# ソフトウェア開発での
# 高品質と高スピードを両立させる
# テストアプローチ

井芹 洋輝

2024/6/28 開発生産性 Conference2024
Special Session

# 自己紹介

## ●経歴

- 開発者、テストエンジニア、コンサルタント、QAエンジニアと様々な立場で様々なプロダクトのソフトウェアテスト業務に従事
- 2021からトヨタ自動車でQAリード/テストテックリードに従事
- JSTQB技術委員、テスト設計コンテストU30クラスファウンダー

## ●著作・講演

- 「テスト自動化の成功を支えるチームと仕組み」
  「シフトレフトテストを支える現代的なテスト設計」
  「テスト設計をより良くするモデリングと観点分析」
  「Androidアプリテスト技法」「システムテスト自動化標準ガイド」
  「テストの視点を活用したTDDアプローチの検討とその検証」など

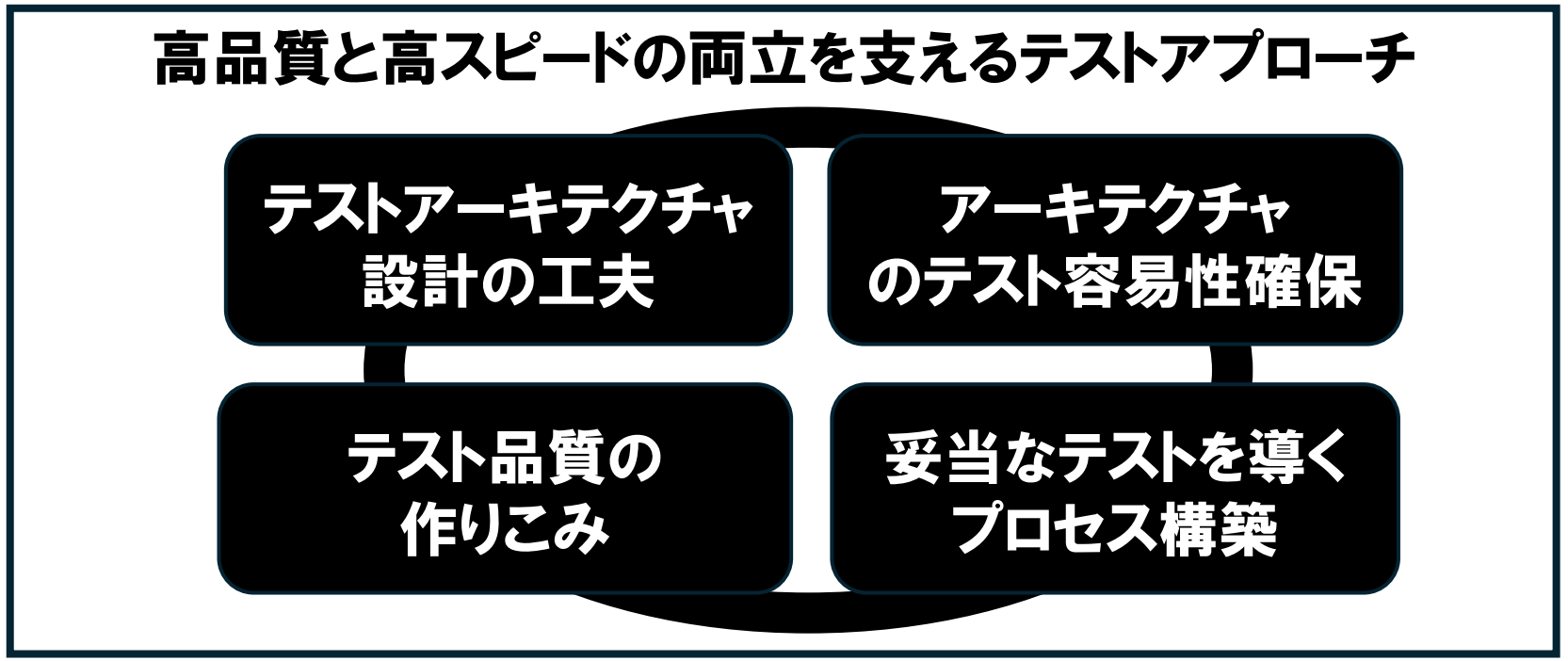
# この講演について

- 現場のテストエンジニア/QAエンジニアとして現場改善をする目線で、
  ソフトウェア開発における高品質・高スピードの両立を
  高度なレベルで実現するテストアプローチを解説します

# 「高品質と高スピードの両立」を高度なレベルで実現したい

- 単純な手元のテストの高速化といった小さな対策、行き当たりばったりの改善の蓄積では、ブレークスルーのような改善は難しい
- システム全体、チーム全体、アーキテクチャレベル、プロセス全体での対策により、ステップアップした高度なレベルの改善を実現できる

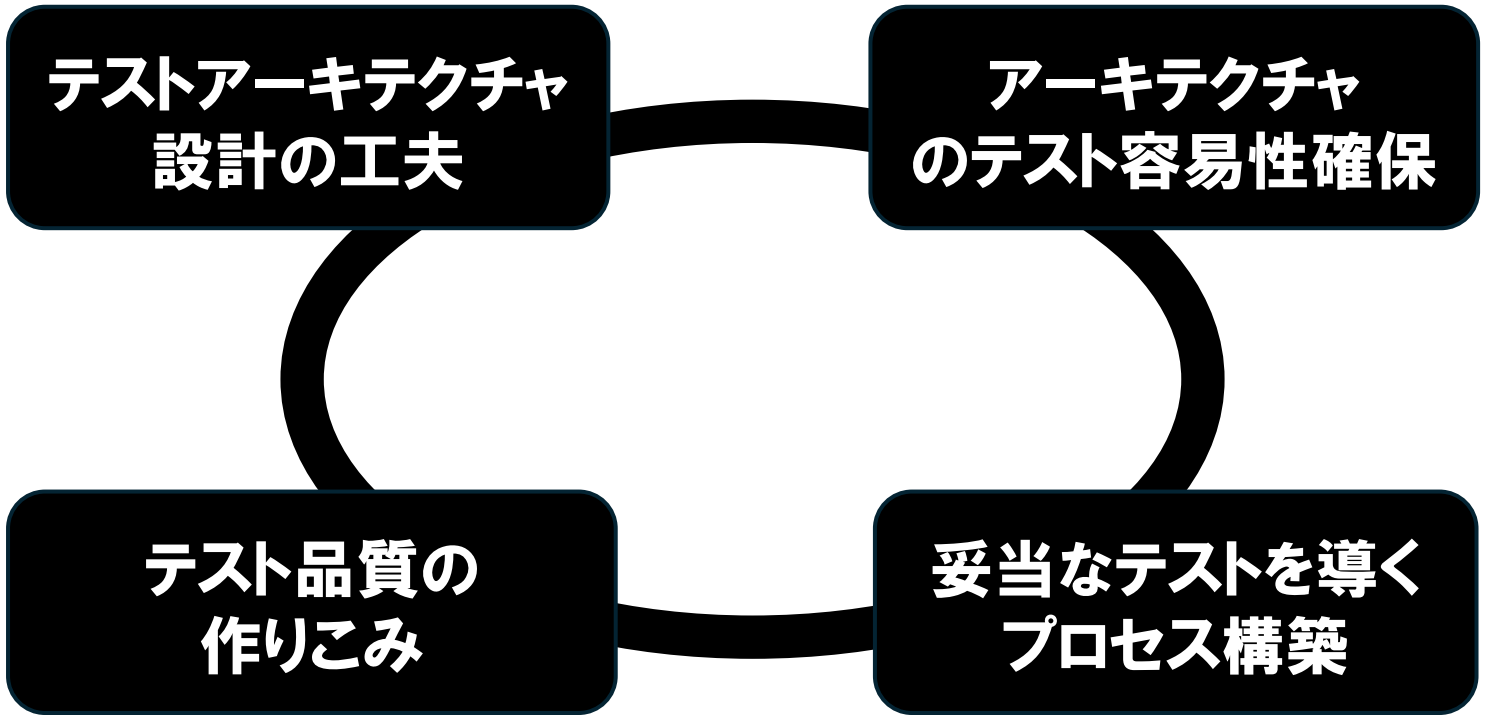
# アウトライン


高品質と高スピードの両立を支えるテストアプローチ
テストアーキテクチャ
設計の工夫
アーキテクチャ
のテスト容易性確保
テスト品質の
作りこみ
妥当なテストを導く
プロセス構築

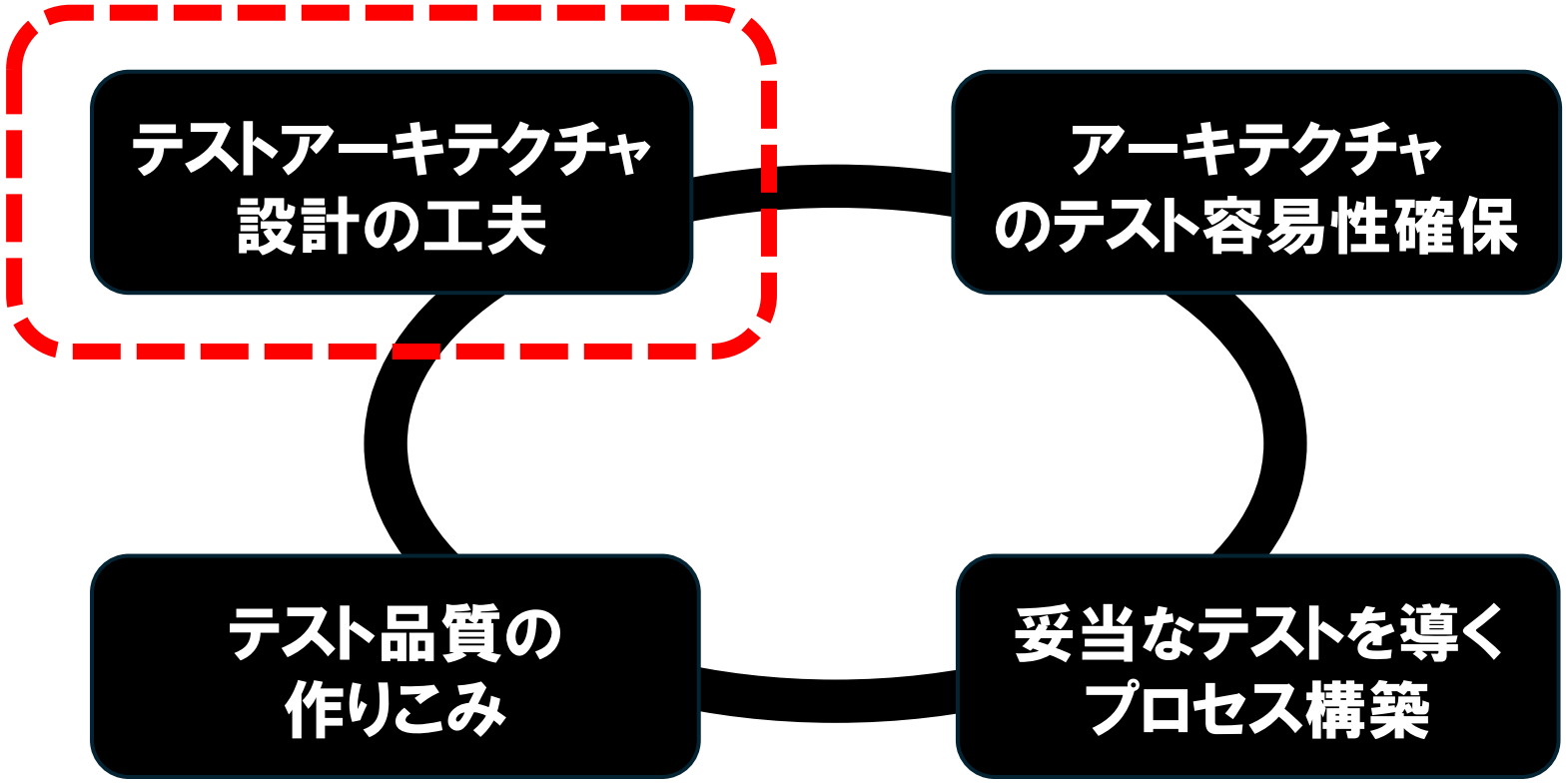

テストアプローチを支える基礎
チームの
テスト能力の確保
Whole Team
アプローチと
チーム文化形成
開発インフラと
テストシステムの
整備
品質マネジメント
の整備

# 高品質と高スピードの両立を支えるテストアプローチ

# ソフトウェア開発での高品質と高スピードを両立させるテストアプローチ



# ソフトウェア開発での高品質と高スピードを両立させるテストアプローチ

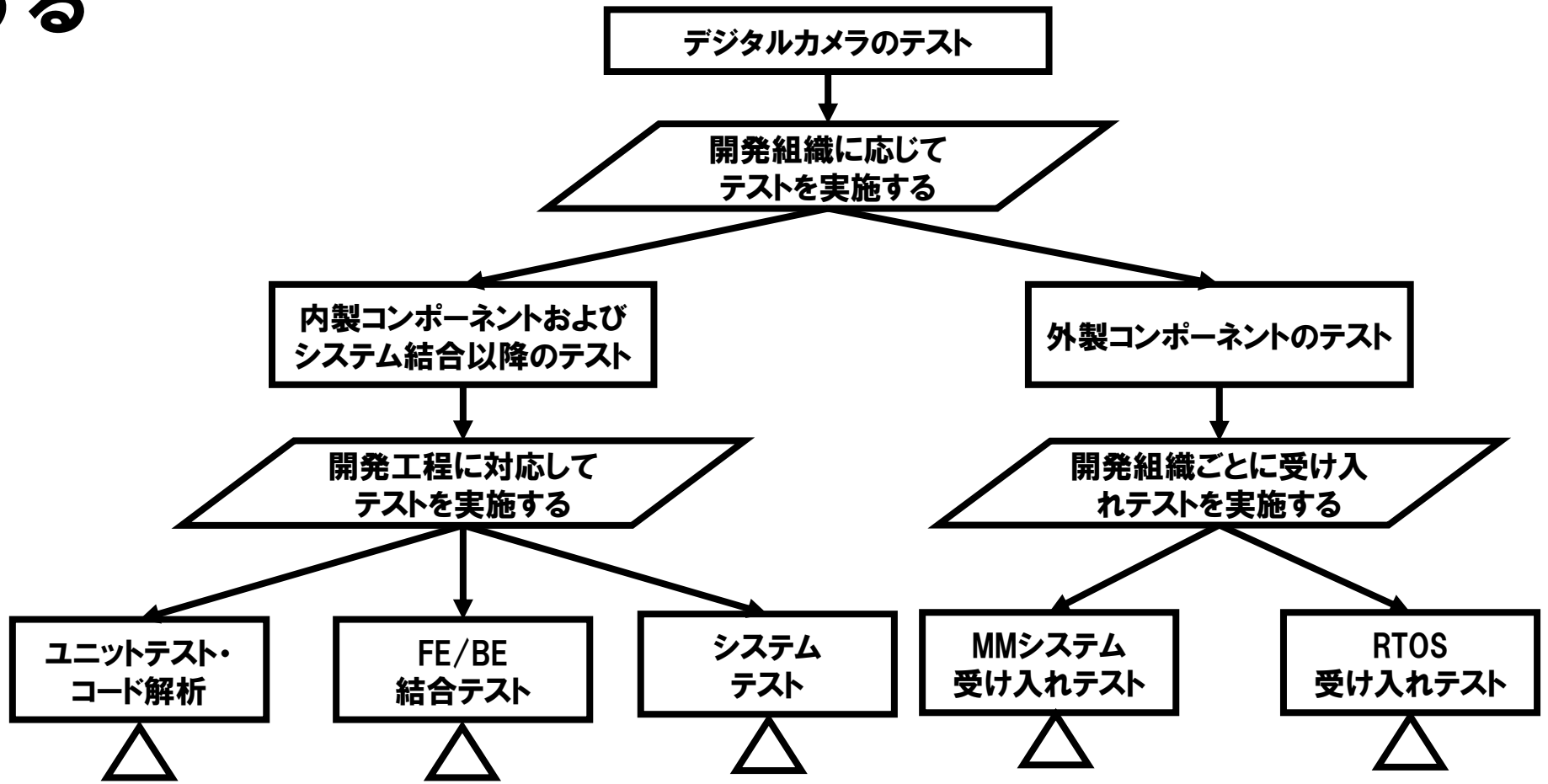


# テストアーキテクチャ設計の工夫

- **テストアーキテクチャ：**
  - ユニットテスト、結合テスト、システムテスト等、様々なテスト活動の全体構造
- **テストアーキテクチャ設計：**
  - テスト活動の全体をアーキテクチャとして扱い、その責務分担や連携の工夫を行う
  - 3つの設計アプローチで進める
    - 責務具体化/関心の分離、連携設計、プロセスとしての整理
  - 品質・スピード両立のための方向性
    - 様々なテスト活動での全体整合性の確保
    - 開発生産性の高いテスト活動の責務を最大化
    - 困難な課題に対して、複数のテスト活動の連携で対応

# テストアーキテクチャ設計アプローチ1：
# テスト責務の具体化・関心の分離

●テストの責務を整理しながら、必要なテストレベル/テストタイプを導出する



# テストアーキテクチャ設計アプローチ2：
# テスト責務の連携設計

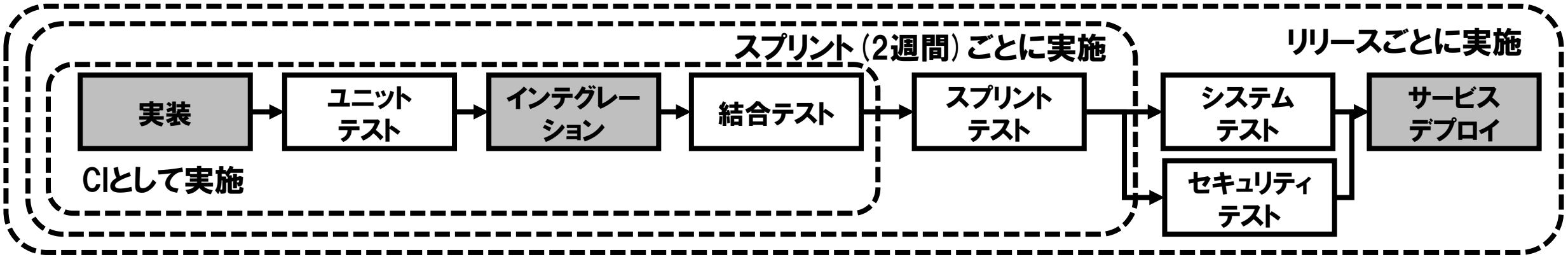
## ●要求や課題に対してテストレベル/テストタイプの連携の工夫を設計

| 要求 | 連携設計のアプローチ |
|---|---|
| 特定の欠陥の検出 | 検出したい欠陥タイプごとにテスト活動を設計 |
| プロダクトリスクの確認 | プロダクトリスクに対し、テスト活動がどう連携するか設計 |
| 品質課題の対応 | 品質課題に対して、テスト活動でどう対応するか設計 |

| 課題の例 | 課題対応方針 |
|---|---|
| グローバル展開する組込み製品の表示文言の品質確保 | 文言の正確性確認、翻訳品質確認：データ静的テスト<br>文言描画の確認：<br>　アプリケーション描画：エミュレータテストで全網羅<br>　実機動作：自動キャプチャーテストで代表パターン確認<br>現地依存の本番環境確認：<br>　ローカライゼーションテスト |

# テストアーキテクチャ設計アプローチ3：
# デプロイメントパイプライン・プロセスの整備

- テストアーキテクチャの構造（依存関係、順序、関係性）を設計し、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



# テストアーキテクチャ設計アプローチ

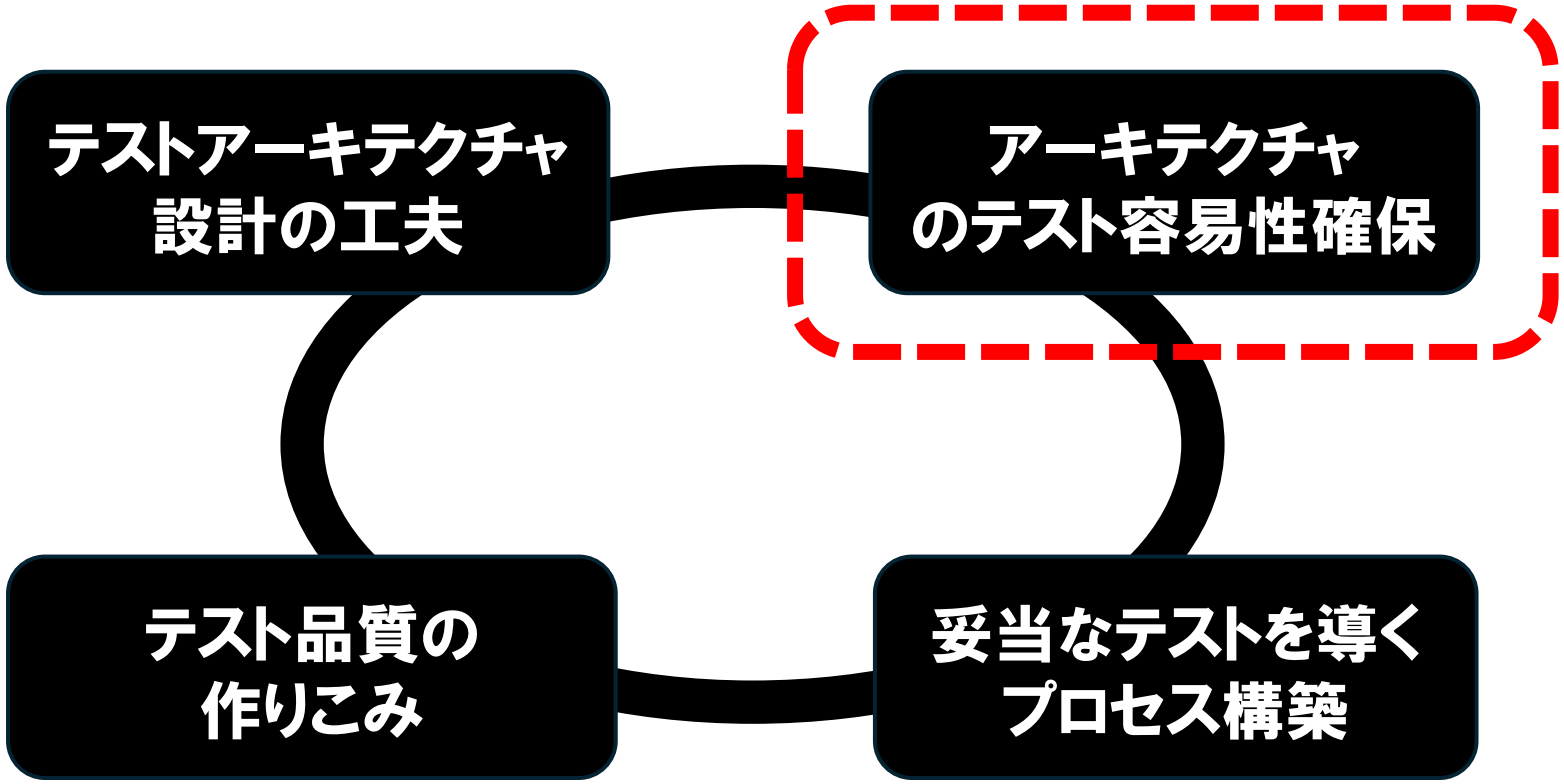
- テスト責務の具体化/関心の分離
- テスト責務の連携設計
- デプロイメントパイプライン/プロセスでの整理

→適切なテスト活動の全体構造を作りこむ

# 高品質と高スピードの両立を目指すテストアーキテクチャ設計

- ●生産性の高いテストレベル/テストタイプの責務を最大化する
  - ・自動化可能。CI/CDのデプロイメントパイプラインに統合可能
  - ・テストの性能効率性・保守性を作りこみやすい
  - ・顧客満足/プロダクト価値に近いテストができる
- ●生産性の低いテストレベル/テストタイプの責務を最小化する
  - ・手動のシステムテストの責務を他のテストに分散する
- ●シフトレフトを推進する
  - ・シフトレフトできるテストレベル/テストタイプを確保し責務を広げる
  - ・システムテストから結合テスト・ユニットテストへテスト責務を移譲へ
    - ・テストピラミッド/テストトロフィーの戦略推進

# ソフトウェア開発での高品質と高スピードを両立させるテストアプローチ



# テスト容易性：テストしやすさについてのプロダクトの品質特性

## ●優れたテスト容易性の効果

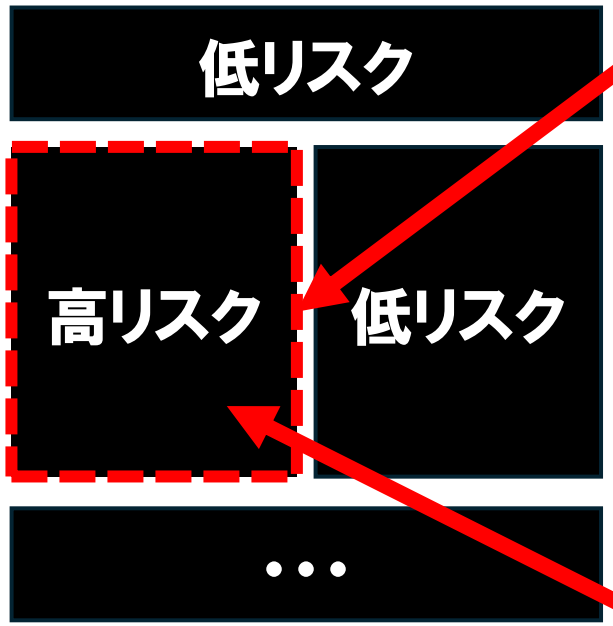
- テストに必要なコスト、期間、リソースを削減し、テストによる品質改善サイクルを加速させる
- テストに関する技術的制約を緩和し、テスト自動化の実現といった、テストについての改善を導入しやすくする
- テストの誤りや冗長性を削減しやすくして、テストによる品質改善サイクルの精度を高める

# テスト容易性：テストしやすさについてのプロダクトの品質特性

| 品質特性 | 内容 | 具体例 |
| --- | --- | --- |
| 観測容易性 | テスト対象の観測のしやすさ | エラーログの充実度 |
| 制御容易性 | テスト対象の操作のしやすさ | APIの充実度 |
| セットアップ容易性 | テストのセットアップのやりやすさ | コンストラクタの単純さ |
| 実行容易性 | 実行の容易さ | テスト実行のブロック要因の少なさ |
| 分解容易性 | テスト対象の分割・置換の容易さ | 接合部の充実度 |
| 網羅容易性 | テストでの網羅のしやすさ | デッドコードの少なさ |
| 安定性 | テスト対象の安定性・バグの少なさ | 変更頻度の少なさ |
| 適時性 | 適時で実行できるか | 実行可能な形式の入手のしやすさ |
| 環境構築容易性 | テスト環境の構築のしやすさ | 環境の冪等性 |
| 問題解析性 | バグの特定のしやすさ | 解析ログの充実度 |

対象によってより具体的な品質特性がある。例）自動化：仮想化容易性、並列化容易性、CI/CD統合容易性

# 高度な高スピード・高品質の両立のためのテスト容易性確保：アーキテクチャのテスト容易性を作りこむ



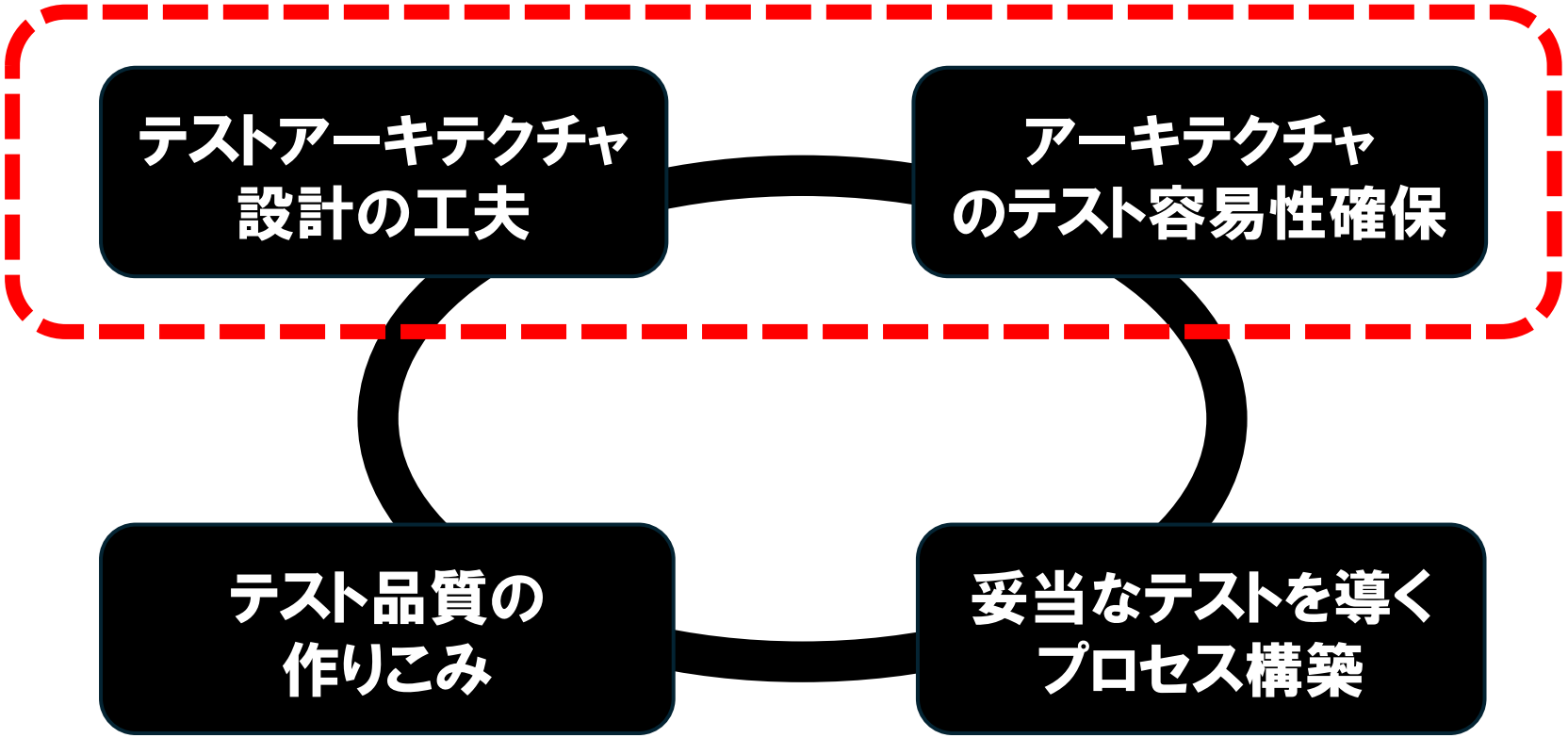

疎結合・高凝集度設計で
モジュール化を推進。
品質リスクを分離する
（例：接合部配備、信頼できるインターフェース手段採用、カプセル化/状態や副作用の局所化、インターフェースのシンプル化/契約による設計）

品質レベルに応じた
アーキテクチャレベルの
テスト容易性確保
（例：自動化対応、CI/CD統合可能、テストの障害排除、網羅容易性確保、シフトレフトテスト対応）

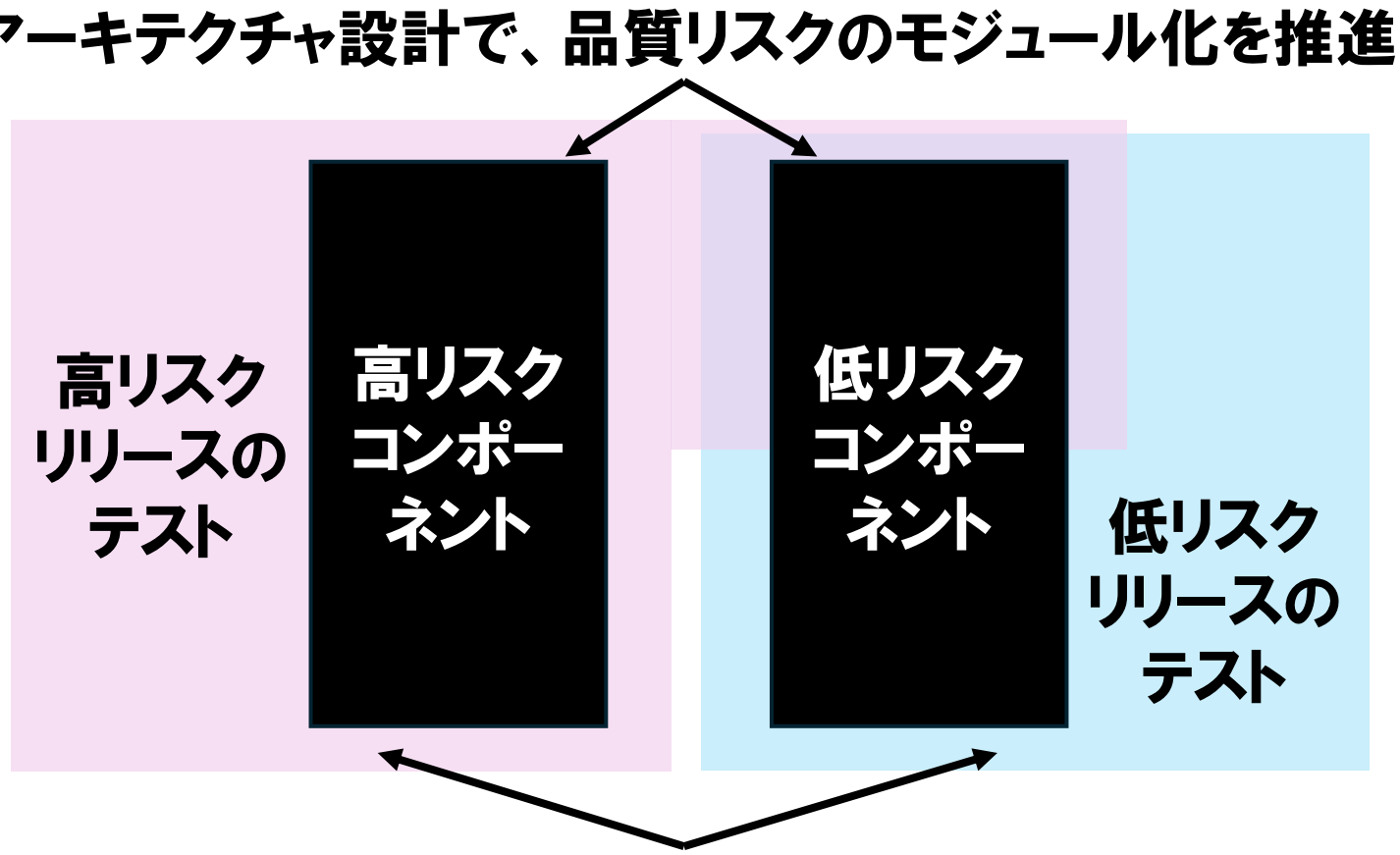
●高リスクで詳細なテストが必要なコンポーネントは、高生産性のテストで品質確保

●高リスクコンポーネントは結合テストレベルで品質を担保。他のコンポーネントを軽快にリリース可能に

# ソフトウェア開発での高品質と高スピードを両立させるテストアプローチ

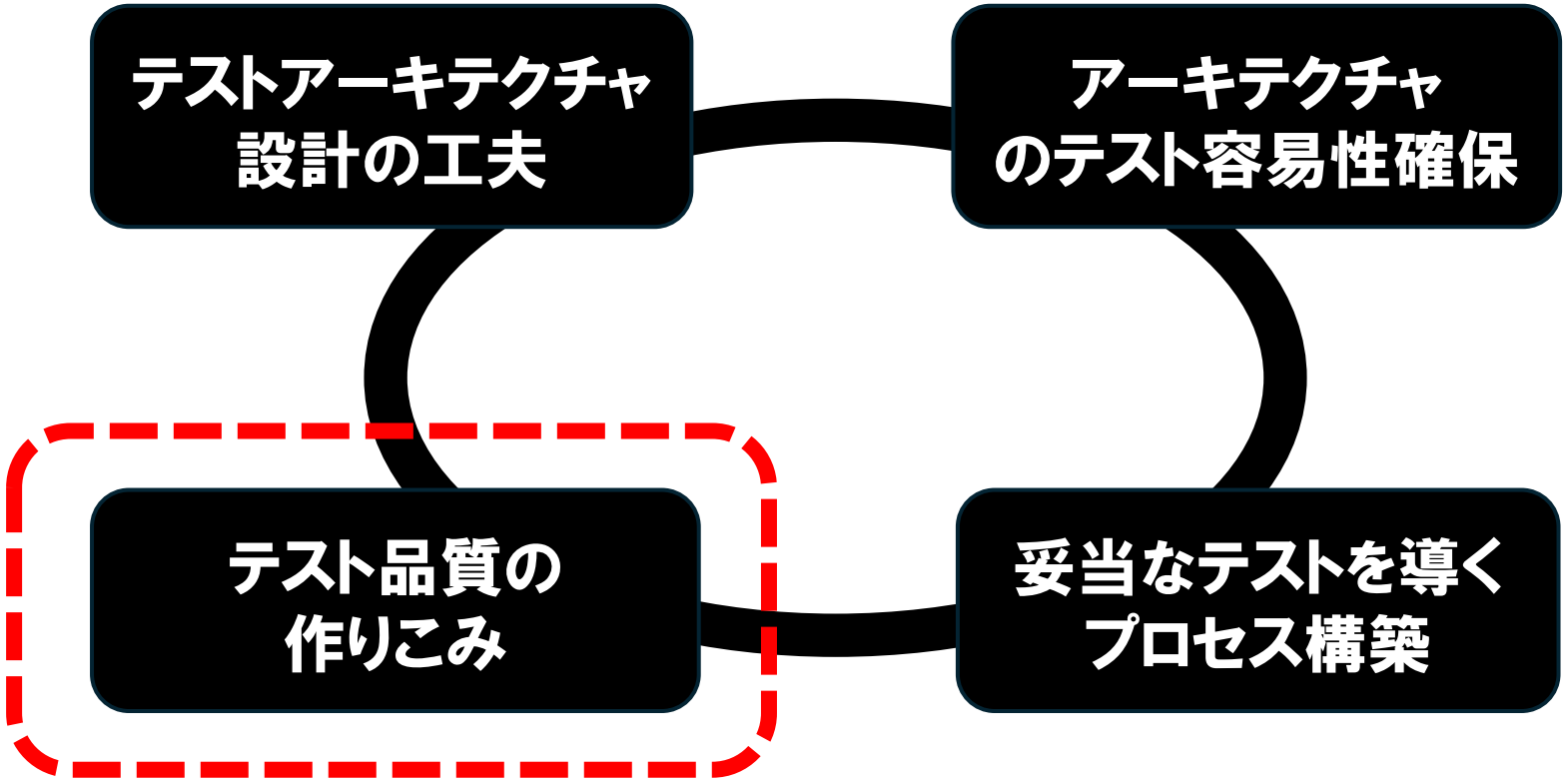


# 高品質と高スピードの両立のため、テストのモジュール化を推進する




**高リスクリリースは詳細なテストで、低リスクリリースはスピード重視のテストで**
**総体として高品質・高スピードのデリバリを実現**

# ソフトウェア開発での高品質と高スピードを両立させるテストアプローチ



# 高スピード・高品質の両立には高度なテストの品質が必要

| 求められるテストの品質 | 品質の内容 |
| --- | --- |
| 妥当性・有効性 | テストの目的を満たせられる。バグの検出やプロダクトリスクの確認、規格等への充足性確認ができる |
| フィードバックの適時性 | 必要なタイミングでテストの成果を出せる。シフトレフトを推進できる |
| 性能効率性 | 時間、コスト、人、機材や環境などをより少なくテストの目的を達成できる |
| 信頼性 | 偽陽性（バグがないのにテストが失敗する）・偽陰性（バグがあるのにテストが成功する）の問題なく、安定してテストの責務を果たし続けられる |
| 持続可能性 | ソフトウェア開発のライフサイクル中、無理なく継続的にテスト活動を続けられる |
| 制約許容性 | テストベースの不足や、テストの制約をより多く許容できる |

上記は一部。使用性や、環境構成に対応するための移植性、セキュリティ等も加えて必要

# 高度なテストの品質の作りこみ

- 高度なテストの品質は、要求やリスクに対する作りこみの蓄積で実現される

# テストの信頼性の高度な作りこみ事例：
# フレーキーテスト

- **フレーキーテスト：テストの信頼性不足の典型例**
  - 変更がないのにテスト結果が不安定なテスト。
    ランダムに偽陽性（バグがないのにテストが失敗するなど）、偽陰性（バグがあるのにテストが成功）の問題が発生する
  - End to Endテストや実機自動テストなど、一定の複雑さを持つ自動テストで課題となる
  - 解決することで、テストの複雑化・高度化の許容範囲が広がる

# テストの信頼性の高度な作りこみ事例：フレーキーテストの原因と対策

| 原因 | 内容 |
| --- | --- |
| 並行処理の不具合 | 並行処理の順序やタイミングのランダム性で発生 |
| 隠れた副作用や状態 | フレームワークの隠れた状態など。テストの順序やタイミングの変化で悪影響を及ぼす |
| 依存コンポーネントの不安定さ | 依存する外部サービスや外部コンポーネント（OS等） |
| 未定義・未既定の処理 | 未初期化の変数値を使うなど、コードの未定義・未既定の振る舞い |
| 本質的にテスト対象が不安定 | 機械制御や物理計測など、対象が本質的に不安定である |

# テストの信頼性の高度な作りこみ事
# フレーキーテストの原因と対策

| 原因 | 内容 |
| --- | --- |
| 並行処理の不具合 | 並行処理の順序やタイミングのラン<br>で発生 |
| 隠れた副作用や状態 | フレームワークの隠れた状態など。テ<br>順序やタイミングの変化で悪影響を |
| 依存コンポーネントの不安定さ | 依存する外部サービスや外部コンポー<br>（OS等） |
| 未定義・未既定の処理 | 未初期化の変数値を使うなど、コー<br>定義・未既定の振る舞い |
| 本質的にテ<br>対象が不安 | |

適切な並行処理の設計・実装
（適切な保護、並行処理のシンプル化、
適切な非同期処理機能の仕様）

テストのモジュール性の向上
（テストの不適切な結合性を削減、状
態や副作用のスコープを制限）

テストからのランダム性の排除
（不安定なコンポーネントはテストダブ
ルに置換）

テストのための実装の基礎力確保
（未定義・未既定を避ける、ロジックや
状態をシンプルに保つ等）

統計的アプローチの導入
（本質的に不安定さを持つなら、相関
分析等で判定するように変更）

テストの自己診断機能の充実
（起動時・接続時の状態チェック、
防御的プログラミングの推進）

# テストの信頼性の高度な作りこみ事
# フレーキーテストの原因と対策

| 原因 | 内容 |
| --- | --- |
| 並行処理の不具合 | 並行処理の順序やタイミングのラン<br>で発生 |
| 隠れた副作用や状態 | フレームワークの隠れた状態など。テ<br>順序やタイミングの変化で悪影響を |
| 依存コンポーネントの不安定さ | 依存する外部サービスや外部コンポー<br>（OS等） |
| 未定義・未既定の処理 | 未初期化の変数値を使うなど、コー<br>定義・未既定の振る舞い |
| 本質的にテ<br>対象が不安 | |

適切な並行処理の設計・実装
（適切な保護、並行処理のシンプル化、
適切な非同期処理機能の仕様）

テストのモジュール性の向上
（テストの不適切な結合性を削減、状
態や副作用のスコープを制限）

テストからのランダム性の排除
（不安定なコンポーネントはテストダブ
ルに置換）

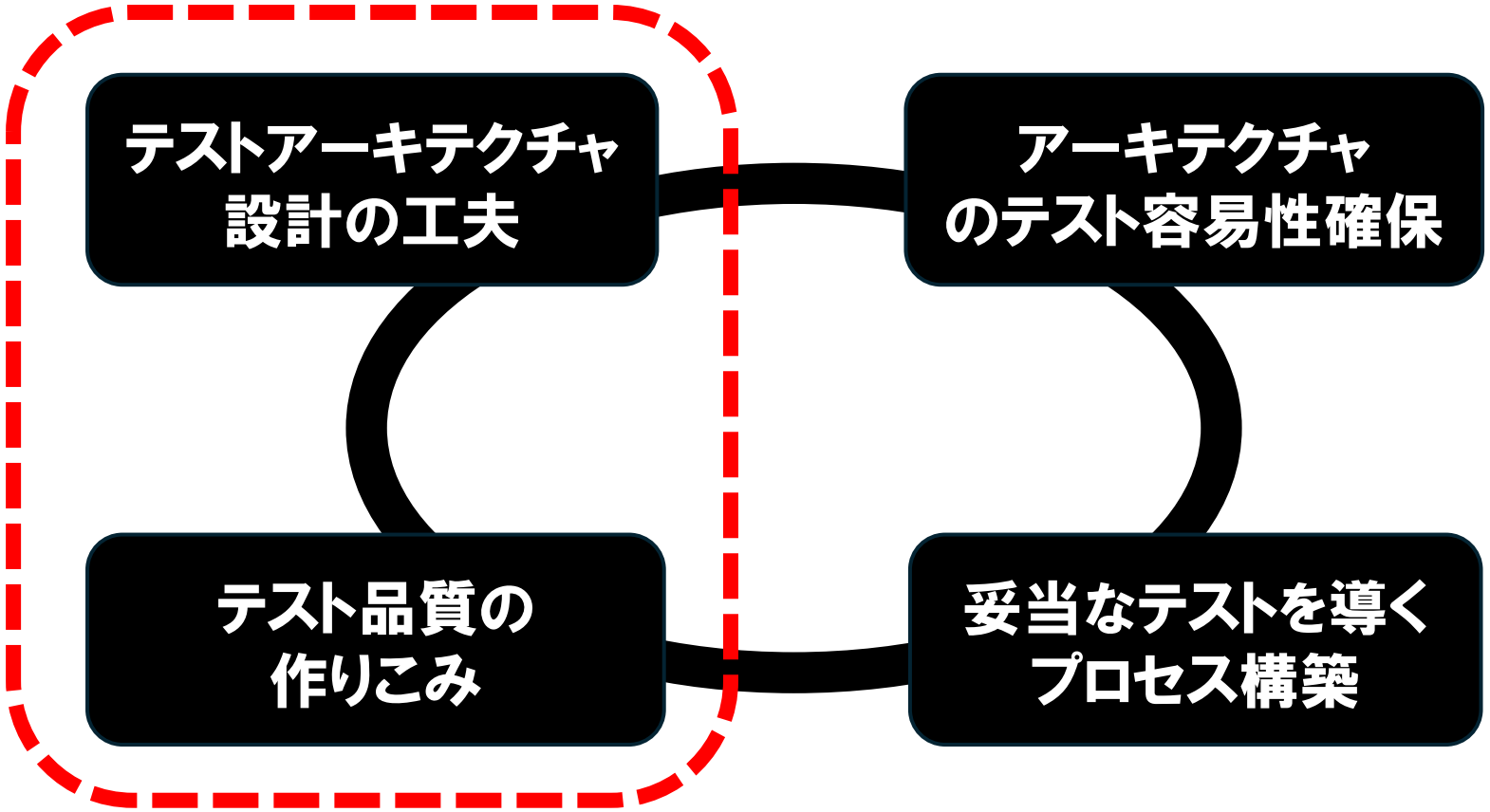
テストのための実装の基礎力確保
（未定義・未既定を避ける、ロジックや
状態をシンプルに保つ等）

統計的アプローチの導入
（本質的に不安定さを持つなら、相関

テストの自己診断機能の充実
（起動時・接続時の状態チェック、
グの推進）

品質改善の積み重ねで高度な信頼性が確保され、高度なテストが実現される

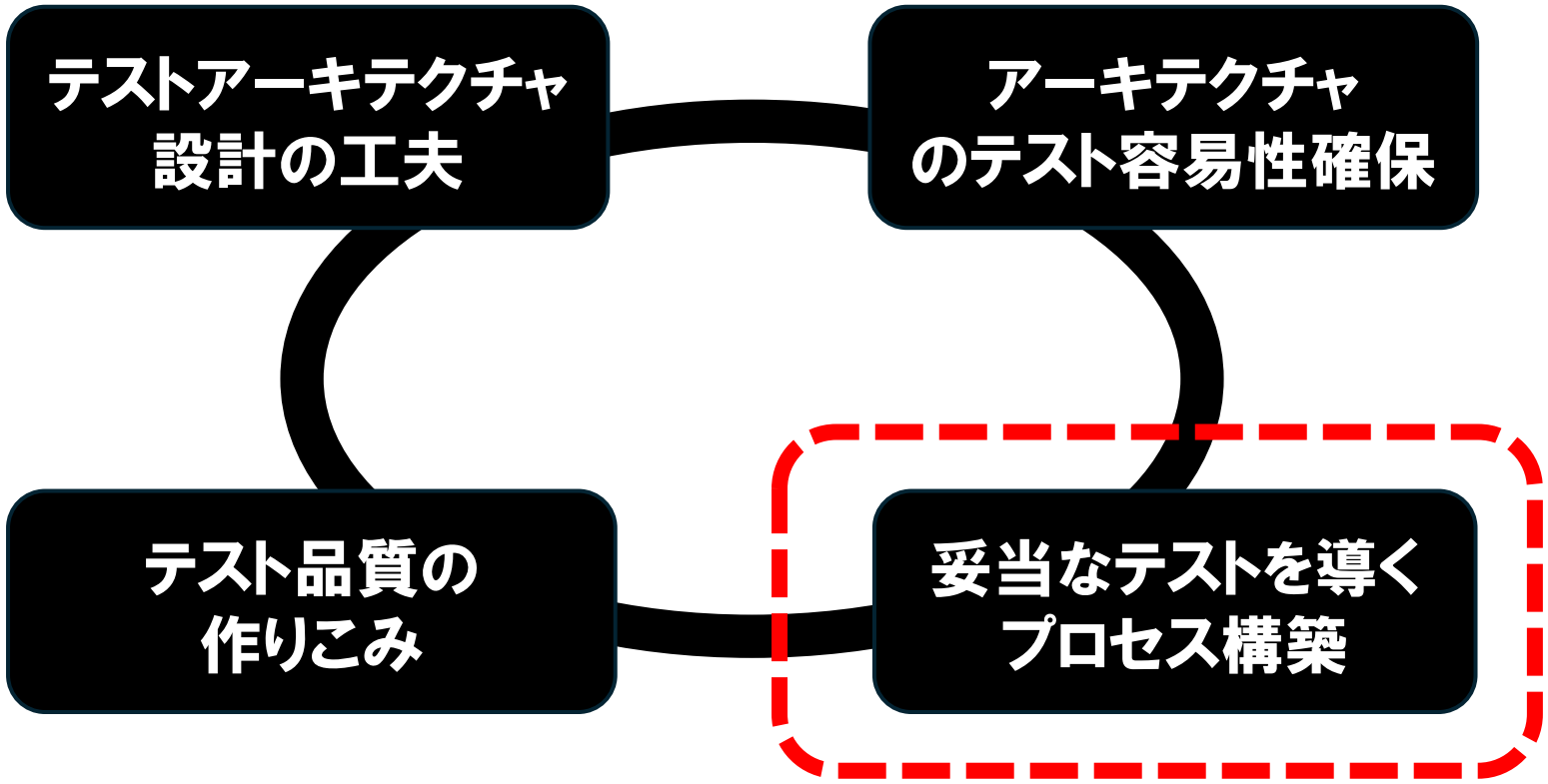
# ソフトウェア開発での高品質と高スピードを両立させるテストアプローチ



# 高スピード・高品質の両立に必要なテストの品質を高度に作りこむ

- ●テストの品質を高度に作りこみ、高スピード・高品質を支えるテストレベル/テストタイプを実現する
  - ・求められるテストの品質
    - ・保守性
    - ・性能効率性
    - ・有効性/妥当性
    - ・自動化容易性
    - ・CI/CD統合容易性
- ●テストアーキテクチャ設計で、品質を確保できたテストレベル/テストタイプの責務を最大化する

# ソフトウェア開発での高品質と高スピードを両立させるテストアプローチ



# 妥当なテストを導くプロセス構築

## ●高品質・高スピードを両立させるテストプロセス構築

1. 顧客満足/ビジネス価値とテストの改善サイクルの形成
   - ユーザ・ビジネスの視座からテストの要求・制約を得る
   - ユーザ・ビジネスからのフィードバックサイクルでテストを方向づけする
   - 上記のフィードバックサイクルを回して継続的にテストを改善する
2. 妥当なテストを生み出すテスト設計プロセス構築
   - 適切なテスト分析/設計アプローチで、テストの要求・制約に対して妥当なテストケースを作り、維持する

# 顧客満足/ビジネス価値とテストの改善サイクルの形成

【単発・内部のフィードバックサイクル】
モックアップ、プロトタイピング、ユーザテスト


【ユーザ、PO/PdM、BAを巻き込んだ協働作業】
ふるまい駆動開発、受け入れテスト駆動開発

【運用を巻き込んだフィードバックサイクル】
DevOps、継続的デリバリ、シフトライトテスト

# 顧客満足/ビジネス価値とテストの改善サイクルの形成

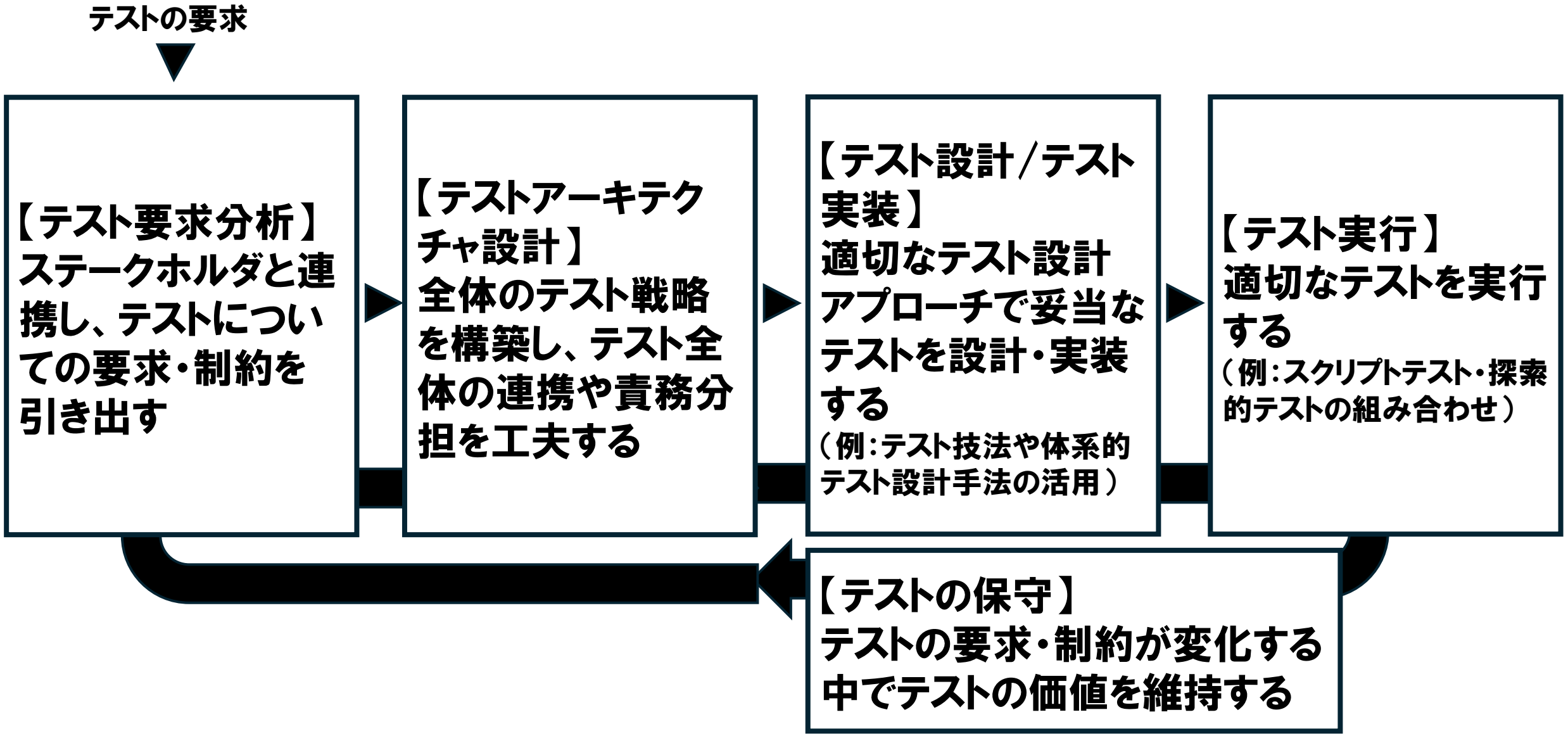
【単発・内部のフィードバックサイクル】
モックアップ、プロトタイピング、ユーザテスト


【ユーザ、PO/PdM、BAを巻き込んだ協働作業】
ふるまい駆動開発、受け入れテスト駆動開発

独りよがりにテストを作らない
ユーザやビジネスにとって妥当なテストを目指す

用を巻き込んだフィードバックサイクル】
)ps、継続的デリバリ、シフトライトテスト

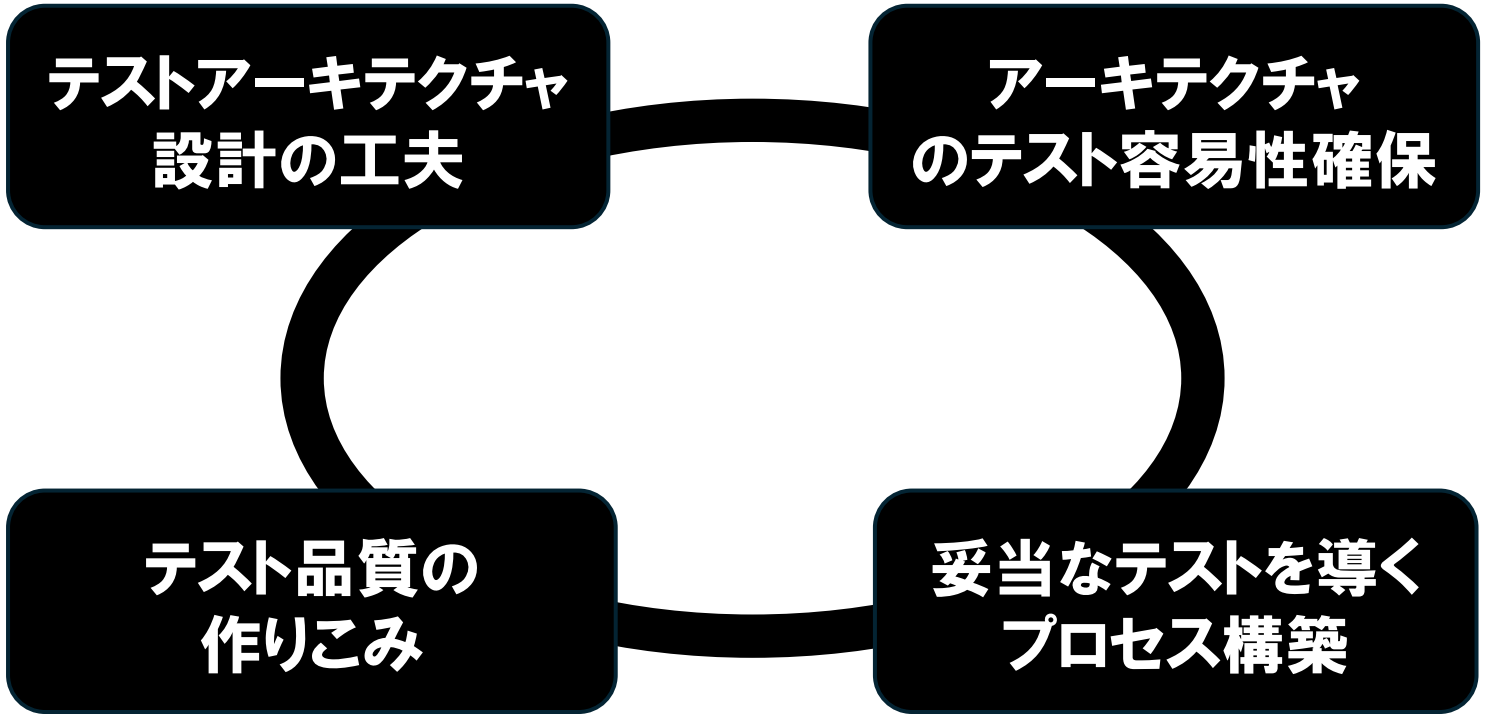
# 妥当なテストを導くテスト設計プロセスの構築



# ソフトウェア開発での高品質と高スピードを両立させるテストアプローチ

・テスト全体の連携や責務分担の工夫
・生産性に優れたテストの責務拡大

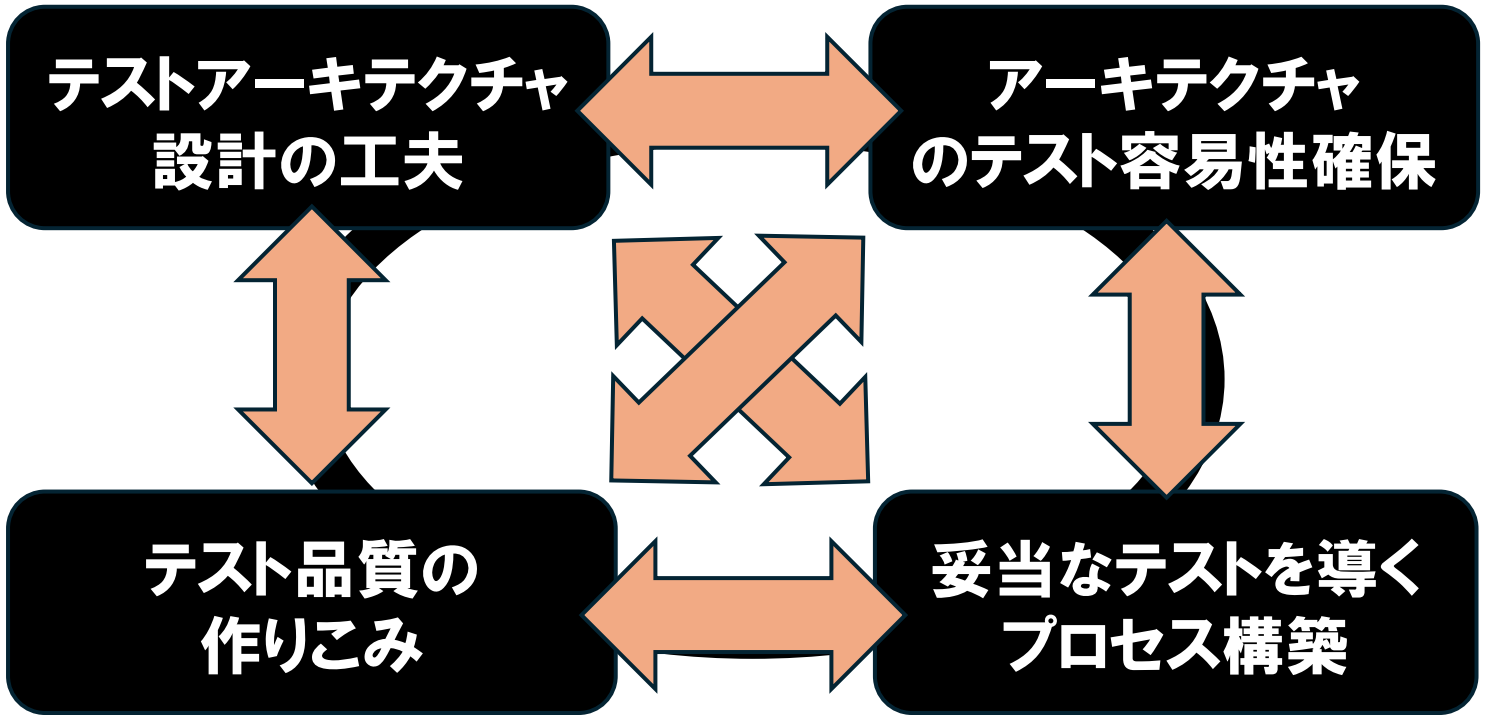
・アーキテクチャレベルでの品質リスクの分離
・品質リスクのモジュール化、テストのモジュール化




・テストの有効性・効率性・保守性・信頼性等
テストの高度な品質確保

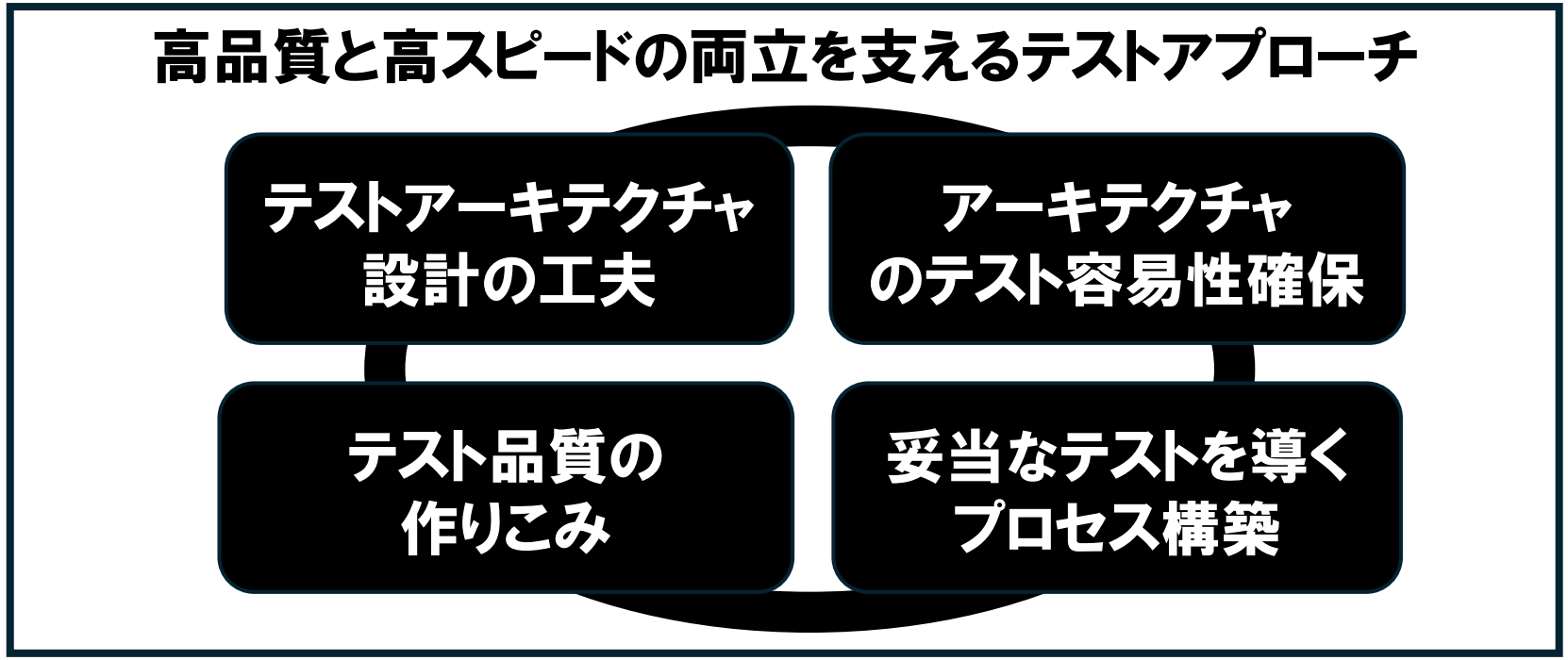
・顧客満足・ビジネス価値の改善サイクルの形成
・妥当なテストを作るテスト設計プロセスの構築

# ソフトウェア開発での高品質と高スピードを両立させるテストアプローチ



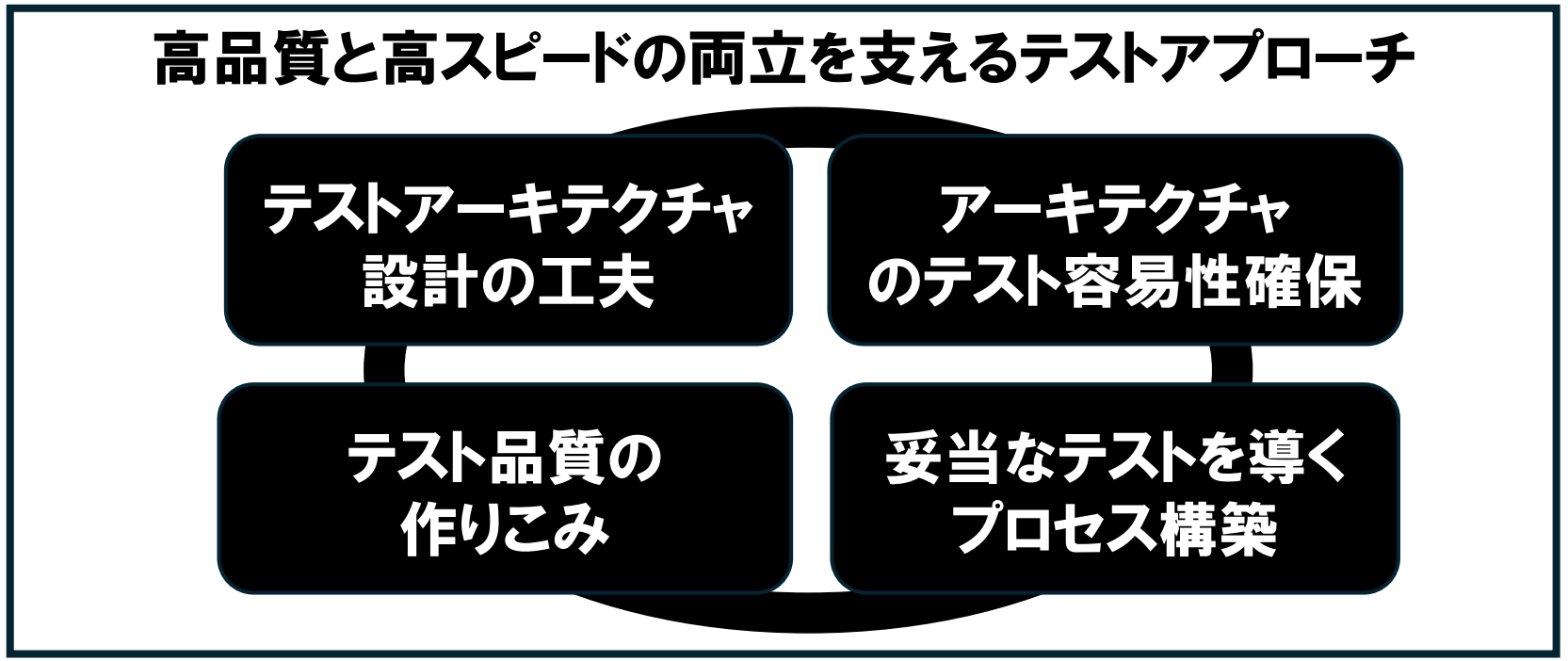
# テストアプローチを支える基礎

# 高度なテストアプローチは、チーム、プロセス、システムの下支えがあってこそ推進が容易になる




# まとめ

# アウトライン


高品質と高スピードの両立を支えるテストアプローチ
テストアーキテクチャ
設計の工夫
アーキテクチャ
のテスト容易性確保
テスト品質の
作りこみ
妥当なテストを導く
プロセス構築


テストアプローチを支える基礎
チームの
テスト能力の確保
Whole Team
アプローチと
チーム文化形成
開発インフラと
テストシステムの
整備
品質マネジメント
の整備