自動テストを活躍させるための テストの基礎作りと テスト設計の工夫

2023/2/10 JaSST'23 Tokai 特別講演

井芹　洋輝

自己紹介

- トヨタ自動車　デジタルコクピットシステム開発室
  - テストチームのテックリードやプロジェクトの品質保証・テストの戦略立て・コントロールを担当
- JSTQB技術委員、テスト設計コンテストU-30クラス初代審査委員長
- 「システムテスト自動化標準ガイド」「Androidアプリテスト技法」「テスト自動化の成功を支えるチームと仕組み」など

- 開発者、テストエンジニア、コンサルタントなど様々な立場でテスト自動化の導入・改善を手掛けてきました
- 今回は教条的な解説をベースに、口頭で自身の経験を織り交ぜながら解説を進めます

この講演について

- 自動テストを活躍させるための経験則の要点を解説します
- アウトライン
  - 【全体像】自動テストを活躍させるための基礎作り
  - 【要点の一つの深堀】自動テストを活躍させるためのテスト設計

- 対象とするテスト自動化：
  - ドメイン：すべて
    - ウェブ、組み込み、アプリといった特定のドメインに依存しない知見を解説します
  - テストレベル・テストタイプ：すべて
    - ユニットテスト〜システムテストを含む、全般的な自動化が対象です

自動テストを活躍させるための基礎作り

自動テストを支える基礎作りの重要性

- テスト自動化はチームの継続的な改善活動
  - 継続的に恩恵を得て、費用対効果をプラスにする
  - 継続的にテスト自動化システム、テスタビリティを作りこんで自動テストを実現する
  - 継続的にステップバイステップで自動テストを拡大・進化させる

→継続性・持続性を支えるための自動テストの基礎作りが重要

自動テストを支える基礎

自動テストの継続的な成功

自動テストの開発の要点

適切な 目的設定	ニーズ・シーズ への気づき	自動テストの 内部品質確保	自動テストの 妥当性確保	テスタビリティ 確保
適時の 環境確保	テストシステム の整備	計画と段取り の工夫	監視とコント ロールの工夫	自動テスト の責務拡大

自動テストを支える基礎作りの要点

プロセスの整備	チームの 構築	自動化インフラ の整備
テストの 基本設計の工夫	良い習慣の 定着	テスト自動化文化 の醸成

自動テストを支える基礎

自動テストの継続的な成功

自動テストの開発の要点

適切な目的設定	ニーズ・シーズへの気づき	自動テストの内部品質確保	自動テストの妥当性確保	テスタビリティ確保
適時の環境確保	テストシステムの整備	計画と段取りの工夫	監視とコントロールの工夫	自動テストの責務拡大

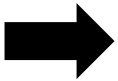
自動テストを支える基礎作りの要点

プロセスの整備	チームの構築	自動化インフラの整備
テストの基本設計の工夫	良い習慣の定着	テスト自動化文化の醸成

【適切な目的設計】
チームの成功につながる目的を設定する

- テストのニーズ・シーズ・制約、チームの状況に応じて：
  テスト自動化のリスクや費用対効果が変わる＆妥当な目的が変わる
- その時のチームとって有益で実現可能な目的を設定するのが重要

例：
【対象】スマフォアプリを使ったサーバクライアントシステム開発
【ニーズ】サーバサービスのリグレッションを検出したい
【制約】UIは未確定で変更が多い
【チーム保有の技術】サーバの実行環境は仮想化対応見込み



【避けるべき自動化】
UIに依存するEnd to Endのふるまいの品質保証のコスト削減

【有効で成功が見込める自動化】
サーバAPIのリグレッションの継続的検出

経験的に多いテスト自動化失敗要因：不適切な目的設定

【適切な目的設計】高い視座で目的を設定する

- 自動テストはチームの継続的な改善活動
  本質的な目的は、テストに閉じた視座でなく、より高い視座（チーム全体・チームライフサイクル全体）から見える

- 例：自動化の目的をチームのライフサイクルで考える
  - テスト自動化施策の費用対効果がマイナスでも、施策の継続・展開でチーム活動でのLTVがプラスにできるならば失敗とは言えない
  - ロックインされるマイナーな高級ツール導入で、案件単発で費用対効果をプラスにできても、中長期に弊害が大きくなるなら失敗である

【適切な目的設計】高い視座で目的を設定する

- より高い視座で自動テストの目的を設定すると：
  - より本質的・効果的な目的を選択できる
  - 目的達成に使える手段や関係者が増える

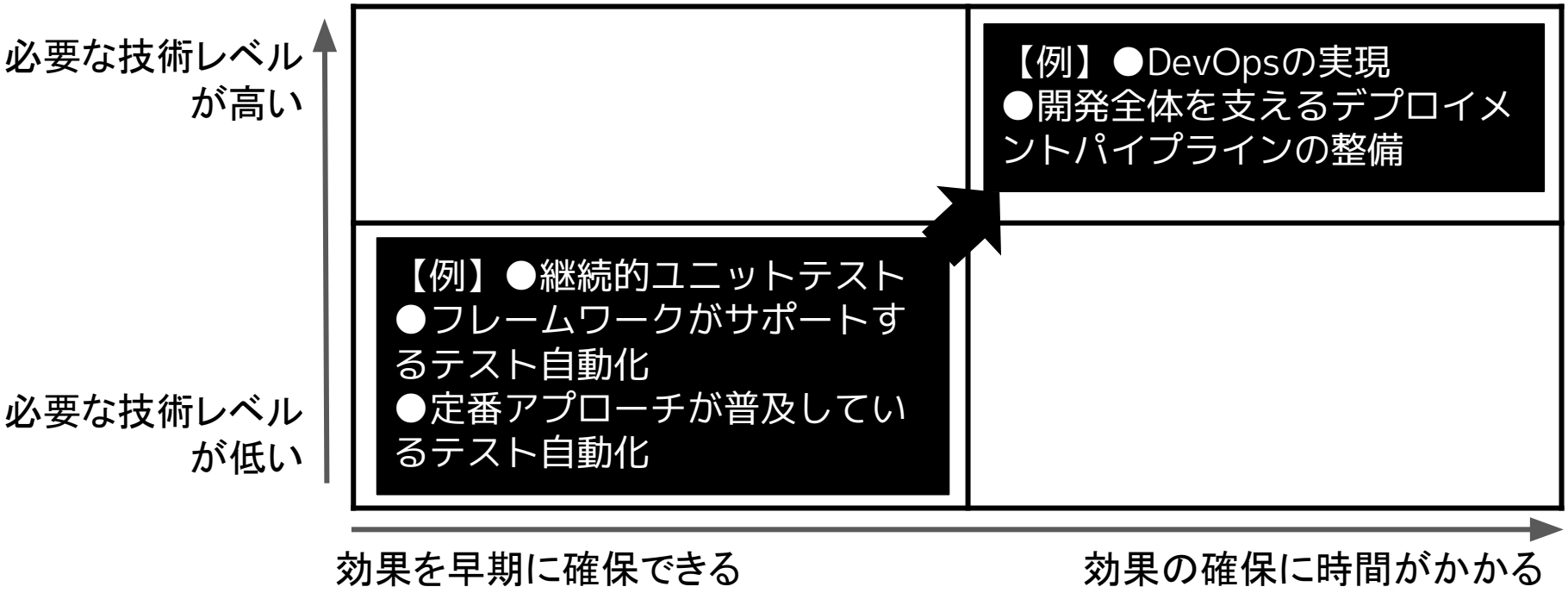
テスト担当に限定した視座
【取りえる目的の例】特定のテスト実行工程の費用対効果の改善
【取りえる手段の例】テスト担当の作業

開発＋テストに広げた視座
【取りえる目的の例】開発や不具合修正の迅速化、Developer eXperienceの向上
【取りえる手段の例】テスタビリティの向上、テスト・開発の連携協力

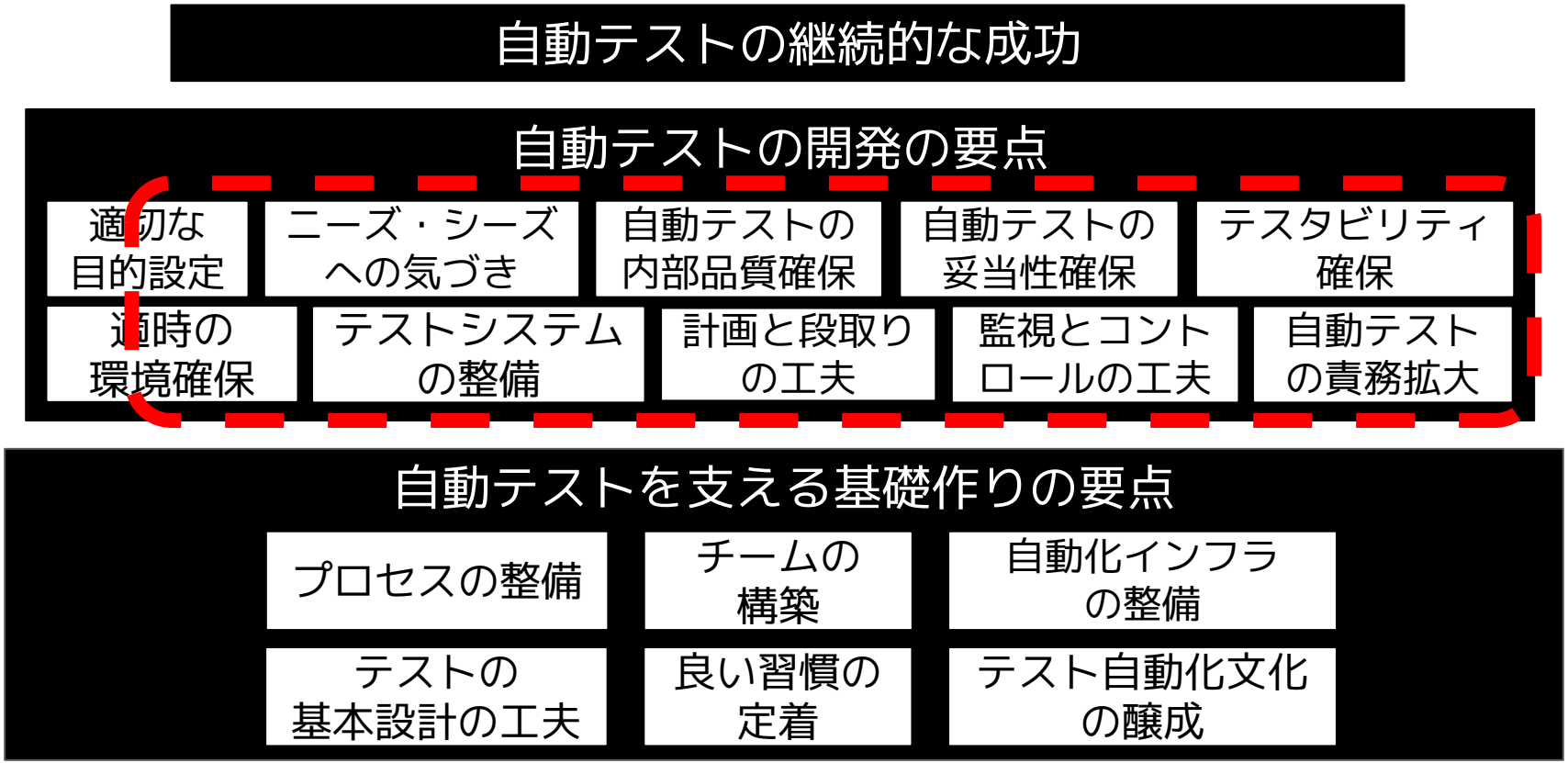
ビジネスを巻き込んだ視座
【取りえる目的の例】ビジネス要求への迅速な追従、高度な製品価値の実現
【取りえる手段の例】ユーザとの連携、プロダクトライフサイクルを通した継続的改善

【適切な目的設計】継続的な視点で目的を設定する

- 自動テストの経験・技術・改善効果を蓄積すると、より難しい目的に対応できるようになる。継続的活動として目的設定するのが有効

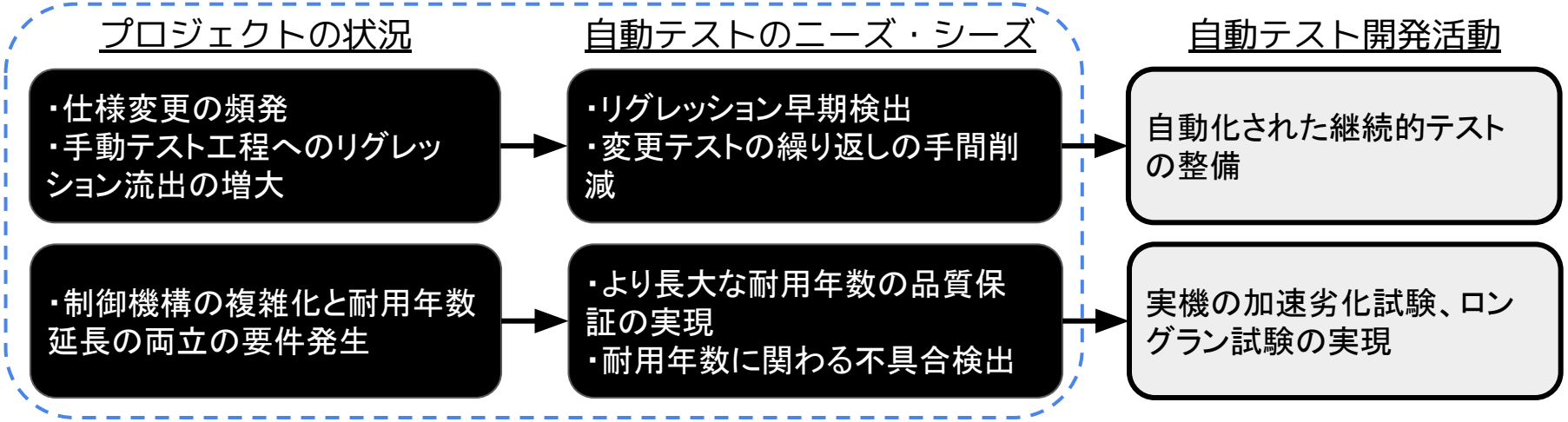


自動テストを支える基礎



ニーズ・シーズに気づくためのテスト技術力を高める

- プロジェクトでは自動テストのニーズ・シーズが逐次発生する。それを継続的を推測・識別し対応可能にするのが自動テスト活用拡大に有効



→重要なのがニーズ・シーズに気づくためのテスト自動化の技術や知見

経験的に多いテスト自動化失敗要因：能力不足によるニーズ・シーズ理解不足

自動テストの内部品質を作りこむ

- 自動テストもソフトウェア開発成果物。ソフトウェア開発のエンジニアリングやノウハウを活用する。自動テストの要点となる品質：

品質特性	確保すべき特性
保守性	自動テストの保守を容易にする ※監視容易性（問題の監視が容易）、試験性（正しいか・妥当かの確認が容易）、解析性・修正性（問題発生時の特定・是正が容易）、可読性（TaDとして運用可能）
信頼性	自動テストの偽陽性/偽陰性の削減、安定性の確保、冪等性の確保 ※継続的自動テストはFragile Test/Flaky Testとの戦いに直面する
機能適合性	必要な機能の提供、精度の確保
性能効率性	フィードバック迅速。並行実行や変更時実行が可能なように必要リソースを削減
拡張性	少ない対価で自動テストのカバレッジを拡張可能にする 例：パラメータ化テスト、データ駆動テスト、環境仮想化、モデル駆動テスト
使用性	チームにとっての使いやすさや魅力を向上する 例：フィードバックの手軽さ、結果の見やすさ

経験的に多いテスト自動化失敗要因：テストの保守性・信頼性不足による保守コスト悪化

自動テストの妥当性の確保：
自動化する価値のあるテストを設計し、価値を維持する

- **価値のある妥当**なテストケースを実現するための要点：

【テスト設計の保守】
テストの要求や制約が変化する開発ライフサイクルの中で、自動テストが有効であり続けるようにテスト設計を保守する

【テスト設計】
適切なテスト設計アプローチで、責務を効率的に充足する自動テストのテストケースを設計する

【テストレベル設計/テスト基本設計/テストアーキテクチャ設計】
自動テストが価値を発揮できるように、テスト戦略を構築し、テスト全体の責務設計を工夫する

【テスト要求分析】
広くステークホルダ（ユーザ、チーム、PjM、PdMなど）と協力し、テストについてのニーズ・シーズ・制約を広く掘り出し、実現すべき要求を識別する

経験的に多い失敗要因：テスト分析・設計の欠落による自動化効果の目減り

【テスタビリティ確保】
テスト対象の自動化しやすさを作りこむ

品質特性	テスト自動化にとっての特性	実装例
実行円滑性	円滑に自動化された処理を実行できる	シンプルなセットアップ手順
観測容易性	自動テストがテスト対象の出力（間接出力、副作用含む）を観測しやすい	エラー認識手段の充実、適切なログ設計、十分な副作用対策(ステートレスなど)、観測用デバッグ機能
制御容易性	自動テストがテスト対象の入力（間接入力含む）を操作しやすい	使いやすいAPI、間接入力の制御手段（Stubなど）、デバッガサポート、環境仮想化
分解容易性	対象から自動テスト可能なテスト対象を切り出しやすい。自動テストの障害を分離しやすい	接合部（DI、Dependency lookup、Link Seam）、適切な結合性設計、モジューラアーキテクチャ
単純性	テスト対象の仕様や構造、実行方法が単純	冗長性の少ない仕様
安定性	自動テストの支障となるテスト対象のバグや不安定さ（変更が頻繁など）が少ない	バグの少ないコンポーネント、変更の少ない仕様IF
理解容易性	テスト対象の仕様や構造、実行方法が分かりやすい	理解しやすい仕様やコード、適切なドキュメンテーション

※品質特性定義：James Bach

経験的に多いテスト自動化失敗要因：テスタビリティ不足

適時のテスト環境確保をマネジメントする

- 適時の自動テスト環境構築のための段取りとコントロールを実施
  - テスト環境確保はEDUF（Enough Design Up Front）
    - 最大責任時点（Most Responsible Moment。対象活動が最も責務を果たせるタイミング）で各活動を実施
    - ≒初期から詳細に環境分析し、スケジュールを組み、調達・構築・検証を進める必要がある
- より高い視座でテスト環境制約に対応するのが有効
  - 例）実機自動化の環境確保の制約が多い
    →実機自動化に拘泥せず、試作環境、ユニットテスト、エミュレータ・シミュレータなど他のテスト自動化環境も分析し、全体のテストの責務を工夫して環境制約を緩和する

経験的に多いテスト自動化失敗要因：環境構築の甘さで自動テストの活躍所縮小

テスト自動化の計画と段取りを工夫する

- 見積もりと戦略策定・計画づくりで、テスト自動化の適切な段取りを構築する。方向づけ・コントロールで段取りの実行を推進する
  - 自動テスト開発の計画とマネジメント
  - リソース（人・物・金・時間）確保の段取りとコントロール
    - なるべく早く、適時に自動テスト環境を揃える
  - リリースの段取りとコントロール
    - デプロイメントパイプラインを構築する
    - 自動テストの投入タイミングを確保する
  - テスト自動化のリスクのコントロール
    - プロトタイピングや反復開発による早期の学習機会の確保
  - ステークホルダの連携の段取り

経験的に多いテスト自動化失敗要因：環境確保の段取りの悪さ

運用と保守の工夫で自動テストを維持・改善する

- 自動テストの品質や成果を監視し、問題検出と是正を迅速化して、効果を確保する
  自動テストを保守し、自動テストの効果を維持・改善するように務める
- 問題検出手段
  - 事前の問題検出
    - プロトタイピング/トライアル、反復開発
    - レビュー/シナリオウォークスルー
      - バグをどのように検出するか/仕様項目をどう保証するか
  - テスト構築時の問題検出
    - レッド→グリーンのテスト駆動サイクル
    - バグの注入（フォールトインジェクション）
  - 運用中の監視
    - 偽陽性の継続的監視
    - テストカバレッジの評価
    - 欠陥流出の評価
  - 事後の評価
    - 目的達成指標ごとの評価
      - 例：欠陥流出率/カバレッジ/費用対効果の事後評価/ステークホルダの主観評価

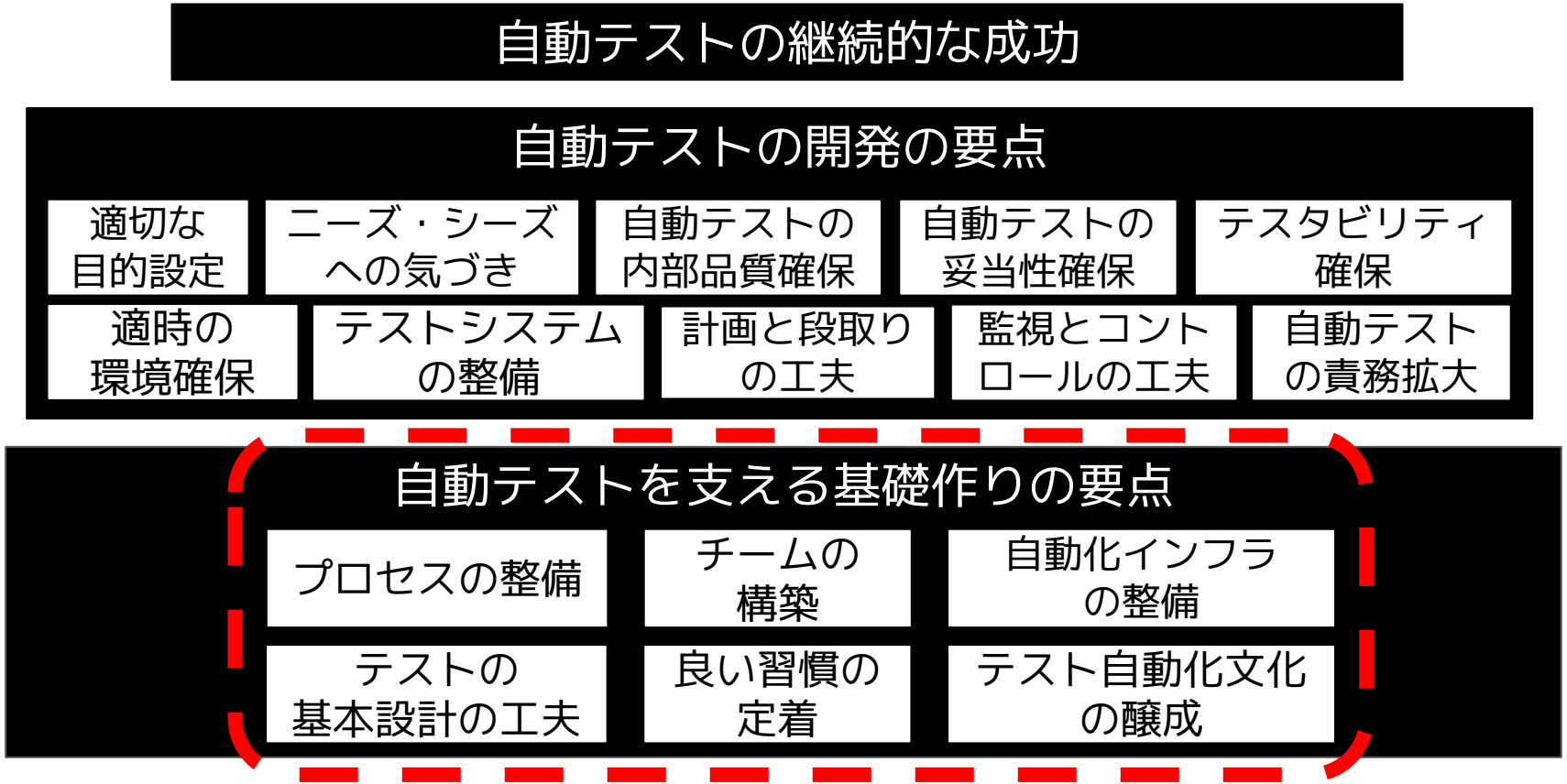
統合的に自動化を支えるテストシステムを整備する

- 自動テストの開発をサポート
  - 自動テストのアーキテクチャやフレームワークを指定・提供
  - テストスクリプト開発を支援
    - pip, npmなどの既存のエコシステム利用の実現
    - Gherkinなどの実装形式のサポート
    - データ駆動テスト、パラメタライズドテストといった構造化支援
    - xUnitやSpec系といったテストスクリプトの基本構造の指定
- 自動テストの運用・保守をサポート
  - パッケージ管理、スクリプト管理などの管理機能の提供
  - 監視、レポート、その他運用に必要なテストユーティリティの提供
  - テストウェアやテストデータの効率的な管理手段の提供

プロジェクトに対する責務を確立・拡大する

- 自動テストの資産は様々な関係者が様々な用途で活用できる
  - 例：自動化に成功した結合テストの用途
    - for 開発者
      - ダブルループのテスト駆動でプログラミングを主導
      - CIに組み込み、開発中のリグレッション発生を監視
      - PR受け入れやパイプライン通過の受入基準に利用
    - for システムテストのテスト担当者
      - スモークテストや、自身のテストの補完手段として利用
    - for 後続プロジェクト
      - Test as Documentationとして保守に利用
- テスト自動化の活用者を広げ、様々な場面で自動テストを活用することで、テスト自動化の費用対**効果**にレバレッジをかけることができる

テスト自動化の基礎


自動テストの継続的な成功
自動テストの開発の要点
適切な
目的設定
ニーズ・シーズ
への気づき
自動テストの
内部品質確保
自動テストの
妥当性確保
テスタビリティ
確保
適時の
環境確保
テストシステム
の整備
計画と段取り
の工夫
監視とコント
ロールの工夫
自動テスト
の責務拡大
自動テストを支える基礎作りの要点
プロセスの整備
チームの
構築
自動化インフラ
の整備
テストの
基本設計の工夫
良い習慣の
定着
テスト自動化文化
の醸成

テスト自動化の能力を持つ人とチームを確保する

- テスト自動化の能力を持つ人材を確保
  - テスト自動化人材のスキルモデルやスキル評価手段の確立
    テスト自動化能力を持つ人材の獲得
    - 主要スキル：
      開発技術、開発インフラ技術、テスト技術、テスト自動化技術
- チームを構築
  - 要となるロールの確保
    - テストアーキテクト/テストアナリスト
      - 品質保証・テスト全体を視座に、あるべきテスト自動化へ先導
    - SET/SETI
      - 開発技術を活かして、テスト自動化およびテスト自動化インフラに注力
  - 必要なコラボレーション手段（例：開発者とテスト担当者）の仕組み確保
- チームの能力の維持・向上の仕組み化
  - プロトタイピングといった学習サイクル確保

経験的に多いテスト自動化失敗要因：開発力不足

自動テストの成功を支えるチーム文化を醸成する

1. 自動テストの価値を理解し、健全な動機付けが行われている
   - 価値の本質を理解し、それに基づいたテスト自動化活動の動機付けを行う
   - 自発的なテスト自動化活動への貢献が行われている
2. テスト自動化の持続可能性を支えている
   - 習慣的に自動テストの技術的負債を返済し、持続可能な状態を保つ
3. Whole Teamで協力しながら自動テストを支えている
   - テストエンジニア、開発者、ドメインエキスパート、マネージャ、様々な立場の人たちがテスト自動化に協力しあう
4. 自動テストのためのミッション・ビジョンが共有・推進されている
   - 本質的な自動テストの価値に基づいてチームが方向づけされている
5. 自動テストのために学習・成長し続けている
   - 継続的に自動テストの技術や能力を伸ばし続けている
6. 自動テストを信頼し責務を任せている

経験的に多いテスト自動化失敗要因：文化とリテラシーの欠落

テスト自動化の良い習慣を定着させる

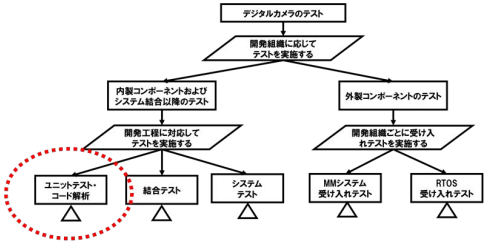
- 習慣で支える自動テストは多い（例：ユニットテスト）
- 作業・チェックの自動化、作業のルール化、クライテリア運用を通して、テスト自動化を支えるグッドプラクティスをチームの習慣として定着させる
- 自動テストコードの良い習慣の例：
  - FIRSTの原則
    - Fast、Independent、Repeatable、Self-Validating、Timely
  - xUTP12の原則
  - 「Googleのソフトウェアエンジニアリング」ユニットテストの原則
    - 変化しないテストを目指せ。
    - 公開API経由でテストせよ。
    - 相互作用ではなく、状態をテストせよ。
    - テストを完全かつ簡潔にせよ。
    - メソッドではなく、挙動をテストせよ。
    - 挙動に重点を置いてテストを構成せよ。
    - テスト対象の挙動にちなんでテストに命名せよ。
    - テストにロジックを入れるな。
    - 明確な失敗メッセージを書け。
    - テスト用コードを共有する場合、DRYよりDAMPに従え。

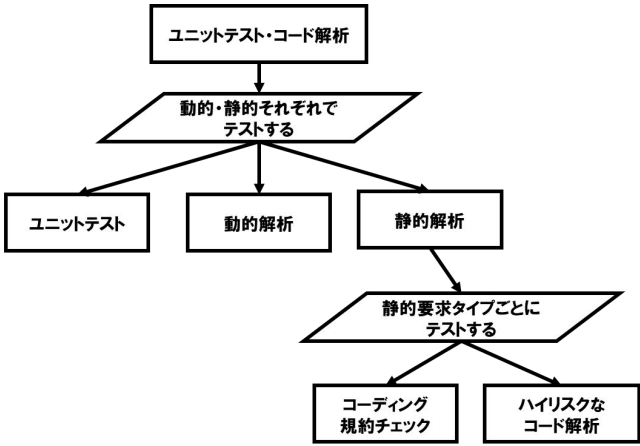
「Googleのソフトウェアエンジニアリング」（オライリー、Titus Winters他）

チームの品質保証・テストの基本設計を工夫する

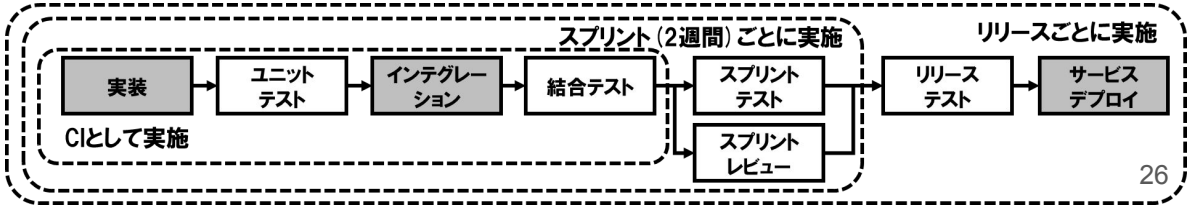
- 全体の品質保証・テストの設計構造を構築・維持し、開発ライフサイクルの中で自動テストのあるべき責務を先導する
  - ある仕様項目の実現はだれが確認するか、あるバグの検出はだれがやるか

品質保証・テスト活動の責務設計





品質保証・テスト活動のプロセス設計



自動化インフラを整備して自動テストを統合する

- 開発を支える自動化インフラに自動テストを組み込む。
  自動化インフラで自動テストの保守や改善効果を底上げする
  - デプロイメントパイプラインへの統合による開発のインテグリティの確保
  - 自動テストの構築・維持に有用なインフラ機能の活用
    - 自動テストのテストウェアの管理
      - テストシステムやテストコードの構成管理
    - 自動テストの品質保証
      - テストコードの動的解析・静的解析、テストシステムのテスト
    - 自動テストの実行
      - 開発イベントに合わせた、効率的な自動テストの実行管理
        - 変更ごとの実行/並列実行
    - 自動テストの監視と問題是正のサポート
      - カバレッジ計測、偽陽性・偽陰性の監視、Flaky Testの監視

テスト自動化のプロセスを整備する

- テスト自動化の良いアプローチや要点をプロセス化してチームに定着させる
  - テスト自動化の能力を伸ばす学習サイクルの確保
    リスクをコントロールするためのフィードバックサイクルの確保
    - プロトタイピングや反復開発
  - 適切なテスト自動化アプローチのプロセス化
    - テストフェーズに関わる連携
    - 適時・適切なツール選定など、必要作業のプロセス化
    - Whole Teamアプローチの活動のプロセス化
      - Wモデルなど各工程でのテストの考慮
  - 必要なテスト自動化活動のクライテリア化

テスト自動化の成功を支える基礎

自動テストの継続的な成功

自動テストの開発の要点

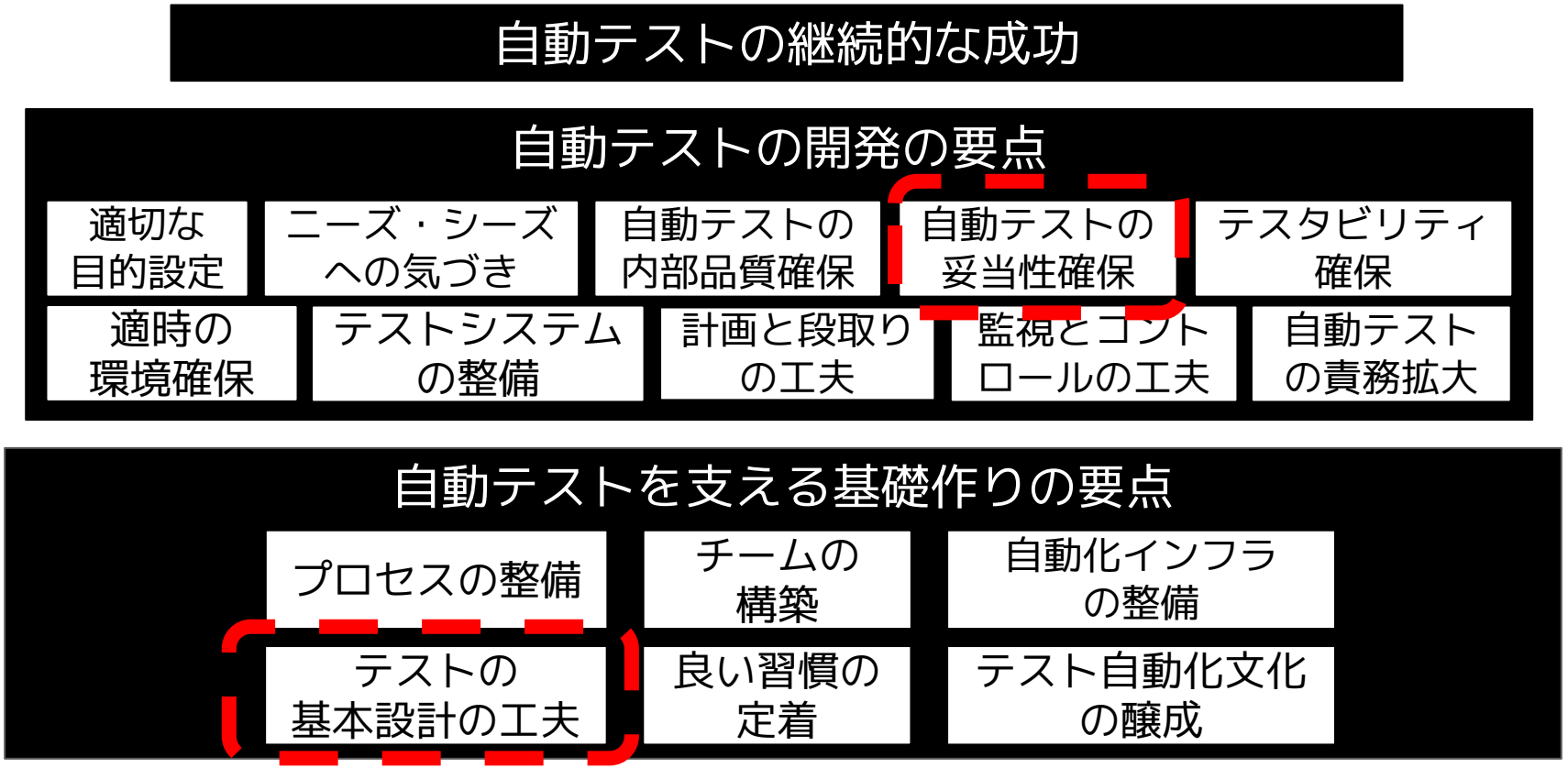
- 適切な目的設定
- ニーズ・シーズへの気づき
- 自動テストの内部品質確保
- 自動テストの妥当性確保
- テスタビリティ確保
- 適時の環境確保
- テストシステムの整備
- 計画と段取りの工夫
- 監視とコントロールの工夫
- 自動テストの責務拡大

自動テストを支える基礎作りの要点

- プロセスの整備
- チームの構築
- 自動化インフラの整備
- テストの基本設計の工夫
- 良い習慣の定着
- テスト自動化文化の醸成

自動テストを活躍させるための基礎
【テスト設計】

テスト自動化の成功を支える基礎



自動テストを活躍させるためのテスト設計の工夫

- 価値ある自動テストを実現するためには、価値あるテストの実現が必要
  →適切なテスト設計アプローチが必要

- 自動テストを活躍させるためのテスト設計のアプローチ
  - 自動テストの強みを活かす
  - 自動テストの弱みを軽減・補完する
  - 自動テストのトレードオフのバランスを取る
- アプローチの適用レベル
  - テスト基本設計（テストレベルやテストタイプレベルの設計）での工夫
  - テストケース設計での工夫

自動テストの強みとテスト基本設計での対応

- テストの有効性の改善
  - 手動で困難なテストを実現する
  - 本質的な目的達成に貢献する
- テストの効率性の改善
  - 時間やリソースを削減する
  - 同じリソースでできる事を広げる
- テストの正確性と信頼性の改善
  - 手動のミスを削減する
  - 同じことを何度も繰り返せるようにする
  - 誤差や不安定さに対応する
- テストの保守性の改善
  - テストの品質保証、維持、修正、再利用を容易にする
  - テストのインテグリティ（一貫性）やトレーサビリティの維持を助ける
- テストの使用性の改善
  - 自動化のステークホルダに合わせたUIを実現する
- その他副次的な効果
  - 開発力の獲得、開発・テストの連携の強化、開発の清流化

●テストの基本設計で強みを活かせるテストを切り出す。その責務を増やす
例）継続的な実行が有効なテストを自動化可能な形でなるべく多く確保する

自動テストの弱みとテスト基本設計での対応

- 探索的アプローチや人間の知能を生かした柔軟な対応ができない
- 具体的に定義されたふるまいや期待結果しか確認できない
- 自動化容易性の確保できる領域しかテストできない
- 多くのリソースとコストを消費する
  - 人・技術：メンバー、スキル、組織
  - 物・環境：場所、機材、ツール、ライセンス、その他テストウェア
  - 労力・時間：
    - 自動テストの実現と運用保守
    - 自動テストを支えるための改善（テスタビリティやテストオラクル確保）
    - その他付随作業(マネジメント/複雑化への対応)

●テスト基本設計で自動テストの弱いテストを切り出し、その責務を減らす。
例）探索的なユーザストーリテスト＋自動テストの組合せで、テスタビリティの低い箇所のテストを担保する

自動テストのトレードオフとテスト基本設計によるバランス調整

- テスト自動化に関わるトレードオフ・制約は多い
- 品質についてのトレードオフの例：

品質特性	テスト自動化のポジティブな効果	テスト自動化のネガティブな効果
保守性	リグレッションの検出・解消確認の効率化 →リファクタリングやデバッグの効率化	テスト対象の変更にテストの保守・変更の手間が加わる→保守コスト増大
性能効率性	時系列での性能監視の実現 →性能悪化の早期検出・早期対策を実現	テスタビリティ実装部による性能低下

●テスト基本設計でポジティブ効果を伸ばしネガティブ効果を減らすよう、テストの責務を調整する。妥当なバランスに調整する
例）UIの変更が多い。その変更対応コストが大きい
→安定したIFに対するテストを増やす。例えばUI依存のテストを減らし、APIテストを増やすなど

テストケース設計での対応：
ツールやフレームワークの強みを伸ばし、弱みを抑える

- テストスクリプトの構造に合わせる
  - テストスイート、テストケースの構造に合わせる
- テストスクリプトの構造要素の凝集性・結合性を合わせる
- テストスクリプトのパラダイムに合わせる
- テストスクリプトの処理フローに合わせる

●主に保守性、性能効率性を改善

ツールやインフラの特性の例	テスト設計の工夫
テストクラス⊂テストメソッドのツリー構造	構造に合わせてテストケースを階層構造化する
テストフィクスチャがテストクラスごとに分離	セットアップを共通化できるようにテストケースをモジュール化する
AAAやGherkinといった記述パターンを指定	パターンに合わせてテストケースを形式化
パラメタライズドテストや構造化プログラミングをサポート	手順を共通化し、テスト条件の差異をパラメータ化する

テストケース設計での対応：
使用可能なテスト技術の強みを伸ばし、弱みを抑える

- 可変点をテストパラメータ化する
- 制御容易性・観測容易性に優れたIFに合わせる
- 安定性・単純性に優れた箇所への依存性を高める
- 蓄積した資産や既存のエコシステムを活用する

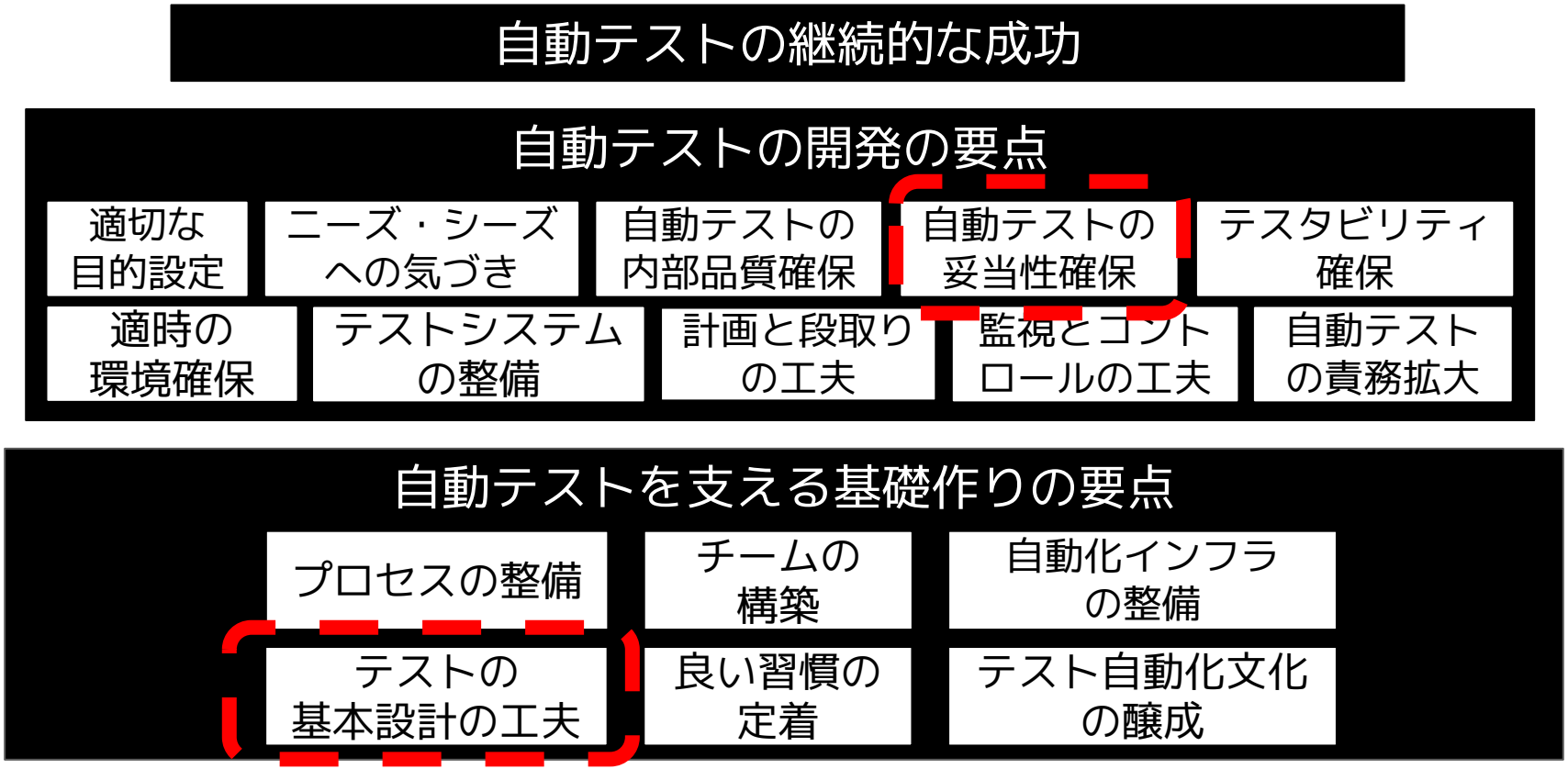
テスト技術の特性の例	テスト設計の工夫
実行環境をコンテナ仮想化	コンテナの環境条件をテストパラメータ化しデータ駆動テストとしてテスト設計
ログ設計が適切で動作ログで異常検出しやすい	ログデータの情報をテスト期待値に追加してテスト設計
膨大な過去の実環境の動作データを活用できる	テスト要求に、保有する動作データを紐づけてテスト設計する

自動テストを活躍させるためのテスト設計の要はテスト基本設計

- テスト基本設計を通して、
  自動テストの強みを発揮できるテスト責務を増やし、
  自動テストの弱みをカバーする仕組みを設計する

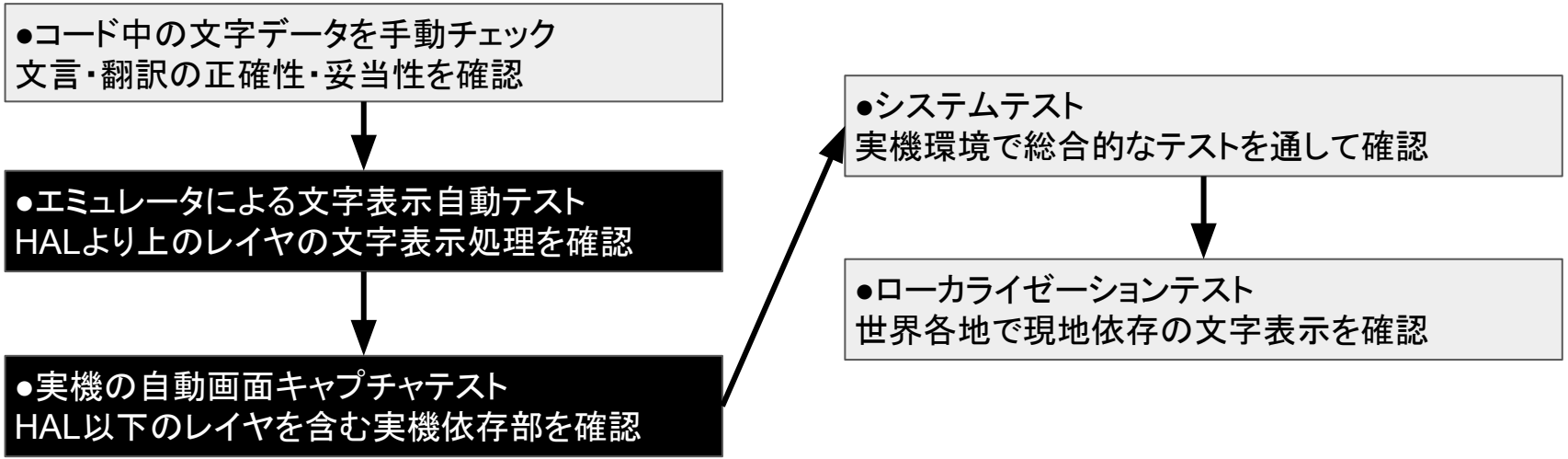
自動テストを活躍させるための基礎
【テスト基本設計】

自動テストを支える基礎



テスト基本設計でのテストの責務設計

- 全体のテストを責務分担する
  - テストで果たすべき責務に対し、保有する能力、保有見込みの能力を持ち寄り、得意な所を活かし、不得意な所を補いあう
    - 自動テストの得意なところを活かして活躍させる
- 例）グローバル展開する組み込み機器の表示文言のテスト

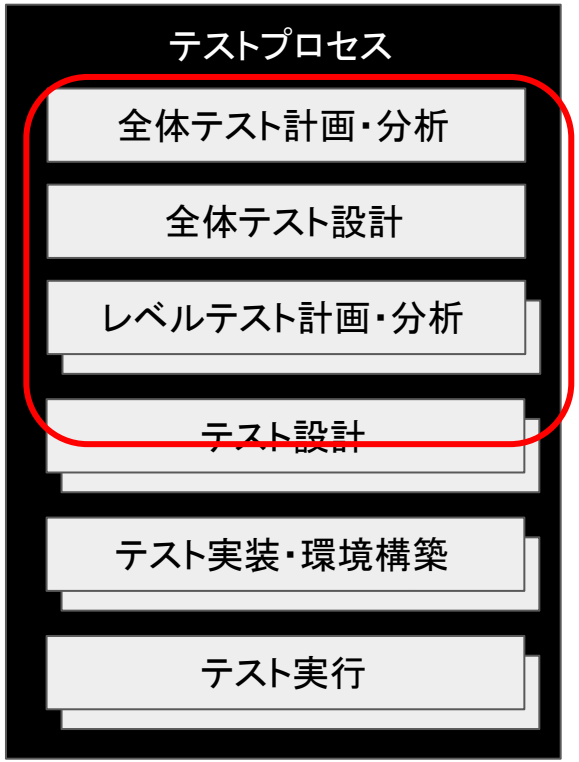


自動テストが品質保証・テストの責務を担うとは

- GoogleのBoyonceルール（Googleのソフトウェアエンジニアリング）
  - 「...インフラストラクチャーの変更の結果として障害や他の問題に陥った場合、問題が継続的インテグレーションシステムでのテストで表面化していなかったならば、インフラストラクチャーの変更に落ち度はない...」
  - ≒「特定の障害が市場で発生したら、すべてCIの自動テストの責任」

- テスト自動化は「できるところまで」がスコープになりやすい。
  しかしそれでは既存手段と責務が競合し、成果が曖昧になる
- 品質保証の責務を引き取り果たしてこそ、投資対効果の明確なプラスを確保できる

テストの責務設計の段取り：



【主要な構成活動】

- テスト要求の分析・すり合わせ
- 責務設計
  - 責務の具体化・関心の分離
  - 責務間の連携の設計
  - プロセス設計
- 評価と改善

【担い手】
チーム全体で考える。開発者やマネージャを巻き込み、高い視座で活動を進める

テストの責務設計に影響するテスト要求の分析

- テストの責務構造に影響するテストの要求・制約を分析する
- 特に基本構造・基本方針を決める重大な要求の識別に注力する
  - テストの全体構造を取り巻く要求・制約は巨大で複雑
  - 全体俯瞰の分析と重点部分の深掘り分析を組み合わせる

要求・制約の種類の例	テスト要求の分析アプローチ
結合テストのテストレベルを識別する	開発プロセス分析、テストベース分析、テスト対象のアーキテクチャ分析、インテグレーションの流れ
標準や法規制への準拠に必要なテストを明らかにする	テストに関わる法規制・標準の識別、法規制・標準の中でのテストへの要求項目の分析
テスト自動化の適用範囲を明らかにする	テスタビリティの調査、リソース計画の分析、保有するテスト自動化技術の調査
システムテストのテストタイプを分析する	テスト要求分析、システムを構成する品質特性の分析、機能構造（インターフェースや静的データなど）の分析
結合テストのテストタイプを分析する	テスト対象のアーキテクチャ調査、インテグレーションプロセスの分析、コンポーネントセットごとの品質目標とテスタビリティの評価

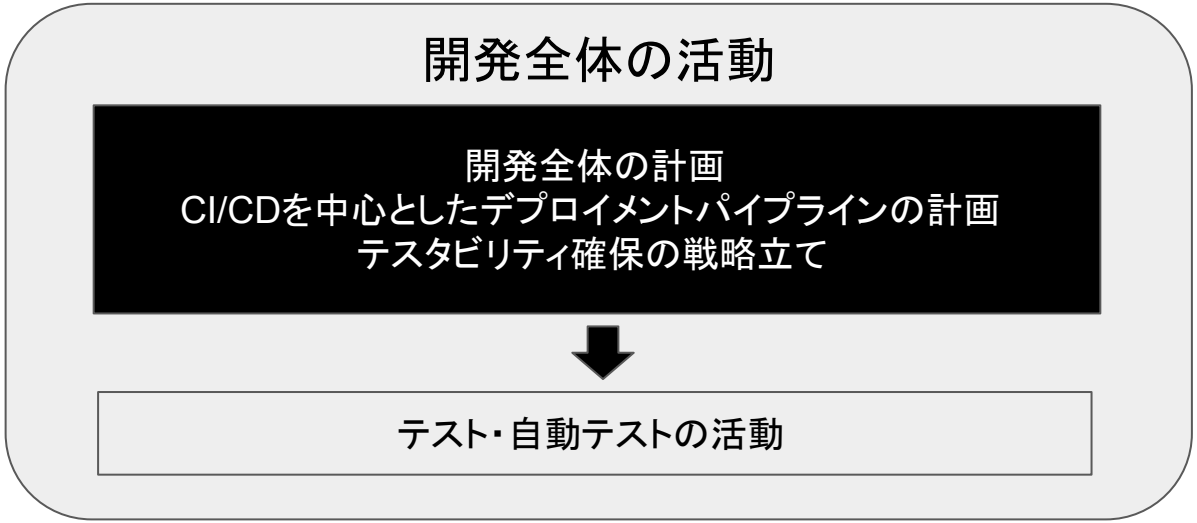
テスト要求の分析：
ステークホルダとすり合わせ要件を見定める

- ステークホルダと相互に要求・制約をすり合わせ、プロジェクトの要求・制約を満たす妥当なテスト基本設計の要件を見定める

すり合わせ相手	すり合わせ内容
ビジネス、ユーザ、PdM	ビジネス・ユーザ要求、テストのフィージビリティ
マネジメント	コスト、リソース、スケジュール、スコープ、体制
開発	テスタビリティ、設計、インテグレーション、開発プロセス、デプロイメントパイプライン、テストベース、開発技術
品質保証	テストの代替・補完手段、品質マネジメント、品質管理
テスト	テストの目的、テスト活動からのフィードバック、テスト技術

テスト要件定義のすり合わせ：
自動テストのための基礎をすり合わせる

- 自動テスト活用ではすり合わせが特に重要
- テスタビリティ確保とCI/CDは自動テストの用途とスコープに影響する。自動テストの前提としてすり合わせし整備の見通しを立てる
- 開発全体の計画・戦略立てに自動テストからの要求を反映する



テストの基本戦略の策定

- 全体のテスト責務構造は複雑なため方向づけする
- 重要な要求に着目し、テスト全体の構造や戦略を方向づけするための、戦略、基本設計方針を策定する
  - 影響の大きな自動テストについて、その成功可能性を高めるための戦略立てを初期から行う

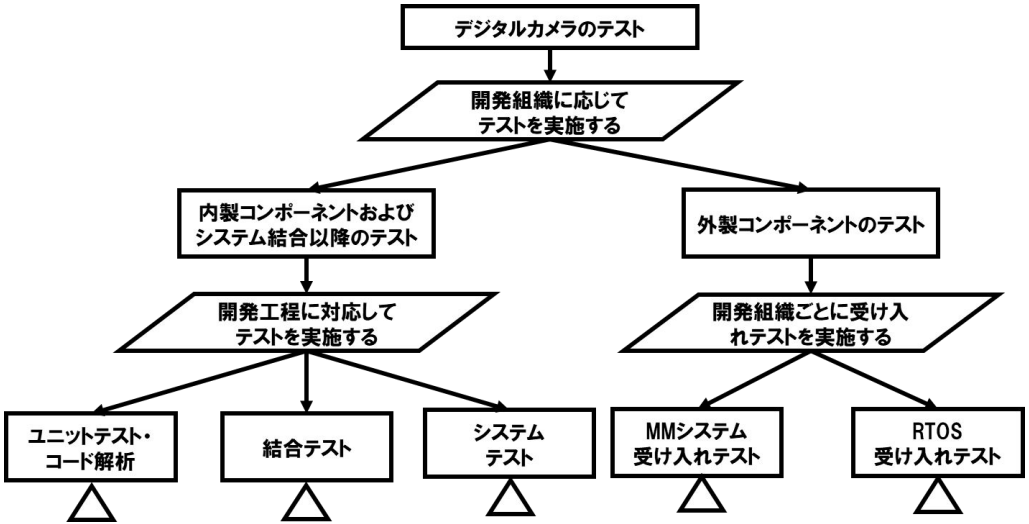
テスト要求の例	基本戦略（テスト責務の設計方針）
継続的なアップデートを通じて、サービスの顧客体験の市場競争力を維持する	各テストレベルのテスト自動化を推進し変更テストを高速化する。テスト対象のテスタビリティ要件を早期に識別・確保してテスト自動化可能な領域を拡大し、それを活用する自動テストの責務を広げる
成功した過去プロジェクトのノウハウを継承する	過去プロジェクトのテスト体制、テスト定義を継承する。

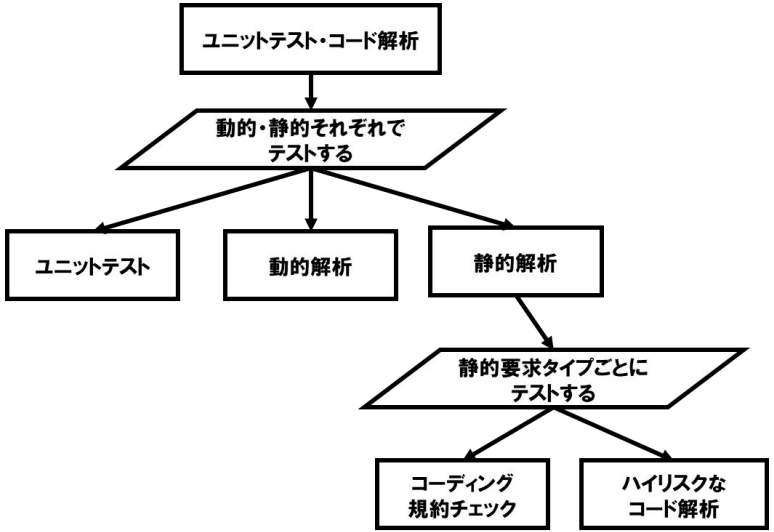
テストの責務設計：３つのアプローチで設計・管理する

- テストの責務の分割・具体化
  - 関心の分離でテストの責務を検討しやすくする
  - 全体の責務を個別対応可能な粒度まで具体化し、テスト責務の構成要素と構造を導出する
- テストの連携の検討
  - 方針、課題、リスクに対して、どのような連携をするか設計する過程で、テスト責務の構成要素の関わり合いを設計する
- プロセス設計
  - プロセスベースでテスト責務を設計し、テスト責務の構成要素と構造を設計する

テスト責務の分割・具体化、関心の分離

- 目的・要求からテストの責務を実行可能な粒度まで具体化・分割する
  - ツリーモデルや表での設計・管理が容易





自動テスト活用における責務具体化・関心の分離：テストレベル設計で検討

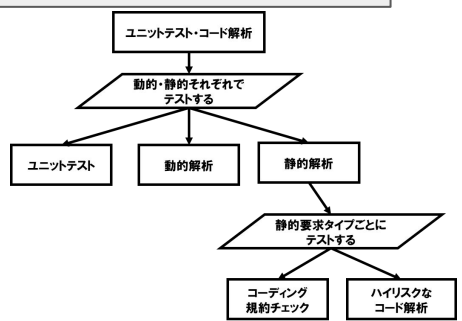
- テストレベル（ユニットテスト、結合テスト、システムテスト等）で関心の分離を実施
  - テストレベルごとにテスト自動化の目的・やり方が変わる。関心の分離を実施しやすい
    - ユニットテスト：方針と開発者の慣習に従って構築
    - 結合テスト・システムテスト：次項のテストタイプレベルで構築
- アプローチ
  - V字モデルベース
    - V字モデル、自工程完結の方針でテストレベル設計
      - 例：開発ステップやテストベースをV字で整理しテストレベルを設計
  - DevOps, CI/CDベース
    - デプロイメントパイプラインを主軸に、効果的なテスト活動を配置
      - 例：テストピラミッドモデル、テストトロフィーモデル

自動テスト活用における責務具体化・関心の分離：テストタイプ設計で保有する自動テスト技術を紐づけ

- テスト自動テストを活用できるテストタイプを蓄積・整理
  - テスト自動化技術の日頃の蓄積とチーム資産化が重要
- テストの責務設計で積極的に紐づけてテストの責務設計を進める

・責務設計に影響するテストの要求や制約
・開発や自動テストの戦略立て
・すり合わせ

テストの責務設計
・具体化・関心の分離
・連携設計
・プロセス設計



プロジェクトが保有する/保有見込みの自動テストのタイプ

- ストレステスト
- 動的解析
- ファジングテスト
- APIテスト
- ビジュアルテスト
- 欠陥注入
- エミュレータテスト
- ・・・

テスト責務の連携の設計

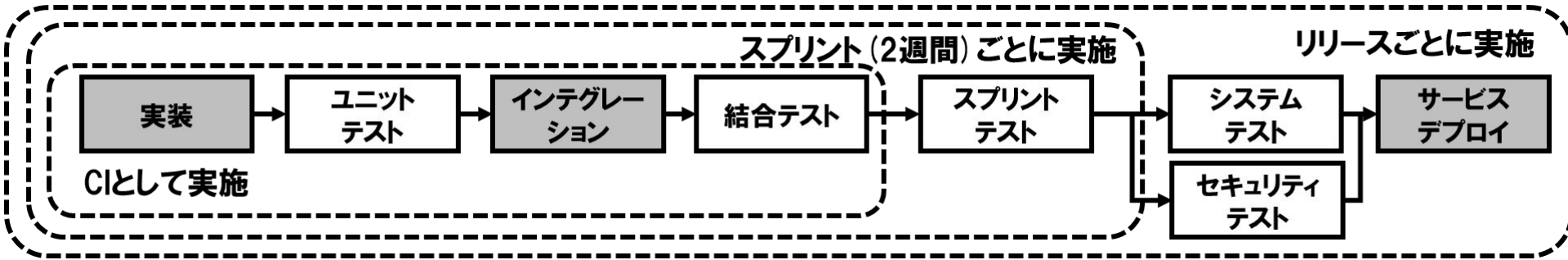
- 要求に基づいて、テストをどう連携させるかを設計する
- 連携設計を通じて自動テストの責務を確保する

要求の種類の例	連携設計アプローチ
様々なテストベースの実現性保証	各テストベースとテスト責務のマッピング
想定される欠陥の検出	検出したい欠陥タイプとテスト責務のマッピング
品質リスクや課題のコントロール	品質リスクや課題に対し各テスト責務がどうコントロールするか具体化

課題	課題対応方針
グローバル展開する組み込み機器の表示文言の品質保証の実現	文言の正確性の確認：コードチェック 文言描画の確認： アプリケーション部分：エミュレータテスト 実機部分：自動キャプチャーテスト 現地依存の本番環境確認： ローカライゼーションテスト

テストのプロセス設計

- テストの構造（依存関係、順序、関係性）を設計し、テスト責務の構成要素や連携を導出する
  - 構造パターン：重ね合わせ、分業、横断的連携、繰り返し
- 自動テストではデプロイメントパイプラインをベースにテスト責務を配置する



評価と改善

- 目的や責務をブレークダウンして評価指標を導出し、責務を担えているか評価。自動テストの改善サイクルを回す

目的の観点の例	評価指標の例
欠陥を検出できるか リグレッションを検出できるか	★欠陥検出率、欠陥流出率 ・擬陽性/擬陰性の検出データ ・欠陥注入による評価結果（エラーシーディング、ミューテーションテスト、TDDサイクル） ・欠陥シナリオに基づいたシナリオレビュー結果
仕様や構造の実現を確認できているか	・仕様/モデルに対する網羅率 ・法規などのテスト要件についての監査結果 ・コードに対する網羅率 ・熟練者による総合的・属人的評価結果
テストの経済指標は妥当か	・費用対効果（自動化案件のROI、手動テストとの比較）
その他自動テストの要件を実現しているか	・求められる性能値や安定性指標 ・コンテナ化など特定の要件達成評価結果

自動テストを活躍させるための評価と改善

- 自動テストは、特に導入時に、フィージビリティや効果が不明瞭な事が多い。その状態で自動テストを活躍させるためには、適応的・創発的な改善サイクルが必要
  - 段取り・プロセスで支える
    - プロトタイピング、フィージビリティスタディ、反復開発を段取りに組み込む
  - テスト責務の変動点を設ける
    - 補完的なテストの確保（例：手動テストでカバー、上位のテストレベルでカバー）、テストのレジリエンスの確保（例：探索的テスト）

自動テストを活躍させるためのテスト基本設計の工夫

- テスト設計の要求分析・すり合わせを通じて、必要な基礎作りを手配
- テスト責務設計で自動テストの活躍どころを増やし、自動テストの弱点をカバーする方針立てをする

まとめ

- 自動テストを活躍させるための基礎作り
- 自動テストを活躍させるためのテスト設計・テスト基本設計