APRESIA®

継続可能なテスト自動化を目指して

JANOG45

次世代技術本部 技術開発部
次世代ソフトウェアグループ
坪田　繁
shigeru.tsubota.zv@apresiasystems.co.jp

Rev.1

自己紹介

◆担当業務
- ◇検証テスト自動化の推進
- ◇次世代NOSの検討調査

◆JANOGと私
- ◇JANOG41
  - –[SP] BDD + Python によるネットワーク機器のテスト自動化
- ◇JANOG43
  - –プログラム委員
    - ただし、インフルエンザで欠席
      - 大変、ご迷惑おかけしました。
      - 運営最後の打ち上げで山梨料理のお店をチョイス頂き有難うございます！

- ◇JANOG45
  - –自動化の2周目以降の事例紹介ならびに議論を実施

目次

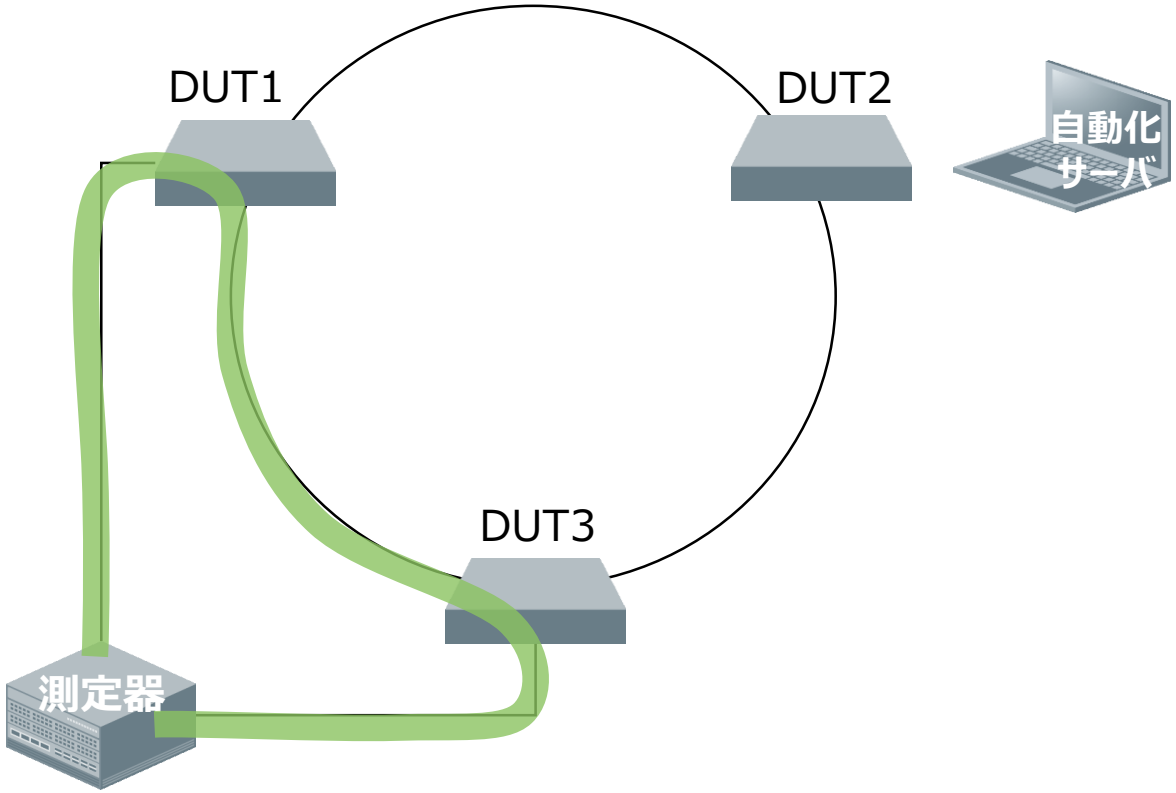
- 背景~対策の提示(Step1~Step3)
- Step1　組織の再構築
  - マネジメント
- Step2　ライブラリのUpgrade
  - 技術
- Step3　プロセスのUpdate
  - マネジメント
- Step1~3による結果と考察

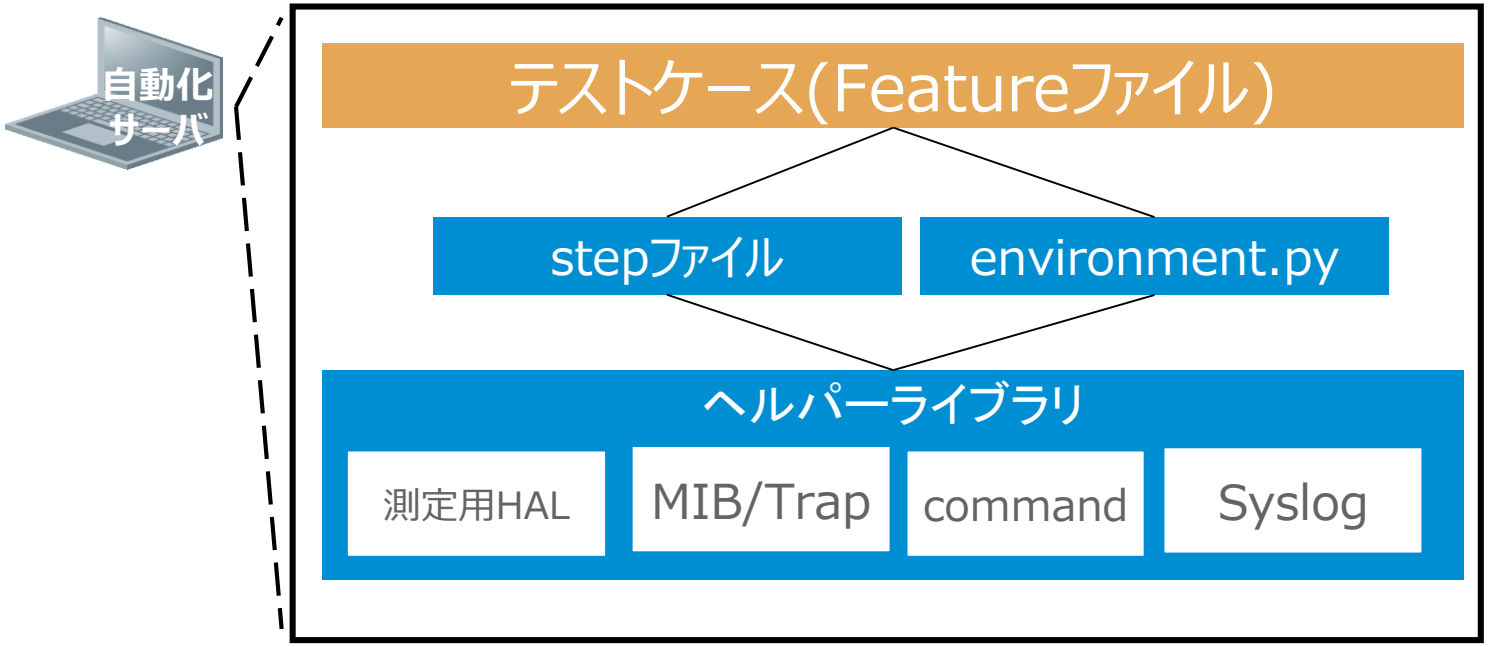
- 継続可能なテスト自動化について

内容が盛りだくさんなため、
適宜集中力に高低をつけていただけると幸いです。

JANOG41振り返り(自動化1周目のプロジェクト)

- behave(BDD)の活用+Pythonライブラリ作成により、結合テストの**67%**を自動化
  - コストは手動テストの**約2倍**(※)





※フレームワークの検討とOnboardingのコスト除く

1周目のプロジェクト終了時にKPTを実施

振り返りとして、
KPTを**メンバー約40名**で実施

Tryの一部を**2か月**かけて**実施**

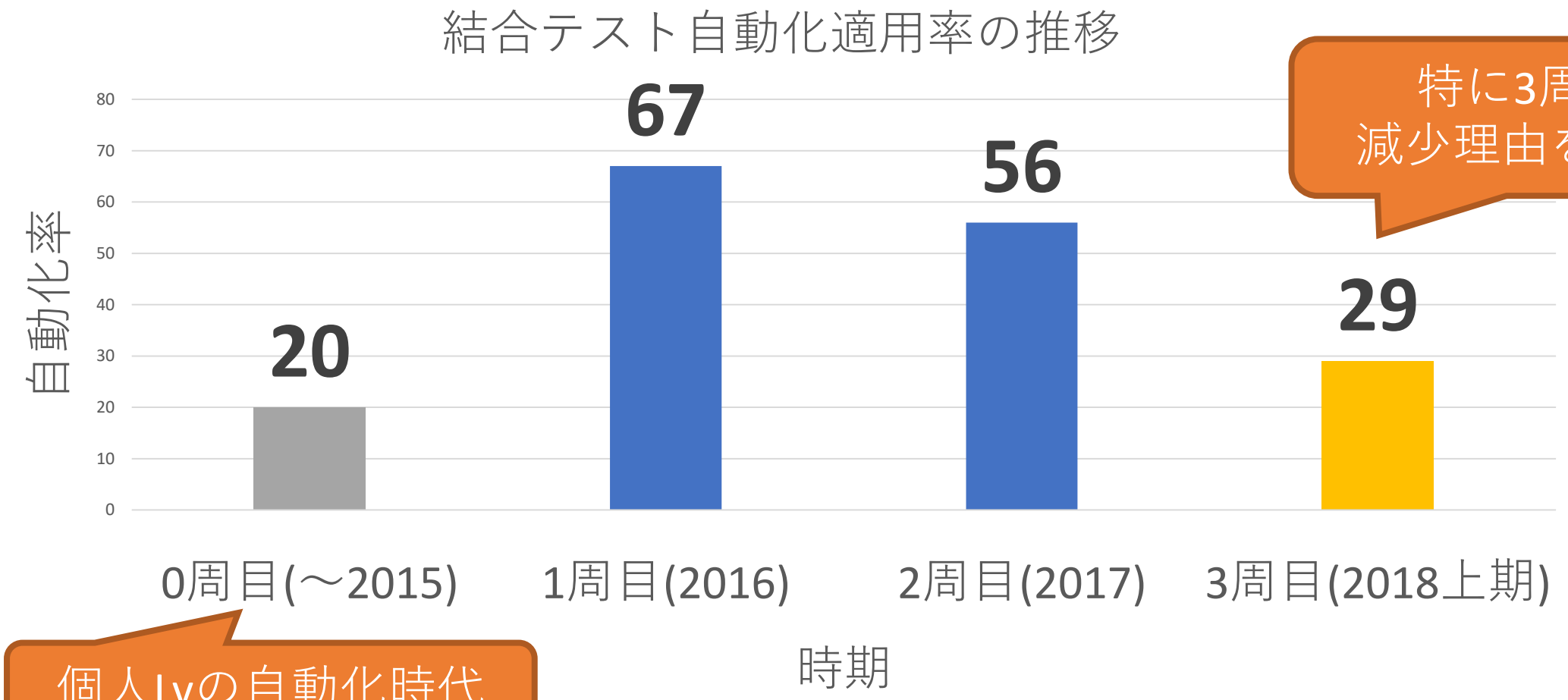
- 自動化未実施機能の**一つを自動化**
- 各機能のテスト実装の**共通資産化**
  - 試験終了時の**エラーログ確認**
  - **測定器**操作
  - **高負荷**試験向け**スレッド**

甘かった予測

- KPTとそのTryを実施！＆2周目プロジェクト向けの自動化方針決めも実施
- 以後の**運用を各プロジェクトに任せてもOK！**
  - プロジェクトによってバラツキはあるが、適宜改善されていくだろう！！？

甘かった予測

単調減少していました。



特に3周目の
減少理由を調査！

個人Lvの自動化時代

原因調査：32名にインタビューを実施

◆良かった点

◇仕様変更の**少なかった**箇所は**流用**できた。

◆問題点

自動化実施の**判断**が**担当者まかせ**

自動化**工数**を**確保できない**
(割り込み作業)

大幅な仕様変更に**追従できず**

テスト実装の**属人性**が強い

テスト実装の**汎用性**が低い

担当者が**変わって**勉強から**やり直し**

環境**構築**の仕方が**わからない**

自動化を推進する
組織が不在

Step1
組織の再構築

ライブラリが不足

Step2
ライブラリのUpgrade

自動化のプロセスが
不明瞭/不足

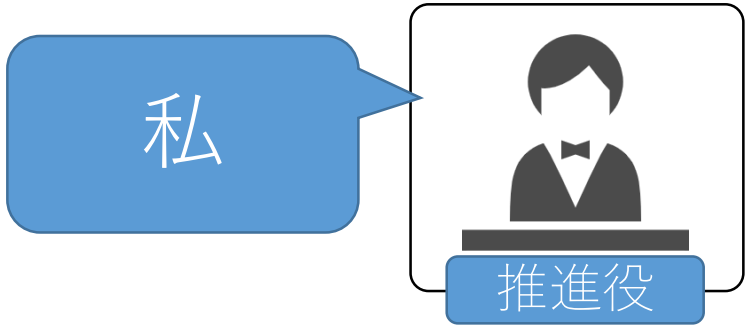
Step3
プロセスのUpdate

Step1
組織の再構築

APRESIA SYSTEMS

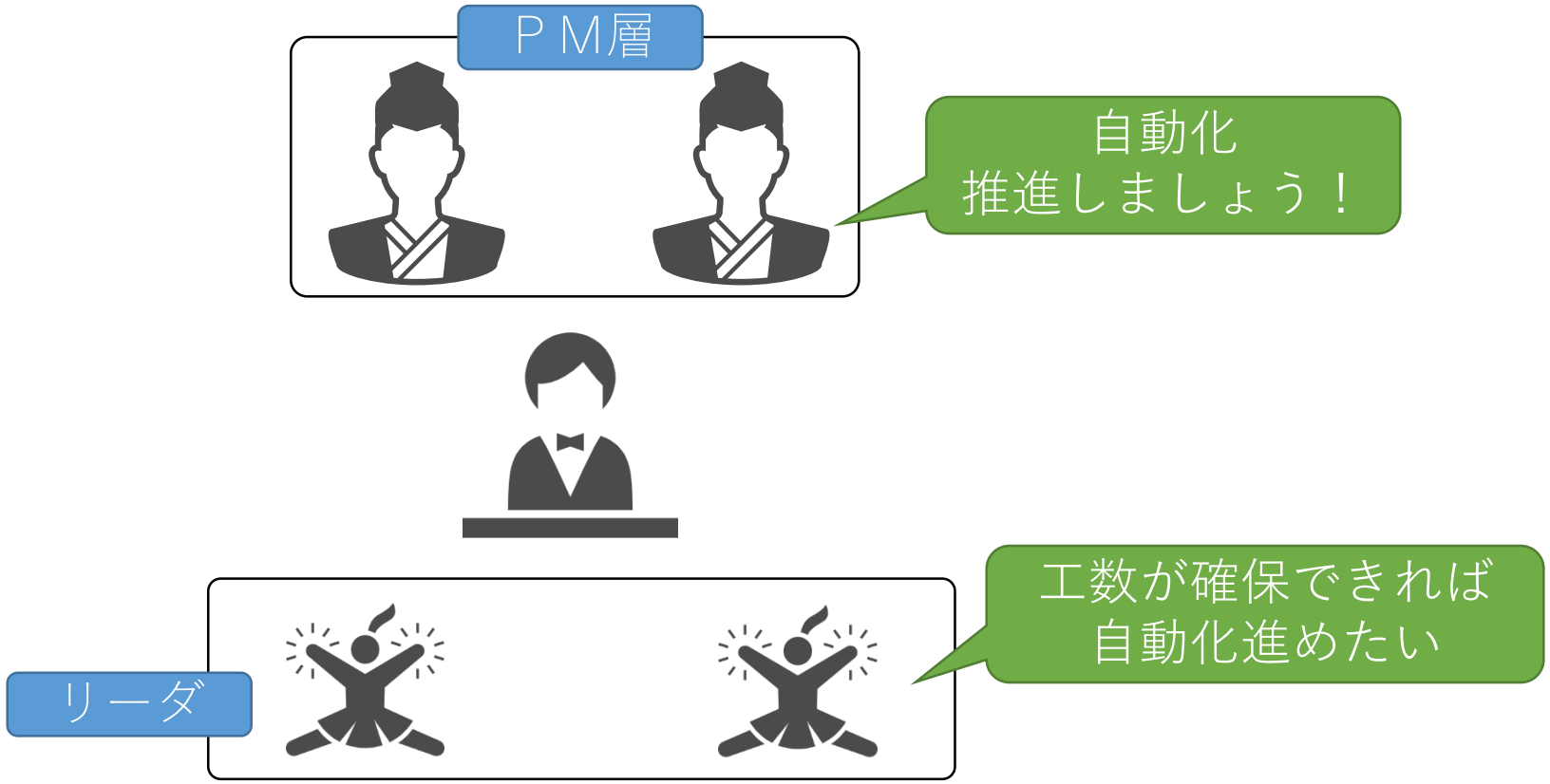
新プロジェクト設立に合わせて自動化推進役の確立

- 目的：組織としての結節点の確立と、自動化に専念するため
- 行動：新プロジェクト開始時に部署異動



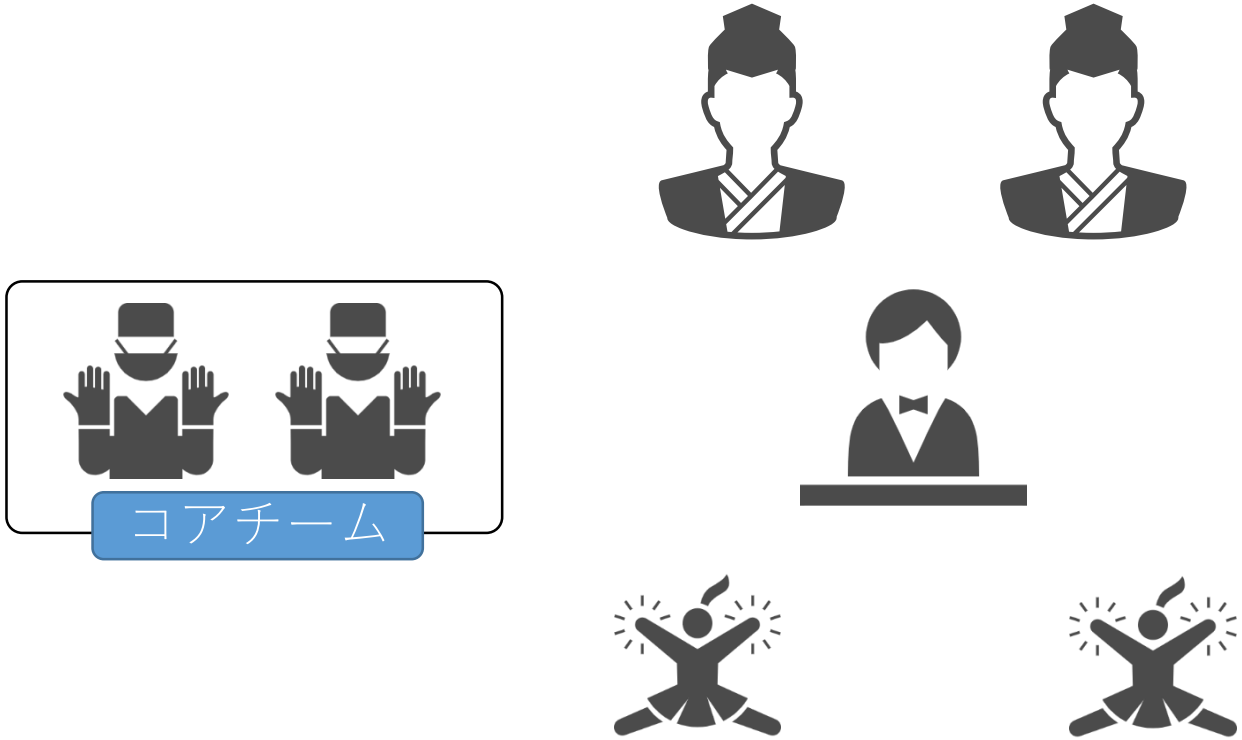
マネージメント層と実装担当のリーダとの合意形成

- 目的：早期の段階で、PM/PMOとの合意形成
- 行動：プロジェクト運営資料に自動化担当を明文化



ライブラリを作成するコアチームを設立

- 目的：追加になったユースケースに耐えうるライブラリにUpgradeするため
- 行動：半年かけて設立（ただし、メンバーは準専任）、運営には優先度付けが必須

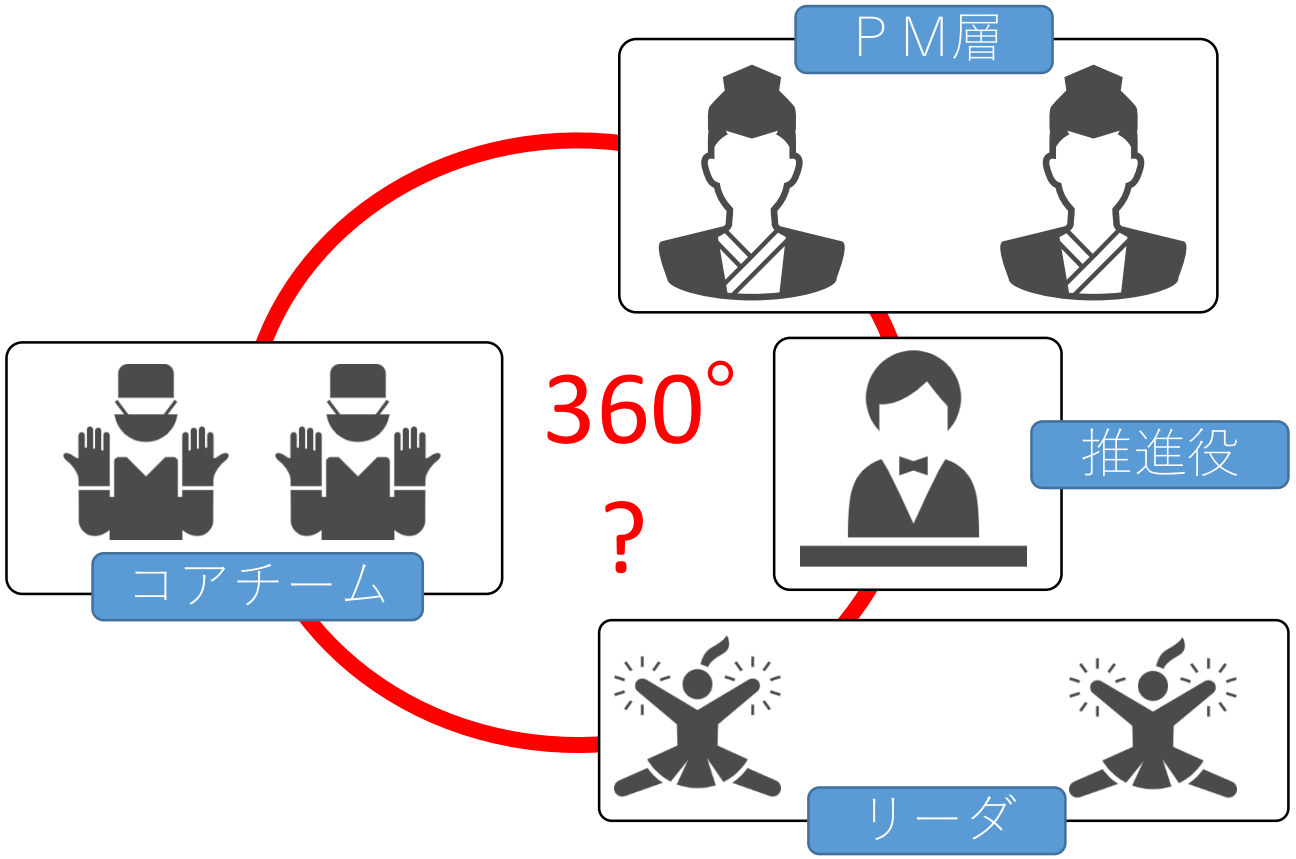


補足事項

- コアメンバーも自動化初心者
- 学びながら既存ライブラリの解説資料を教育資料をとして作成

甘かった予測

- PM/PMOや実装チームのリーダとの合意形成、コアチームの設立をもって360° connectが形成できたとばかり・・・

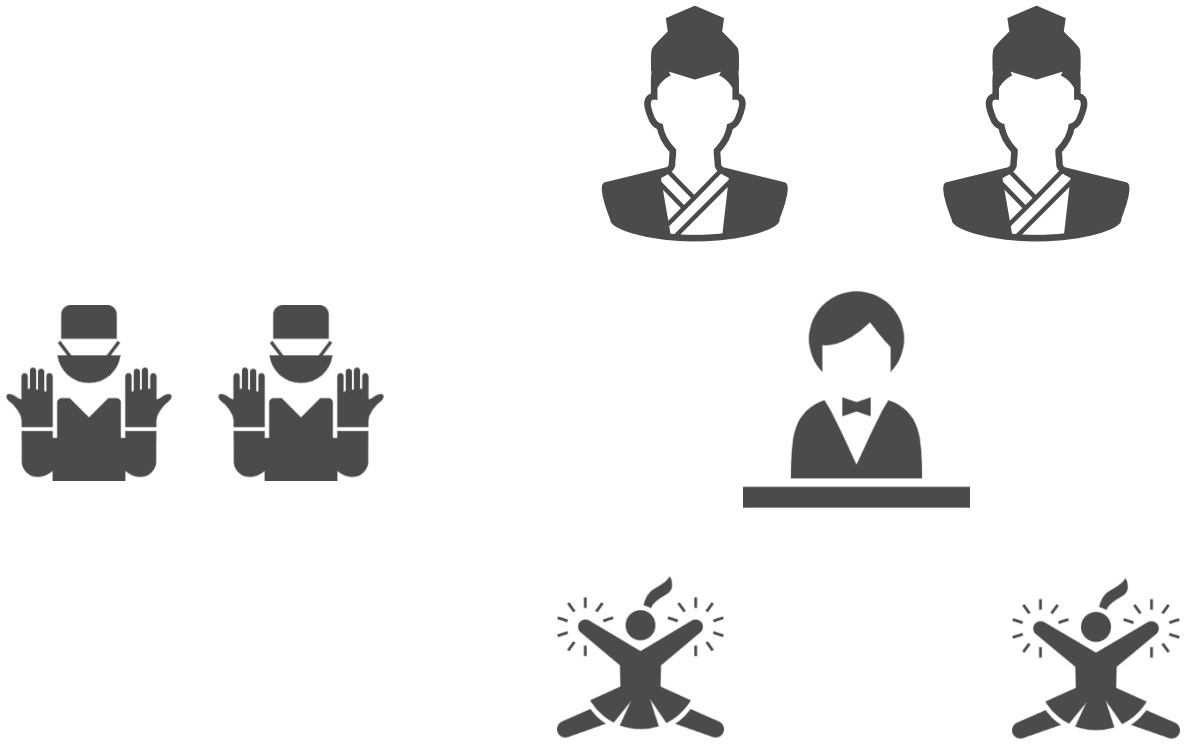


PL層の反発

◆1周目の時は、一時期、PLと自動化推進者を兼任していたため問題はなかった。

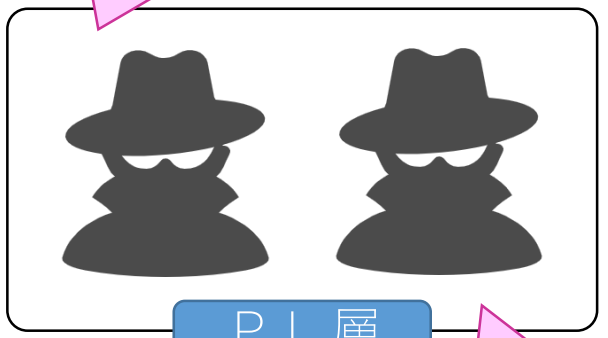
PL層の反発

◆1周目の時は、一時期、PLと自動化推進者を兼任していたため問題はなかった。



完全自動化されたら利用する

倍のコストは割けない



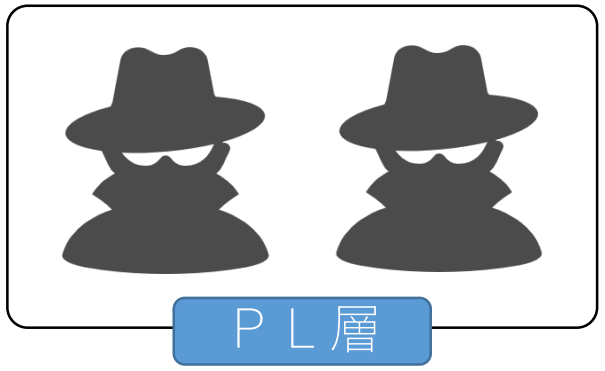
自動テストの
バグ摘出率が低い

自動化されれば
利用する

PL層へのご説明

◆完全自動化は、CI環境を構築したら利用可能

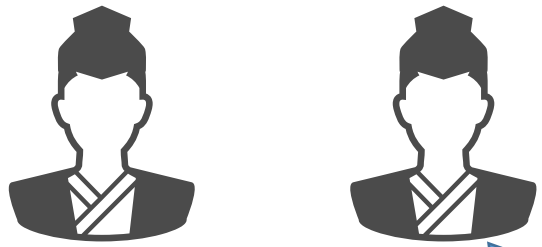
完全自動化されたら利用する



自動化されれば
利用する

PL層へのご説明

- 完全自動化は、CI環境を構築したら利用可能
- コストは、PM/PMOのサポート頂きながら1.2倍まで許容
  - 2周目の複数PJでは、0.9～1.2のコストで自動化実施
  - PJ内に回帰試験有



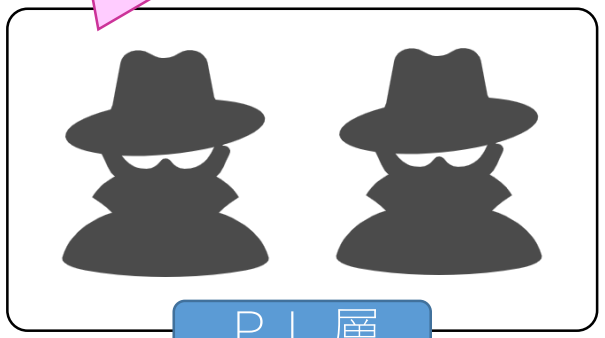
コアチームも支援します

段階的な開発を数回実施するため、
回帰試験で元が取れます

コスト比で手動テスト
の1.2倍まではＯＫ

完全自動化されたら利用する

倍のコストは割けない

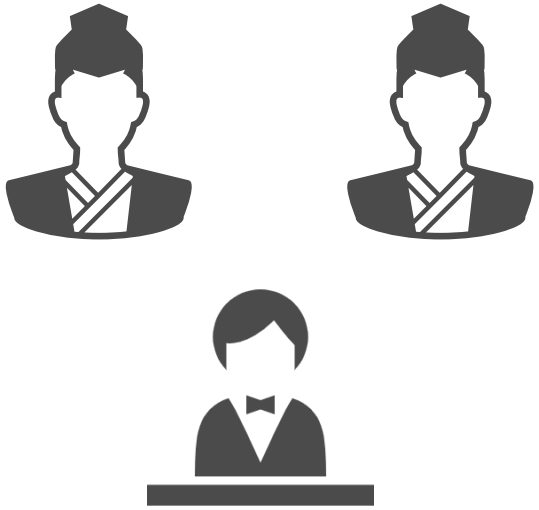


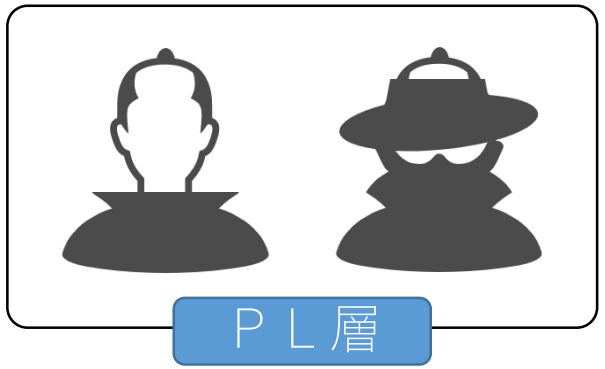
ＰＬ層

自動化されれば
利用する

PL層の反発の軟化

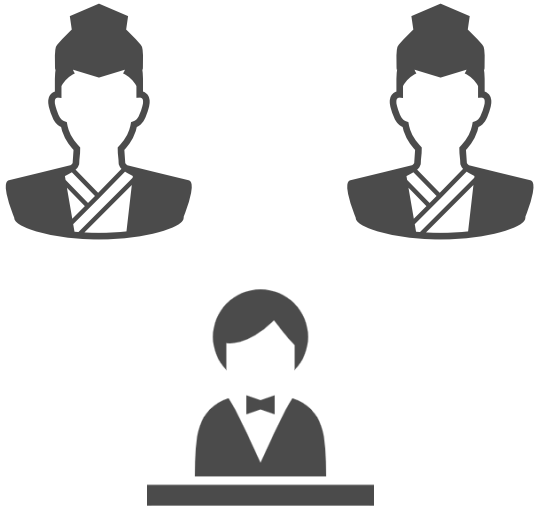
◆積極的には賛成ではないが、ご納得いただけた。

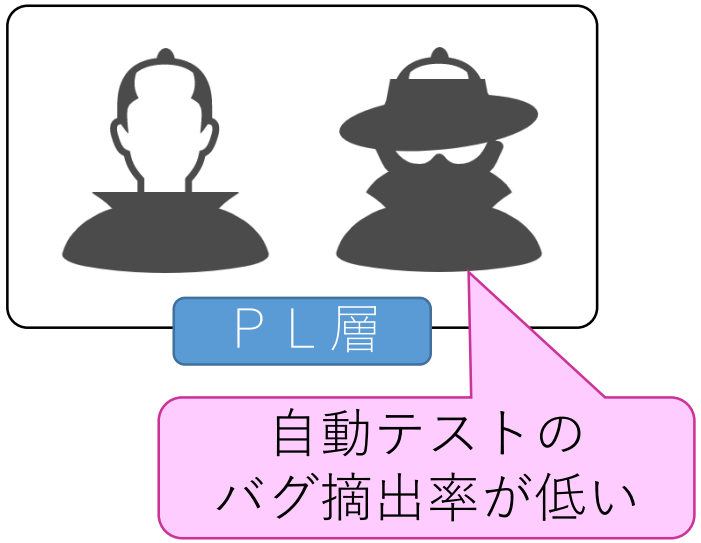




PL層の反発

◆どうしてもご納得いただけない方もおられ・・・

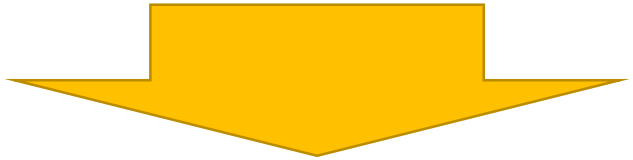




バグ摘出と、手動テストについて

◆「システムテスト自動化　標準化ガイド」より

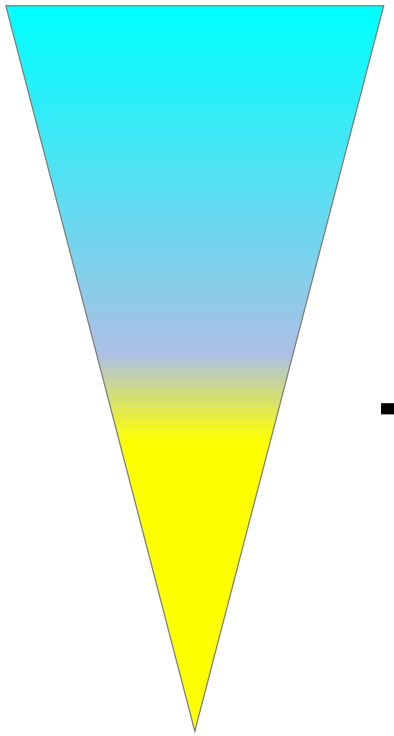
1. 手動テストの方が自動テストよりも多くの欠陥(＝バグ)が見つかる
   - ◇テストは**最初に実行**したときが**一番欠陥を見つけやすい**
   - ◇**自動化**したら、**最初にそのテストケースのテスト**を通常実施するが、それは**手動**



   - ◇よって、テストケースの欠陥は、**手動**で見つかることが多い

2. 手動テスティングは**なくならない**
   - ◇**物理操作、結果の確認**が**難しい**
   - ◇**予期しないイベント**が発生

テスト自動化する試験・しない試験（自動化経験者の議論 etcより）



するべき

パラメータが変化する試験

何度も行う試験(基本動作, 性能評価)

仕様変更の予感がする場所の試験

次の開発で使いそうな試験

新機能開発などバグが多そうな試験

複数回行う試験

一回で終わる試験

そもそも実施が難しい試験

一回ごとに結線が変わる試験

物理操作を伴う試験

コマンド一回で終わる試験

結果の確認が難しい試験

しなくても良いかな……

二度とやらない試験

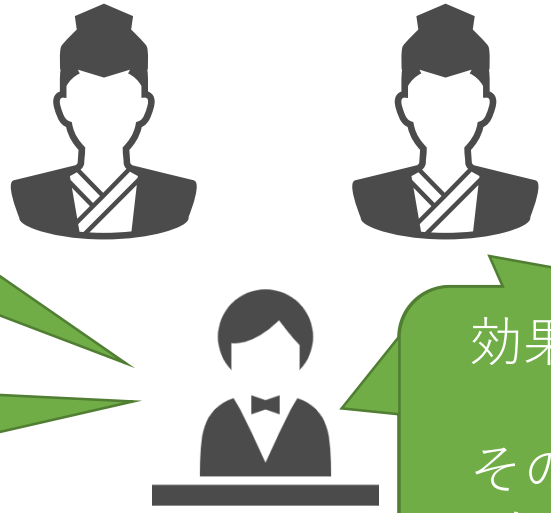
自動試験した方がバグを誘発しそうな試験

PL層の反発

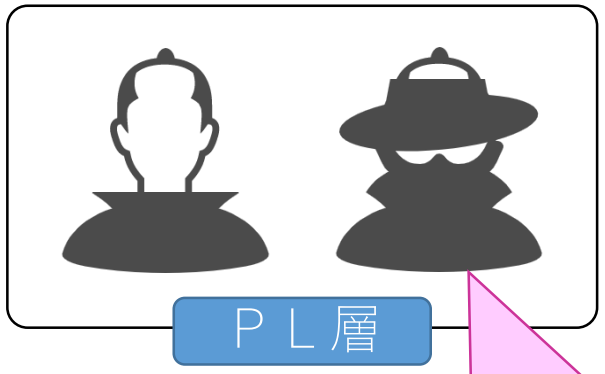
◆どうしてもご納得いただけない方もおられ・・・

手動テストの項目書を
自動化しているため、
基本的に同じはず

将来的には
自動テストによって
効率化された時間は
別作業に割り当て可能



効果的に手動テスト
と
その観点を取り込む
ことはできませんか？



自動テストの
バグ摘出率が低い

リーダ層とはいっても、開発外の部署の方だったため、
無理して進めることは可能かもしれないが・・・

組織について

◆「経営者の役割」チェスター・バーナード主著 より

◆組織とは「意識的に調整された2人またはそれ以上の人々の活動や諸力のシステム」

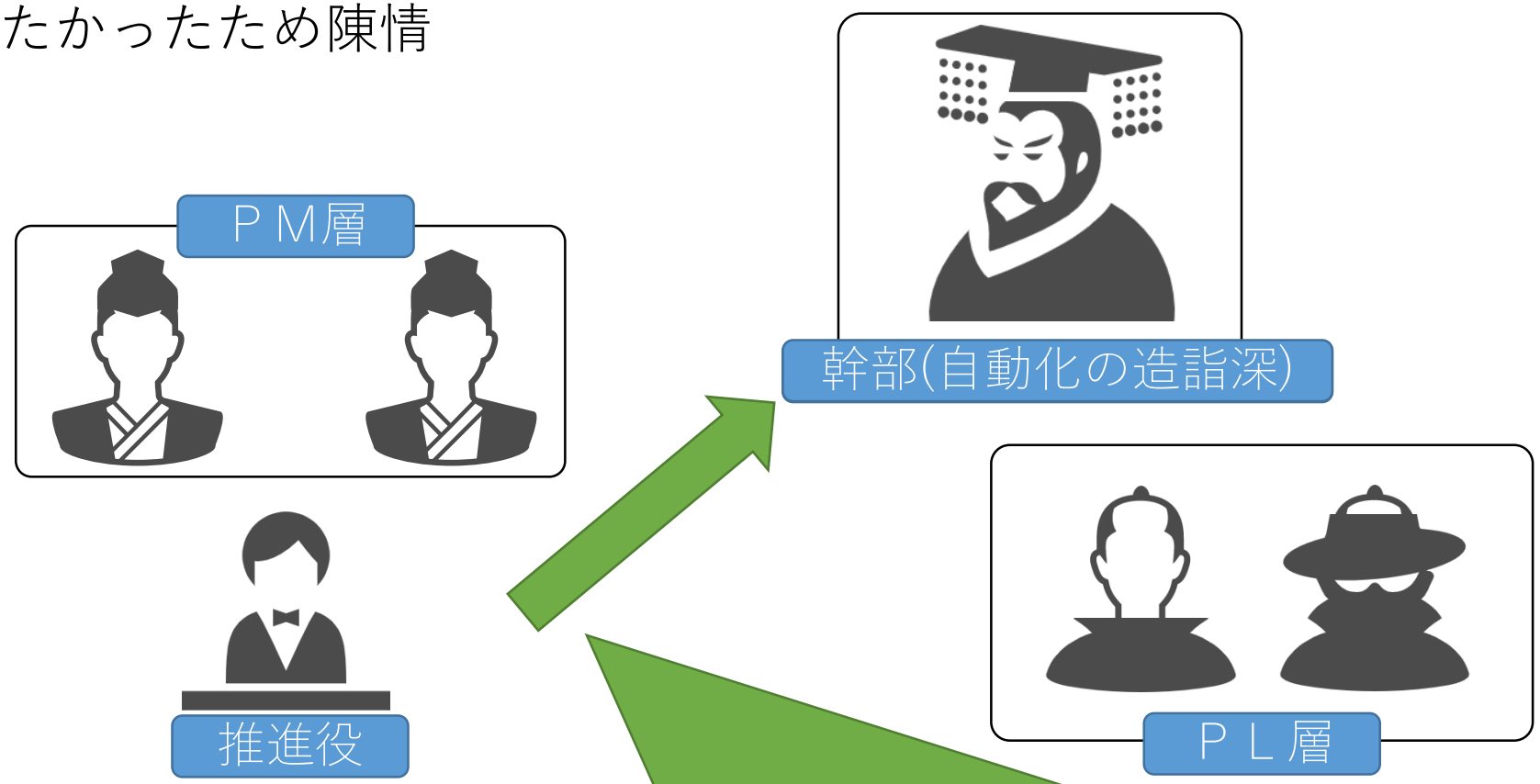
◇成立のための条件

1. **共通目的**（組織目的）
2. **協働意思**（貢献意欲）
3. コミュニケーション

組織の3要素の均衡が組織成立の条件であり、存続の前提

プロジェクトとして合意して進めるため、幹部に陳情

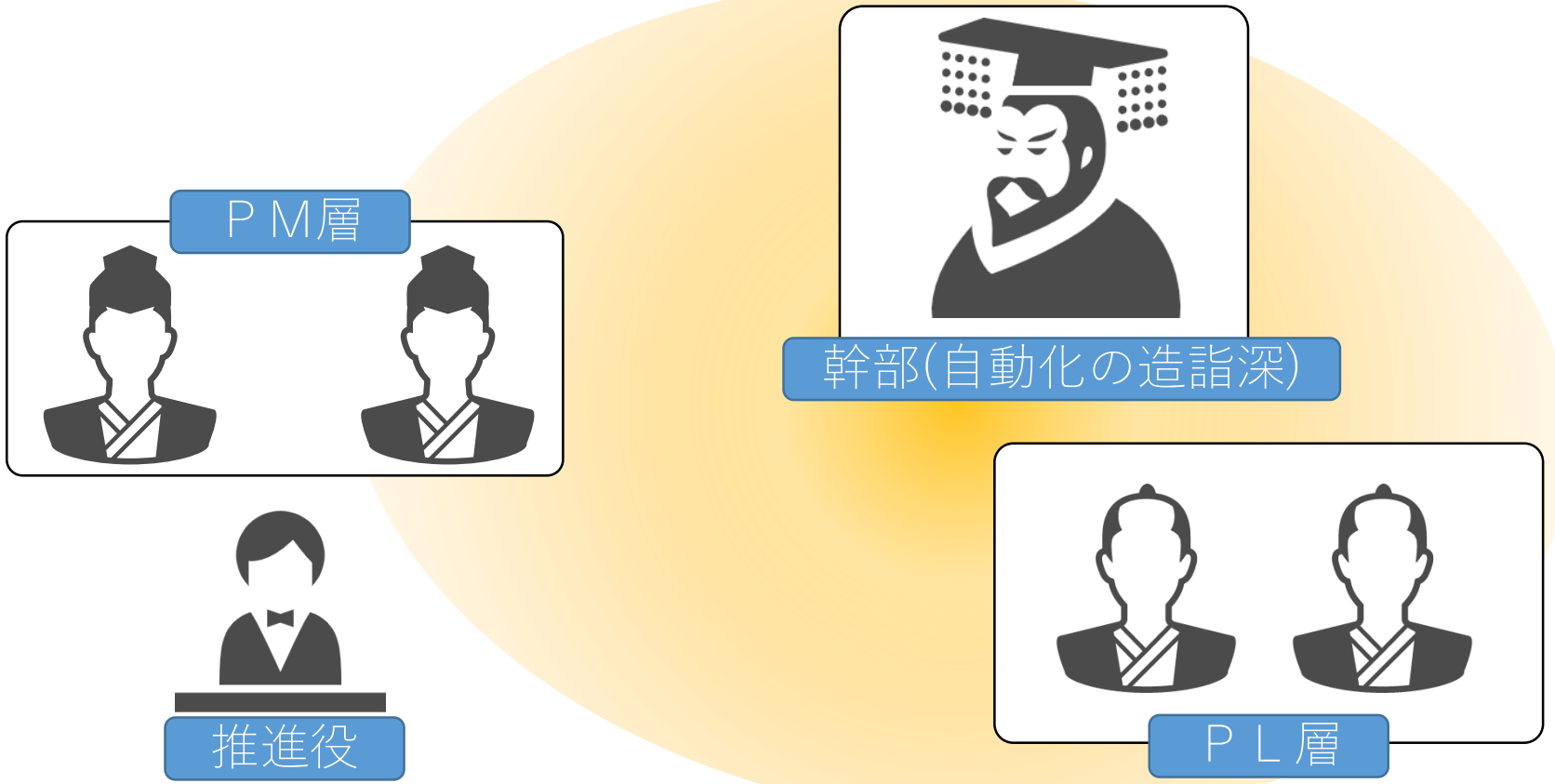
◆共通目的をもって進めたかったため陳情



・**技術(Step2)/プロセス(Step3)**の**準備自体**は**実施済み**です。

・社会的にも**自動化**の**必要性**は**高まって**おり、**会社**として**自動化**を**進めさせていただきたい。**

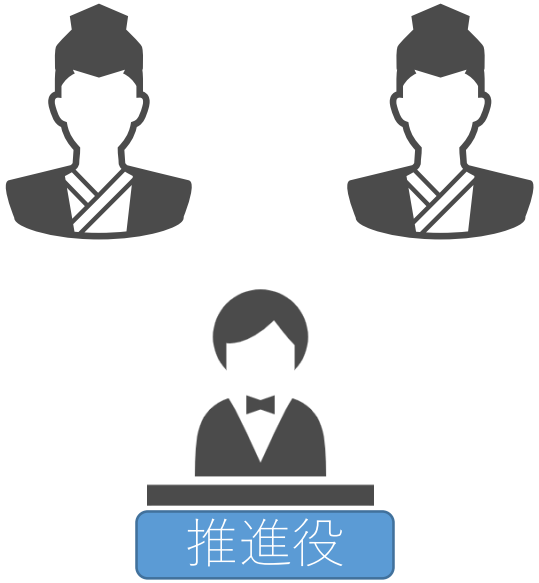
PL層の合意も獲得

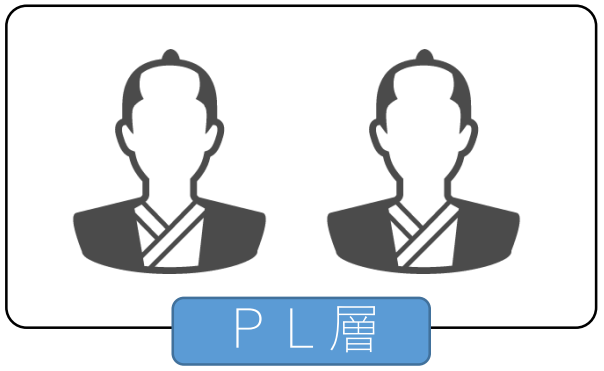


テスト自動化を**推進**する**組織構築**だったため、
自動化に造詣が深い幹部の存在がキーポイントでした。

改善目標値が明記されました。

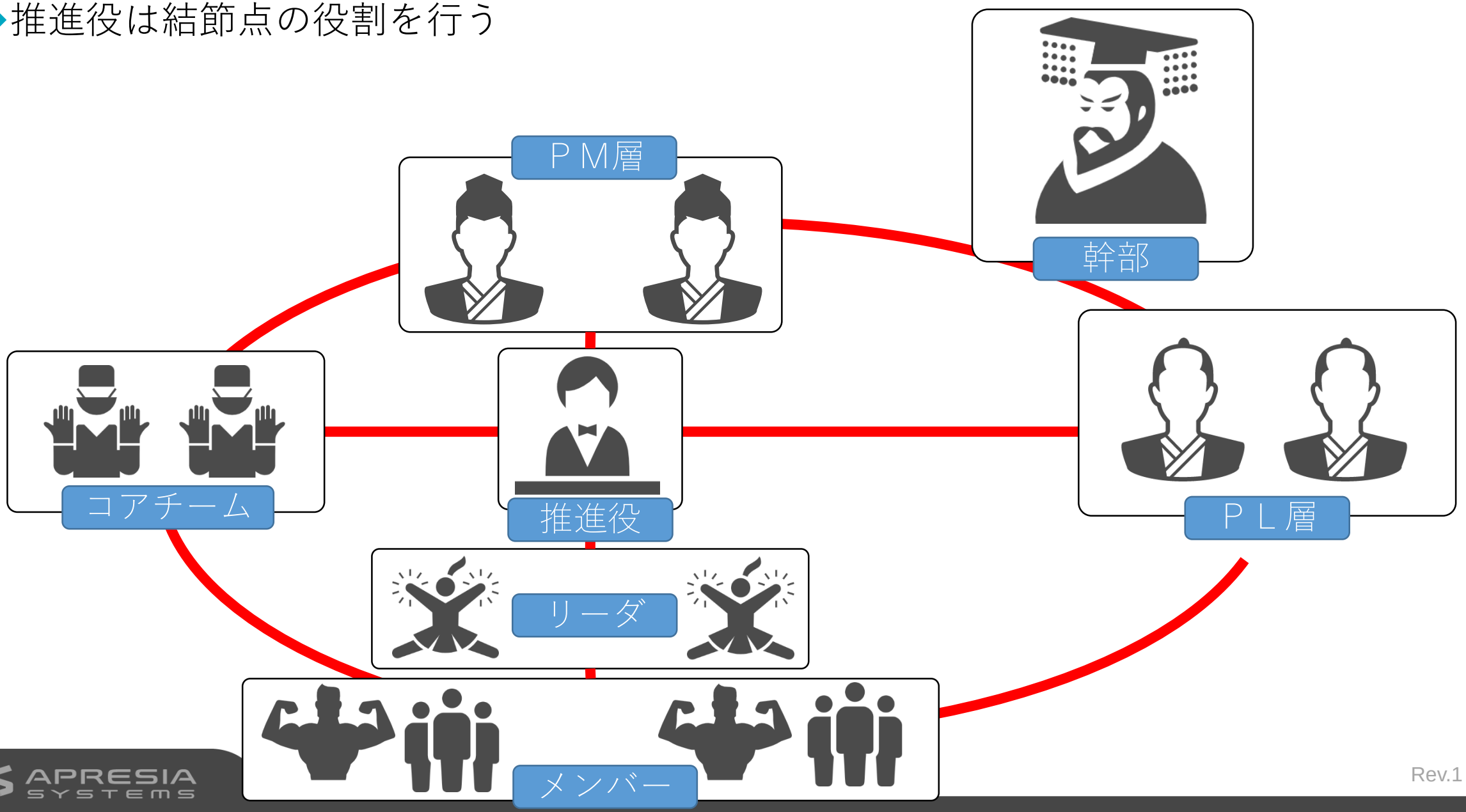
◆前年度比で**30%**のテスト自動化工数改善





開発メンバーもアサインして360°cross connectが完成！！

◆推進役は結節点の役割を行う



Step2
ライブラリのUpgrade

APRESIA SYSTEMS

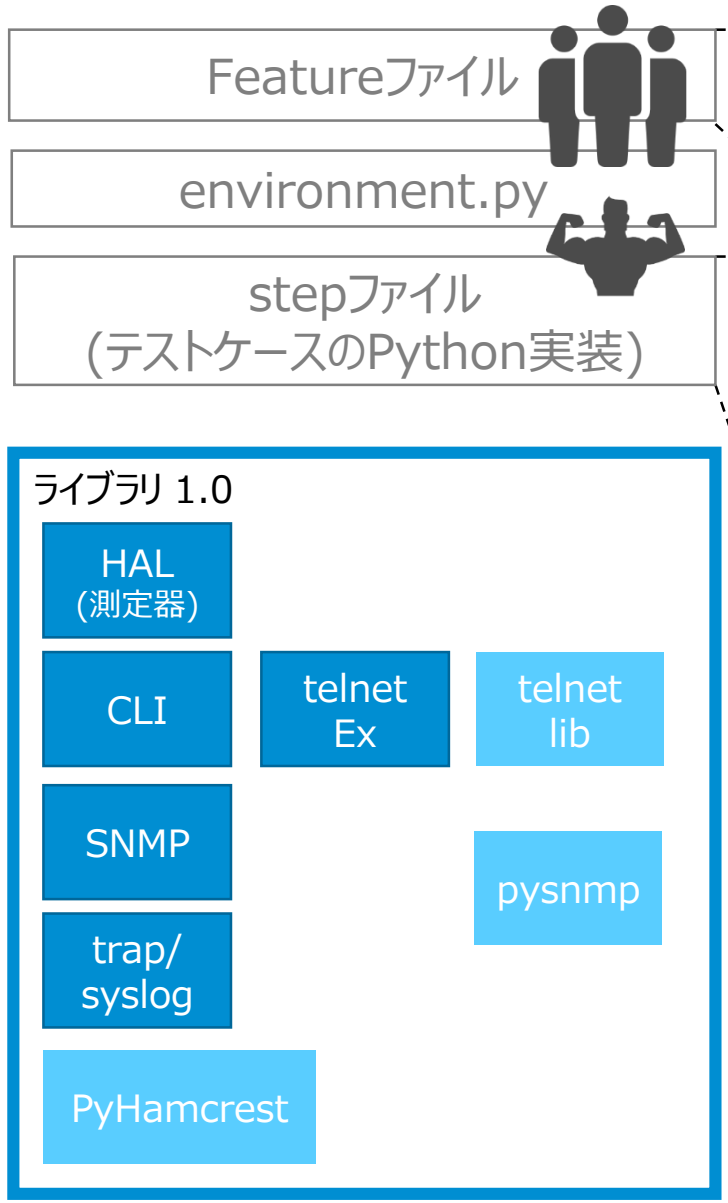
Rev.1

JANOG45

テスト自動化作業の品詞分解/分析

独自クラス　Python 公開パッケージ

- 確認したい内容(テストケース)を記述し、
- テスト環境を構築(setUp)し、
- テスト内容に沿って機器を操作し、
- 結果を収集して、
- 期待値と合っているかを確認すること。



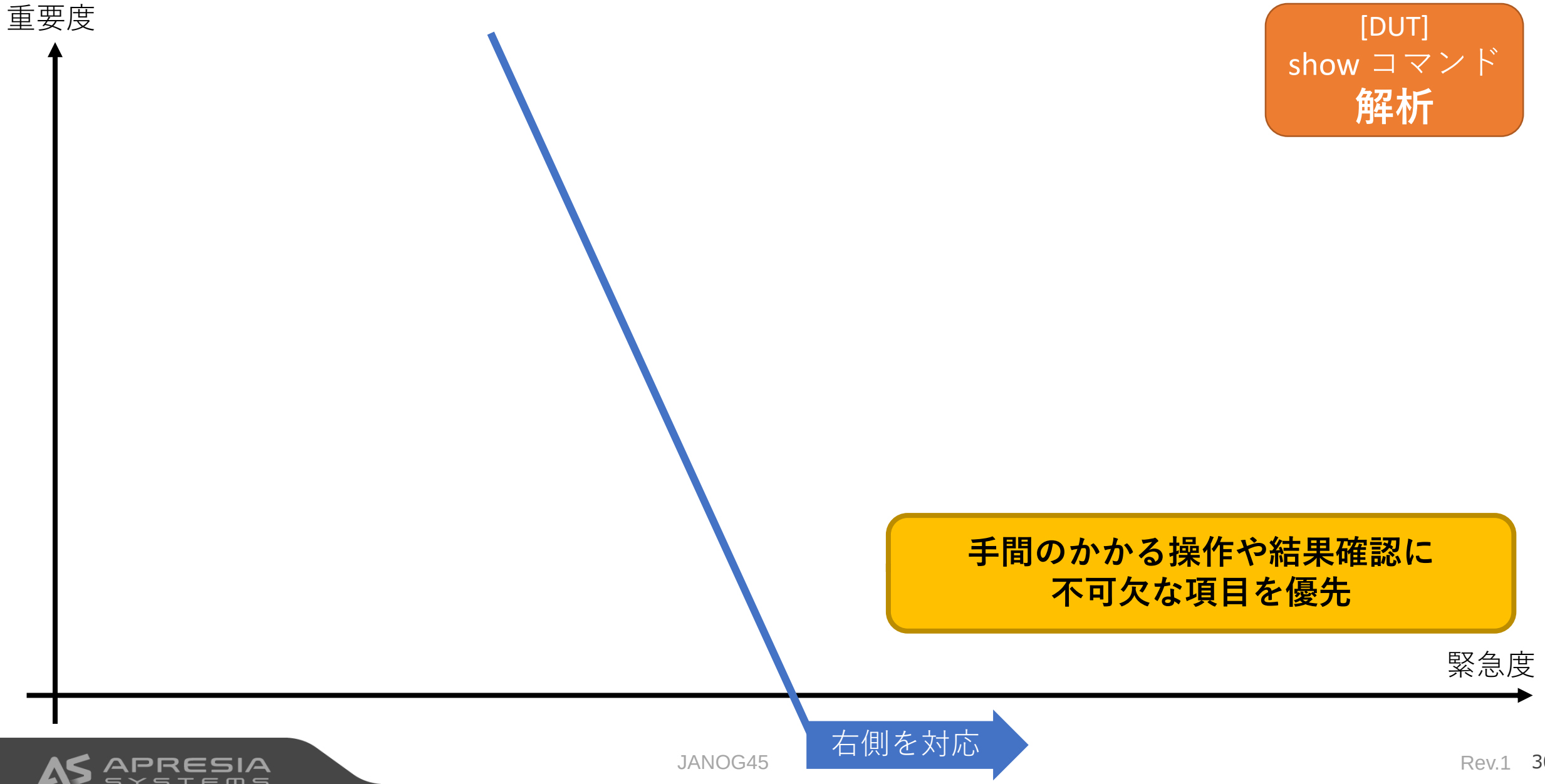
2周目以降でテストケースの**パターン増**

当時コアチーム不在のため対応するには、**メンバーに**よるstepファイルの**実装(新規/作り直し)が増**

自動化率 Down
⇩
ライブラリの Upgradeを検討

テスト環境の解放(teardown)はsetUpの逆手順のため省略

ヒアリングに基づくライブラリ化要望機能の緊急度・重要度マトリクス図



showコマンドの解析の効率化

◆機種Aの装置の状態表示(show status card)

表形式のパーサがあればOK!

```
Card         Type          Status          Boot SEM Config Temp     Firmware
---------- ---------- -------------- ---- --- ------ -------- --------
Manage 1     MM1-QC        Active         -    On   Sync   Normal   6.10.01
Manage 2     MM1-QC        Power down     -    -    -      -        -
Linecard 1   XG26040c-QC   In service     -    On   Sync   Normal   6.10.01
```

1行目：Key

2行目：最大文字数

3行目~：Value

◆機種Bでも問題なかったのですが、機種Cで実行したところ、

◆実行時エラーになり、原因を解析すると

showコマンドの解析の効率化

表形式のパーサがあればOK!

◆機種Aの装置の状態表示(show status card)

```
Card         Type          Status         Boot SEM Config Temp     Firmware
---------- ---------- ------------- ---- --- ------ -------- --------
Manage 1     MM1-QC        Active       -    On  Sync   Normal   6.10.01
Manage 2     MM1-QC        Power down   -    -   -      -        -
Linecard 1   XG26040c-QC   In service   -    On  Sync   Normal   6.10.01
```

1行目：Key

2行目：最大文字数

3行目~：Value

◆機種Cの装置の状態表示(show status card)

```
Card         Type              Status         Config Temperature      Firmware
                                                     Major    Minor
---------- -------------- ------------- ------ -------- ------ --------
Manage 1     Ap20000-8X4T-AC Active       Sync   Warning  Normal 7.00.02
```

2行に！！

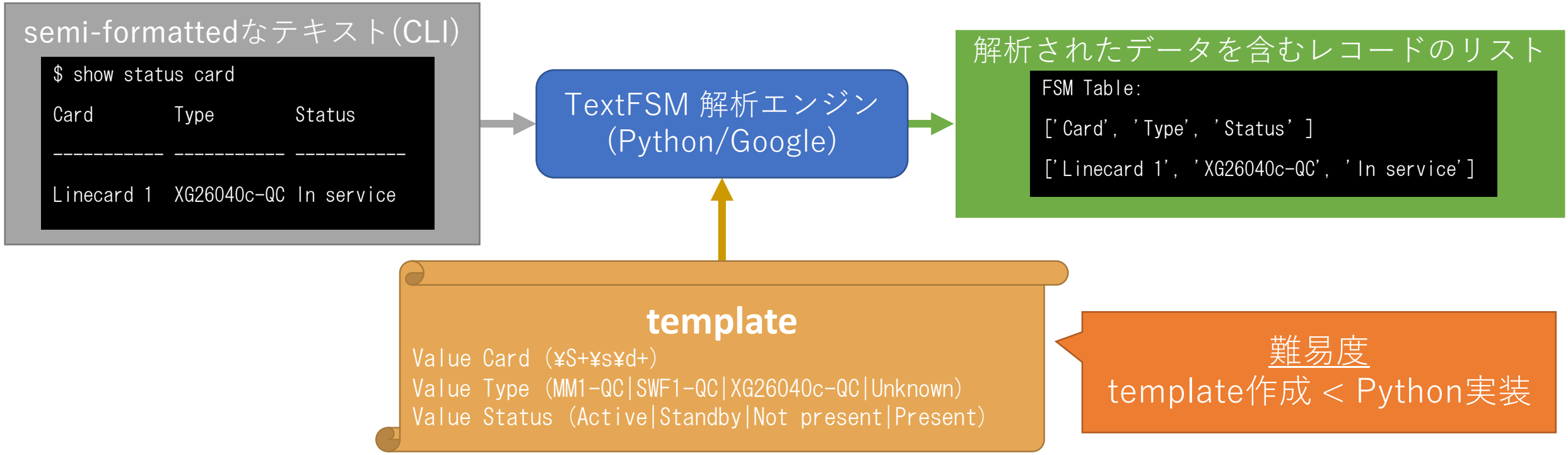
最大文字数の
判定行が浮動

Key取得の複雑化

全フォーマット変更を吸収可能な**統一的なパーサ**を作るのは**難しい**
（個々に正規表現でPython実装するのは大変！）
⇩
JANOG41で紹介されていた**TextFSM**にて負荷軽減

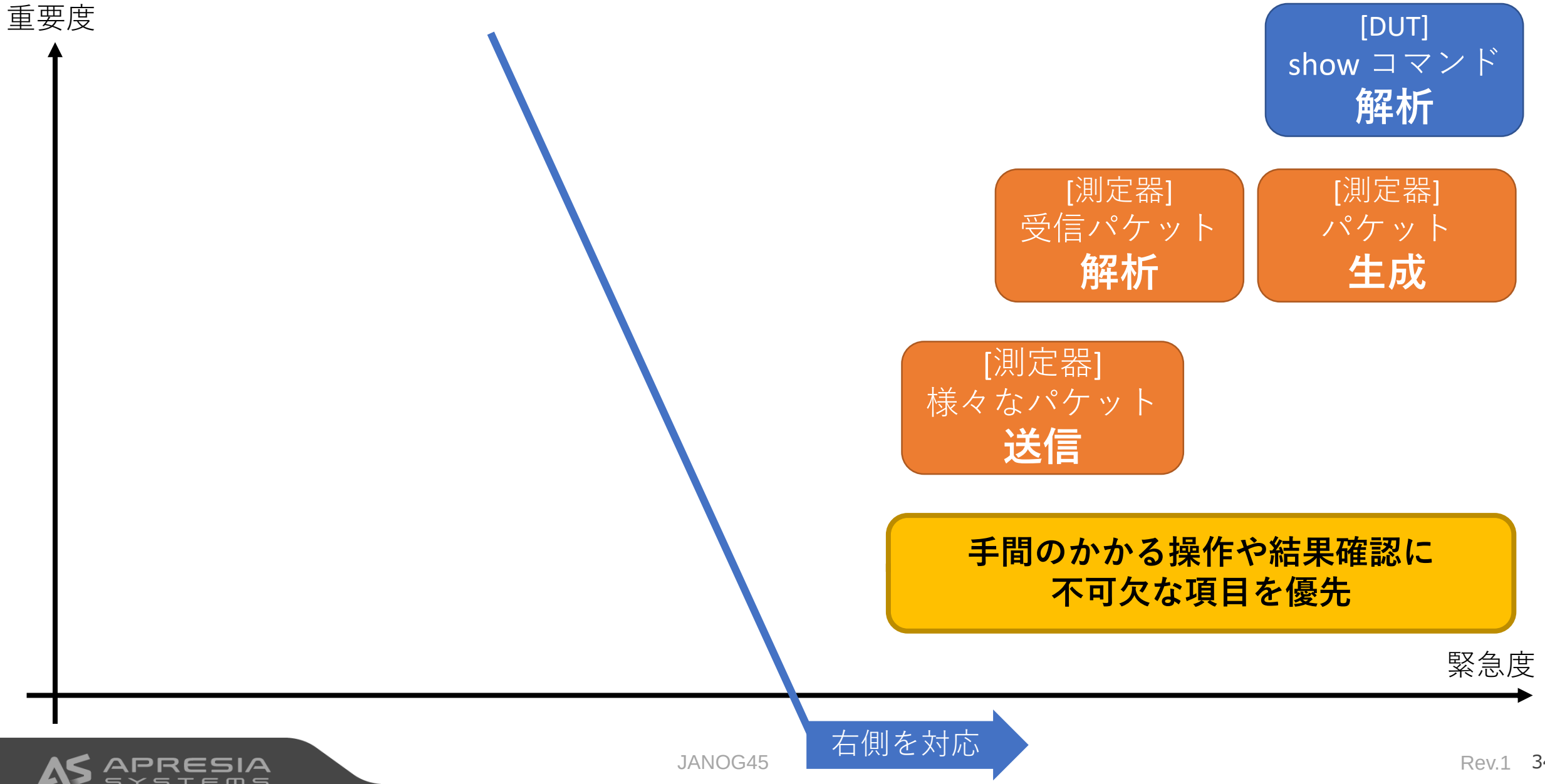
TextFSM (Finite State Machine)について

◆TextFSM(Finite State Machine)のおさらい



◆数十万行に及ぶshowコマンド(例：show fdb)を解析するため、TextFSMをバッチ処理ではなく、イテレーティブに実行できるように独自拡張

ヒアリングに基づくライブラリ化要望機能の緊急度・重要度マトリクス図



測定器制御ライブラリの高度化（パケットの生成・送信・解析）

- 生成・送信制御
  - 既存プロジェクトテスト項目書の調査＆ヒアリングより下記を作成
    - **L2~L４までのパケットを階層的に定義**可能
      - 各要素は**辞書形式**で**指定**可能

Featureファイルの例

```
Scenario:
    Given Set L2 parameters
      |name       |da                |sa                |tpid1 |vid1|vmode1|sa_mode|
      |test-frame1|01:80:C2:00:00:02|00:40:66:26:50:60|0x88a8|4094|vIdle |       |
    Given Set L3 parameters
      |name|proto|sip         |dip         |v4 tos|v4 frag|v4 ttl|v6 TC|v6 FL|v6 HL|
      |L3_1|IPv4 |192.168.1.1 |192.168.1.2 |255   |2      |128   |     |     |     |
    Given Set PBB parameters
      |name |proto|ttl|eid|itag|isid |customer da      |customer sa      |tpid |pri|cfi|vid |
      |PBB_1|PBB  |   |   |0   |11201|00:00:5E:AA:AA:01|00:00:5E:BB:BB:01|     |   |   |    |
```

これまでは、複雑なパケットはbyte列で指定する必要があった

      - ストリーム／（ストリームを束ねた）**グループ単位**送信のラッパAPI

測定器制御ライブラリの高度化（パケットの生成・送信・解析）

◆受信パケットの解析

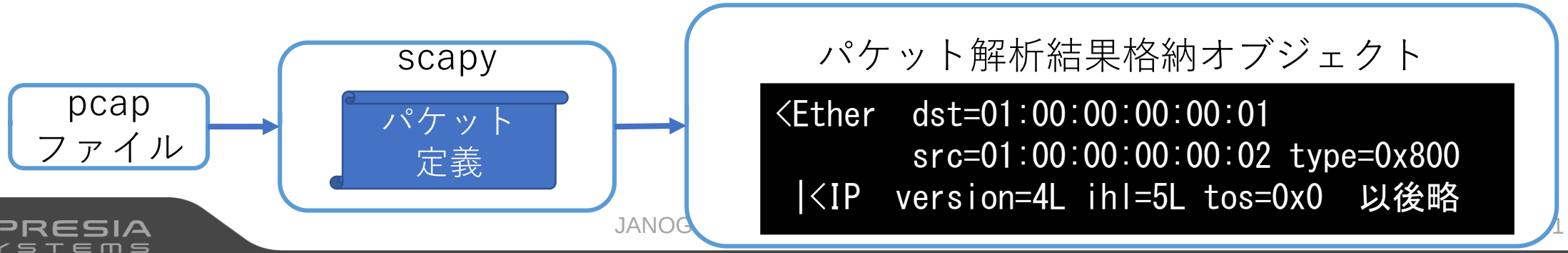
◇ライブラリを調査し、**バランス**的に**scapy**を採用

ライブラリ	構造化	解析速度	利用難易度	備考
scapy	◎	△	◎	送信機能有, GPLv2
pyshark	×	不明	ー	
construct	○	×	ー	
hachoir	△	△	ー	
Kaitai Struct	◎	◎	×	MIT(Runtime)

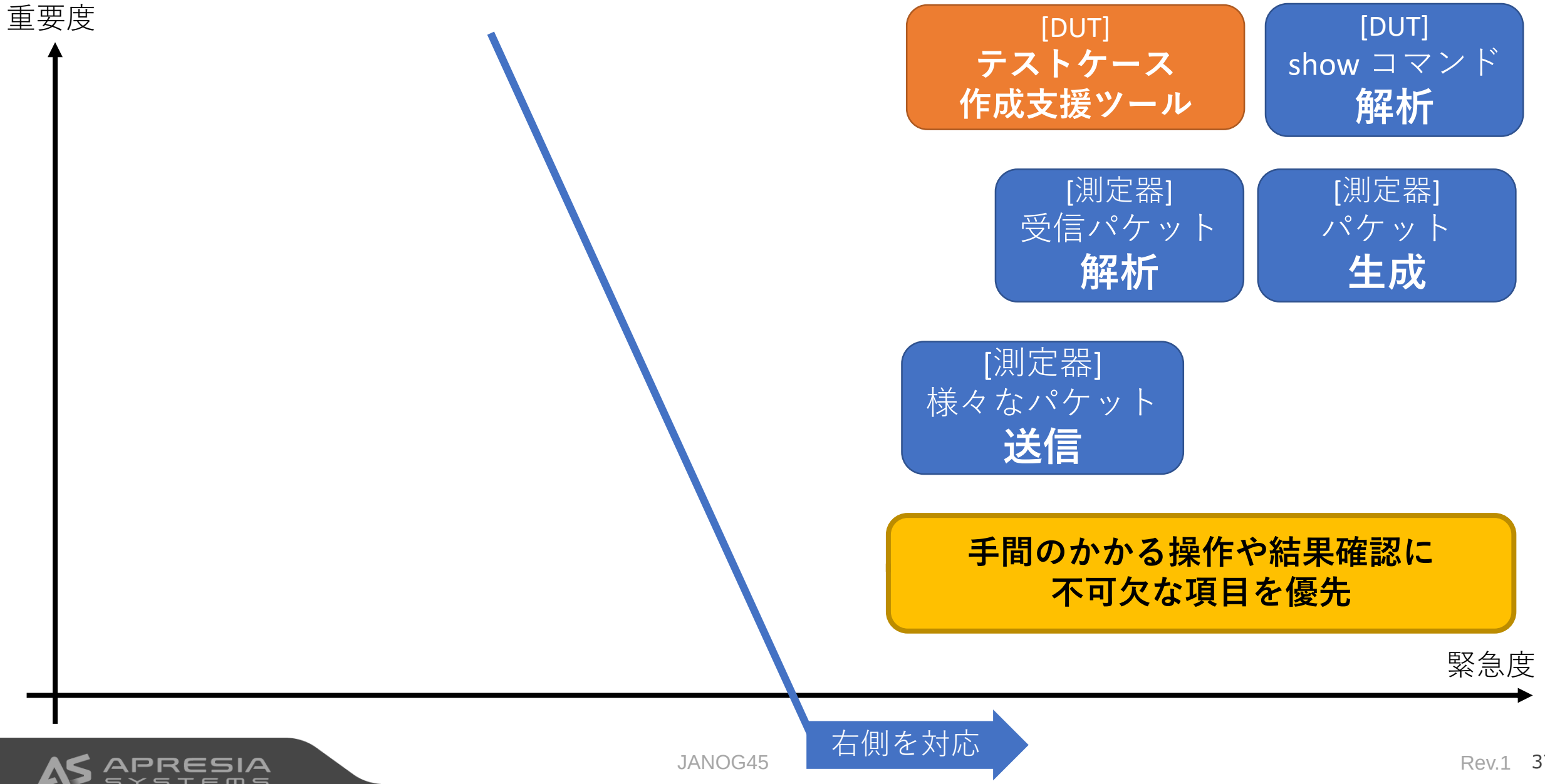
基準は私見

◇補足[構造化]

ーscapy内にパケットの独自定義があり解析に利用

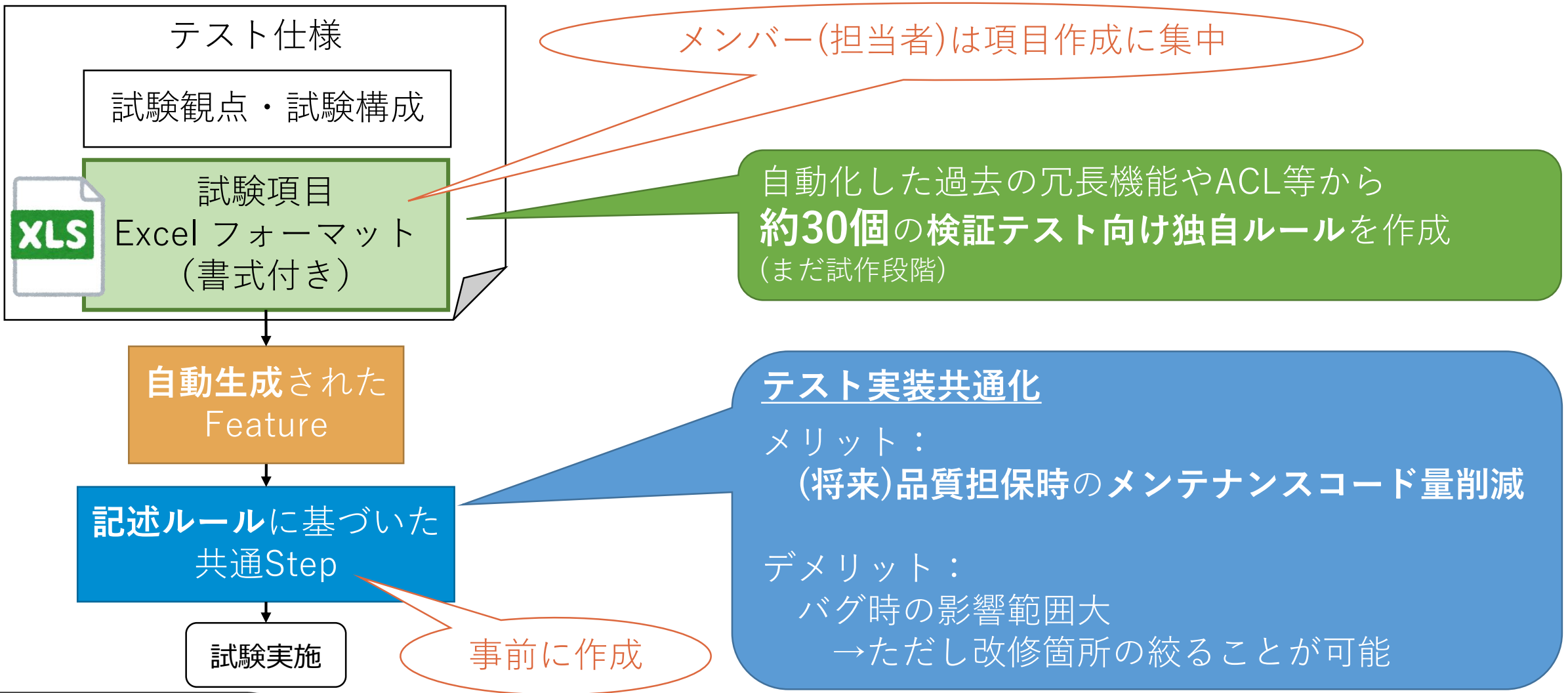


ヒアリングに基づくライブラリ化要望機能の緊急度・重要度マトリクス図

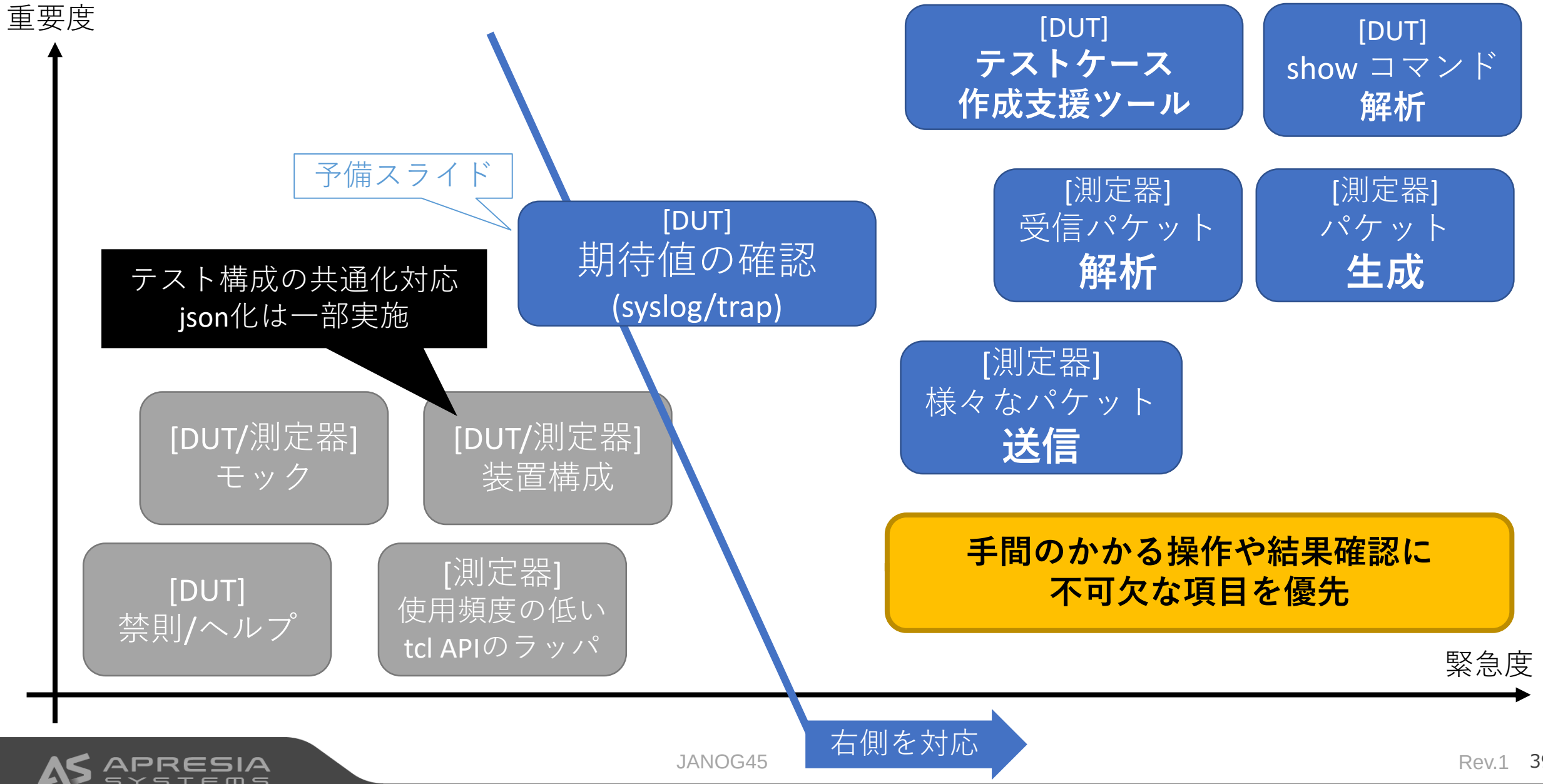


テストケース作成支援(テスト項目書 兼 テストケース自動生成)ツールについて

◆実はJANOG41の発表のなかで、いつかできたらいいなと申し上げていた内容です。



ヒアリングに基づくライブラリ化要望機能の緊急度・重要度マトリクス図



テストケース作成支援ツールを含めた全体像

テスト項目書 兼 テストケース生成支援ツール

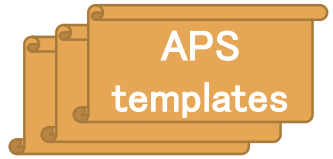
自動生成されたテストケース (Featureファイル)

共通stepファイル

Featureファイル

stepファイル

environment.py



ライブラリ 2.0 (最新)

Traffic generator Arbistration Interface(TAI)

Generator | Sender | Receiver (Analyser)

SSH

syslog v2

trap v2

PyHamcrest Ex

TextFSM Ex

独自クラス

ライブラリ 1.0 (2016年当時)

HAL (測定器)

CLI

logging Ex

telnet Ex

logging

telnet lib

shslog/ trap

SNMP

pysnmp

PyHamcrest

外部クラス

TextFSM

Step3
プロセスのUpdate

APRESIA SYSTEMS

対策した内容一覧

従来(1周目)のテストプロセス

- 計画
  - 粗見積もりの作成
  - テスト戦略の決定

- （対象者）教育

- 設計

- 実装
  - 項目書作成
  - (自動化)テストケース作成
  - Python実装/開発環境構築

- 実行

- 実績記入(総項目数、実行時間 etc)

- 振り返り(アンケート＆インタビュー)

ソース管理

全行程共通

作業時間
記録

Q&A
作業管理

1. プロセスの簡略化

◆計画
- ◇粗見積もりの作成
- ◇テスト戦略の決定

◆（対象者）教育

◆設計

◆実装
- ◇**(項目書作成)：省略可(PL層で判断)**
- ◇(自動化)テストケース作成
- ◇Python実装/開発環境構築

◆実行

◆実績記入(総項目数、実行時間 etc)

◆振り返り(アンケート＆インタビュー)

ソース管理

全行程共通

作業時間
記録

Q&A
作業管理

2. 目標設定

この箇所についてご説明

◆計画
- ◇ **粗見積もりの作成**
- ◇ **テスト戦略の決定**

◆（対象者）教育

◆設計

◆実装
- ◇ **(項目書作成)：省略可(PL層で判断)**
- ◇ (自動化)テストケース作成
- ◇ Python実装/開発環境構築

◆実行

◆実績記入(総項目数、実行時間 etc)

◆振り返り(アンケート＆インタビュー)

ソース管理

全行程共通

作業時間
記録

Q&A
作業管理

2. 目標設定

Start
開発の初期段階で自動化の粗見積もり

2. 目標設定～テスト自動化で取得しているデータ一覧～

テスト戦略作成

項目			計画	実績	備考
機能名			○	○	対象外になった時のみ実績更新
担当			○	○	
項目関連	①	総項目数	○	○	
	②	**流用項目数**	○	○	
	③	**流用自動化項目数**	○	○	
	④	新規自動化項目数	○	○	
	⑤	自動化率	○	○	計算：（③＋④）／①
時間関連 [単位は時]	❶	自動化対応工数	ー	○	
	❷	自動試験実施工数	ー	○	
	❸	手動試験実施工数	ー	○	
	❹	回帰試験時に削減可能な自動化対応工数	ー	○	想定
	❺	全項目手動試験実施時の工数	ー	○	想定
	❻	コスト効果	ー	○	計算：❺ ー（❶ ＋ ❷ ＋❸）

（注）不具合関連はソフトウェア開発のプロセスで記録

2. 目標設定～テスト戦略～

Start
開発の初期段階で自動化の粗見積もり

予想自動化率が**60%超**

Yes

既存自動化にて対応
（ライブラリ2.0も適宜活用）

No

要因分析

・他機能のテスト実装を使えないか？
・技術面/教育面でサポートできるか？etc

予想自動化率が**50%超**

Yes

他機能のテスト実装や
ライブラリ2.0を活用して対応

No

No

自動化対象外
・そもそも時間的に厳しい
・物理操作中心機能

テストケース作成支援ツールを
各機能の担当者にて活用

2. 目標設定～将来対応にした箇所～

品質担保のプロセスは重く辛い・・・

◆**品質**は将来対応

◇デバッグツール扱い、テスト実装のコーナーケースをつくことはあまり無い

◇テスト項目書の結果記載は手動のため（目で結果を見ている）

◇共通Stepファイルを作成していくことで準備は進める。

◇参考：CMMI(https://www.sea.jp/CMM/publish/CMM-J99.html)

成熟度レベル		特性（抜粋）
Lv.1	初期	場当たり的、成功は個人の努力に依存
Lv.2	反復できる	基本的なプロジェクト管理プロセスが確立
Lv.3	定義された	プロセスが文書化、標準化、統合化
Lv.4	管理された	成果物品質に関する詳細な計測結果収集
Lv.5	最適化する	革新的なアイディアや技術の試行、プロセスからの定量的フィードバックによって継続的なプロセス改善が可能

今回の目標

Updateしたテストプロセス

◆計画
- ◇ **粗見積もりの作成**
- ◇ **テスト戦略の決定**

◆ **（対象者）教育**

この箇所についてご説明

◆設計

◆実装
- ◇ **(項目書作成)：省略可(PL層で判断)**
- ◇ (自動化)テストケース作成
- ◇ Python実装/開発環境構築

◆実行

◆実績記入(総項目数、実行時間 etc)

◆振り返り(アンケート＆インタビュー)

ソース/**教材**管理

全行程共通

作業時間
記録

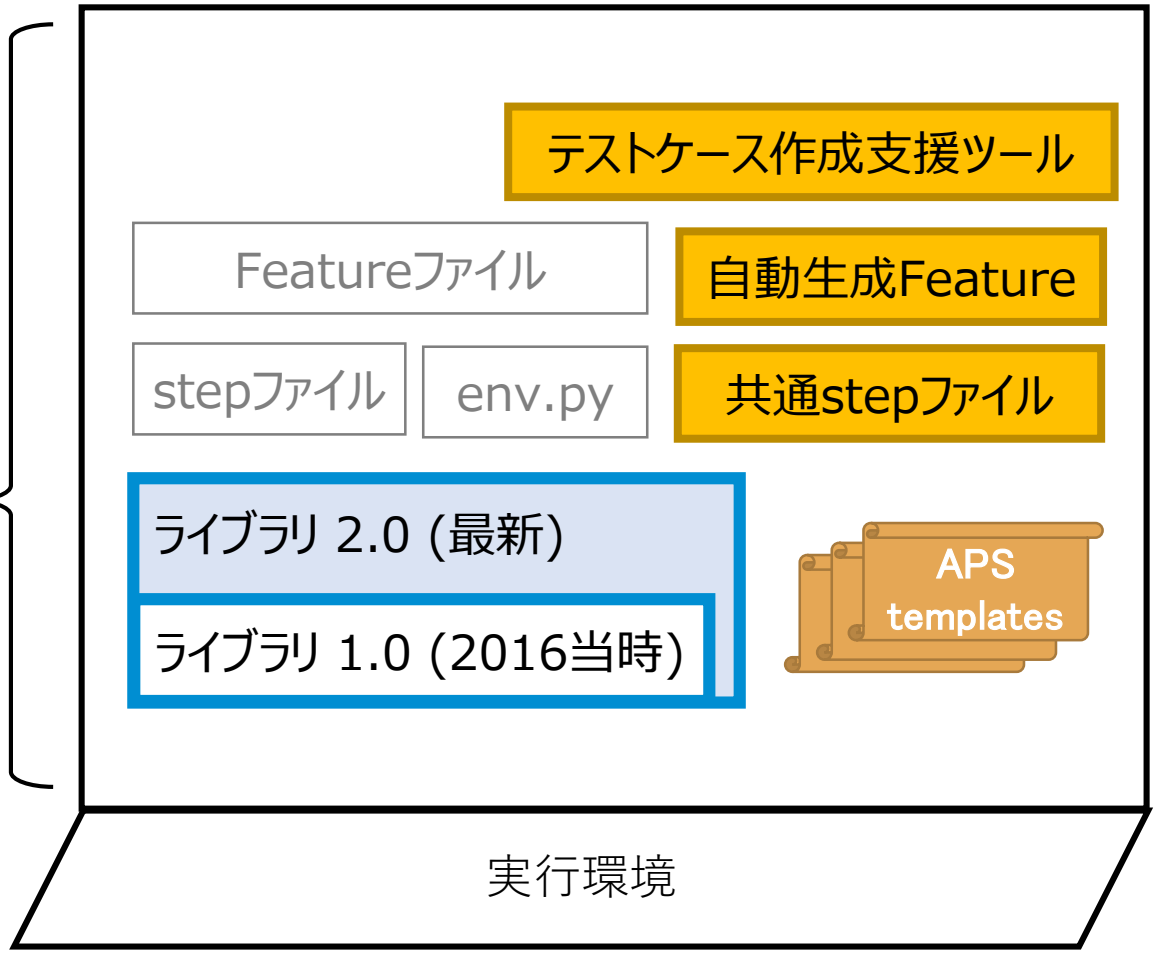
Q&A
作業管理

3. 教育

◆**教材を作成**し適宜**トレーニング**を実施
◇各教材に**サンプル**を作成し**GitLab**管理

◆テストの基礎技術の解説/環境構築
◇**BDD**の基本説明、**behave**の解説
◇**インストールガイド**(**Python3版**)

◆ライブラリ関連
◇**APIリファレンス**と**解説**(ライブラリ1.0)
◇テストケース作成支援ツールの**仕様書**
◇TextFSMの**template作成**方法解説



3. 教育 ~TextFSM の普及活動について~

- ◆**教材**を作成
  - ◇template
    - –**showコマンド**を分類/作成
      - 難易度**高**を中心に**約50個**
      - **テスト結果込み**でGitLab管理
  - ◇templateの記述方法
    - –基本編
      - 変数定義
      - 状態/アクション定義
    - –応用編
      - Optionの利用方法 etc

- ◆**トレーニング**を**複数回**実施
  - ◇内容は同じだが、1回目で分かりにくかった方は2回目にも参加

ライブラリは作ったら終わりではなく、**トレーニング**も実施することで**チームに浸透！**

Updateしたテストプロセス

◆計画
- ◇ **粗見積もりの作成**
- ◇ **テスト戦略の決定**

◆ **（対象者）教育**

◆設計

◆実装
- ◇ **(項目書作成)：省略可(PL層で判断)**
- ◇ (自動化)テストケース作成
- ◇ Python実装/開発環境構築

◆実行

◆実績記入(総項目数、実行時間 etc)

◆振り返り(アンケート＆インタビュー)

ソース/**教材**管理

全行程共通

作業時間
記録

Q&A
作業管理

対策のまとめ

Step1

組織の再構築

Step2

ライブラリのUpgrade

Step3

プロセスのUpdate

360° cross connect

ライブラリ 2.0

プロセスを一部省略可能 +
教材の更新 /トレーニング実施

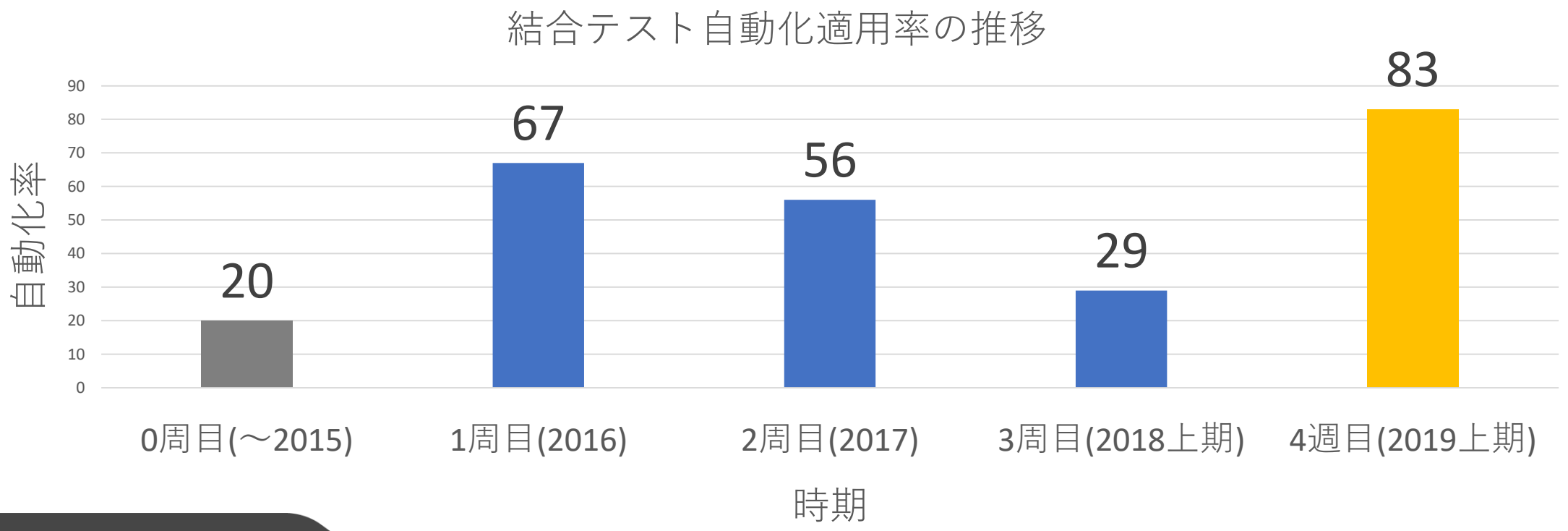
結果と考察

APRESIA SYSTEMS

結果

- テスト自動化率が**83%**に改善
  - 単独PJで**10万件以上**のテスト自動化を実施
- **手動テスト**(実行）に比べて自動テスト(実装+実行) のコスト は**29% 削減**
  - また、自動テスト間でも**前年度比(正規化)**で**48.6%** 改善

(注)ライブラリ作成コスト除く：外部ツール扱い



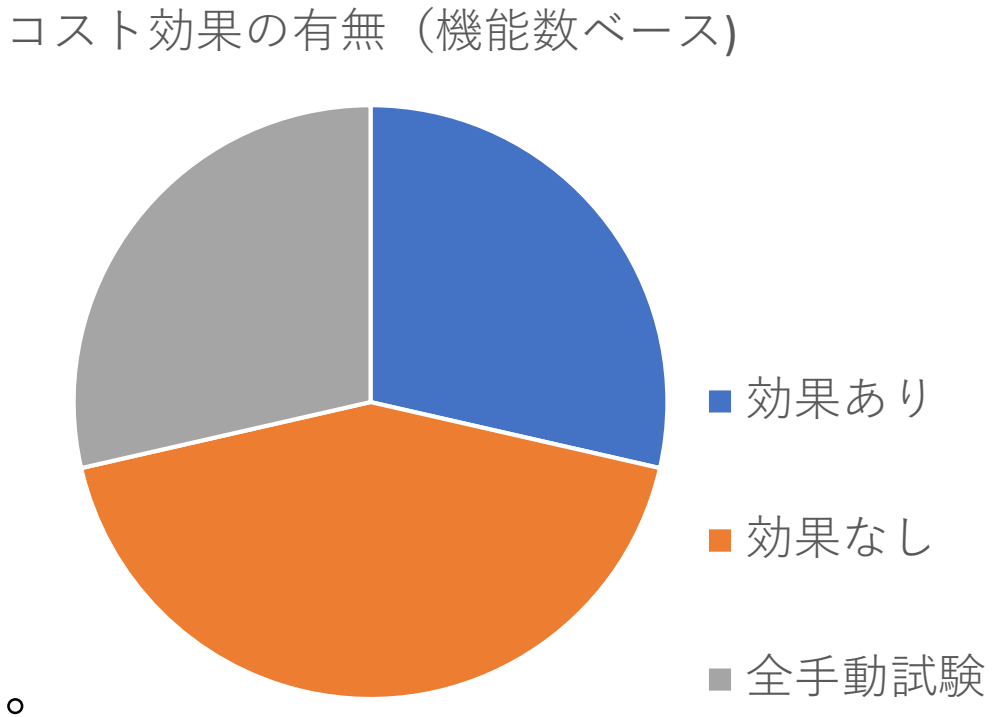
◆**今回(4周目)**、**3割位**の機能でコスト効果が出ました。

◇効果の出やすい機能の例

– 以前のプロジェクトからの**流用率が高い**

– **単機能で1万項目**を超える場合は、自動化新人でもコスト効果あり

– **経験者**が担当する



◇**1周目**でコスト効果を出すのは**やはり難しい**です。

所感

- 自動化を全社的に合意して推進するには、**自動化に造詣が深い幹部の存在がキー**でした。
  - 体制の安定化含め社内の合意をとるのに**10か月以上かかり**苦労しました。

- 常に**最新の教育コンテンツ**が必要だと感じました。
  - 新人向けにも経験者向けにも、自動化技術を更新し続ける限りは必須です。

- もし、自動化の活動をやり直すならば**テストケースの網羅的な整理を最優先**します。
  - 自分たちの特徴や実績の把握、実現可能なマイルストーンを作成するため

継続可能な自動化を目指して

APRESIA SYSTEMS

Rev.1

JANOG45

組織の承認を得るためには、コスト問題は避けては通れなさそう。

- **1周目で成果を出すのは困難**
  - 自動テスト可能なテストケースやテスト実装（ライブラリ含む）等が必要
- 自動化のコスト効果は**回帰性**が重要
  - プロジェクト**内/間**での回帰試験
  - (現状)プロジェクト間で、なるべく**共通な構成**を組む
- テストケース/テスト実装が流用できる**テスト設計**も重要
- (**問題なく**)**省略可能な工程**を増やしていく仕掛けが必要
  - 例えば、テスト設計からFeatureファイルの自動生成
- **参考程度**として効果が出始める項目は**2,000項目近辺**

自動化推進者は必要

- **関係者**が**安心して**テスト自動化を進められる環境づくりが必要
  - ただ、テスト自動化が当たり前の時代になれば、推進者は不要になると信じて・・・

定量的な振り返りは有効で、そのためにはデータ収集が重要

◆**定量的な振り返り**をすることで**効果**が測定できるようになる

◇例

– 1周目：**自動化率0%⇒Try実行時**に自動化**一部**実施

– 4周目：**コスト/自動化率**の両面で**最強**！

◇どのユースケースに対応するか？を**実装レベル**で**判断可能**！

◆定量的な振り返りには**データ収集**が重要

◇**特に時間データは無いよりはあったほうがマシ**位の気持ちの方が楽

◇時間記録については、**Redmine**の**作業時間**も利用可

自動化推進者からみたPJレベルの継続可能な自動化を目指すとは・・・

以下の状態に近づいていくことでは？と考えます。

：プロジェクト全体としてコストに向き合いつつも、

：担当者が安心して、言いたいことを気兼ねなく言えて、
(心理的安全性)

：定量的な振り返りにより、テスト実装量を徐々に減らし、

：(目標と手法に)迷わず自動化に取り組める。

御礼とお願い

- 登壇の機会を頂きありがとうございます。
  - 自動化を始めて4年間の整理ができました。

- 整理して強く思ったことは、
  - 道半ばですが、**チームの一人一人の力**があったからこそ、ここまで来ることができました。
  - 少しお時間をいただければ、メンバーの写真を最後に写させていただけると幸いです。

開発担当者の写真(1/2)

開発担当者の写真(2/2)

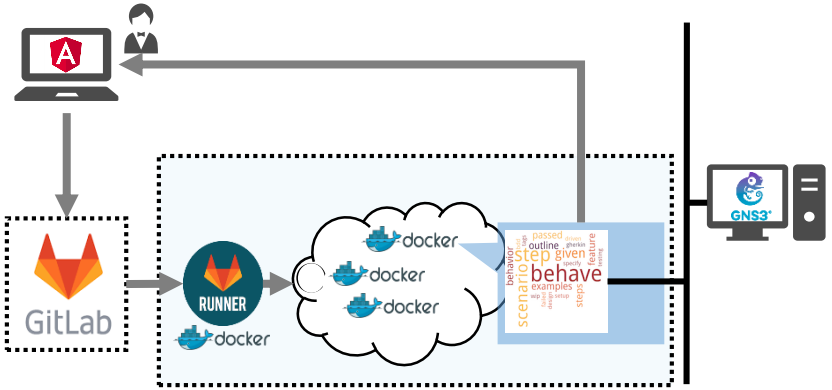
Thank you !

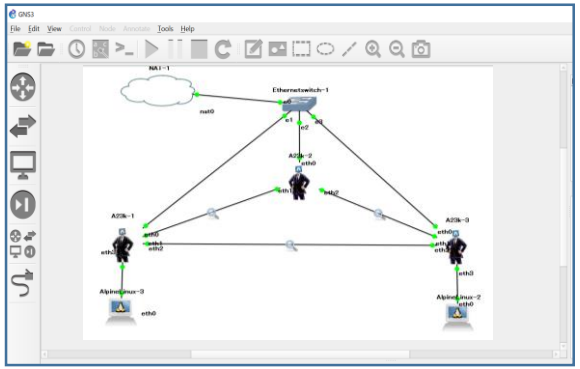
以後、補足資料

APRESIA SYSTEMS

今後の課題

- テスト工程の効率化
  - テストケース作成/テスト実装
    - テストケースならびに装置構成ファイルの自動生成
  - テスト実行
    - Docker＋GitLab CIやAWXによるCI環境
    - L1SWやメディア挿抜装置の活用による回帰試験効率化
  - その他
    - エミュレーション環境の構築
    - 統合開発環境の作成





- 機械学習の利用による効率化・分析
  - データを継続的に取得することによる各種分析
  - 画像処理を利用したLED監視テスト自動化

- 導入支援
  - 新規のテスト自動担当者向けマニュアル・サンプルの充実
  - プログラミング教育

カスタムマッチャの作成のポイント

◆PyHamcrestの**contains**の実際の動作は完全一致(contains_exactly:Java)

○ソースコード
```
assert_that([1, 2, 3], contains(1, 2))
```

○実行結果
```
AssertionError:
Expected: a sequence containing [<1>, <2>]
     but: Not matched: <3>
```

◆PyHamcrestのマッチャーのdefault APIについて

◇「**要素の順序は確認**」しつつ、「**対象要素間の個数は任意**」というAPIはない

– AがDより前に出現

– ただし、AとDの間に要素数は任意(0以上)

◆**PyHamcrestを拡張**（カスタムマッチャー）を作成し、syslog/trapに適用

◇例：状態遷移ログを定義しておいて、それが順序通りに出力されているかを確認

ライブラリ・ツール拡張時の全体像

テストケース設計 to テストケース/装置構成ファイル

テスト項目書 兼 テストケース自動生成ツール

将来対応

featureファイル

装置構成ファイル

自動生成されたテストケース (featureファイル)

APS templates

stepファイル

environment.py

共通stepファイル

ライブラリ 2.0 (最新)

Traffic generator Arbistration Interface(TAI)

Generator

Sender

Receiver (Analyser)

SSH

syslog v2

trap v2

PyHamcrest Ex

TextFSM Ex

HAL (測定器)

CLI

shslog/ trap

SNMP

独自クラス

logging Ex

telnet Ex

外部クラス

logging

telnet lib

pysnmp

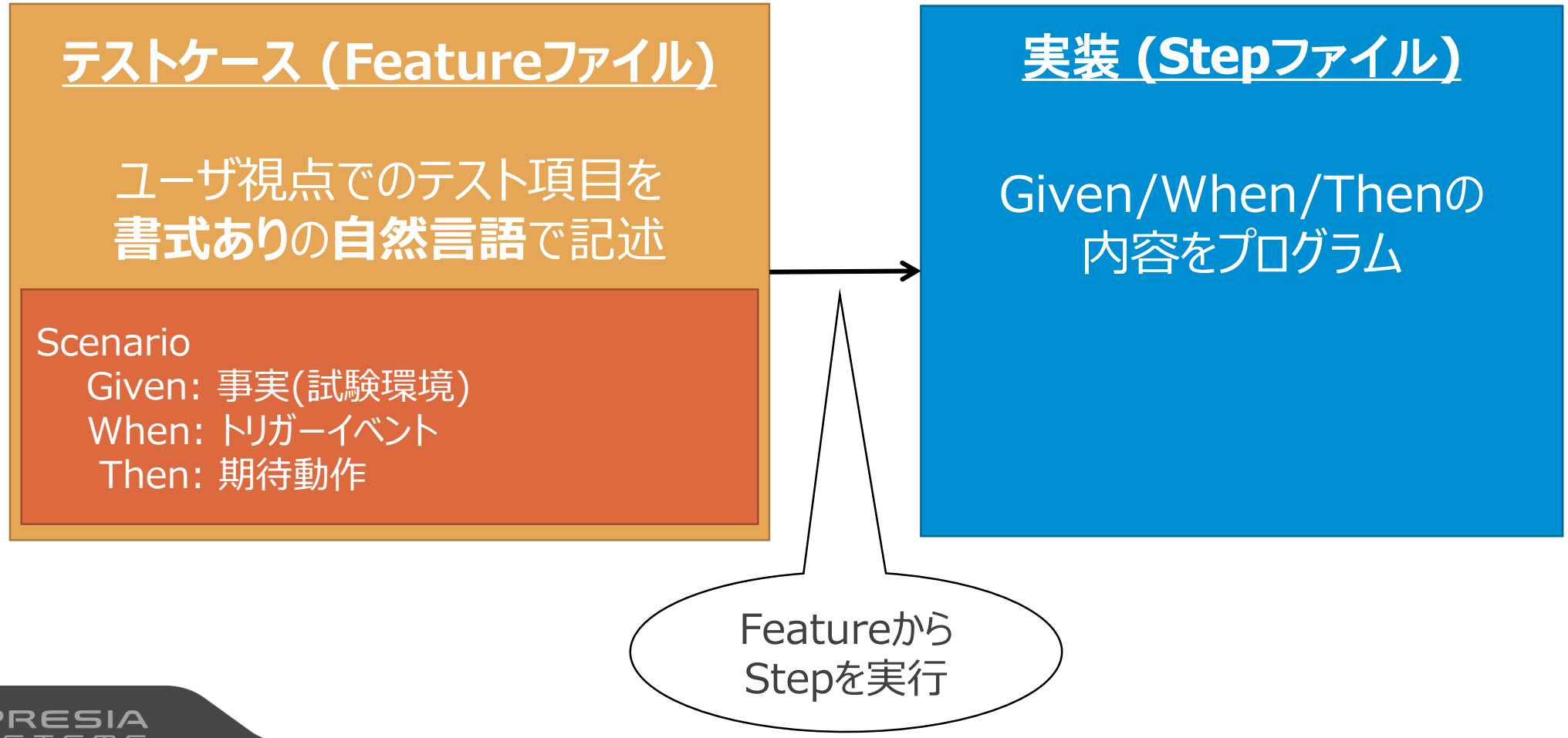
PyHamcrest

TextFSM

ライブラリ 1.0 (2016年当時)

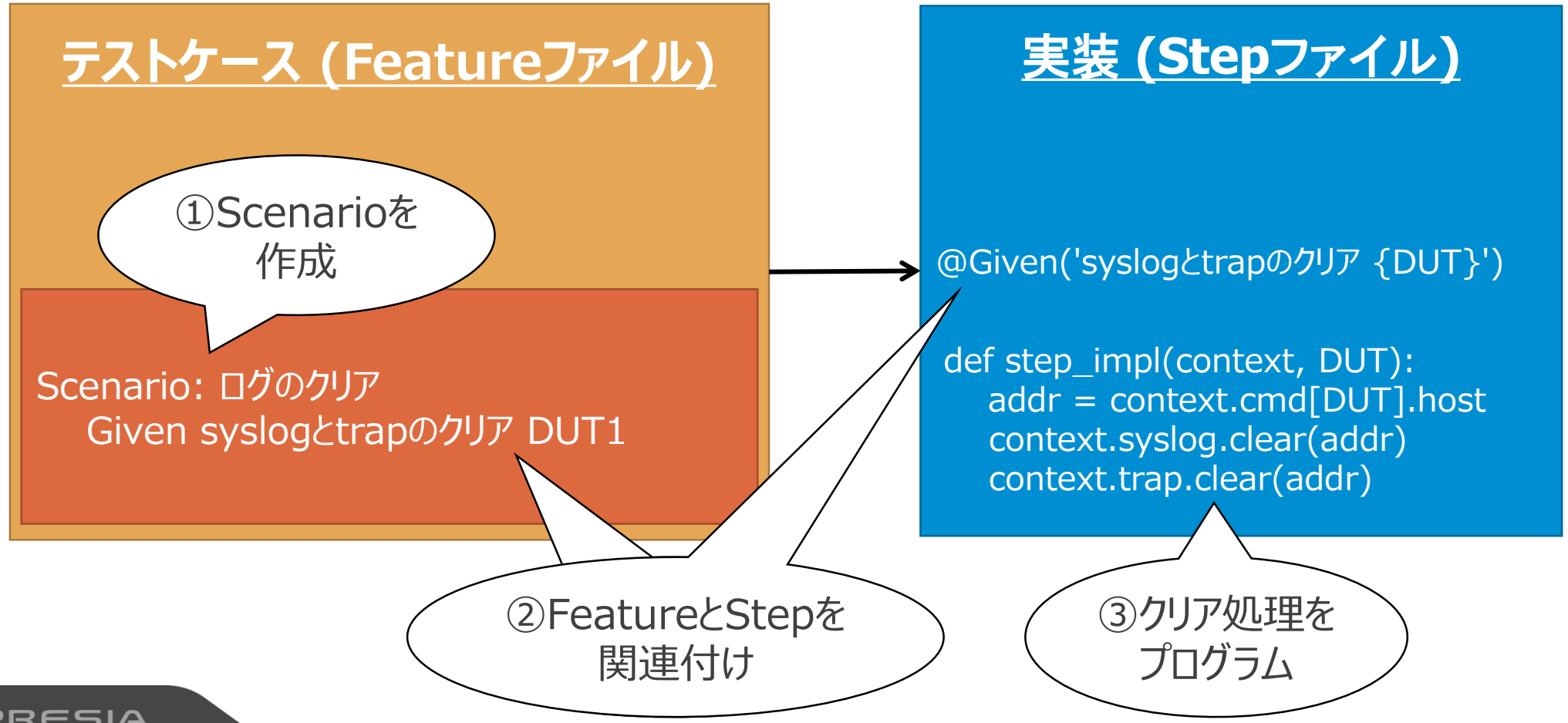
BDD (Behavior Driven Development) のイメージ

- **テストケース**と具体的な**実装**を分けるための仕組み
  - Webの受け入れテストで、よく使われるフレームワーク



BDD (Behavior Driven Development) のイメージ

◆例：syslogとtrapのクリア



TextFSMのテンプレートとは

- テキストをパースする定義で、下記の2セクションからなる
  - ntc-templates （GitHub 上の別リポジト）に Cisco, Arista 等のテンプレートが公開

テンプレート

変数定義：抽出するデータ定義

VALUE ＜変数名＞ ＜マッチパターン＞

状態/アクション定義：データ解析するための1つ以上の状態

Start　　※Start 状態から開始、後続の独自状態定義可
<Rule> -> <Action>

Actionについて
Next　：行にマッチした場合でも、次の行を読む
Record：マッチしたら一旦記録し、最初に戻る
同じパターンの繰り返し出力時の最後に使用
※<Action>省略時はNext(Default)