MITSUBISHI ELECTRIC
Changes for the Better

JaSST‘20 Hokuriku

継続的テストに向けたテスト自動化
～テスト自動化の事例と推進への課題～

2020/01/24

三菱電機株式会社　情報技術総合研究所
都築　浩平

MITSUBISHI ELECTRIC CORPORATION

はじめに

- 今日お話しすること
  - ◎テスト自動化の事例
    - ・パブリッククラウド活用【JaSST'18 Hokkaido発表内容】
    - ・テスト管理との統合の事例など
  - ◎継続的テスト・組織的な展開に向けた考察
    - ・課題分類し、今後何をすべきか考えてみた話
- 対象外
  - ×最新の自動化技術研究
  - ×開発の内容詳細
  - ×組み込み・産業機器制御
  - ×定量的な評価

MITSUBISHI ELECTRIC
Changes for the Better

目次

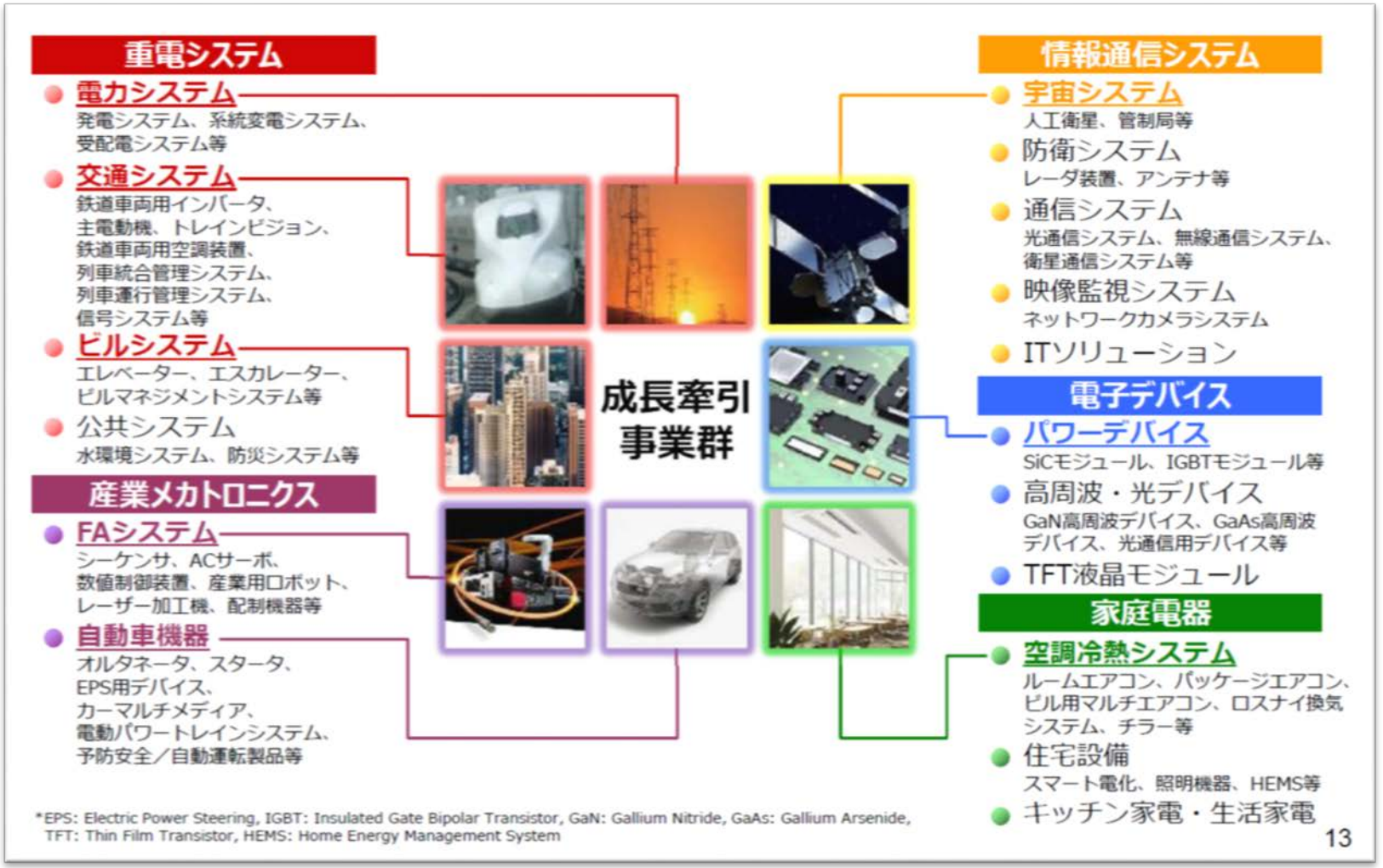
- 三菱電機紹介
- テスト自動化取り組み（～50分目標）
- 継続的テストへの課題（～80分目標）

MITSUBISHI ELECTRIC
Changes for the Better

三菱電機紹介

事業分野

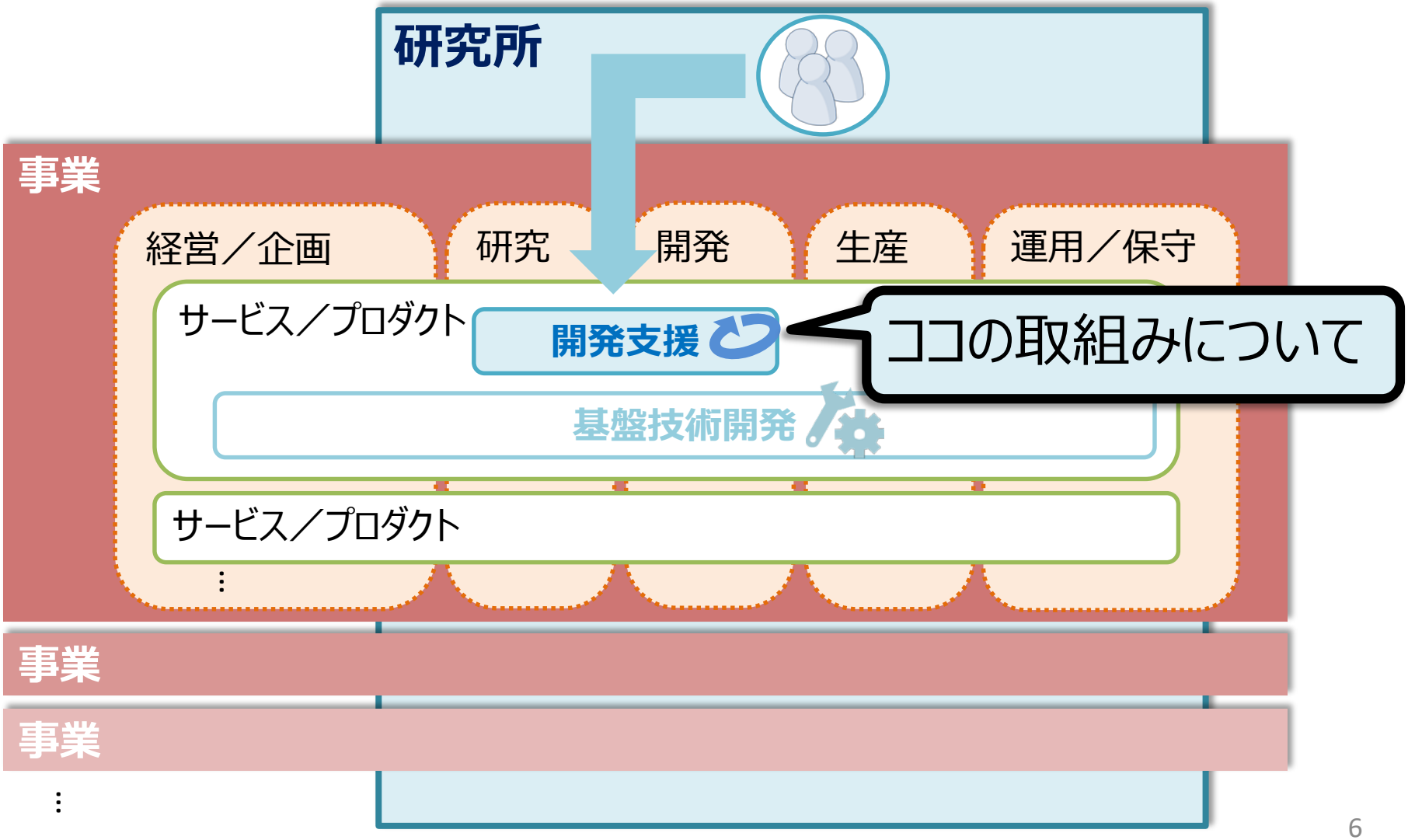
複数の事業群による事業活動の展開



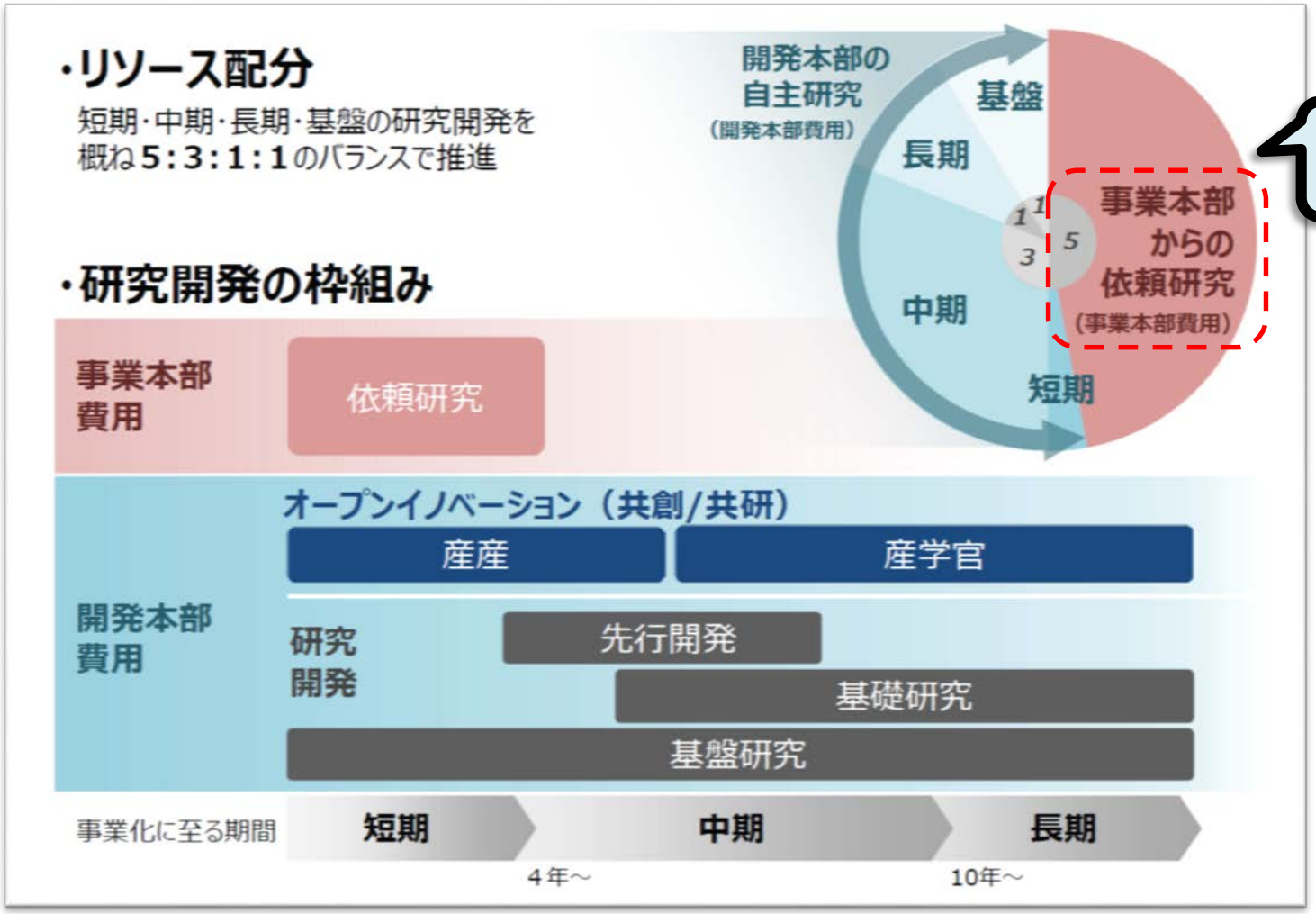
引用：三菱電機の経営戦略（2019年5月），
(https://www.mitsubishielectric.co.jp/ir/data/management_report/index.html#strategy)

事業と研究所

事業／サービスをまたいだ技術開発支援



研究開発と取り組みの位置づけ



引用：三菱電機の研究開発戦略（2019年2月），
(https://www.mitsubishielectric.co.jp/ir/data/management_report/index.html#strategy)

MITSUBISHI ELECTRIC
Changes for the Better

テスト自動化取り組み

取り組み全体

自動化検討・構築・実装と展開を実施

IoTシステム開発
リグレッションテスト自動化

アジャイルでテスト設計
しながら自動化検討

製品開発
移行

ハンズオンで
技術展開

別開発に展開
ペアプロで技術展開

IoTシステム開発
受け入れテスト自動化

Webシステム開発
テスト管理との統合

2017 2018 2019

MITSUBISHI ELECTRIC
Changes for the Better

IoTシステム開発
リグレッションテスト自動化

製品開発
移行

IoTシステム開発
受け入れテスト自動化

Webシステム開発
テスト管理との統合

2017 *2018* *2019*

MITSUBISHI ELECTRIC
Changes for the Better

IoTシステム開発
リグレッションテスト自動化

開発背景など

IoTシステムのアジャイル開発でテスト自動化検討

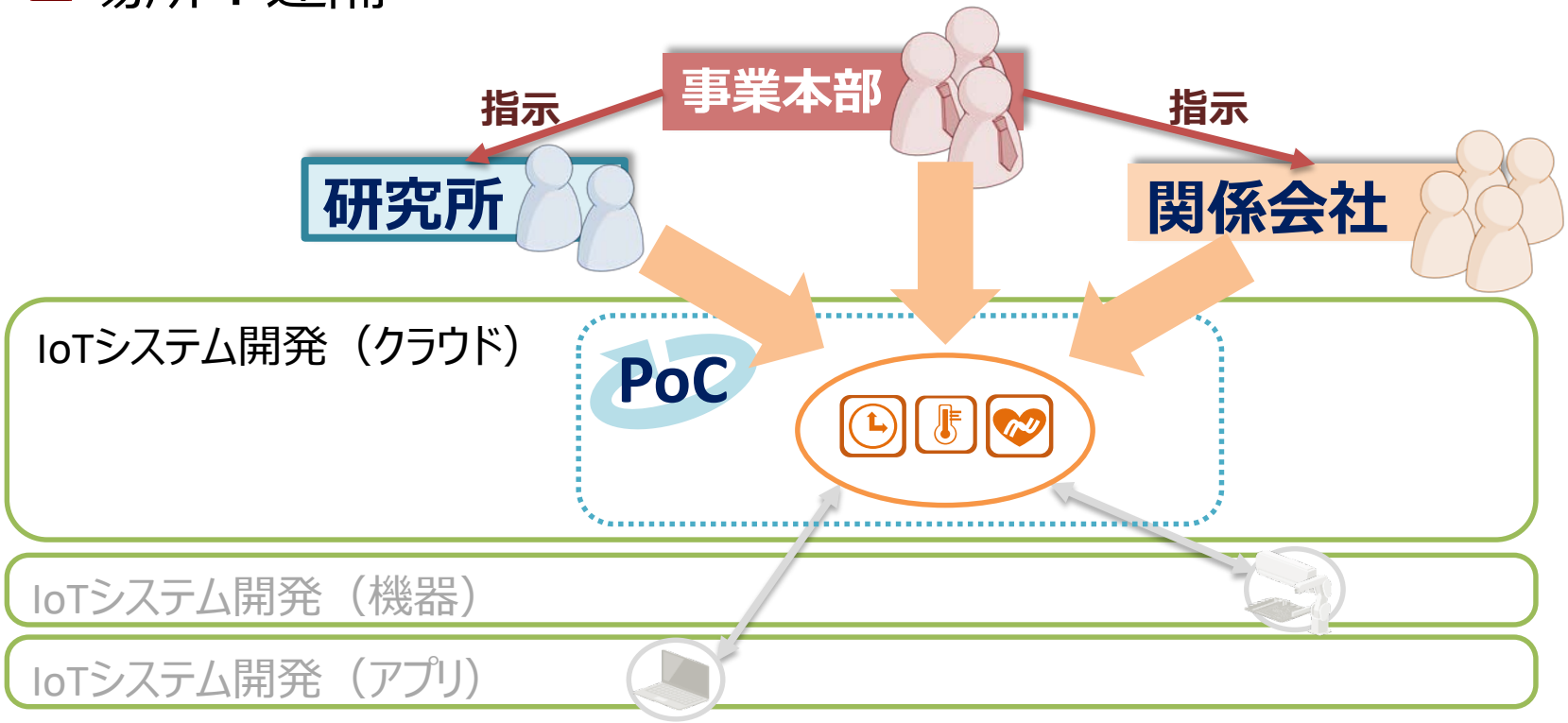
- 開発について
  - オンプレで稼働していたIoTシステムの、パブリッククラウドへの移行を目的としたプロト開発
    - 本開発ではAWS (Amazon Web Services)を利用
  - クラウドネイティブなアーキテクチャを検討、**アジャイル開発**にて動作確認しながら進めていくことに
  - **リグレッションテストの自動化検討**を担当(1人)
    - 機能(API)テスト部分

IoTシステムイメージ

複数組織またいだメンバでPoC実施

開発体制

- 人　：事業場所(設計など), 研究所(設計支援・テスト設計), 関係会社(実装・テスト)
- 場所：遠隔



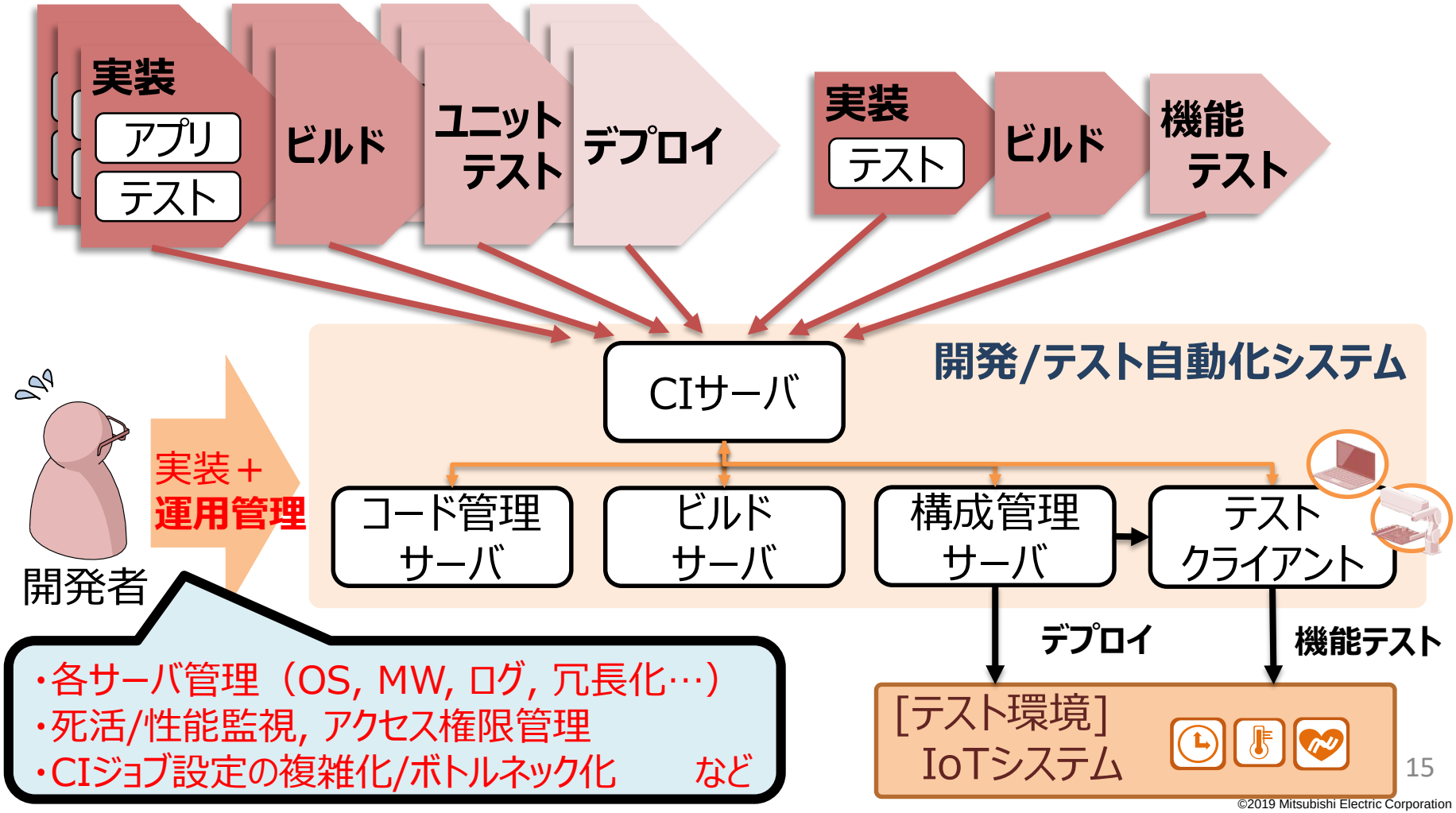
IoTシステム開発
リグレッションテスト自動化

自動化内容

課題

開発/テスト自動化システムは複雑・運用負荷高い

■開発プロセスと自動化システム

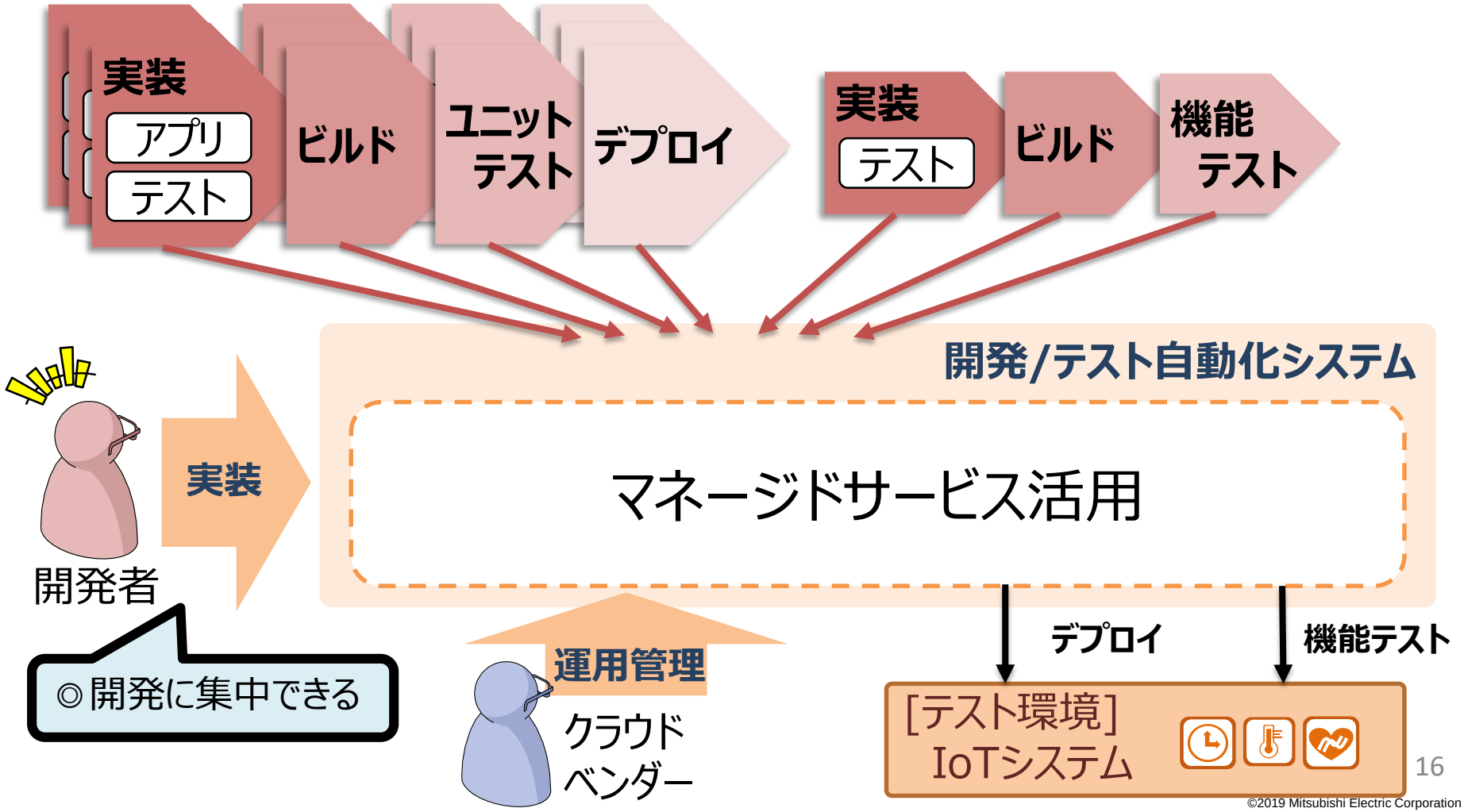


・各サーバ管理（OS, MW, ログ, 冗長化…）
・死活/性能監視, アクセス権限管理
・CIジョブ設定の複雑化/ボトルネック化　　など

解決アプローチ(1)

マネージドサービスによる自動化基盤で負荷抑制

■開発プロセスと自動化システム

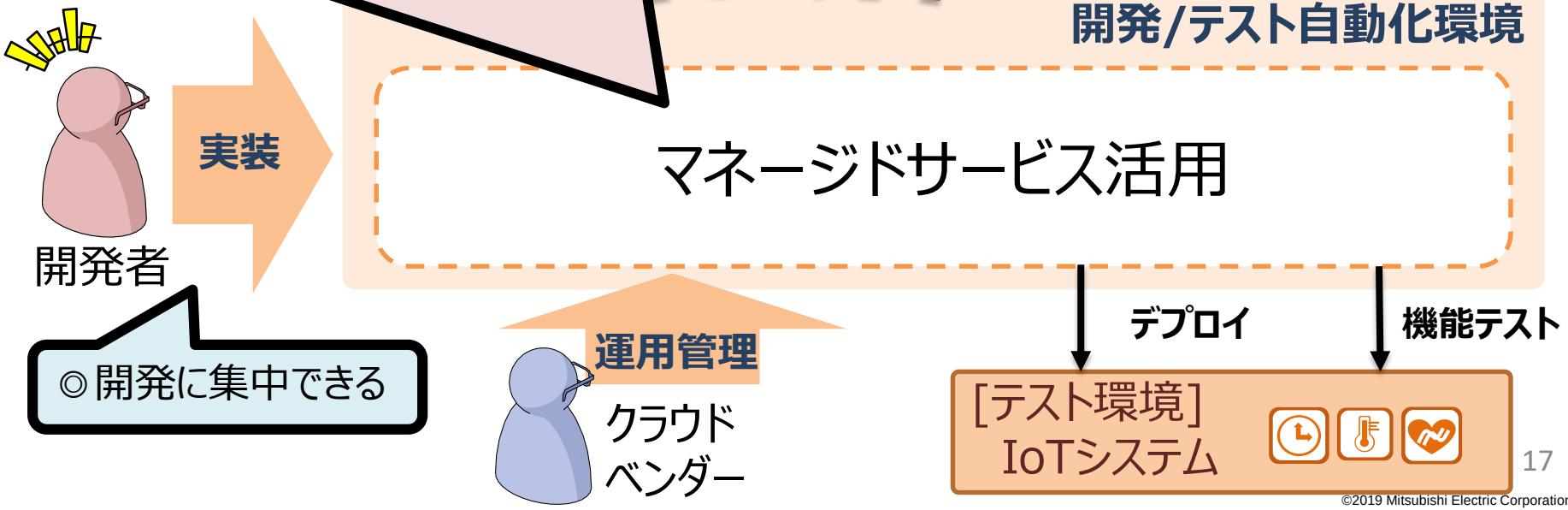


解決アプローチ(1)

マネージドサービスによる自動化基盤で負荷抑制

マネージドサービスですべて解決？

⇒NO



解決アプローチ(2)

機器側の独自仕様への対応を考慮した切り分け

- テスト自動化するための要件を整理
  - マネージドサービスでどこまで可能か？

要件

- API模擬
- 成否判定
- …

独自プロトコルなど考慮しなきゃいけなさそう…
連携拡大に向けてより柔軟な拡張性がほしい

- レポート機能
- アジャイル適用
- …

マッチするマネージドサービスはなさそうだ
保守を考えると作り込みはしたくない…

- 自動実行/トリガー
- サーバ管理
- …

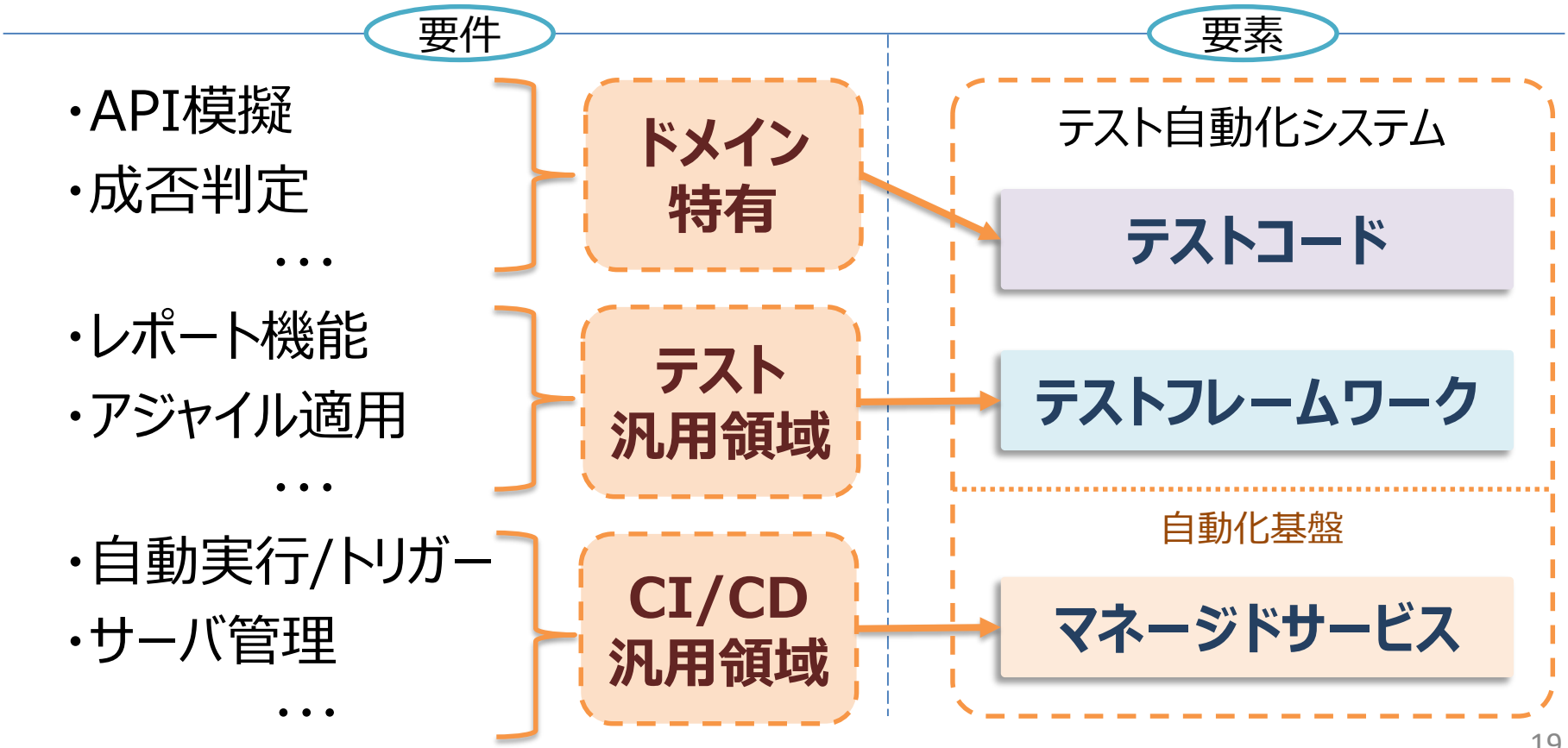
マネージドサービスで解決できそう◎

[*] CI：継続的インテグレーション　CD：継続的デリバリー

解決アプローチ(2)

機器側の独自仕様への対応を考慮した切り分け

- テスト自動化するための要件を整理

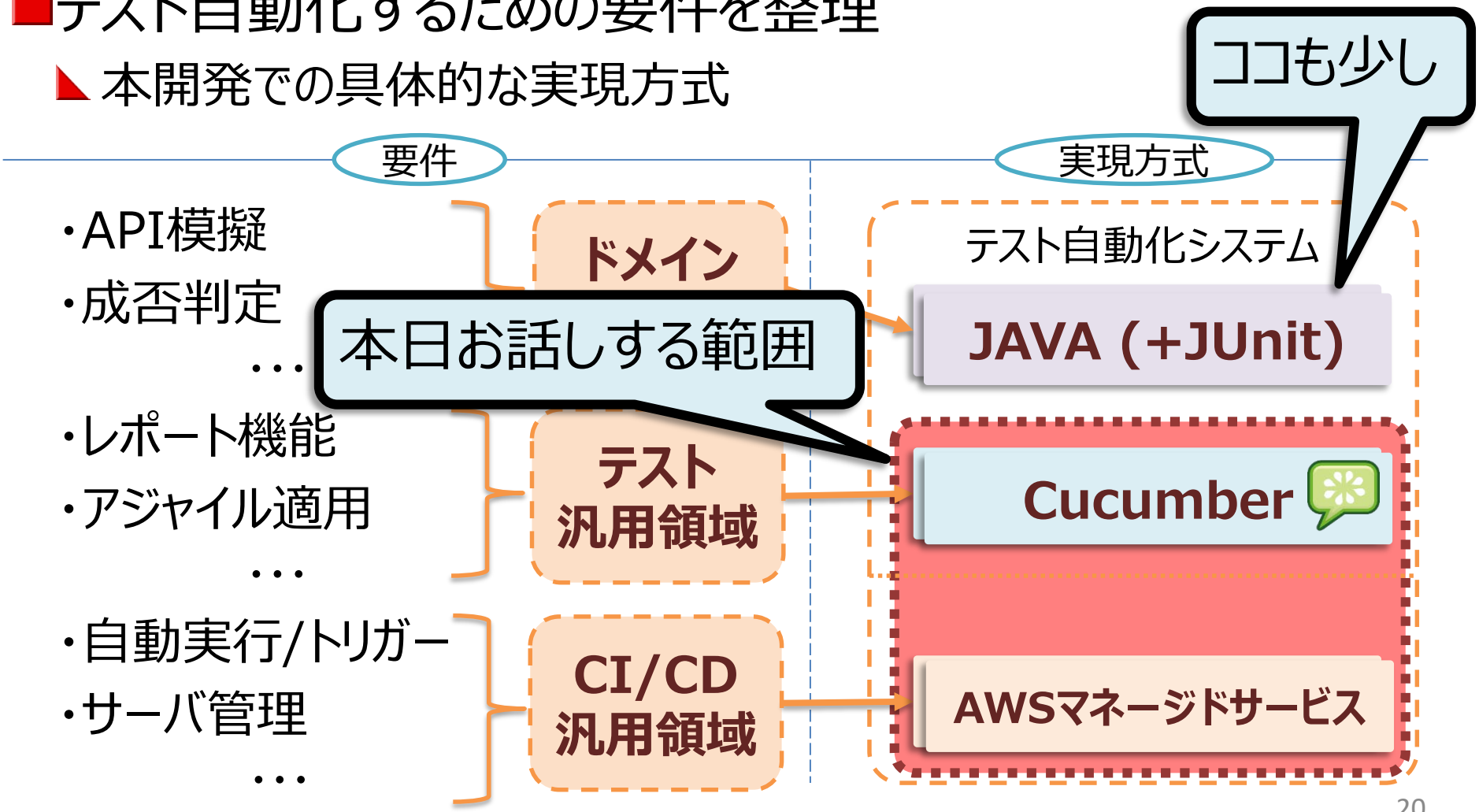


[*] CI：継続的インテグレーション　CD：継続的デリバリー

解決アプローチ(2)

機器側の独自仕様への対応を考慮した切り分け

- テスト自動化するための要件を整理
  - 本開発での具体的な実現方式



[*] CI：継続的インテグレーション　CD：継続的デリバリー

実施内容

テスト自動化システム

JAVA (+JUnit)

Cucumber

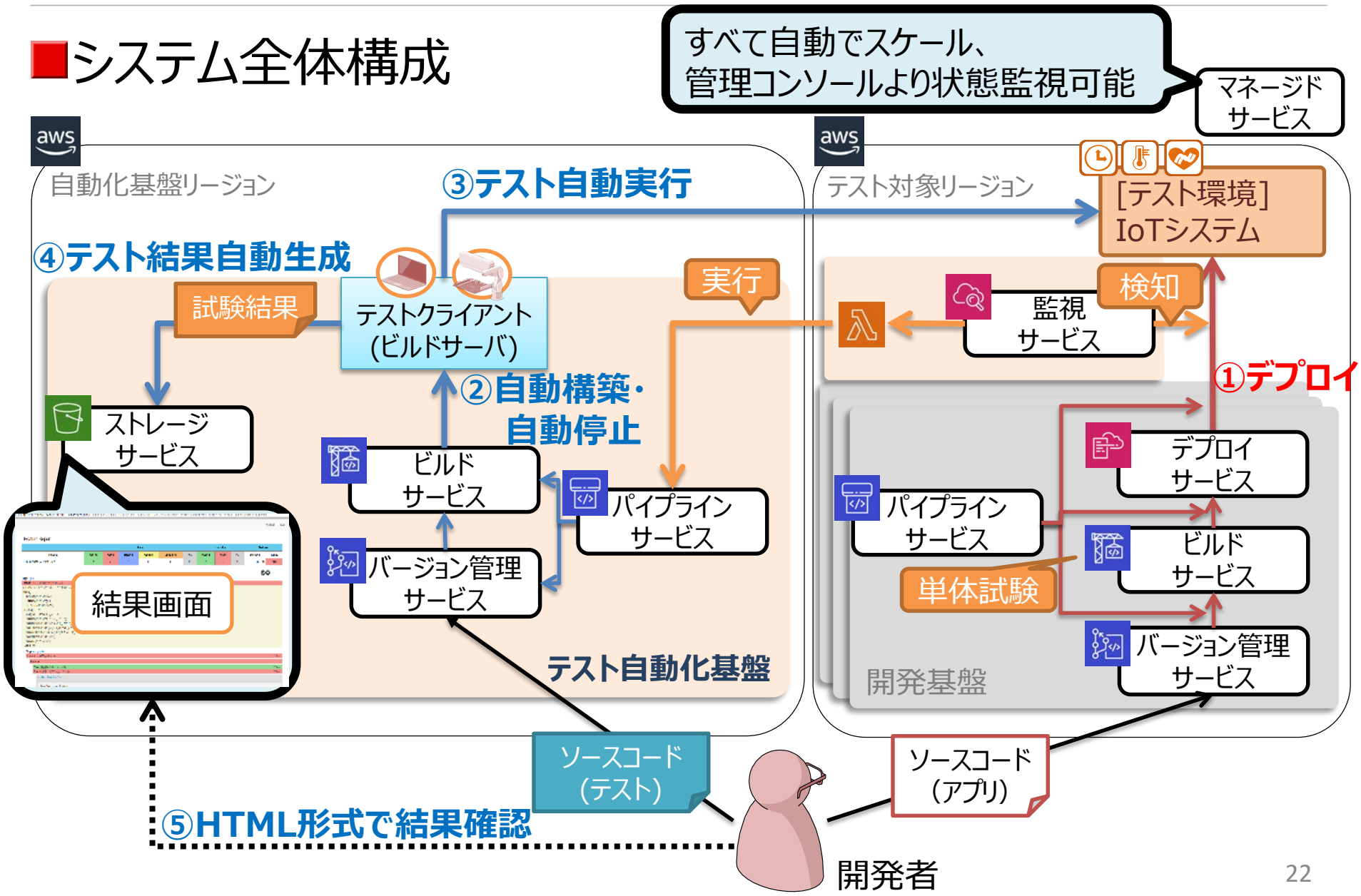
AWSマネージドサービス

要件

- **レポート機能**
  - 導入容易性
- **アジャイル適用**
  - 効率性
- **自動実行/トリガー**
  - 独立性・保守性
- **サーバ管理**
  - OS，MW，NW
  - 冗長化, 監視
  - ログ管理

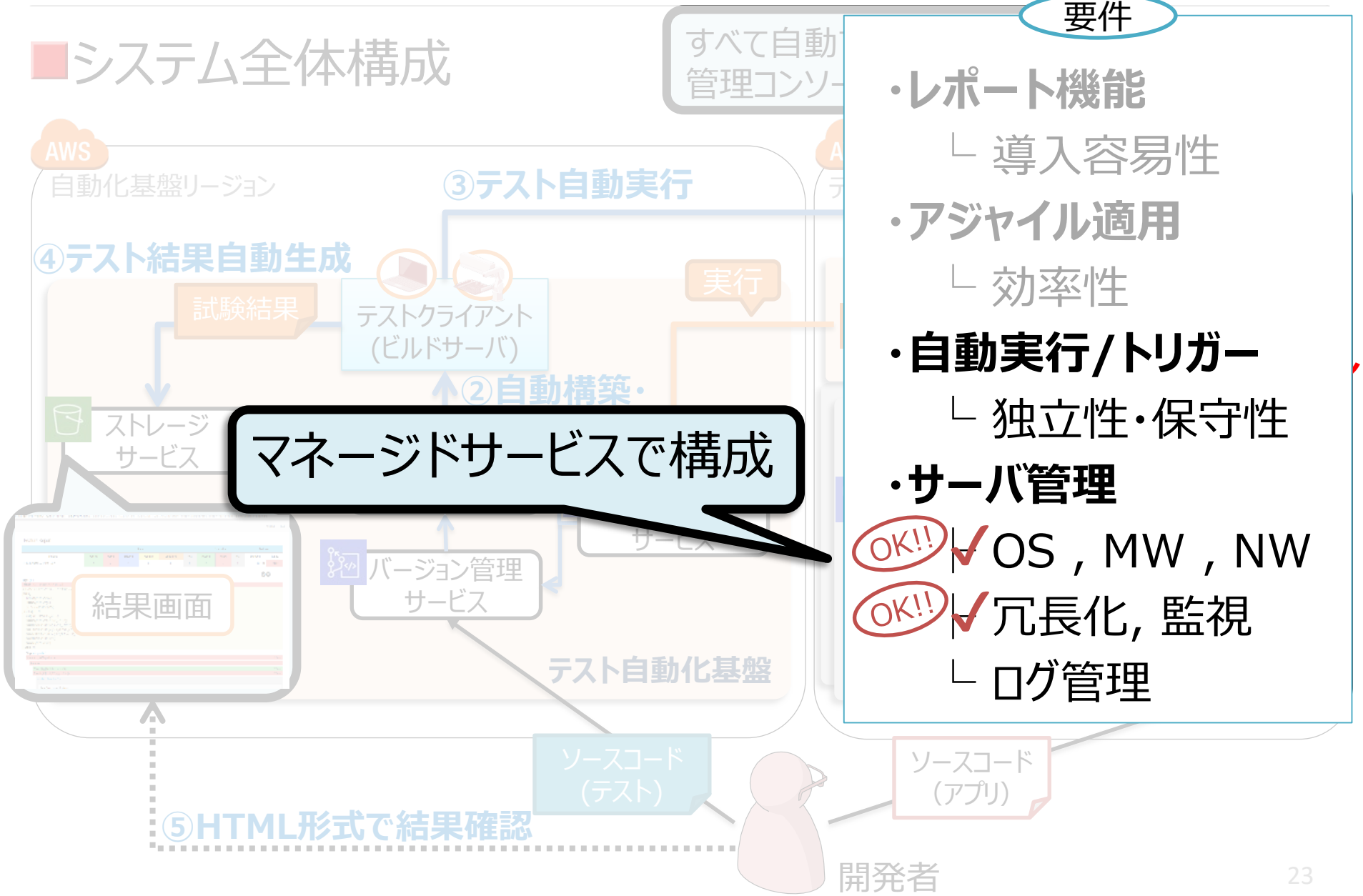
実施内容

システム全体構成



実施内容

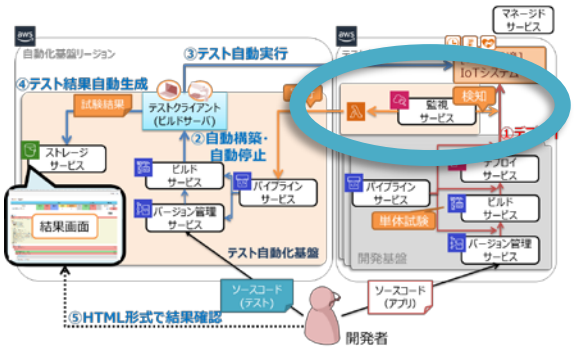
システム全体構成



要件

- **レポート機能**
  - 導入容易性
- **アジャイル適用**
  - 効率性
- **自動実行/トリガー**
  - 独立性・保守性
- **サーバ管理**
  - OK!! ✓ OS，MW，NW
  - OK!! ✓ 冗長化，監視
  - ログ管理

実施内容
本自動化基盤のポイント(1)



- 監視サービスによるトリガー
  - 構成情報の変化を検知してテスト実行
  - アプリ開発とテストのパイプラインを分離

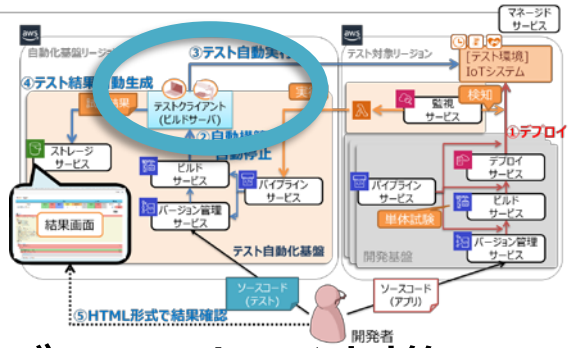
⇒開発基盤側の変更(追加/削除)に左右されず、**保守性**◎
分散したアーキテクチャにも対応可能

- マネージドサービスの設定変更の検知・テスト実行も可能
  - 各サービスの管理用APIの実行を監視している (CloudTrail)

⇒テスト対象側の**サーバレス/マネージドサービス特有の設定**や
障害原因となりやすい設定値の変更を監視可能

  - 例：Lambdaのメモリ容量やタイムアウト値
    DynamoDB（データベースサービス）のキャパシティ　など

実施内容
本自動化基盤のポイント(2)



- テスト実行ログ
  - ビルドサーバの実態はコンテナ
  - ビルドサービス(AWS CodeBuild)は、ログ管理サービス(CloudWatch Logs)とデフォルトで連携

⇒ビルドサーバでの標準出力がログとして自動保存されるため、API実行ログを標準出力するだけでよく、**ログ管理が容易に**

ビルドサービスの
管理コンソール画面例

実施内容
本自動化基盤のポイント(2)

- テスト実行ログ
  - ビルドサーバの実態はコンテナ
  - ビルドサービス(AWS CodeBuild)は、
    ログ管理サービス(CloudWatch Log

⇒ビルドサーバでの標準出力がログとしてス
に自動保存・管理されるため、API実行
テスト実行時に出力すると**ログ管理が容**

要件

- **レポート機能**
  - 導入容易性
- **アジャイル適用**
  - 効率性
- **自動実行/トリガー**
  - OK!! ✓ 独立性・保守性
- **サーバ管理**
  - ✓ OS , MW , NW
  - ✓ 冗長化, 監視
  - OK!! ✓ ログ管理

実施内容

テスト自動化システム

JAVA (+JUnit)

Cucumber

AWSマネージドサービス

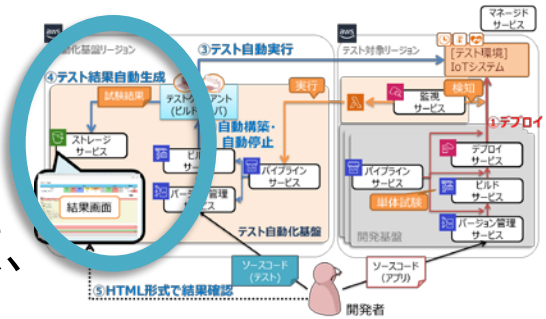
要件

- **レポート機能**
  - 導入容易性
- **アジャイル適用**
  - 効率性
- **自動実行/トリガー**
  - ✓独立性・保守性
- **サーバ管理**
  - ✓OS，MW，NW
  - ✓冗長化，監視
  - ✓ログ管理

実施内容
テストフレームワークのポイント(1)

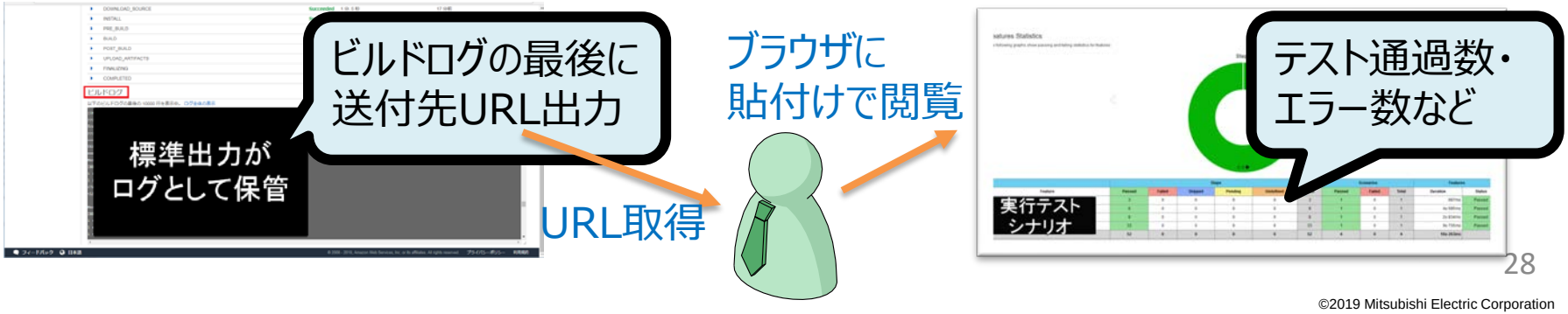
レポーティング

- Cucumber利用
- 実行したテストシナリオ別のテスト通過数やエラー数、各ステップでかかった時間など確認可能
- レポートはHTMLやJSON形式で出力可能(プラグイン利用)



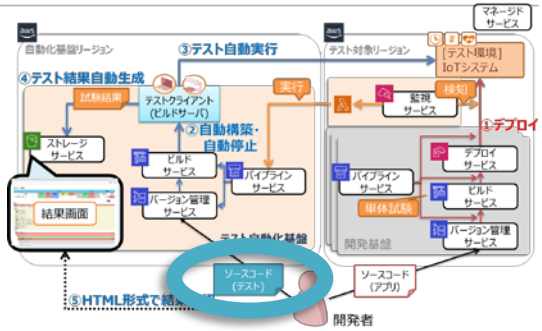
⇒静的ホスティング・IPアクセス制限させたストレージサービス(S3)に送付することで、**レポートの閲覧・管理が容易に**

開発者が別途準備しなくても、最新のテスト結果を常に確認可能



実施内容
テストフレームワークのポイント(2)

- Cucumber



  - 振る舞い駆動でテストファーストな開発を実現するために考案されたフレームワーク
  - テストケースを自然言語で記述
    - 顧客と技術者との間の、言葉と理解の共有容易化
    - そのままテストコードとなるのでPull Requestベースのやり取りが可能

⇒要件とテスト実装の乖離を抑制　・・・　**手戻り削減**
　余計なドキュメント作成減らせる　・・・　**オーバーヘッド抑制**

実施内容
テストフレームワークのポイント(2)

Gherkin書式

✓ 下記のようなファイルがそのままテストコードとして扱われる

sample.feature

```
@feature-tag
Feature: ログインしてユーザを識別できる

  @scenario-tag01
  Scenario: ユーザ登録してログインする
    Given: “ユーザ登録”ページを表示している
    When: “ログイン名”フォームに”melco”と入力する
      And: “E メール”フォームに”xxxxxxxxxx@xxx.xx.xxx”と入力する
      And: “新規作成”ボタンを押下する
    Then: “melco ユーザが作成されました。”と表示されていること

  @scenario-tag02
  Scenario: ログイン済みユーザの情報表示する
  …
```

Whenなどのステップの
組み合わせで記載する

参考：Cucumber公式Webサイト (https://cucumber.io/)

実施内容
テストフレームワークのポイント(3)

保守性の高いテストコード記述が可能

- Cucumberの応用的な使い方
  - 各ステップを振る舞い・データのまとまりとしてモデル化して記載すると、**キーワード駆動として保守性の高い**テスト記述とすることが可能

    例：**“API種別を表すキーワード”＋”データセットを表すキーワード”**とすると、**顧客側と認識共有しやすく、コード構造化もしやすい**
    - Stateパターンとして、データ部を状態として実装するとテストケースが増えてもロジックに変更なく追加できる…など

 # テストケース記載例

```
Scenario: ユーザログインする
    When: ログイン用APIを”正常な顧客Aデータ”で実行
      #API種別：ログイン用APIを<>で実行
      #データ　　：正常な顧客Aデータ
```

実施内容
テストフレームワークのポイント(3)

保守性の高いテストコード記述が可能

- キーワード駆動テストとしての活用
  - テストケースの”Given”や”When”,
    のまとまりとしてモデル化して記載する
    性の高いテスト記述とすることが可能
    例：対象APIがSOAP(Simple Obj
    にデータボディのXMLで詳細データ
    “API種別を表すキーワード”＋”テ
    とすると、**顧客側と認識共有しや**
    - Stateパターンとして、データ部を状態
      もロジックに変更なく追加できる…な

```
# テストケース記載例
Scen
  W
                                    行
  #データ     ：正常な顧客Aデータ
```

目指していた自動化
が実現できた

Excellent!!

要件

- **レポート機能**
  - OK!! ✓導入容易性
- **アジャイル適用**
  - OK!! ✓効率性
- **自動実行/トリガー**
  - ✓独立性・保守性
- **サーバ管理**
  - ✓OS，MW，NW
  - ✓冗長化，監視
  - ✓ログ管理

実施内容（追加）

テスト自動化システム

JAVA (+JUnit)

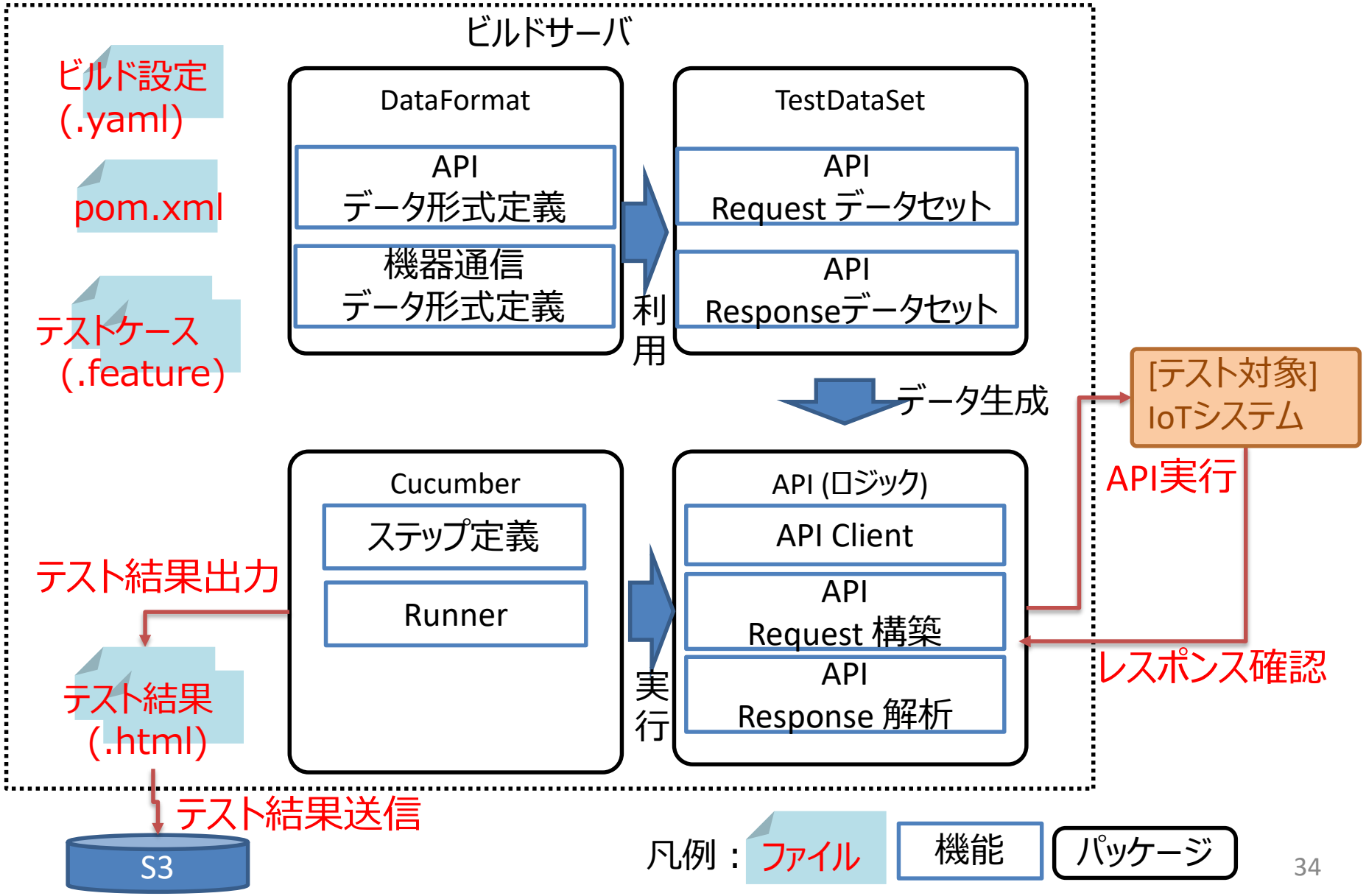
Cucumber

AWSマネージドサービス

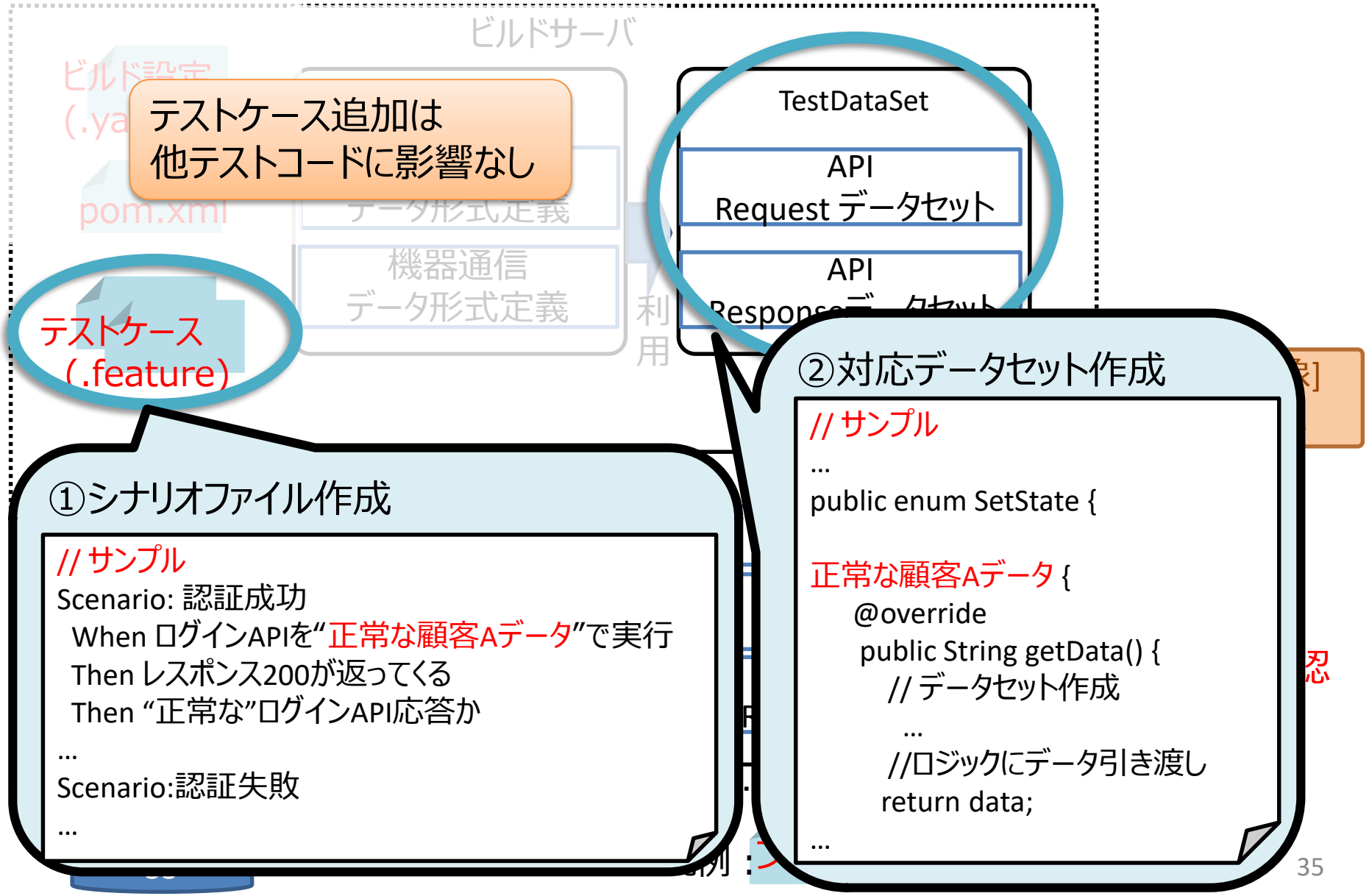
要件

・API模擬、成否判定
　└ 保守性

実施内容
テストコード構成イメージ



実施内容
テストコードのポイント



MITSUBISHI ELECTRIC
Changes for the Better

効果とまとめ

～30分

開発者の負荷抑制しながらテスト自動化を実現

◎テスト自動化基盤にマネージドサービス活用

- **回帰テストの自動化を実現**
  - 手動で確認していたテストが、気がつけば終わっている状態になった
- **自動化環境の運用負荷抑制**
  - テストコード以外はほぼ気にしなくてよくなった

◎マネージドサービス利用コスト

- 毎日1回テスト実行(5分)、テストコード・レポートなどそれぞれ月10GB以下すると、
  - **5USD/月以下**(マネージドサービスそれぞれ月1USD未満)

△テストコードの実装負荷が高め

MITSUBISHI ELECTRIC
Changes for the Better

IoTシステム開発
リグレッションテスト自動化

製品開発
移行

IoTシステム開発
受け入れテスト自動化

Webシステム開発
テスト管理との統合

2017 *2018* *2019*

MITSUBISHI ELECTRIC
Changes for the Better

IoTシステム開発

受け入れテスト自動化

開発背景など

IoTシステムの開発で受け入れテスト自動化検討

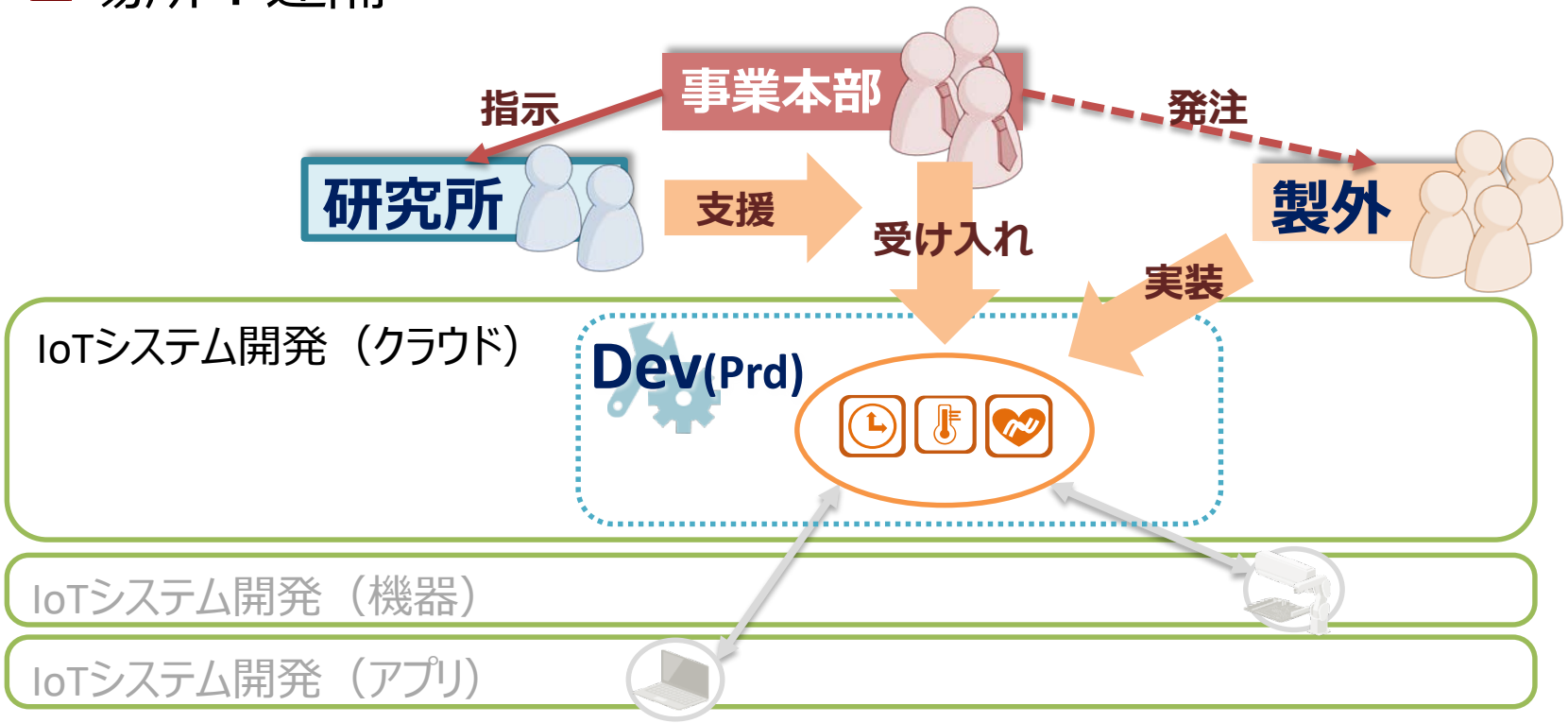
- 開発について
  - IoTシステムのクラウド側開発
    - 本開発ではAWS (Amazon Web Services)を利用
  - 製外実装による、リリースを複数回に区切った反復開発
  - 機器依存でクラウド側の状態・応答が複雑に変化するため、テスト手順が複雑・手動実施負荷が高い
  - **受け入れテストの自動化検討**を担当(2人)
    - 機能(API)テスト部分

IoTシステムイメージ

複数組織またいだメンバで開発実施

開発体制

- 人　：事業場所(設計・受入テスト), 研究所(テスト自動化設計), 製外(実装・テスト)
- 場所：遠隔



IoTシステム開発

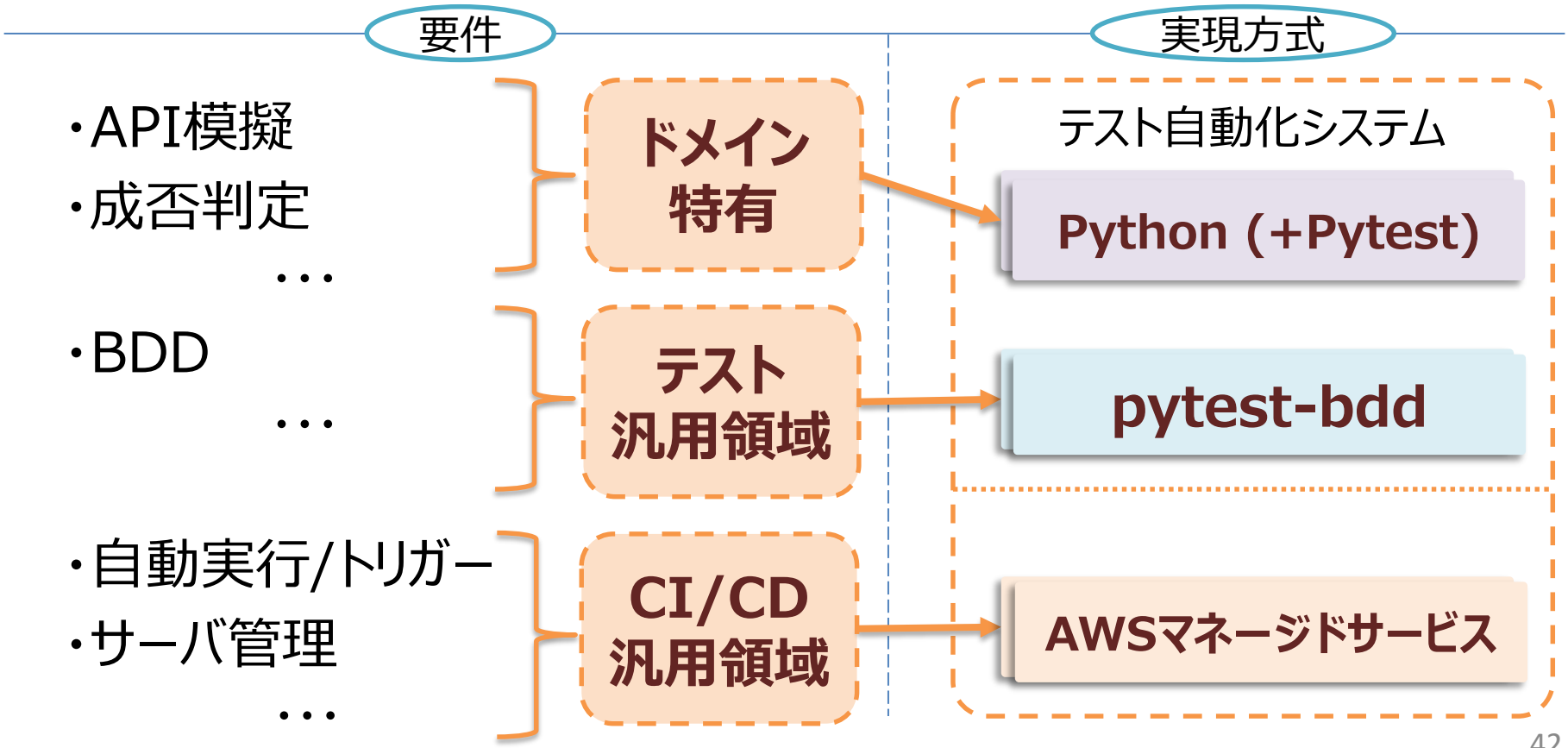
受け入れテスト自動化

自動化内容

課題と解決アプローチ

マネージドサービス利用で自動化負荷軽減

- 基本的に同じアプローチで実施
  - 開発側に合わせて実装はPython, フレームワークも合わせる



[*] CI：継続的インテグレーション　CD：継続的デリバリー

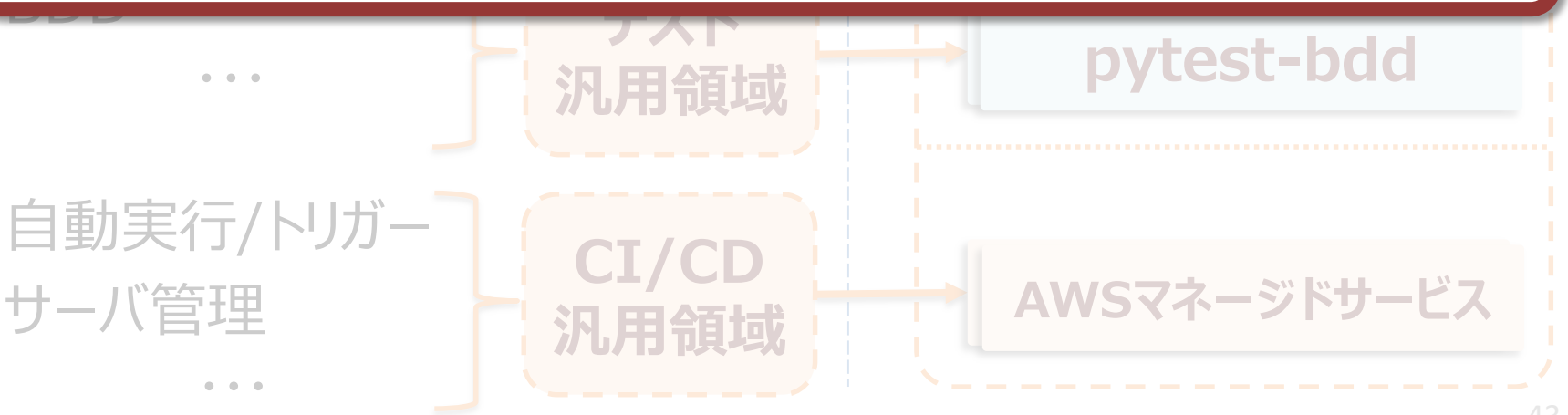
課題と解決アプローチ

マネージドサービス利用で自動化負荷軽減

- 基本的に同じアプローチで実施

pytest-bdd

- ✓ Pythonで利用可能なBDDライブラリ
- ✓ Pytestのプラグインなので、レポート出力のpytest-htmlや、並列テストのpytest-xdistとそのまま連携可能
- ✓ CucumberのJSON形式レポートを出力することが可能



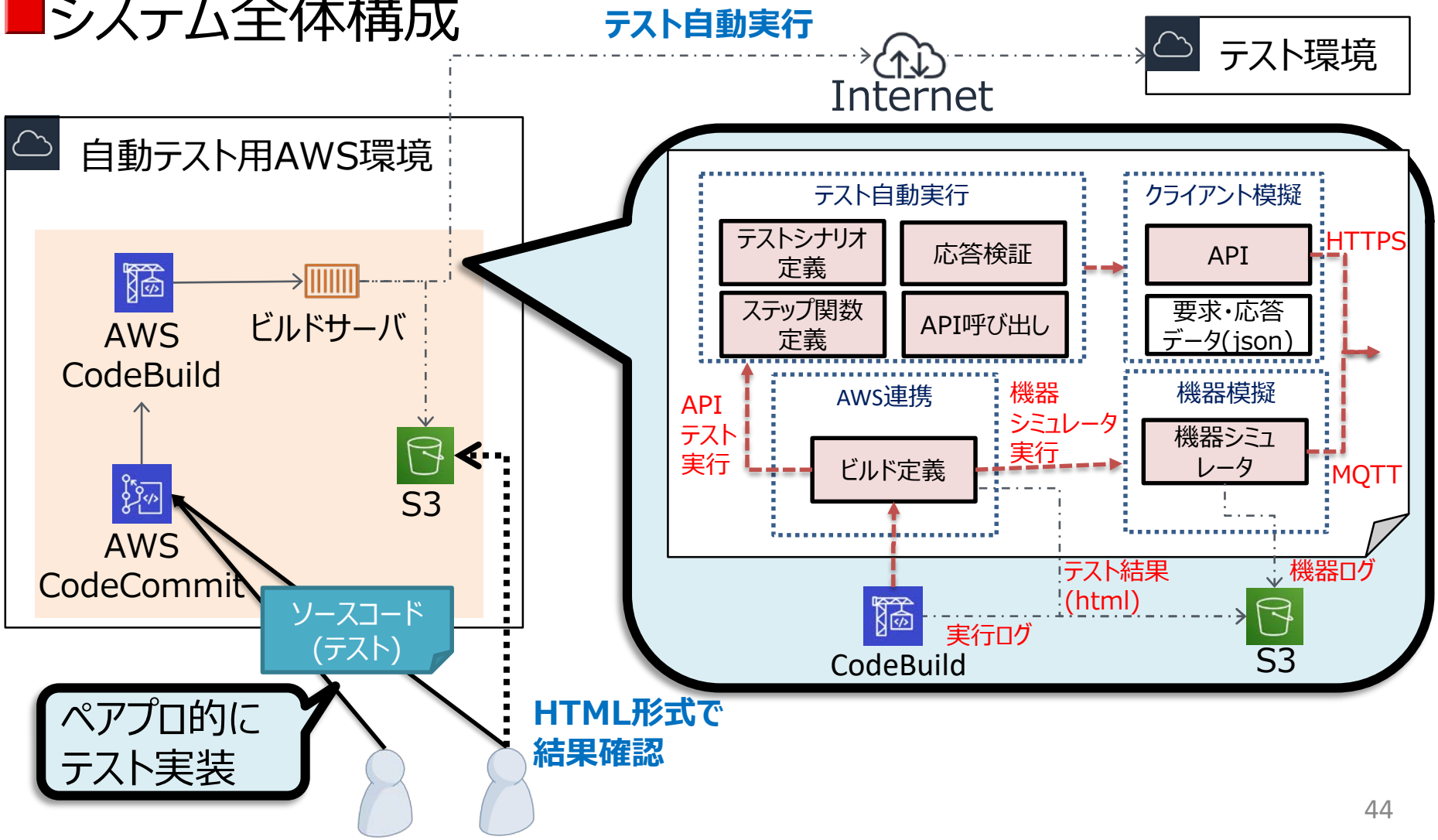
[*] CI：継続的インテグレーション　CD：継続的デリバリー

MITSUBISHI ELECTRIC
Changes for the Better

実施内容

ビルドサーバ内で機器シミュレータも実行

システム全体構成



効果とまとめ

～40分

複雑なテスト手順も自動化により効率化

◎テスト自動化基盤にマネージドサービス活用
- 自動化基盤は前例そのまま活用で効率的に
- 導入時はペアプロでのテストコード実装でスムーズにチーム内展開

△テストケース追加時の負荷がそれなりにある
- 要求・応答用データの作成に手間がかかる
- 複数人でステップ追加していくと、重複部分が増えていく

△非機能試験にそのままスケールしない
- ビルドサーバ(コンテナ)内でクライアント・機器すべて模擬

△エラー解析に時間がかかる
- クライアント・機器模擬など疎結合で改善しやすい反面、ログ分析が面倒

MITSUBISHI ELECTRIC
Changes for the Better

IoTシステム開発
リグレッションテスト自動化

製品開発
移行

IoTシステム開発
受け入れテスト自動化

Webシステム開発
テスト管理との統合

2017 2018 2019

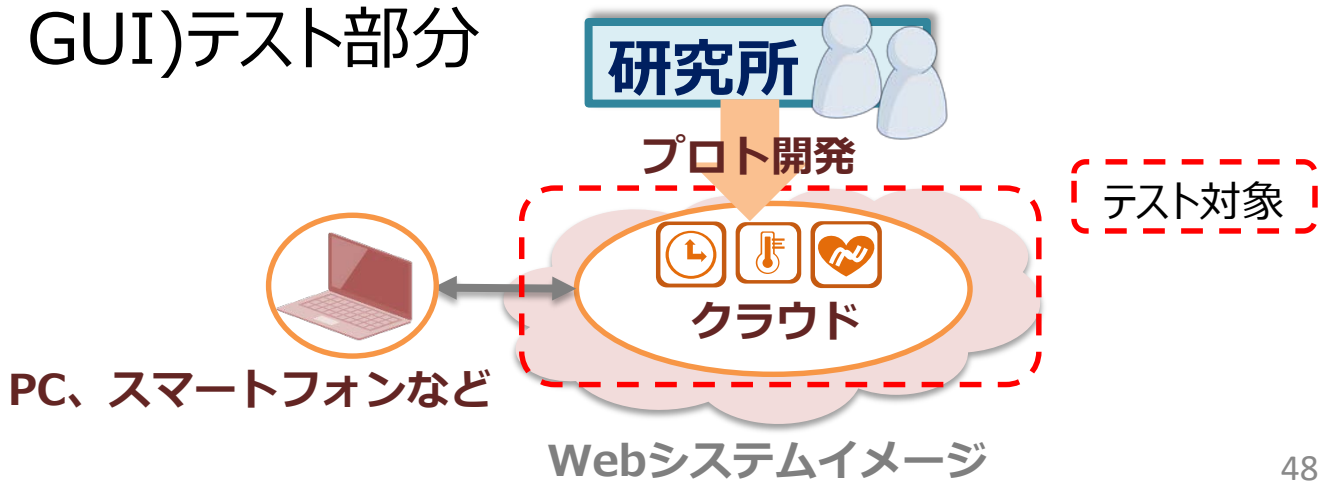
MITSUBISHI ELECTRIC
Changes for the Better

Webシステム開発
テスト管理との統合

開発背景など

Webシステムのプロト開発でテスト自動化検討

- 開発について
  - データ可視化・分析関連のWebシステム
    - 本開発ではAWS (Amazon Web Services)を利用
  - 研究所内に閉じた、リリースを複数回に区切った反復開発
    - 自由度高くツール／サービス利用可能
  - **テスト自動化とテスト管理との統合検討**を担当
    - 機能(API, GUI)テスト部分



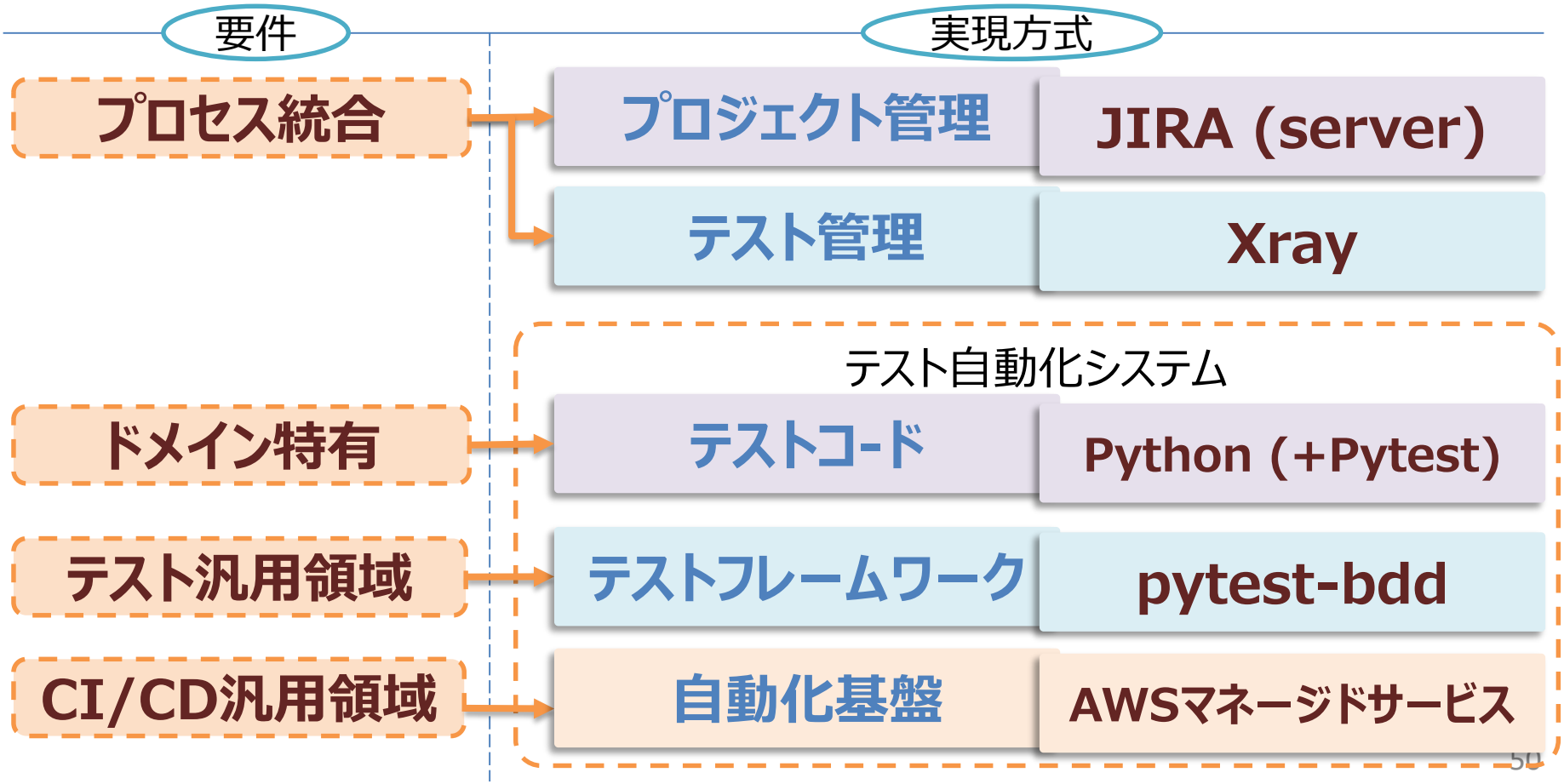
Webシステム開発
テスト管理との統合

実施内容

課題と解決アプローチ

自動テストとプロジェクト管理ツールの統合

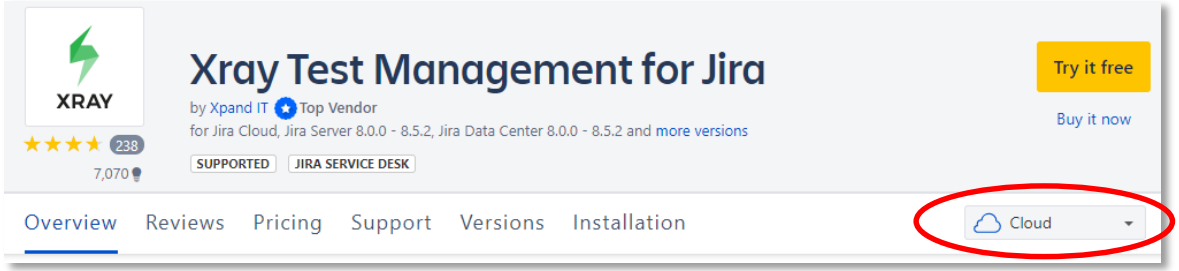
- 自動テストは同じアプローチ、テスト管理にJIRA利用
  - テスト結果のシームレスな管理を目指す



課題と解決アプローチ

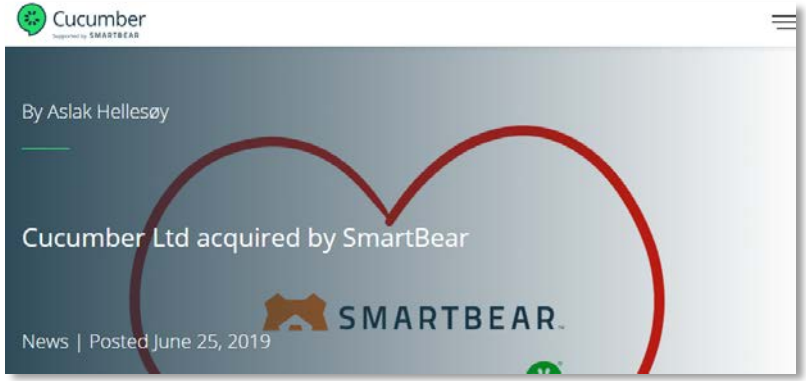
テスト管理ツール

- ✓ 検討当時はJIRA Server版にしかXray対応してなかった
  ⇒ 現在はCloud版でも利用可能な模様



https://marketplace.atlassian.com/apps/1211769/xray-test-management-for-jira?hosting=cloud&tab=overview

- ✓ CucumberがSmartBearに買収された(2019/6)ので、今後はZephyrも注目？

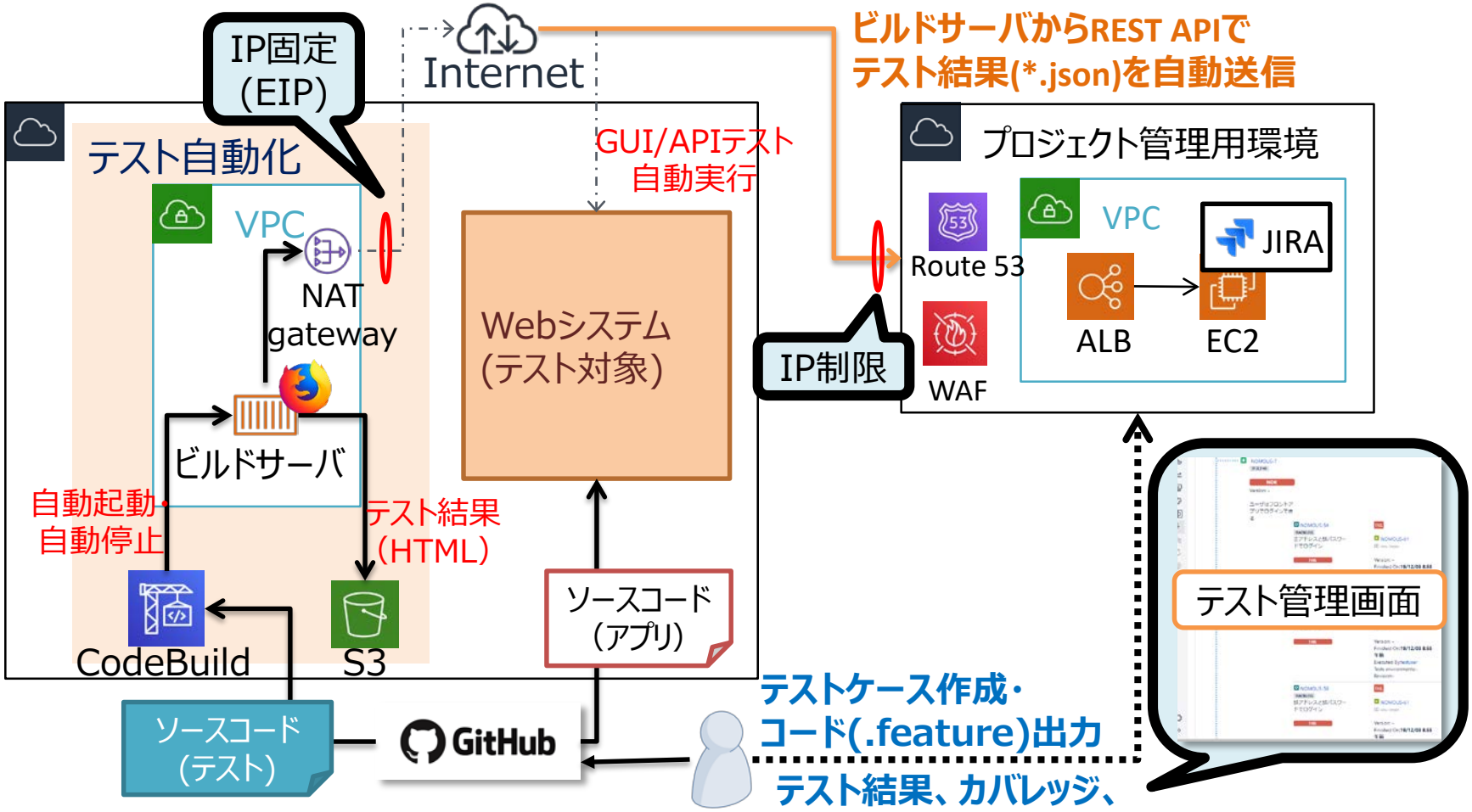


https://cucumber.io/blog/news/cucumber-acquired-by-smartbear/

実施内容

テスト結果を自動で集計、トレーサビリティ確保

システム全体構成



実施内容
テスト管理との統合ポイント

テスト結果管理・集計の負荷を削減

画面例

※画面は発表用のサンプル

MITSUBISHI ELECTRIC
Changes for the Better

効果とまとめ

要件に紐づいたテスト状況を管理可能に

◎テスト自動化基盤にマネージドサービス活用
- 自動化基盤は前例そのまま活用で効率的に
- テスト自動化とプロジェクト管理を統合、テスト管理の負荷を削減

△プロジェクト管理サーバの管理負荷
- Cloud版のJIRA+Xrayなど利用すれば負荷削減できるかも

△プロジェクト管理の設定、テスト管理連携の設定・管理負荷
- 色々できる反面、設定項目も多くて少し煩雑

現在も試行錯誤中…

事例全体

今後の課題

MITSUBISHI ELECTRIC
Changes for the Better

今後の課題

～50分

継続的テストを実現していきたい

- 自動化対象
  - 機器含めたE2Eテスト、非機能テストの自動化
- 導入時
  - テストコード実装負荷
    - プロジェクト計画時にテスト自動化のコストも意識
- 保守／展開時
  - テスト・テスト対象複雑化によるエラー発生時の解析難化
    - 非同期のテストなど
    - テストでもAPMやログ分析などモニタリング技術との連携
  - テスト自動化基盤の運用ルール・ガバナンス
    - 自動化基盤自体のログは？セキュリティは？……
      - 自動化担当者まかせでいいの？
  - 使われない・保守されなくなることも

というわけで、
後半に続きます

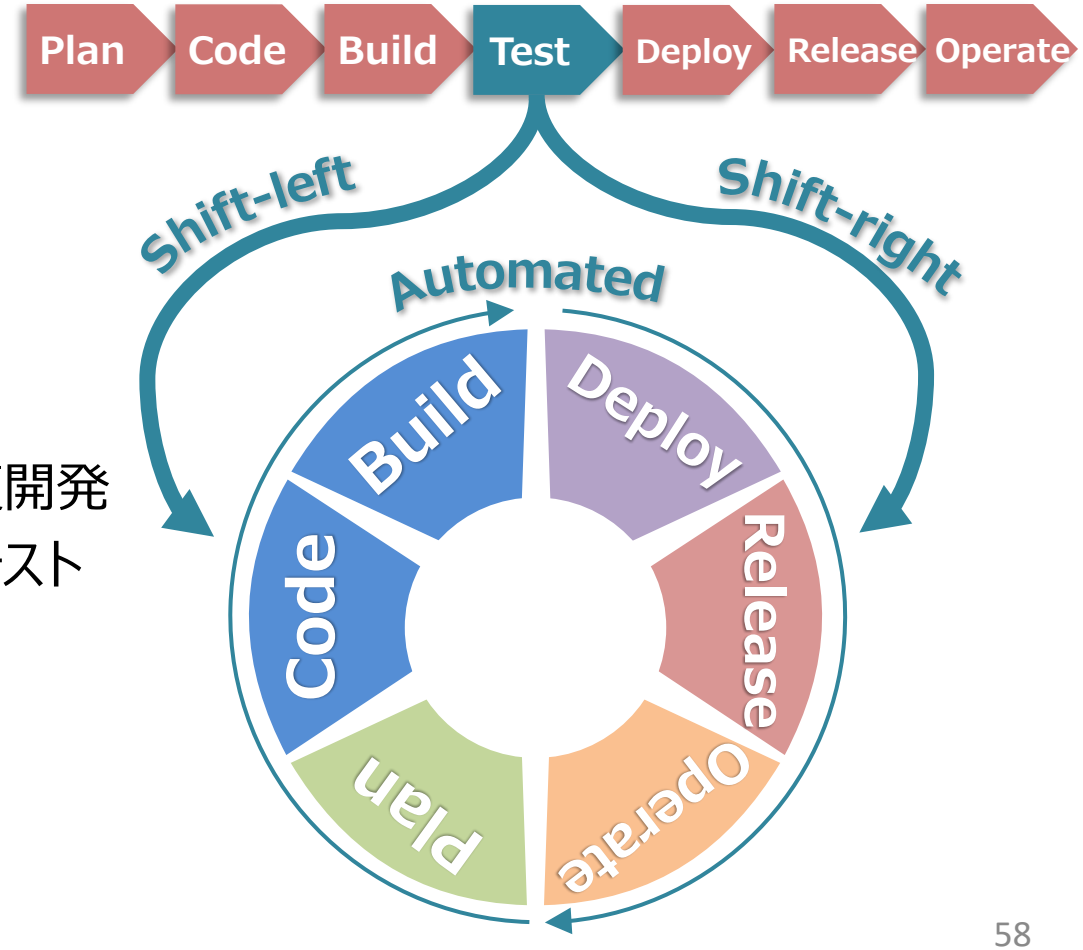
継続的テストへの課題

MITSUBISHI ELECTRIC
Changes for the Better

継続的テストとは？

様々なテスト観点をまとめたもの

- 明確な定義はないが、以下の３つを含んだ考え方
  - **Automated/CI**：
    開発環境の変化をトリガーとして実行され、他の開発プロセスに組み込まれる
    ※独立して追加されるものではない
  - **Shift-left**：
    より上流工程でのテスト
    - 動くものをテストしながら反復開発
    - 非機能要件も早い段階でテスト
  - **Shift-right**：
    本番環境上でのテスト
    - A/Bテスト、カナリアテスト…
    - カオスエンジニアリング



話の流れ

- 課題の分類観点を抽出
  - 取り組みから
  - 世の中の状況から

網羅的にはならずとも、
ある程度俯瞰的に見たい

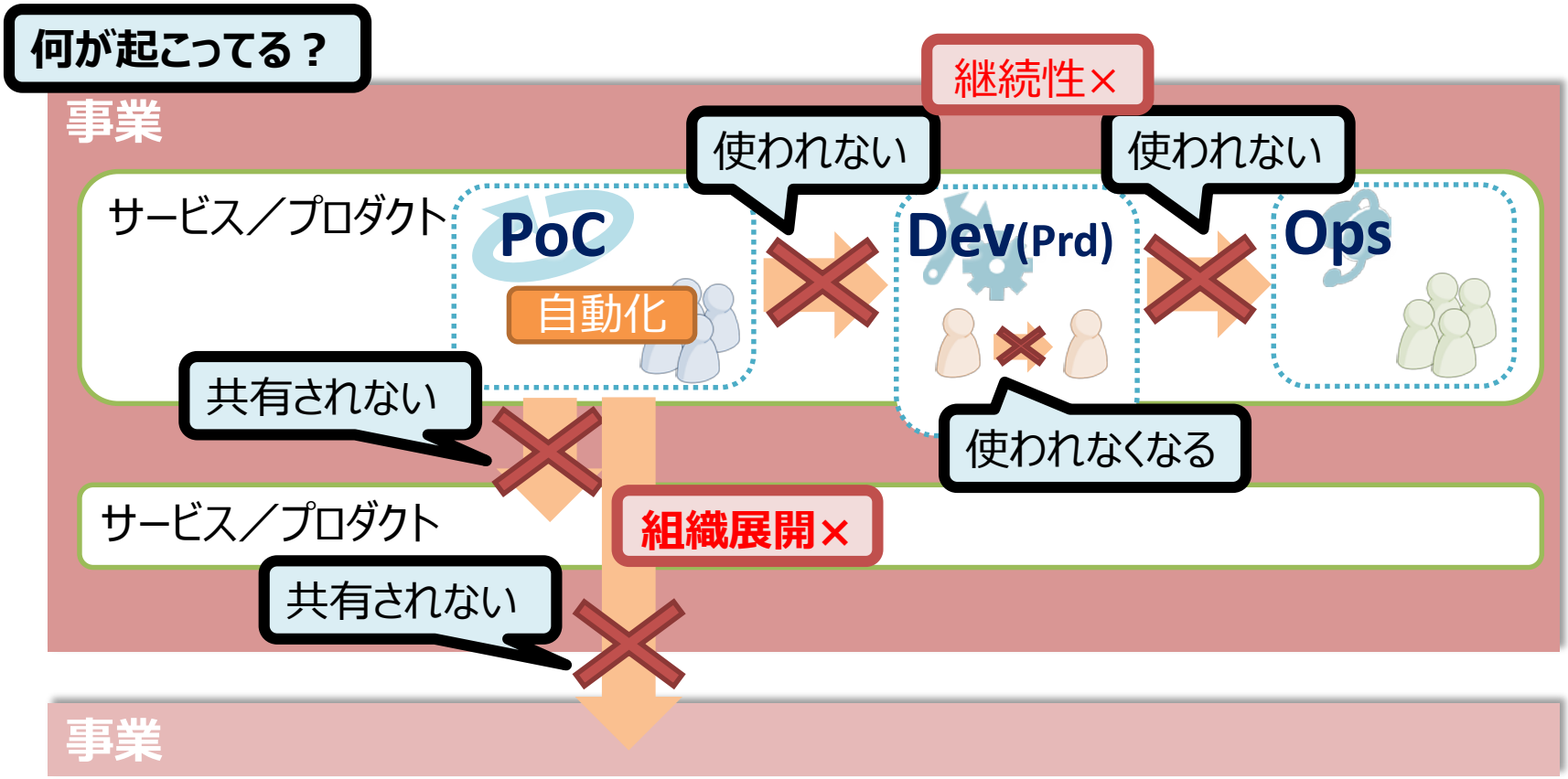
- 分類観点のまとめ
- 課題検討
- 継続的テスト実現に向けてすべきこと

MITSUBISHI ELECTRIC
Changes for the Better

課題の分類観点を抽出：取り組みから

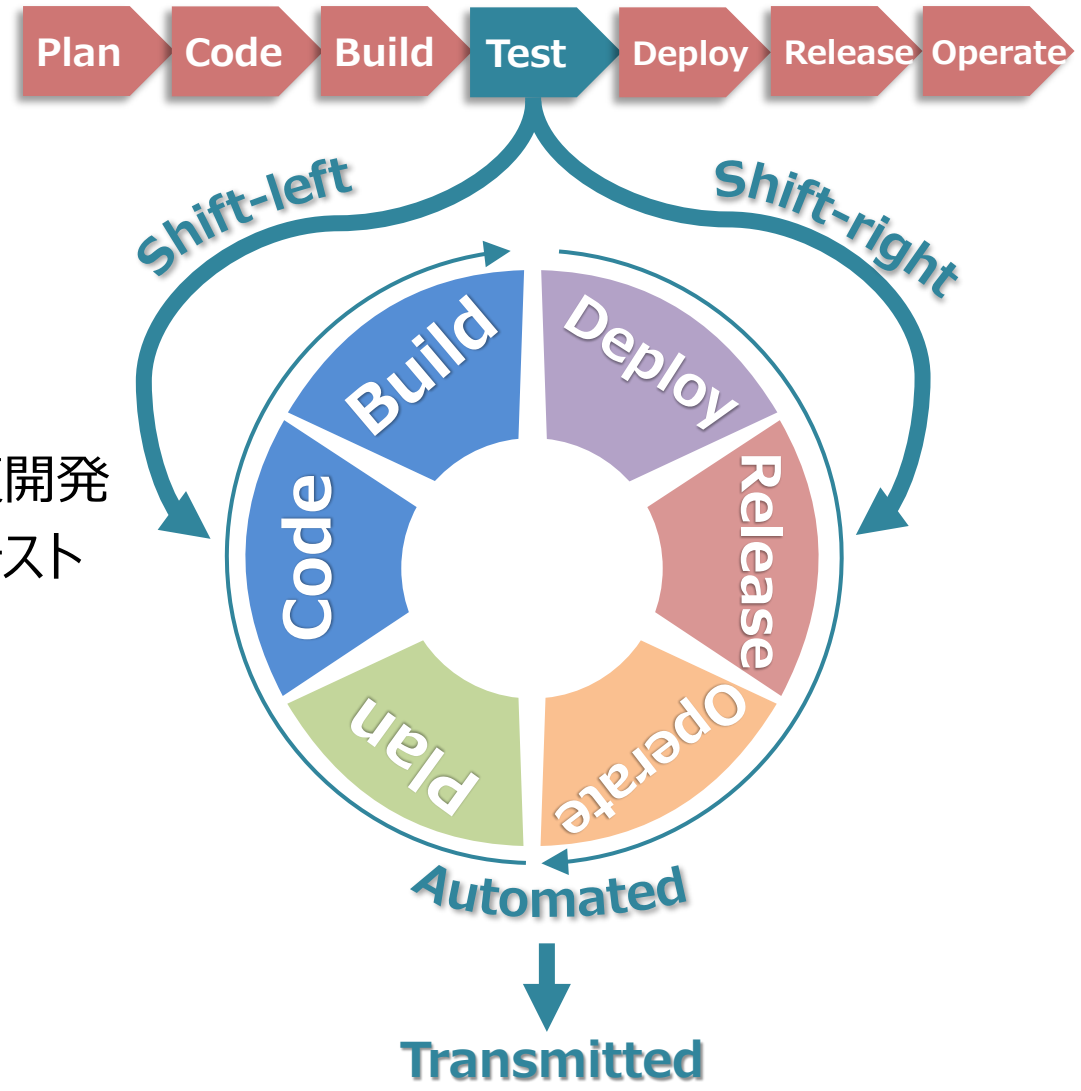
プロト開発以降で活用されない、横展開されない

- サービス開発内で継続的に実施されない
- サービス／事業を跨いだ組織的な展開ができない



本発表における継続的テスト

- **Automated/CI**：
  開発環境の変化をトリガーとして実行され、他の開発プロセスに組み込まれる
  ※独立して追加されるものではない
- **Shift-left**：
  より上流工程でのテスト
  - 動くものをテストしながら反復開発
  - 非機能要件も早い段階でテスト
- **Shift-right**：
  本番環境上でのテスト
  - A/Bテスト、カナリアテスト…
  - カオスエンジニアリング
- **Transmitted**：
  横展開可能なテスト

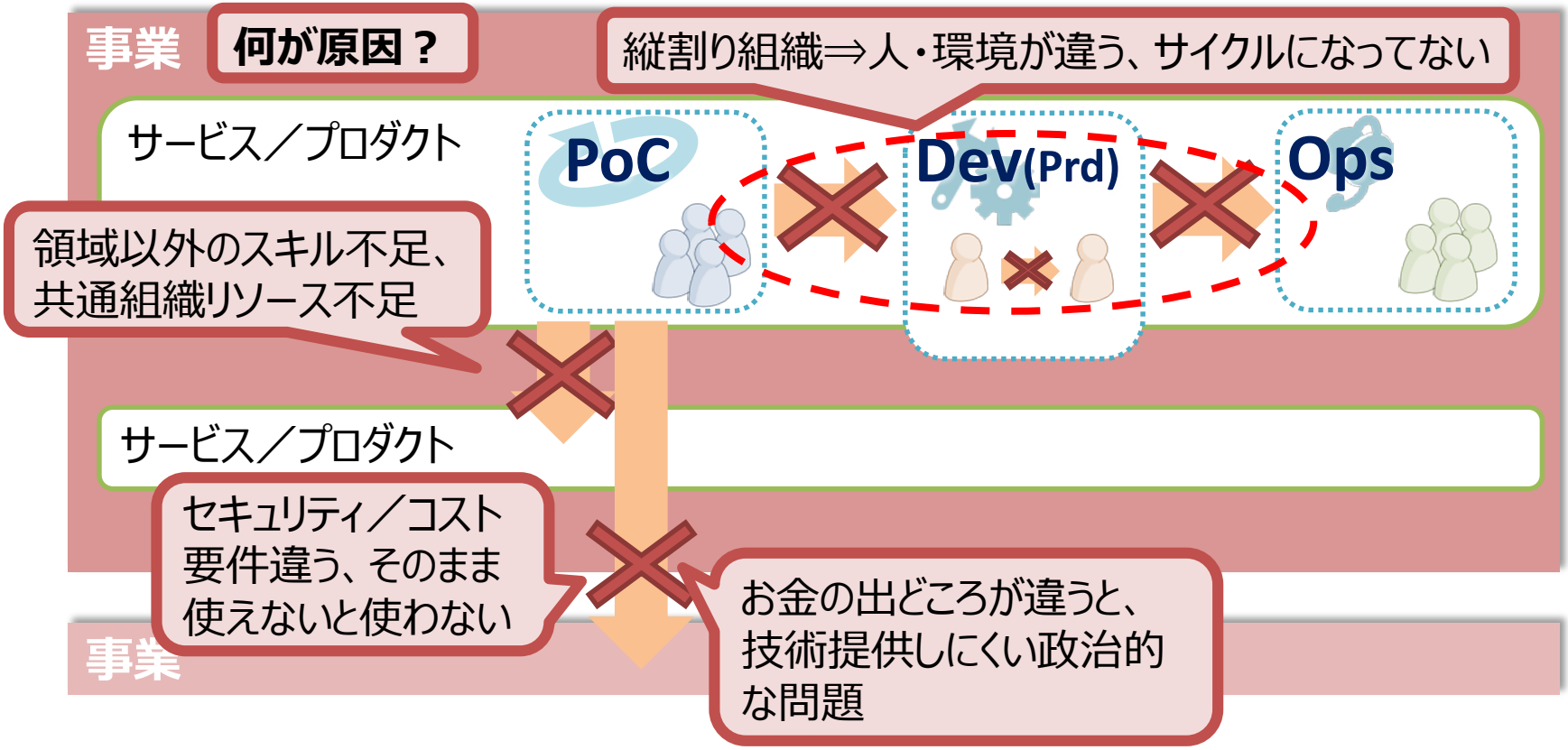


MITSUBISHI ELECTRIC
Changes for the Better

課題の分類観点を抽出：取り組みから

サービス開発のための仕組み・規程整備が必要

- 既存のHW製造プロセス・体制そのまま継承してしまっている
- ドメインごとの知識・技術で局所最適化されてきており、共通的な情報システム技術を取り扱うためのリソースが圧倒的に不足



課題の分類観点を抽出：取り組みから

サービス開発のための仕組み拡張／整備が必要

- 既存のHW製造プロセス・体制そのまま継承してしまっている
- [illegible]情報

組織体制・規程だけの問題？
具体的に何？
⇒**継続性・組織展開**の観点を
もう少し掘り下げてみる

サービス／プロダクト

セキュリティ／コスト
要件違う、そのまま
使えないと使わない

お金の出どころが違うと、
技術提供しにくい政治的
な問題

事業

課題の分類観点を抽出：世の中の状況から

DevOpsの文脈でも組織へのスケールがポイント

Jennifer Davis, Ryn Daniels, Effective DevOps,
(https://www.oreilly.co.jp/books/9784873118352/)

４つの柱
・コラボレーション
・アフィニティ
・ツール
・**スケーリング**

3つの道
・フローの加速
・素早いフィードバック
・実験、失敗からの学習、反復と練習
が熟達には不可欠という考え方を並行して育てていく**企業文化を作ること**

ジーン・キム,ケビン・ベア,ジョージ・スパッフォード,
The DevOps 逆転だ! 究極の継続的デリバリー,
(https://shop.nikkeibp.co.jp/front/commodity/0000/P85350/)

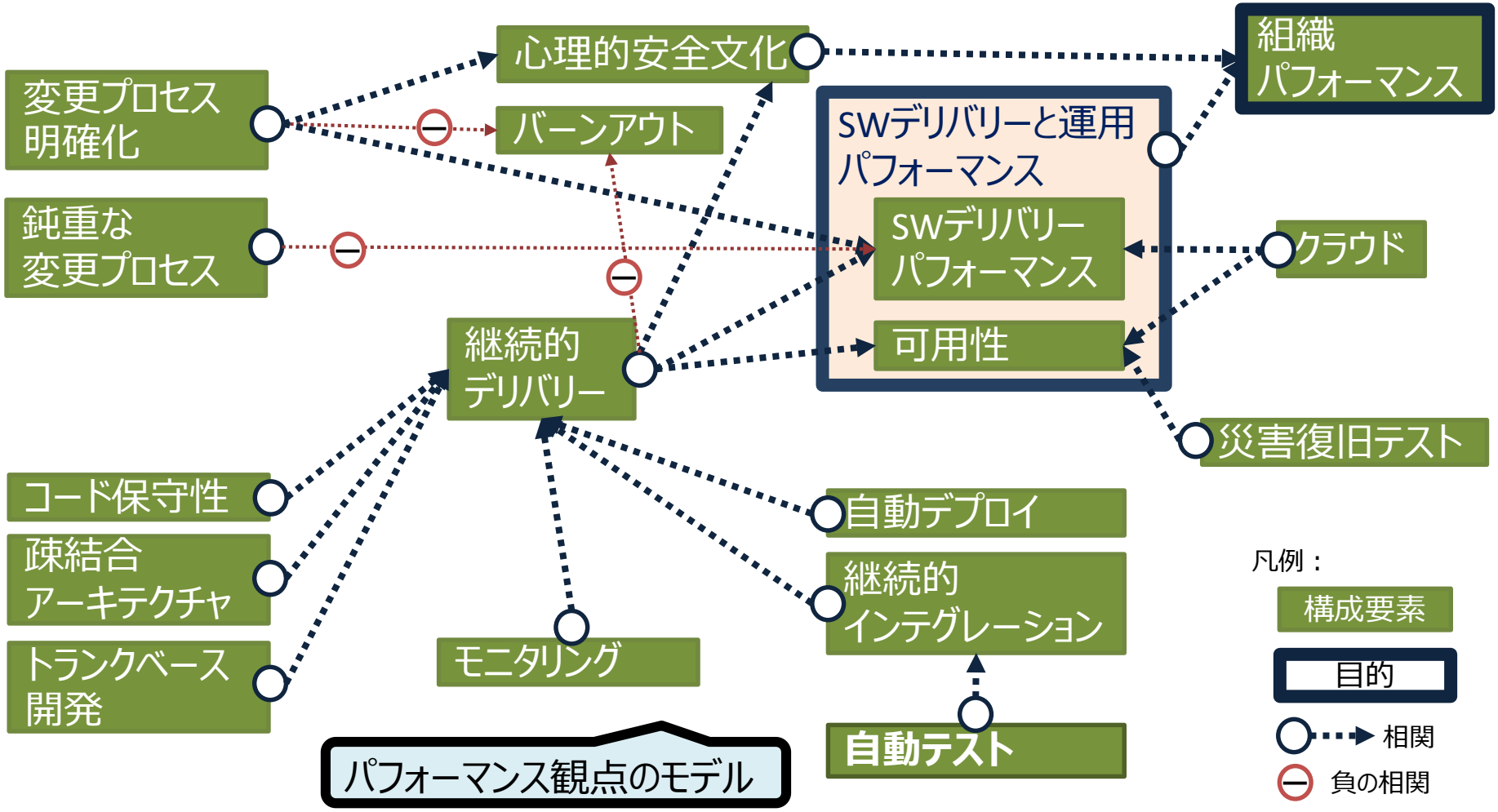
ジーン・キム,ジェズ・ハンブル,パトリック・ボア,ジョン・ウィリス,
The DevOps ハンドブック 理論・原則・実践のすべて,
(https://shop.nikkeibp.co.jp/front/commodity/0000/P85480/)

世の中の状況から課題の分類観点抽出

継続性：DevOpsのプラクティスから

課題の分類観点を抽出：世の中の状況から

DevOpsと関連要素の相関モデル(1)



参考：
Dr. Nicole Forsgren, Dr. Dustin Smith, Jez Humble, Jessie Frazelle, “Accelerate: State of DevOps 2019”, (https://cloud.google.com/devops/)

課題の分類観点を抽出：世の中の状況から

State of DevOps Report

- ✓ 当該レポート作成しているDORA(DevOps Research and Assessment)は、2018/12にGoogleが買収している

DORA joins Google Cloud

We are super excited to announce that DevOps Research and Assessment (DORA) has been acquired by Google. As part of Google Cloud, DORA will continue to create delightful experiences for developers and operators through data-driven insights that deliver value using DevOps in the cloud.

"The best, most innovative organizations develop and deliver their software faster, more reliably, more securely, and with higher quality, standing as high performers in technology," said Dr. Nicole

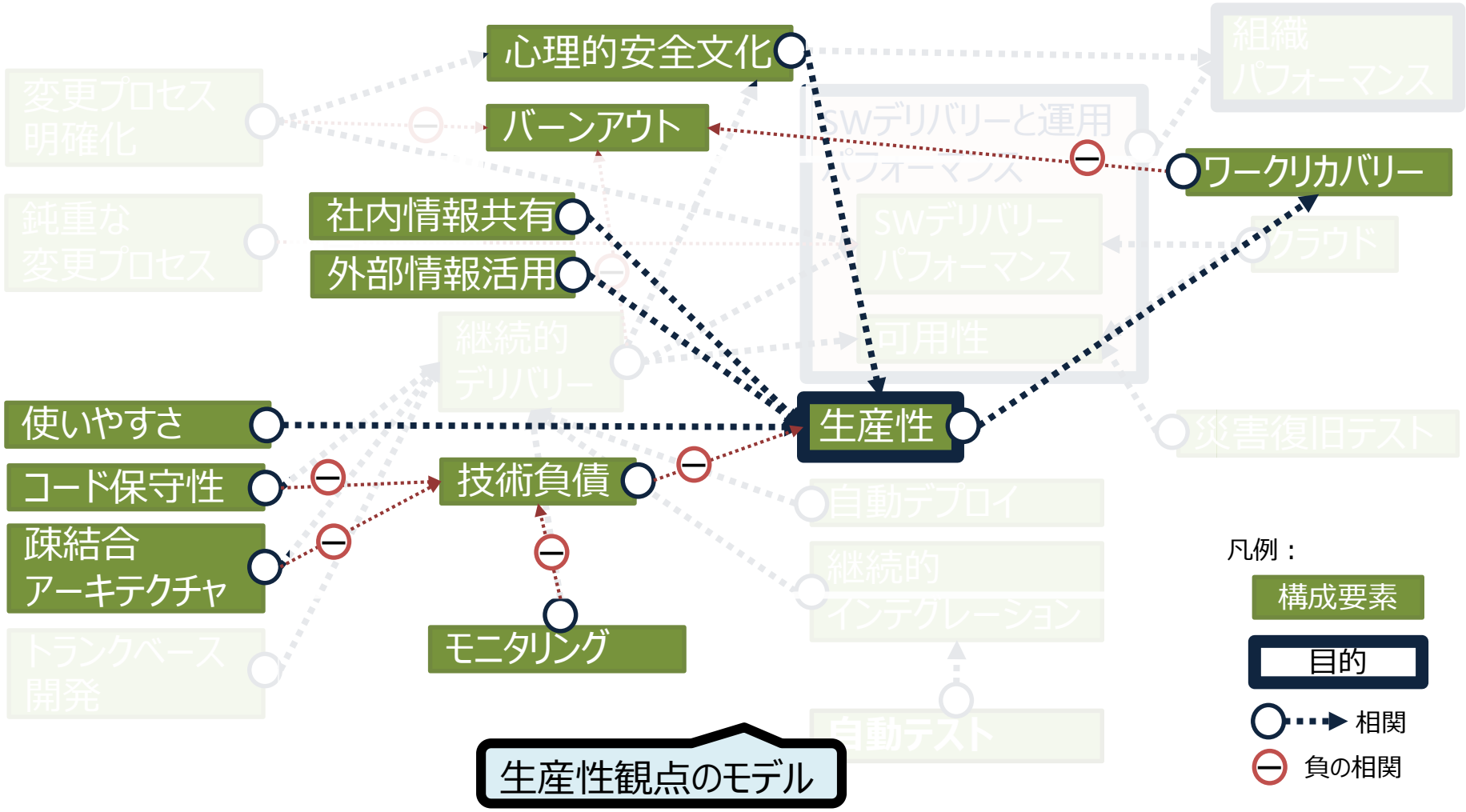
https://www.devops-research.com/dora-joins-google-cloud.html

参

Dr. Nicole Forsgren, Dr. Dustin Smith, Jez Humble, Jessie Frazelle, "Accelerate: State of DevOps 2019", (https://cloud.google.com/devops/)

課題の分類観点を抽出：世の中の状況から

DevOpsと関連要素の相関モデル(2)



参考：
Dr. Nicole Forsgren, Dr. Dustin Smith, Jez Humble, Jessie Frazelle, “Accelerate: State of DevOps 2019”, (https://cloud.google.com/devops/)

MITSUBISHI ELECTRIC
Changes for the Better

課題の分類観点を抽出：世の中の状況から

プロセス内の自動化は前提、どう連携するかが重要

DevOpsパフォーマンス別の**自動化・インテグレーションプラクティス**

	Low	Medium	High	Elite
自動ビルド	64%	81%	91%	92%
自動ユニットテスト	57%	66%	84%	87%
自動受入テスト	28%	38%	48%	58%
自動性能テスト	18%	23%	18%	28%
自動セキュリティテスト	15%	28%	25%	31%
テスト環境自動構築	39%	54%	68%	**72%**
自動デプロイ(本番環境)	17%	38%	60%	**69%**
チャット統合	29%	33%	24%	**69%**
本番環境モニタリング統合	13%	23%	41%	**57%**
上記以外	9%	14%	5%	4%

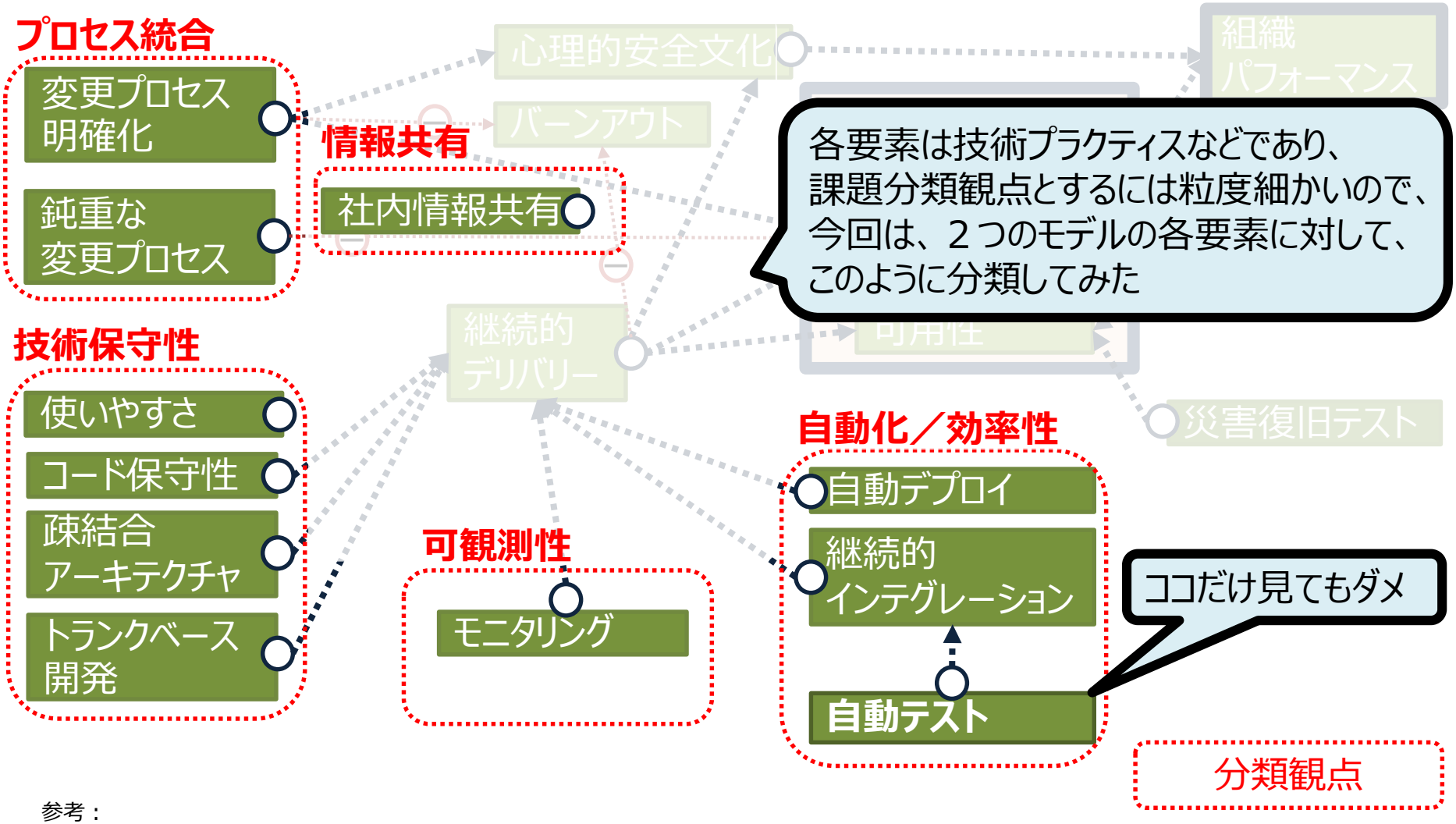
チャットツールやチャットボットと、自動化環境を**連携**

本番環境監視ツールや可視化ツールと、自動化環境を**連携**

参考：
Dr. Nicole Forsgren, Dr. Dustin Smith, Jez Humble, Jessie Frazelle, “Accelerate: State of DevOps 2019”, (https://cloud.google.com/devops/)

課題の分類観点を抽出：世の中の状況から

自動化だけではなく複数要素の考慮必要



参考：
Dr. Nicole Forsgren, Dr. Dustin Smith, Jez Humble, Jessie Frazelle, “Accelerate: State of DevOps 2019”, (https://cloud.google.com/devops/)

世の中の状況から課題の分類観点抽出

組織展開：DevOpsの戦略から

MITSUBISHI ELECTRIC
Changes for the Better

課題の分類観点を抽出：世の中の状況から

実践的なスキル習得やリソース確保の観点も重要

リソース不足、継続サポートの負荷

理論と実践の断絶、排他的な専門家

より実践的にスキル習得できているかに差がある

失敗要因

DevOpsパフォーマンス別の**スケーリング戦略**

	Low	Medium	High	Elite
トレーニングセンター	27%	21%	18%	14%
COE (研究所)	34%	34%	20%	24%
PoCプロジェクト	41%	32%	20%	16%
PoC as Template	16%	29%	29%	30%
PoC as a Seed	21%	24%	29%	30%
実践的なコミュニティ	24%	51%	**47%**	**57%**
ビックバン (トップダウン)	19%	19%	11%	9%
ボトムアップ活動	29%	39%	**46%**	**46%**
マッシュアップ	46%	42%	34%	38%

戦略・リソース不足

参考：
Dr. Nicole Forsgren, Dr. Dustin Smith, Jez Humble, Jessie Frazelle, “Accelerate: State of DevOps 2019”, (https://cloud.google.com/devops/)

分類観点のまとめ

分類観点のまとめ

自動化／効率性	マニュアル作業をいかに減らせるか
可観測性	本番環境からの迅速なフィードバックが可能か
規程整備	目的に合った規程が整備されているか
プロセス統合	プロセス間に無駄な作業がないか
技術保守性	他の人がすぐ使える技術か 変更しやすいか
リソース	十分な人材・期間・金が確保可能か
情報共有	ナレッジを迅速に共有可能か
スキル習得	迅速・実践的なスキル習得が可能か

単一組織内での**継続性**に関連

組織展開に関連

複合的で重なっているところ多そうだが、
ひとまず今回はこれをベースに課題掘り下げてみる

MITSUBISHI ELECTRIC
Changes for the Better

課題検討

課題検討

規程整備　プロセス統合　自動化／効率性

- サービス開発にあった組織体制・規程になってない

継続性

・開発/変更管理時の、プロセス間の必要成果(outcome)自動化非対応
→自動出力した試験レポートをそのまま成果とできない、そもそも自動でバージョン管理されていても別途成果物として出力が必要
→承認したというアウトプットが必要／会議での承認が必要だったりと、チャットボット連携した承認フロー自動化など難しい
→一時的なPoCプロジェクト形態で、開発・本番環境間での自動化・連携が困難

組織展開

・必要成果、承認フローは規程・規則として定められている
→チームレベルで変更不可／例外的・一時的な措置としてのみ許可
→このフローを前提とした運用外部委託、継続的／横展開が困難

課題検討

規程整備　プロセス統合　自動化／効率性

製造業とプロセス規程

- ✓ input⇒[プロセス]⇒output⇒[outcome]⇒input⇒[プロセス]⇒…
  例：[単体試験]⇒試験結果(データ)⇒[単体試験成績書⇒レビュー]⇒…

- ✓ HW製造では、法律・規約⇒…⇒HW を遵守するために細かにドキュメント化・承認フローを実施、効率化のために分業・縦割りで最適化
  - ➢ 製造過程に入ってしまってからの手戻りは致命的なため、プロセスの必要成果に問題がないかを、手動ベースの多重チェックで保証・管理してきた
    - 管理責任は開発チーム外、外部の人間が理解できる形に落とし込む必要
  - ➢ プロセスを厳格に守らせるために、必要成果・承認形態を規程・規則として定め、それに沿った組織体制に
    - 抜け漏れなく・作成手間を省けるように、様式化⇒手動作成が前提
    - 人・環境が異なると、コア技術以外の暗黙知も毎回明文化する必要

過剰に厳格、管理責任がチーム外に存在

課題検討

技術保守性 **スキル習得** **リソース**

- サービス／事業ごとに要件が異なり展開しにくい
- 事業コア技術外のスキル不足、共通技術支援のリソース不足

組織展開

・**事業所ごとの要件をすべて取り入れた標準化は困難**
→ツール・技術の標準化・提供は、提供側での保守が前提のため誰もやりたがらない
→HW製造では事業ごとに一から十まで標準化／ガイドライン化してきている
→継続的なサポート体制の構築にはリソース不足
・**事業ごとに自動化⇒要件違う部分が疎結合にできてない⇒横展開できない**
→クラウド／オンプレ、OSS利用、独自作りこみ、有償ツールのライセンス費用…
・**自動化技術者と、テスト実行・実装・保守担当が異なる**
→自動化導入に伴うスキル習得をコストと捉えられてしまう
→使い方の明文化を求められる⇒提供側のコスト
・**縦割りの組織体制による分業・マルチタスクで、スキル習得する余裕がない**
→サービス／プロダクトごとの機能実現に対する検討で手一杯
→プロセス改善のためのタスクとして自動化の優先度下げられてしまう

課題検討

技術保守性　スキル習得　リソース

Googleの事例

- ✓ トランクベース開発＆社内共有、他チームからの改善・コミット
- ✓ 定期的なリファクタリングによる技術負債の軽減と、新メンバのスキル習得

Software Engineering at Google

31 Jan 2017

Fergus Henderson
<fergus@google.com> (work) or
<fergus.henderson@gmail.com> (personal)

Abstract

We catalog and describe Google's key software engineering practices.

https://www.infoq.com/jp/news/2019/11/google-software-engineering/?itm_source=infoq&itm_medium=popular_widget&itm_campaign=popular_content_list&itm_content=

→プロセス改善のためのタスクとして自動化の優先度下げられてしまう

課題検討

情報共有　**可観測性**

- 標準化や情報共有、事業ごとへの適用に時間がかかる
- 運用からのフィードバックの仕組みがない

組織展開

・情報共有はしていても、要覧・ガイドラインの共有まで
→全社共通の要覧・ガイドライン化には、関連事業ごとに確認必要で時間がかかる
→疎結合になっておらず情報過多、ナレッジからの落とし込み・適用が難しい
・標準化に向けたコミュニティなどは数多くあるも、定期的な情報交換にとどまる
→一緒になって実践的に取り組むためのリソース・場所・権限がない

継続性

・プロト開発などはその場限りの体制で、フィードバックの仕組みがない
→研究所・横通しの組織へのフィードバックがない／遅い、高頻度の担当者配置換えでカバーしている現状
・運用外部委託な組織体制で、本番稼働中のサービスに関する情報取得が困難
→Shift-rightな、継続的改善を目的とした仕組みが考慮できてない(丸投げ)

継続的テスト実現に向けてすべきこと

課題と1対1対応する解決策検討は難しそうだけど、
何ができるか・すべきか考えてみる

継続的テスト実現に向けて

・開発/変更管理時の、プロセス間の必要成果(outcome)自動化非対応
・必要成果、承認フローは規程・規則として定められている

・運用外部委託な組織体制で、本番稼働中のサービスに関する情報取得が困難

・プロセス統合に向けた規程整備
→小さく早く改善していくことを前提とした、開発プロセスOutcomeの整理・再定義
→プロセスごとの承認者・管理責任者の整理・再定義
→変更管理や構成管理、リリース管理の自動/軽量化

・継続的なサービス改善サイクルのための組織編制
→運用時におけるトイル削減を成果とするような、各役割の評価制度の再定義

トップダウンによる変革必要

・プロセス間の自動化
→テスト結果レポート自動作成、承認フロー連携（チャット連携）、テスト管理連携

ボトムアップでもやっていけそう

継続的テスト実現に向けて

・事業所ごとの要件をすべて取り入れた標準化は困難
・事業ごとに自動化⇒要件違う部分が疎結合にできてない⇒横展開できない
・自動化導入に伴うスキル習得をコストと捉えられてしまう
・プロセス改善のためのタスクとして自動化の優先度下げられてしまう

・全社共通の要覧・ガイドライン化には、関連事業ごとに確認必要で時間がかかる
・疎結合になっておらず情報過多、ナレッジからの落とし込み・適用が難しい

・仕様書・ドキュメントからコードベースの情報共有へ
→自動化・コード化はコスト削減手段ではなく、コア技術であるとの文化醸成
→当たり前に自動化コードを書ければ、費用対効果の悪化を抑制できる
→コードベースなら既存のCIの仕組みで効率的に管理・統合可能

・社内共有リポジトリによるコードベースの技術共有
→基盤／ツール提供ではなくコードの共有
→チーム外(各事業所)からの修正・コミットで継続的改善
→リファクタリングによるスキル習得

ボトムアップでとりかかれそう

継続的テスト実現に向けて

・標準化に向けたコミュニティなどは数多くあるも、定期的な情報交換にとどまる

・プロト開発などはその場限りの体制で、フィードバックの仕組みがない

・より実践的なコミュニティ活性化に向けたLab環境の整備
→パブリッククラウド活用、チーム内への展開から実施
→活発な情報発信を促すための、参加・発言の敷居を下げる仕組み／文化醸成

・本番環境の可観測性を考慮したモニタリング技術
→テストとAPM・サービス監視の統合、Shift-left
→本番環境テストで、ビジネス判断を簡単にできるような仕組み、Shift-right

ボトムアップでとりかかれそう

今後も継続して活動・課題検討を進めていきます

さいごに

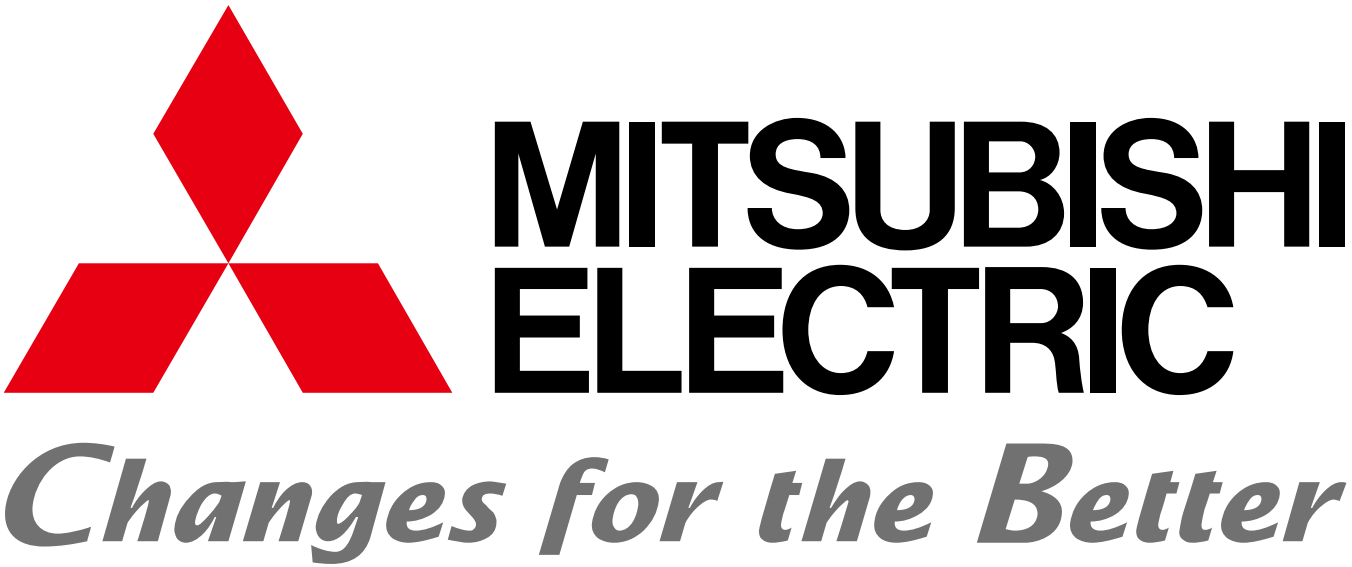
継続的テスト実現に向けて

継続的なボトムアップ活動、トップダウンへ働きかけ

ボトムアップ・草の根活動から
組織全体展開のためには？

⇒事業サービス責任者に、
成功体験を

・ボトムアップ活動での成功前例づくり
・事業で失敗しないように、ボトムの段階でうまく失敗しいていく技術
・失敗をマイナスとしない評価観点の変革
などが必要


MITSUBISHI
ELECTRIC
Changes for the Better