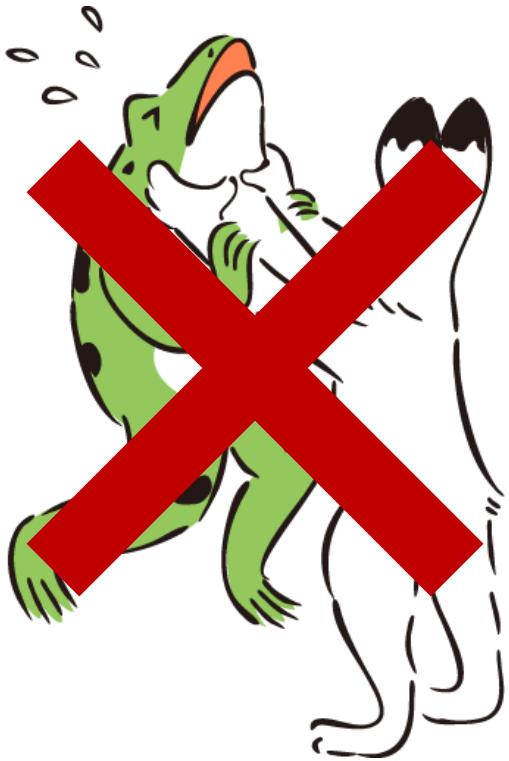


- バックエンドは別会社だから APIテスト必要ない
- 継続的デリバリーも自分達で やっているから必要ない

- いきなりお払い箱！？
- 寧ろ、最初からフロントエンドの テストだと言われていた（らしい）

現状について説明

- フロントエンドに特化した
  テストの知見は多くありません
- 特に自分が得意とする部分が
  使えず価値を直ぐには出せません
- （正直、厳つ過ぎて毎週のMTG、
  胃がキリキリします）



お客様からの返答

- 最初から完璧な答えがあると思っていない
- テスト自体や自動化の知見があるのであれば、自分達だけでやるよりも、早く辿り着ける
- 一緒に考えながら取り組ましょう

やってやるぞ！

目次

現実は厳しい

触ったことのあるもの

- HTML
- CSS（ちょびっとSass）
- JQuery

現実

- JS！！JS！！JS！！
  - React（Next.js）＋Mobx
- Atomic Design
  - 無数の部品、何万行のコード
- 無数のツール群
  - Storybook, Post CSS, etc

歯が立たない

とにかく勉強だ！

実はバックエンドと一緒なのかも

- ステートレスな部品を作る
- 外部からデータを与えることで変化を起こす
- それらを組み合わせて機能を作る
- 機能を組み合わせて全体(ページ)を作る

フロントエンドにも当てはまる？

- 構造が一緒ならテストピラミッドはFEにも当てはまるのでは
- 部品をUT、部品結合をITで担保BEと結合しないといけない部分だけをSTで対応
- STだけだとリリーススピードに追いつけないからITでやろう

ITに寄せて自動テストを作成・運用

- 機能単位でテスト設計を実施
- STではなく、ITで討ち取れるものは積極的にITに寄せる
- 残ったものはE2E + 手動テスト
- CIに乗せてPR毎にテストを実施

思ったより効果が上がらない

- 確かにSTでやる範囲は減った
- ただ、かなり労力を
  使ってしまっている
- これだと十分な範囲をカバー
  出来ず継続もしないかも

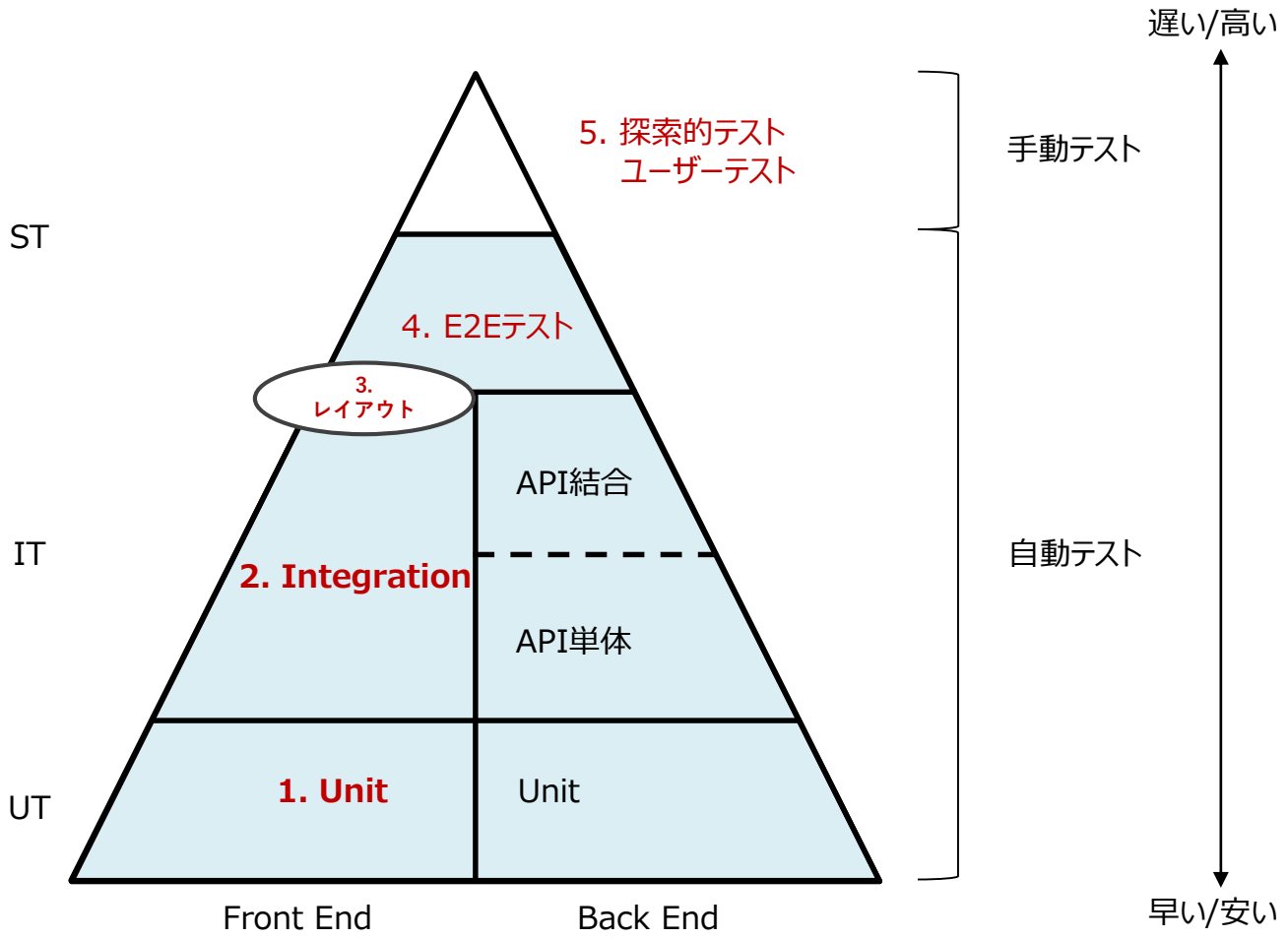
もっと詳細化する

- 本来やらなくてもいいことまでITでやっている
  - 「全部STで」が「残り全部ITで」に
- 構成がテスタブルじゃない
- どこを重点的にテストするかの指標がない

分割・分類・障害蓄積

- 分割して其々でやることを定める
- 分割した工程に分類
- 障害を蓄積

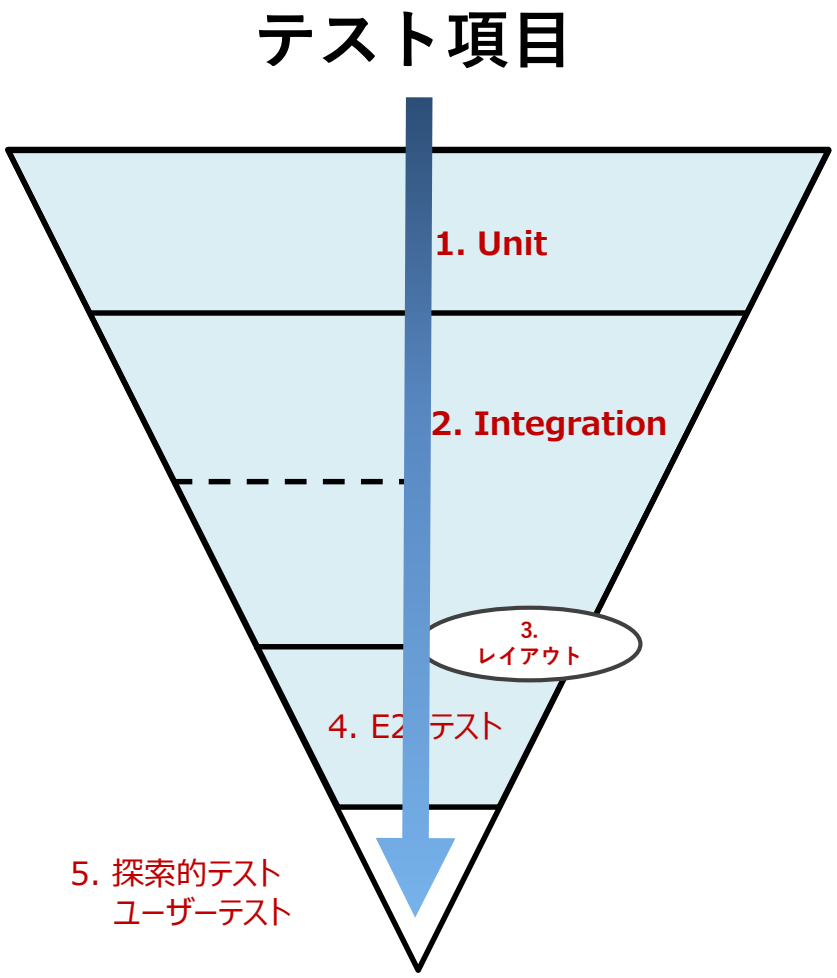
5段階にテストを分割


遅い/高い
5. 探索的テスト
ユーザーテスト
手動テスト
ST
4. E2Eテスト
3.
レイアウト
API結合
IT
2. Integration
自動テスト
API単体
UT
1. Unit
Unit
Front End
Back End
早い/安い

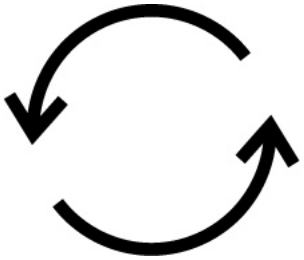
それぞれの役割

UT	最小の部品単位であるComponent単位でテスト
IT	複数ComponentとStore（状態）を組み合わせて 最小で使用する機能単位でテスト
レイアウト	実際の描画をスクリーンショットで比較し レイアウトの微妙なズレを検知
E2E	BEと結合した状態でテスト
探索的	通常の探索的テストに加えて、障害をFBして作った UATの項目書を使って忘れがちなエッジケースにも対応

ろ過するように項目をマッピング



テスト項目は仕様と障害から



① 仕様からざっくりテスト項目を洗い出す

② ろ過 => テスト実装・実行 => リリース

③ 障害が出たら、原因を分析した上で、
項目に追加して重要度別に分類・対策

よし、これで完璧だ！！

目次

継続運用の壁

① テスト自体の質が担保出来ない

② メンテ出来ない

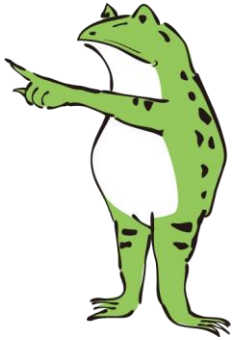
③ 対策しているのに本番障害が出る

① テスト自体の質が担保出来ない

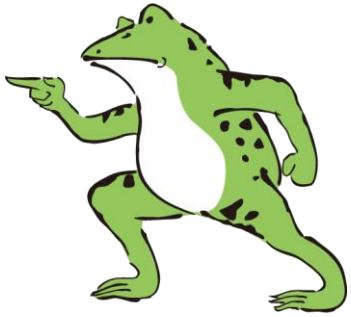
テスト自体の質をあげる

- ルール策定とサンプルテストコードの作成
- 抜き打ちレビュー
- レビューのレビューでレビュアーを育成

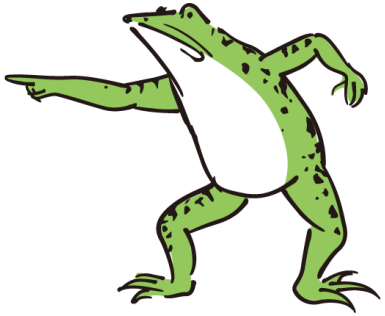
ルール守って



お前がな



お前や！！



・・・

完全主義を手放す

- 「網羅的」なテスト設計に拘らない
- 完璧主義の自動化を辞める
  - 費用対効果に合わせて全ては取りきれなくても重要度高のものが取れれば良しとする
    - UT/ITのスナップショットを中心に
    - レイアウトはスクリーンショットを利用
    - E2Eは無理にユーザーの動きを再現せず安定する小さい粒度で
- 判断基準は障害というファクトベース

③ 対策しているのに本番障害が出る

ファクトベースの振り返りで改善

- 怖いからもっとテストしようではなく障害から対策や優先順位を決めて対応
- そのために障害管理を徹底
- 振り返りながら一個ずつ潰していく

目次

継続こそ力なり

- 適切な対象を適切なタイミングと手法でテスト
- そのためにテストの単位と深さを分割する
- 一回で完璧を追い求めず、少しずつ改善する
- 改善はファクトをベースとしつつ振り返りを軸に

起きた変化・成果

- 半年以上継続
  - 試行錯誤しつつテストを書きながら開発をするサイクルを継続
  - 1万ケース以上のテストケースがあり、ケース自体も改善が継続
- 積極的なリファクタリングが可能に
  - 1年前からコードとしては別物の構成に（疎結合でテスタブルに）
  - TypeScriptへの移行も実施 etc
- 品質はQAだけではなくチームで守ることが浸透
  - 実際にテスト設計や実装・レビューまでやることで意識が芽生える
  - 自分達でテストレベルやテスト密度・手法について検討するように

本セッションのまとめ

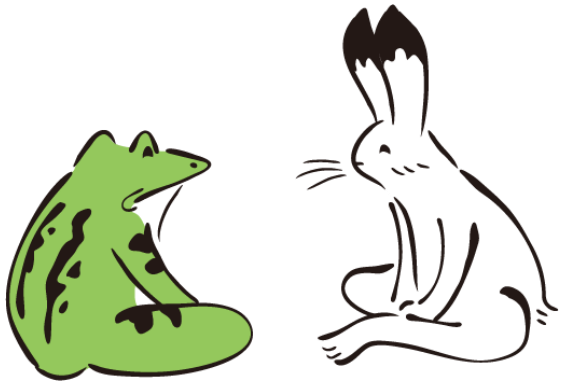
株式会社ＳＨＩＦＴ
山下 裕晃

変わるテスト戦略と技術

- 開発の変化に合わせてテストも変化が必要
- 一方で自動化を闇雲に導入しても
  効果や持続可能性がなくなる
- 自社・顧客の制約やテスト対象にあった
  適切な単位に対して手法を組み合わせる

一緒に次のステージへ

- 話せていない・話せない内容がまだまだあります
- 完璧な答えがあるわけではありません
- 具体的な課題と向き合うことで改善していきます
- 皆さんのお話を聞かせて下さい



以上