MITSUBISHI ELECTRIC
Changes for the Better

JaSST'18 Hokkaido

アジャイル開発効率化に向けたテスト自動化基盤
～パブリッククラウドサービスの活用～

2018/09/07

三菱電機株式会社　情報技術総合研究所
都築　浩平

MITSUBISHI ELECTRIC CORPORATION

MITSUBISHI ELECTRIC
Changes for the Better

はじめに

- 今日お話しすること
  - プロト開発でテスト自動化した経験
  - パブリッククラウド活用についてがメイン
  - クラウド上のシステムへのAPIテストを自動化

- 対象外
  - 開発の内容詳細
  - 組み込み・機器のテスト
  - 各利用サービスの詳しい説明
  - 定量的な評価×

MITSUBISHI ELECTRIC
Changes for the Better

目次

- 背景
- 課題
- 解決アプローチ
- 実施内容
- 効果とまとめ
- 今後の課題

IoTシステムのアジャイル開発でテスト自動化検討

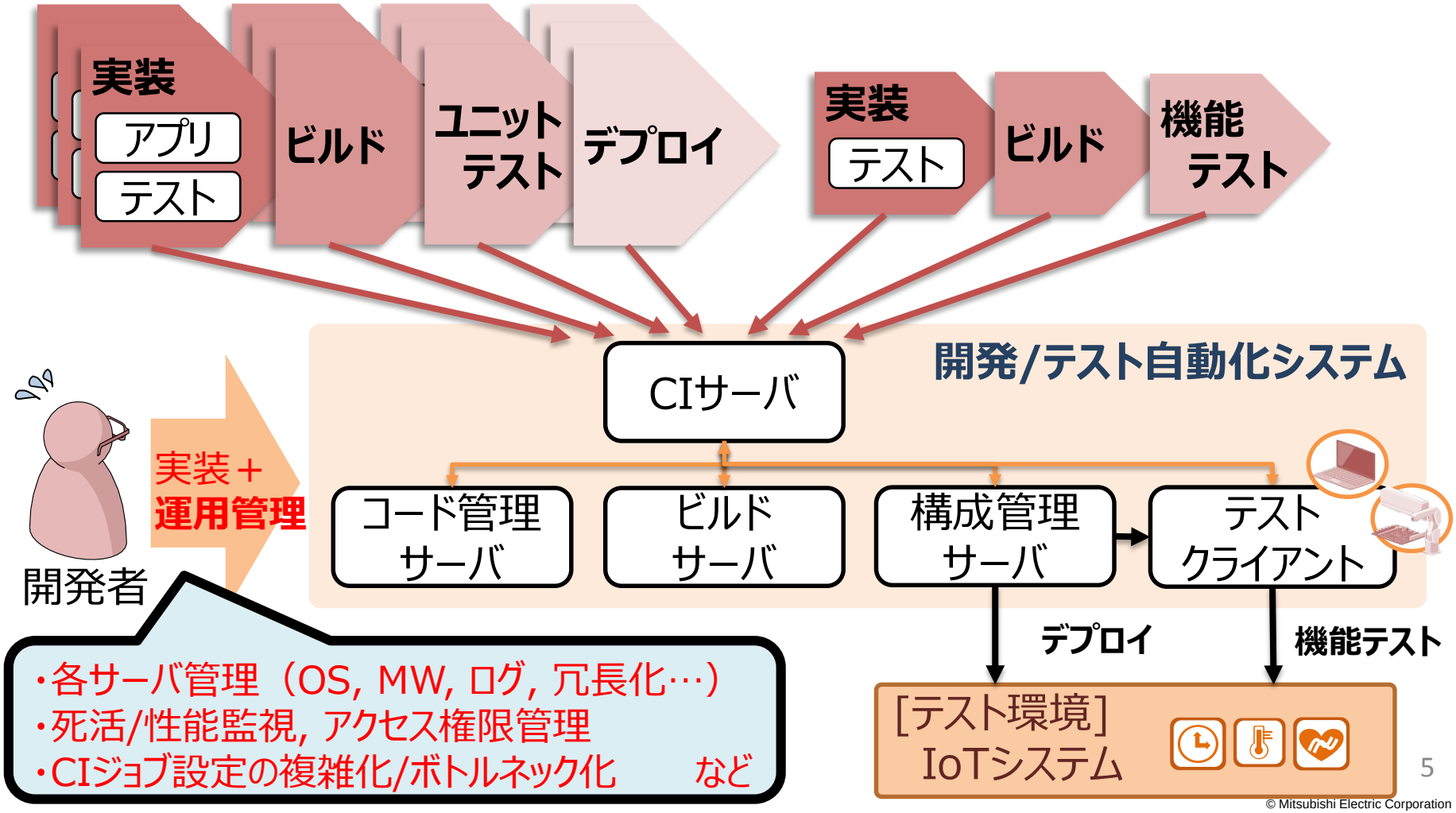
- 開発について
  - オンプレで稼働していたIoTシステムの、パブリッククラウドへの移行を目的としたプロト開発
    - 本開発ではAWS (Amazon Web Services)を利用
  - クラウドネイティブなアーキテクチャを検討、**アジャイル開発**にて動作確認しながら進めていくことに
  - **回帰テストの自動化検討**を担当
    - 機能(API)テスト部分

IoTシステムイメージ

課題

開発/テスト自動化システムは複雑・運用負荷高い

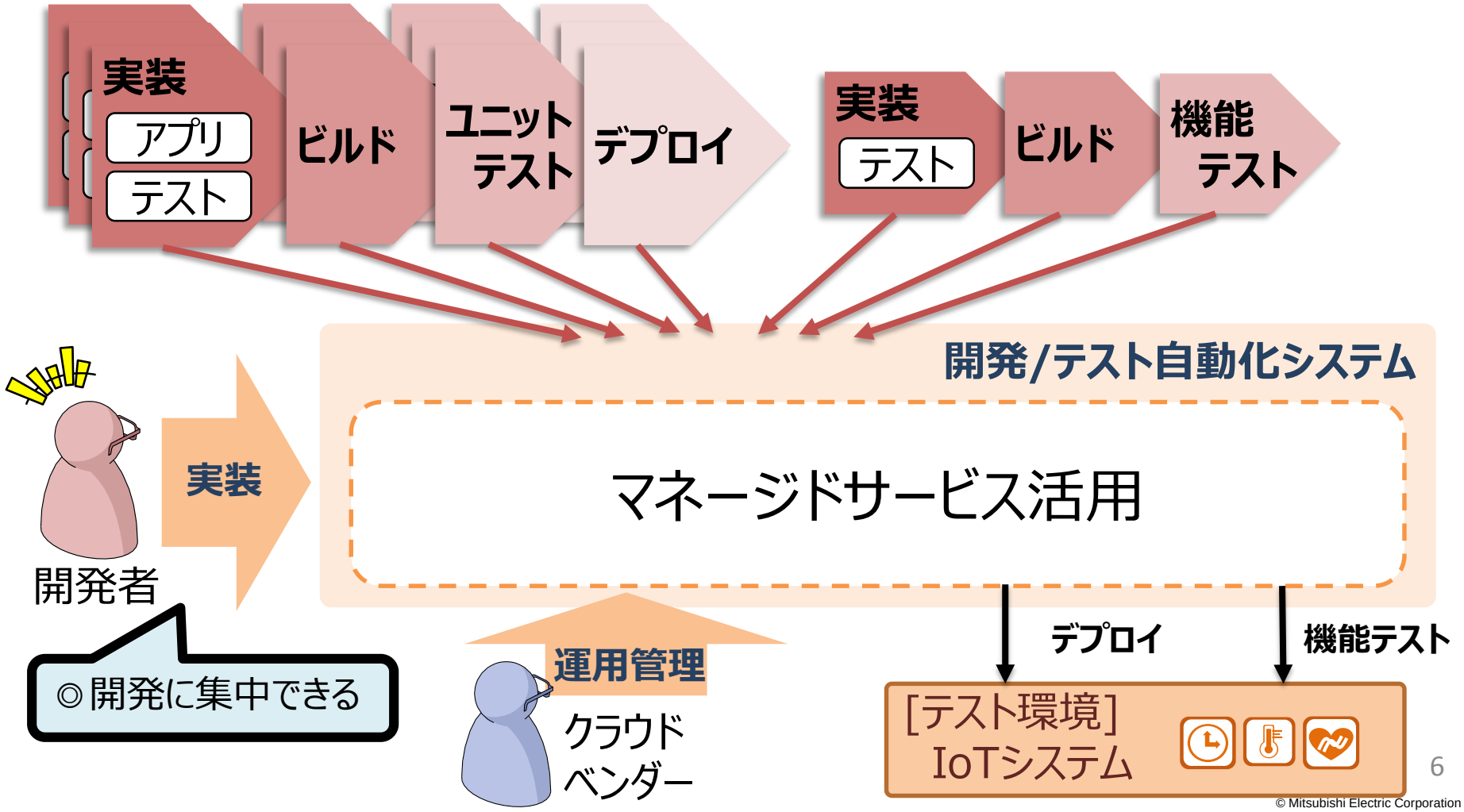
■開発プロセスと自動化システム



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

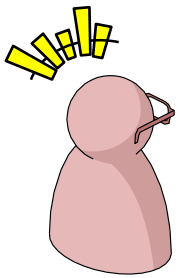


解決アプローチ(1)

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



開発者

実装

開発/テスト自動化環境

マネージドサービス活用

◎開発に集中できる

運用管理

クラウド
ベンダー

デプロイ

機能テスト

[テスト環境]
IoTシステム

解決アプローチ(2)

機器側の独自仕様への対応考慮した切り分け

- テスト自動化するための要件を整理

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
- …

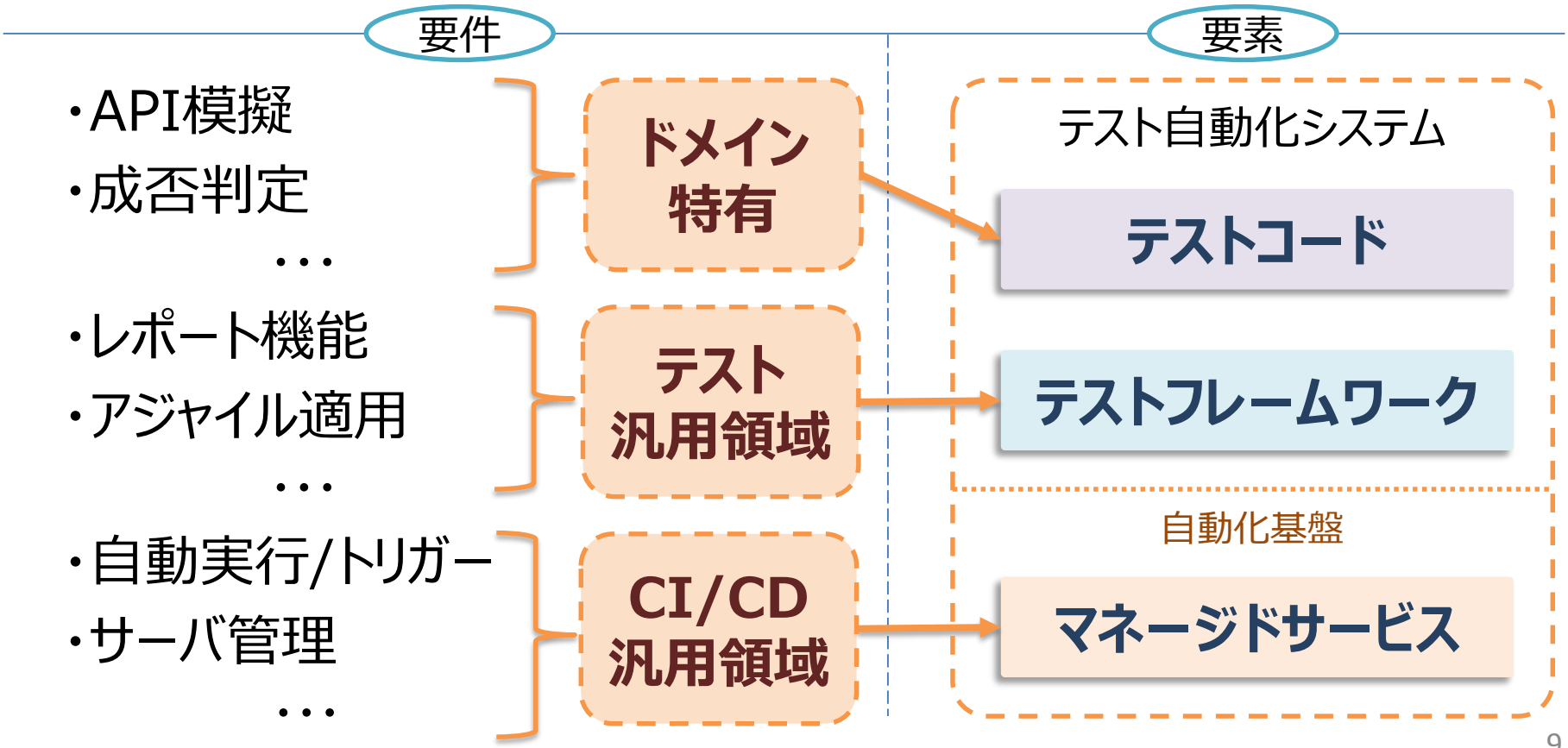
→ マネージドサービスで解決できそう◎

[*] 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？

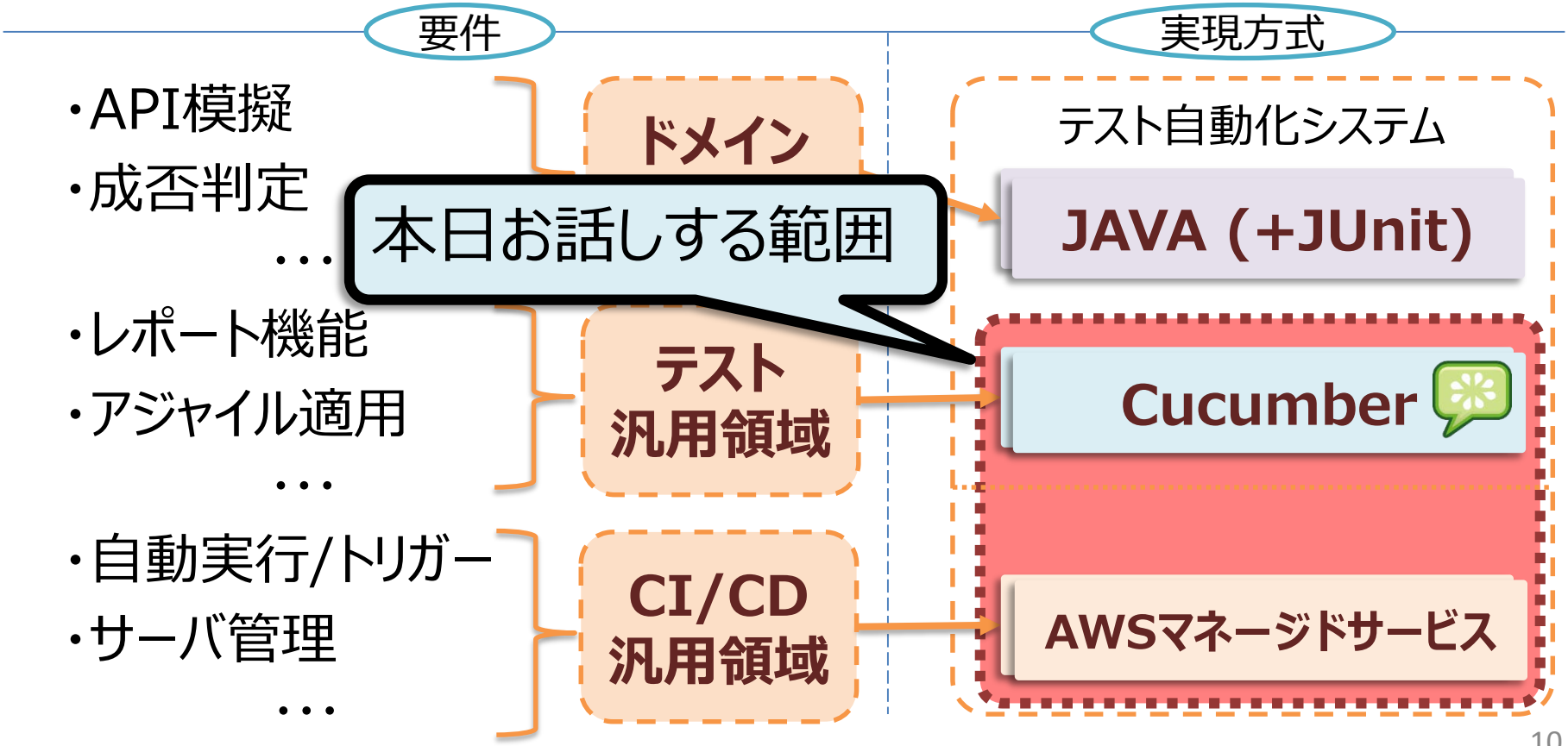


[*] CI：継続的インテグレーション　CD：継続的デリバリー

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

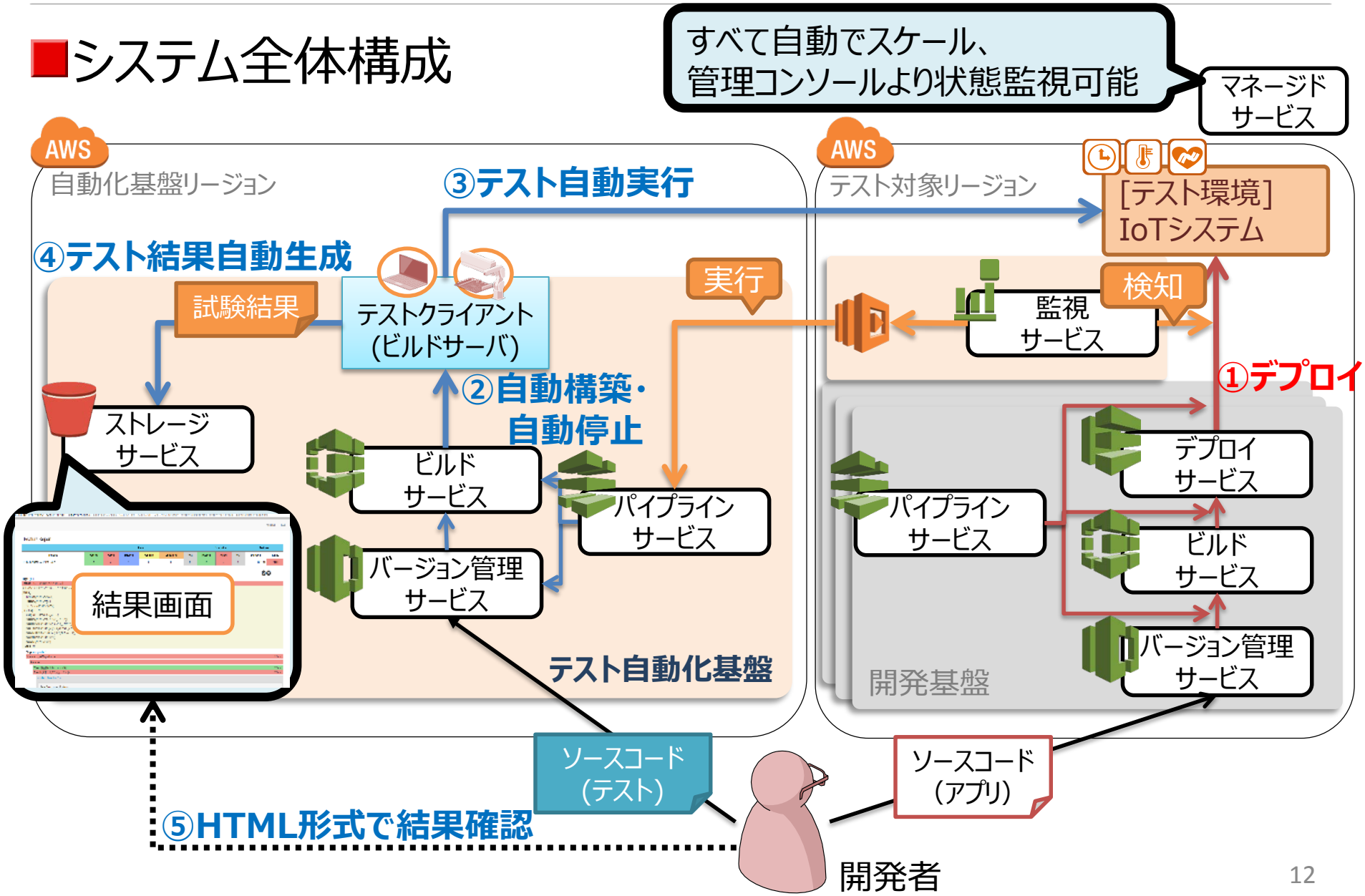
AWSマネージドサービス

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

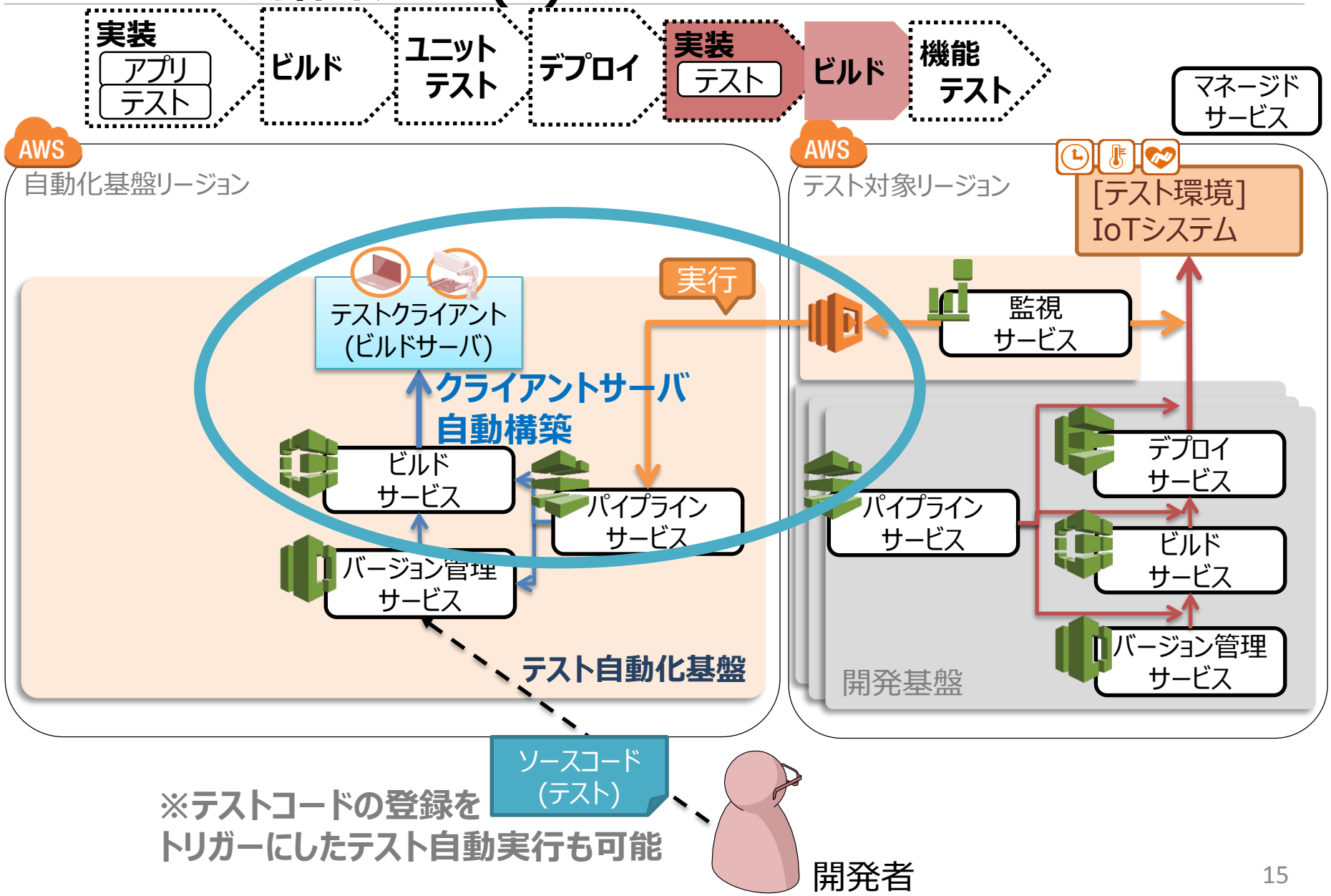
システム全体構成



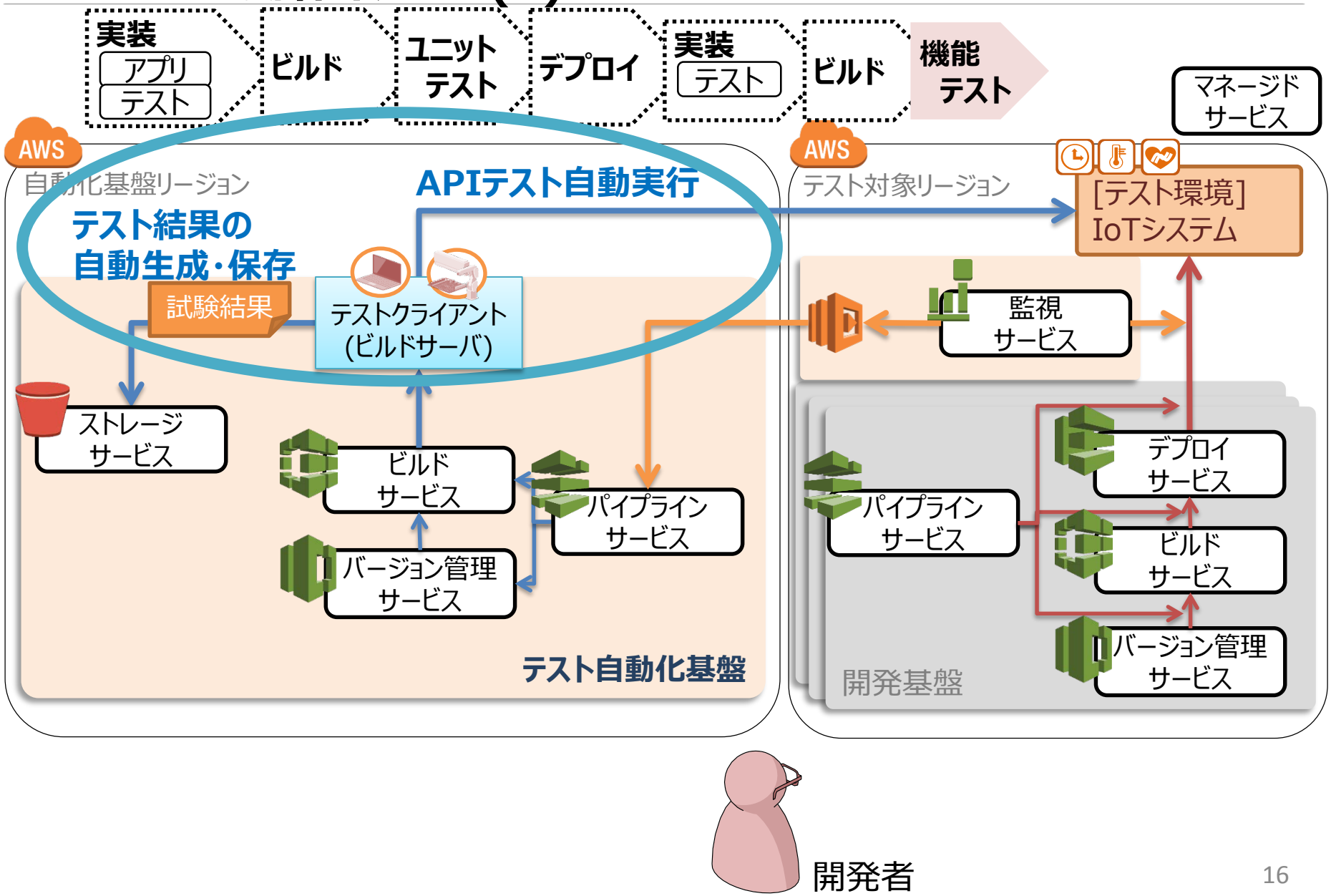
実施内容
動作イメージ(1)

実施内容
動作イメージ(2)

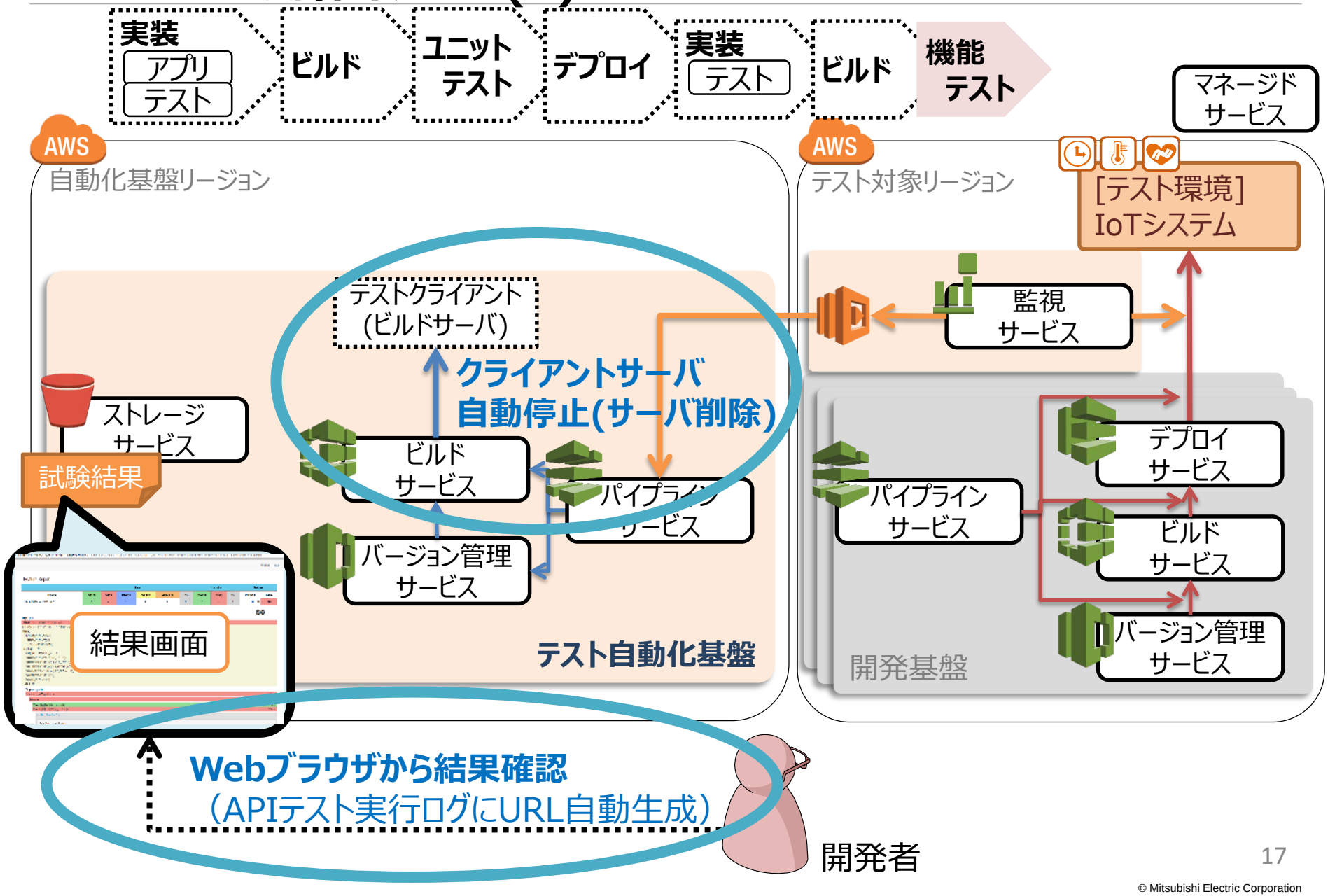
実施内容
動作イメージ(3)



実施内容
動作イメージ(4)

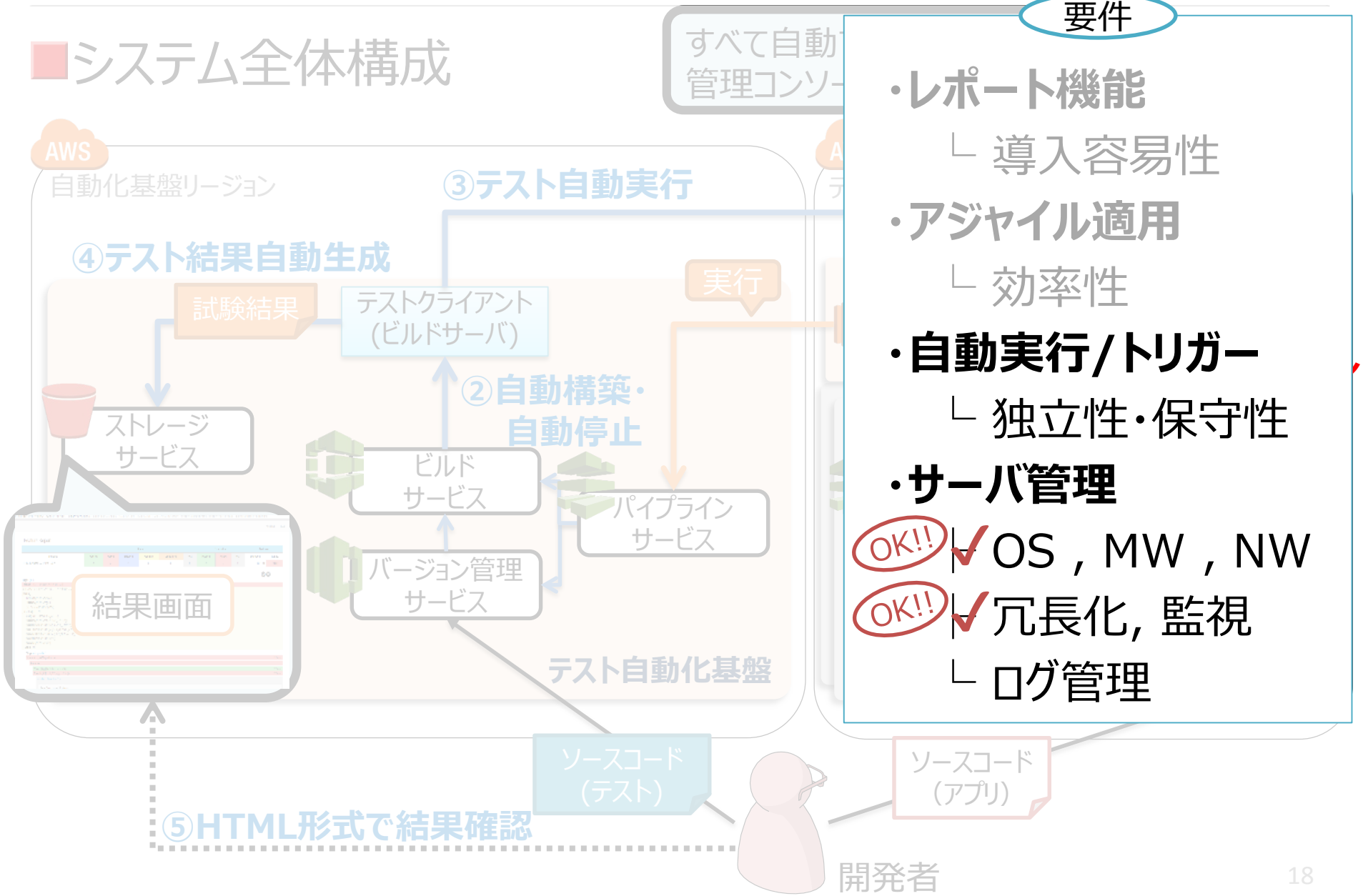


実施内容
動作イメージ(5)



実施内容

システム全体構成

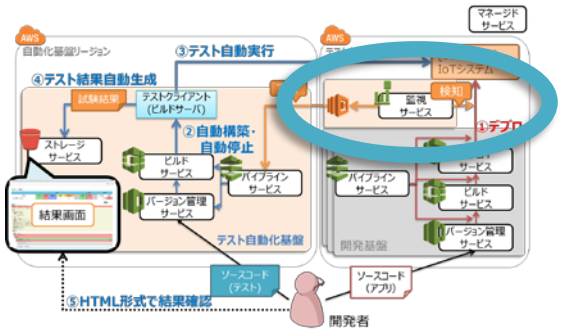


要件

- **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



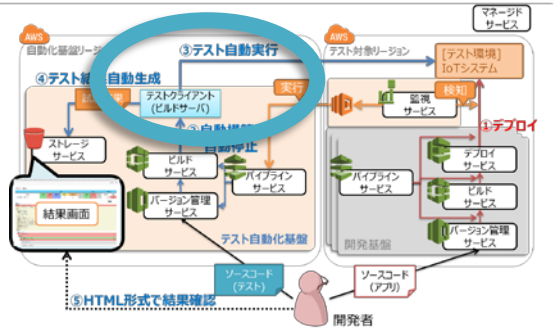
⇒開発基盤側の変更(追加/削除)に左右されず、**保守性**◎
マイクロサービスを考慮したアーキテクチャにも対応可能

- 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ストレージサービス(S3)に自動保存・管理されるため、API実行ログを標準出力としてテスト実行時に出力すると**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

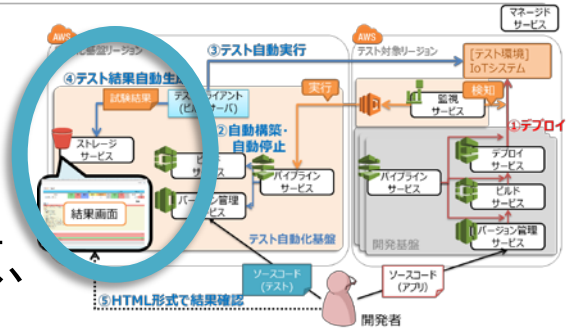
要件

- **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)

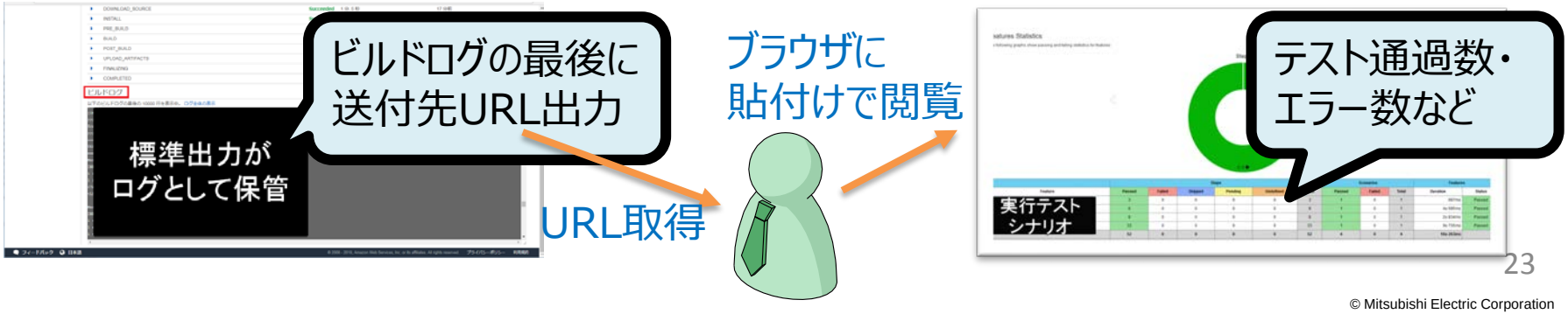
レポーティング

- 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)

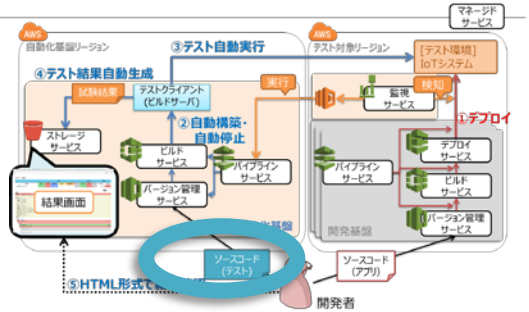


⇒静的ホスティング・IPアクセス制限させたストレージサービス(S3)に送付することで、**レポートの閲覧・管理が容易に**

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**

 # テストケース記載例

```
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: “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
```

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
  - ”Given”や”When”, “Then”などを振る舞い・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：対象APIがSOAP(Simple Object Access Protocol)のようにデータボディのXMLで詳細データをやりとする場合、
    “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

効果とまとめ

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パブリッククラウド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)

今後の課題

- テスト種別の拡大
  - 機器含めたE2Eテストの自動化
- テストコード実装負荷が高め
  - システム側とほぼ同等規模
  - 計画時にテスト自動化の為のコストも意識
- テスト管理強化
  - JIRA(+Xray Plugin)によるテスト管理お試し中
    - Cucumberと連携可能でJIRA上でテストケース記述、テスト結果の集計(要件カバレッジ・要件トレーサビリティ確認など)可能

⇒JIRAで可能なこと多い反面、運用負荷が高くなる

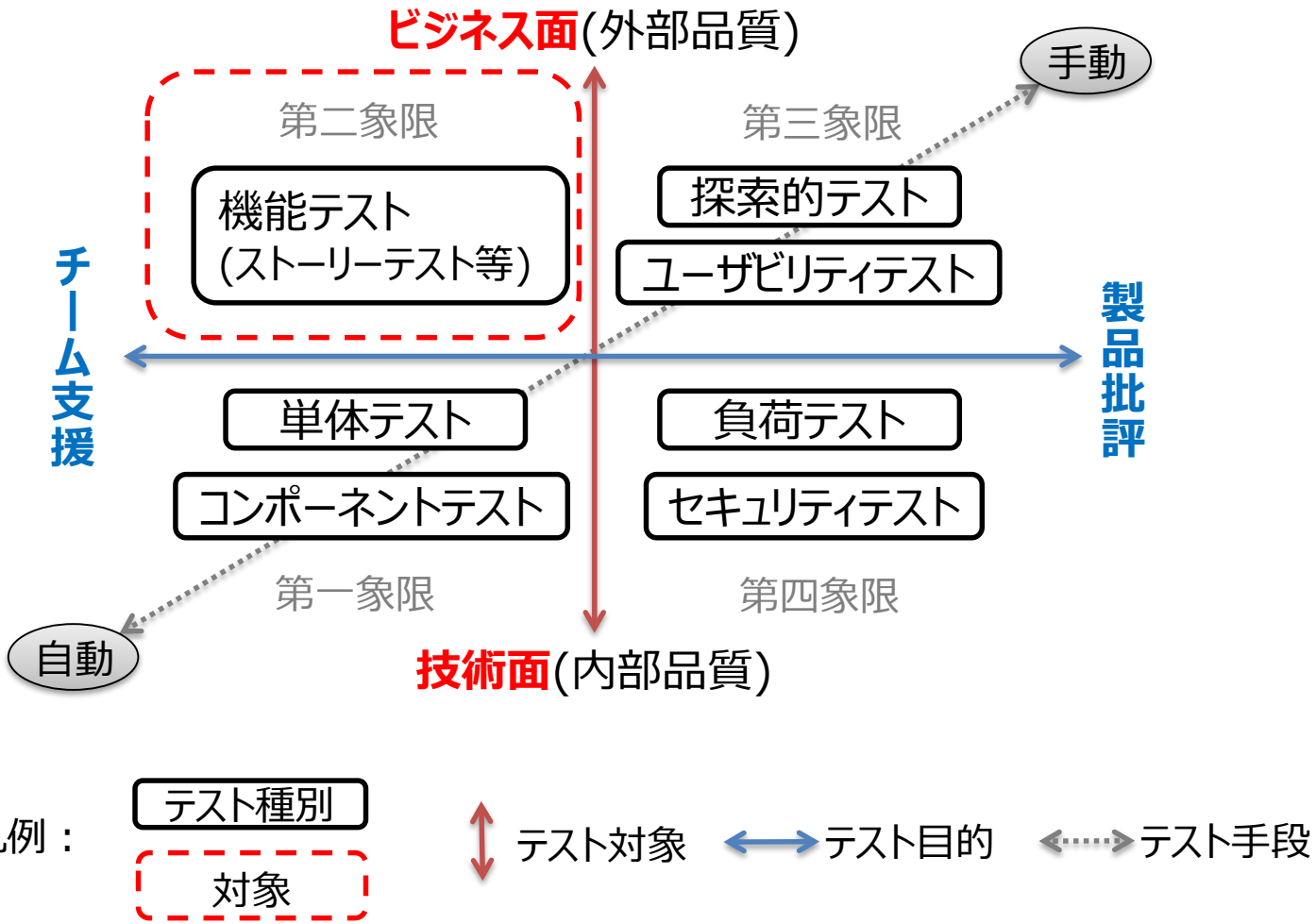
MITSUBISHI
ELECTRIC

Changes for the Better

補足

補足：背景

アジャイルテストの4象限



参考：リサ・クリスピン, ジャネット・グレゴリー, 榊原彰（2009）「実践アジャイルテスト テスターとアジャイルチームのための実践ガイド」, 翔泳社

補足：Cucumber

Gherkin書式でのテストケース記載例

- 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
  …
```

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

補足：システム構成詳細
テスト管理サーバ(JIRA)含めた構成