テスト設計チュートリアル　テスコン編 ’26

ytte Lab
yumotsuyo test engineering laboratory

2026.5.13

© YTTELAB LTD 2026-

講師紹介

株式会社ytte Lab

湯本 剛

❑経歴

1991年　工作機器メーカーで経理部に所属し、生産管理システムの受入テストに従事
1995年　メーカー系列ソフトハウスにてテストリーダーとしてアプリ開発に従事
　　財務会計のパッケージソフト、プリンタ、スキャナ、製造業受託開発（在庫管理、生産管理など）、顧客向けプロダクト販売Webサイト構築など
2003年　ソフトウェアテストコンサルタントとなりテストプロセス改善コンサル、テストツール導入支援、テスト教育に従事
2010年　外資系ITソフトウェア部門にて、テスト管理、自動テスト、性能テストツールのプリセールスに従事
2014年　上記の外資系ITサービス部門にて、デリバリー案件のテストマネジメント、QA組織立ち上げのコンサル＆デリバリーに従事
2017年　外資系損保　ソリューションデリバリー部テスティング課　課長（合併によるシステム統合のテストマネージャー兼務）
2019年　SaaS型クラウドサービス　QAエンジニアに従事

❑現在（2026年5月）

株式会社ytte Lab 代表取締役／コンサルタント
freee 株式会社 QAマネージャー
NPO法人　ASTER：ソフトウェアテスト技術振興協会　理事、JSTQB(認定ソフトウェアテスト技術者資格制度)技術委員
ISO/IEC JTC1/SC7 WG26　エキスパート（ISO29119：テストプロセス標準の策定）

❑著書

▪書籍（6冊）：　ビジネス主導のテストプロセス改善、現場の仕事がバリバリ進むテスト手法、ソフトウェアテストの基礎、JSTQBソフトウェアテスト教科書、基本から学ぶソフトウェアテスト、ソフトウェアテスト293の鉄則
▪学術論文：「データ共有タスク間の順序組合せテストケース抽出手法」電気学会　論文誌C　Vol. 137　　No. 7
▪国際学会：　A Test Analysis Method for Black Box Testing Using AUT and Fault Knowledge　など多数
▪雑誌、ネット記事：　日経ITProなど多数

© YTTELAB LTD 2026-

はじめに

- テスト設計コンテストは2011年からやっています。テスト設計技術を広く共有し、さまざまな意見を交換することでさらに進化させていくことが目的です。
- テスト設計のためのチュートリアルは毎年開催されていて、多くの資料が参照可能です。
- 本日は、これまでのチュートリアルの資料での説明とは違った切り口で、各活動で本質的になにをやるかを説明します。
  - 具体的な手法などは過去のチュートリアルが参考になるので確認してください。
  - 例：テスト観点モデル、テストコンテナ、テストフレーム

本チュートリアルでは、属人的に行われがちなテスト設計を整理し、
再現性のあるプロセスとして捉えることを目的とします!

© YTTELAB LTD 2026-

アジェンダ

テストの役割と全体像

テスト開発プロセス

その他注意して欲しいこと

[参考]とついているスライドは、有用な追加情報ですがチュートリアルの中では解説しません

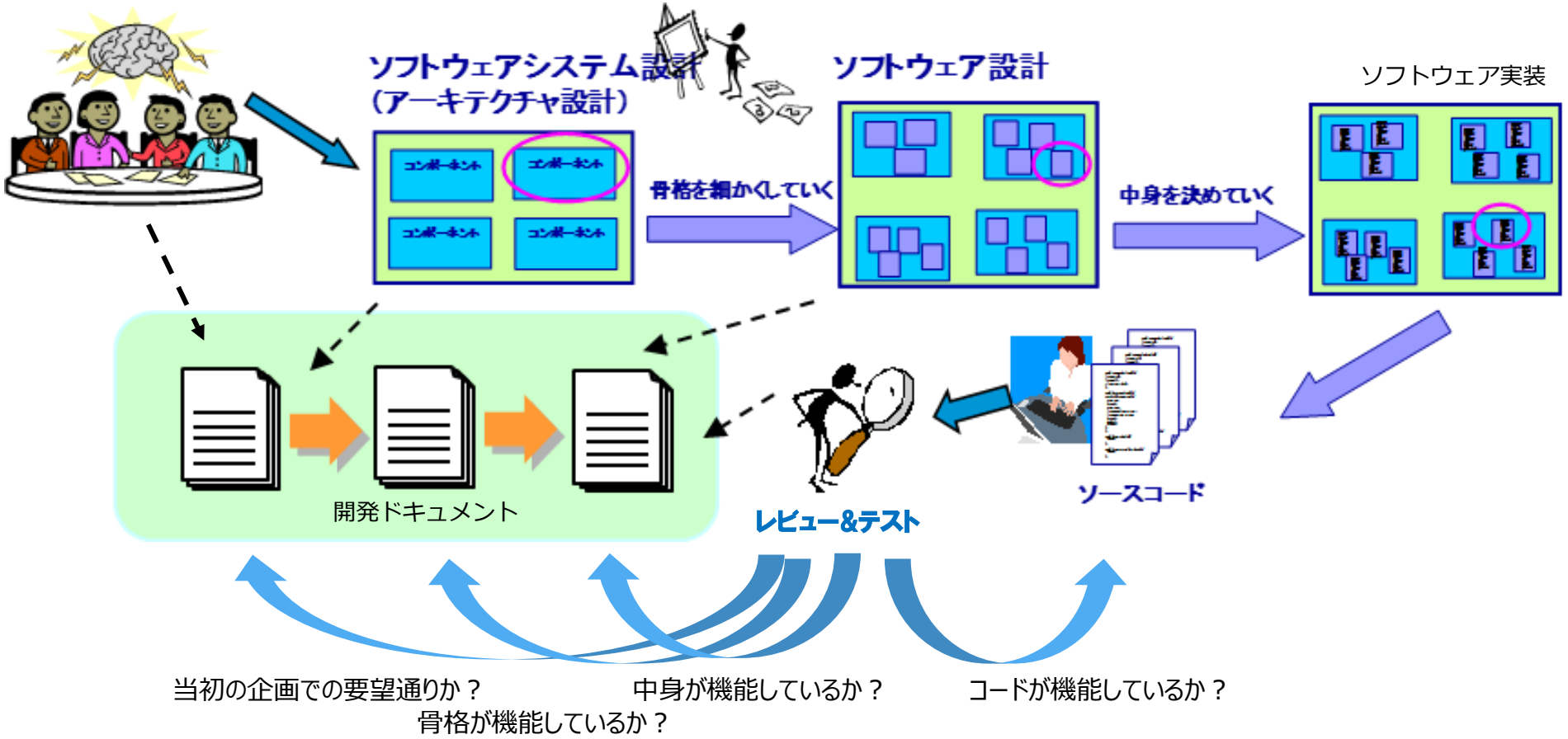
© YTTELAB LTD 2026-

テストの役割と全体像

テスト開発プロセスの解説の前に知っておいて欲しい大事なことを説明します！

© YTTELAB LTD 2026-

ソフトウェアテストとは？

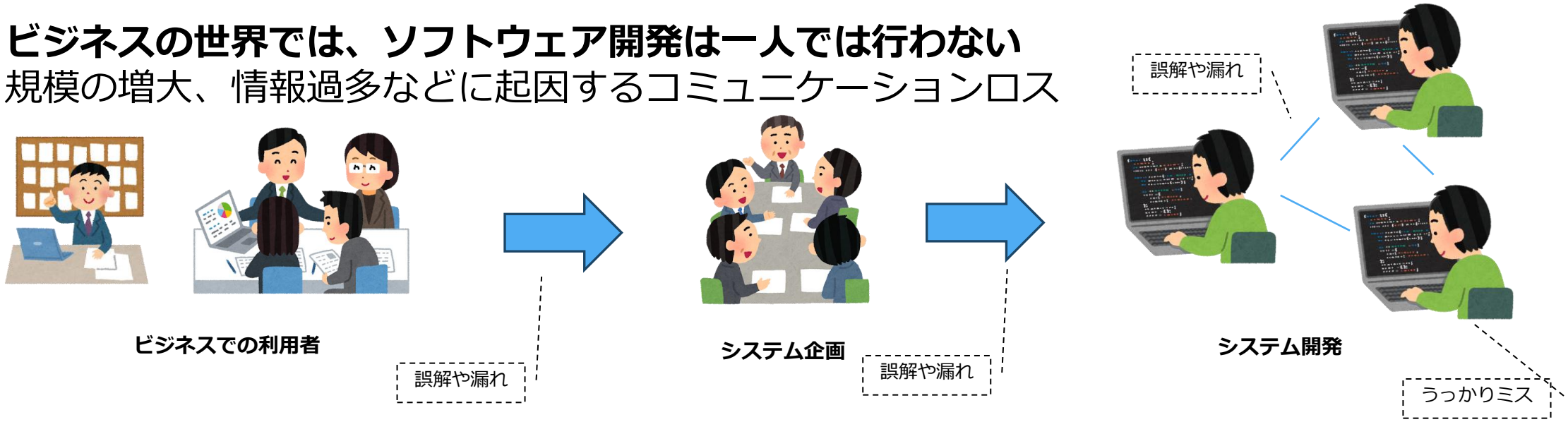


テストは、一連の開発活動の中で**大丈夫か？**を確認する
エンジニアリング活動の一つ

© YTTELAB LTD 2026

なぜソフトウェアテストが必要か？

ビジネスの世界では、ソフトウェア開発は一人では行わない
規模の増大、情報過多などに起因するコミュニケーションロス



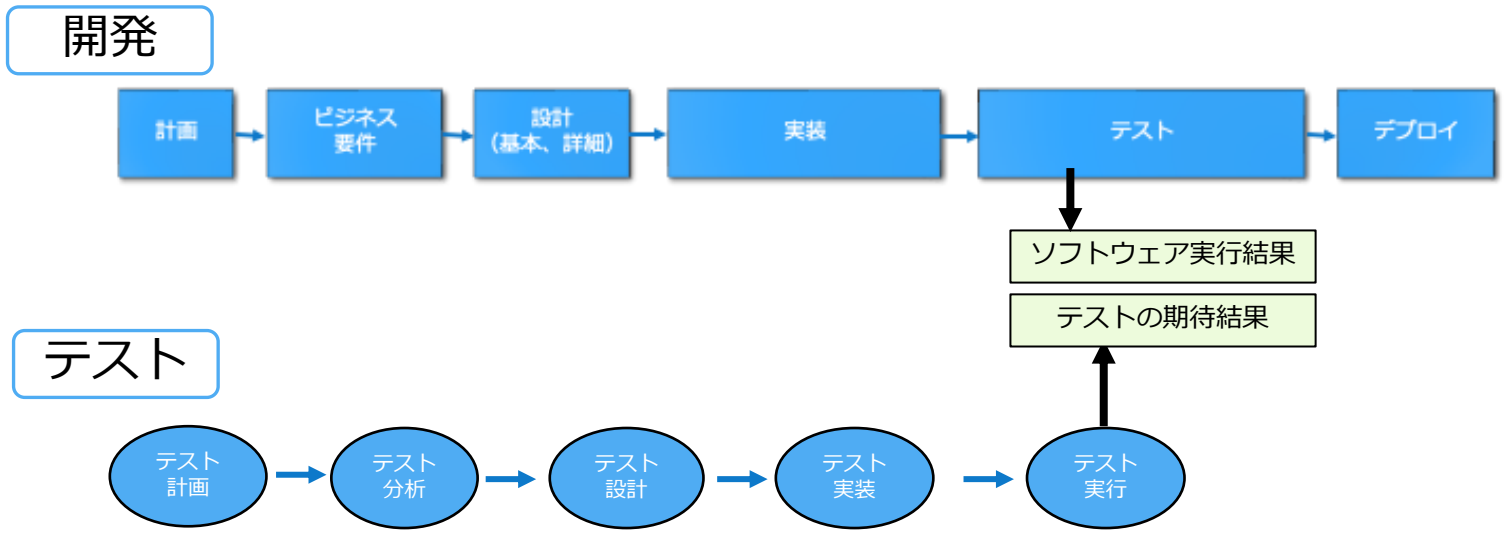
口頭だけで決めて開発してしまう
うっかり勘違いを誘発

人は間違えるものだから、現物が大丈夫かをテストしないといけない

© YTTELAB LTD 2026

二重にやることで間違える可能性を最小限にする

二つの同じものを開発して、実行結果を突き合せれば高い確率で誤りの有無を判断できる。
このときに**比較するのは結果のみ**で良いから、**コーディングは片方だけあれば十分**である。もう片方は
コードではなく「期待結果」を用意すればよい。この**コーディング不要なほうの活動を「テスト」と呼ぶ。**



テストでこの1%の誤りを見つける

テストの期待結果		開発成果物の実行結果	
		正99%	誤1%
	正 99%	98.01%	0.99%
	誤 1%	0.99%	0.01%

秋山浩一　仕事の基本-5.ソフトウェアテスト より引用

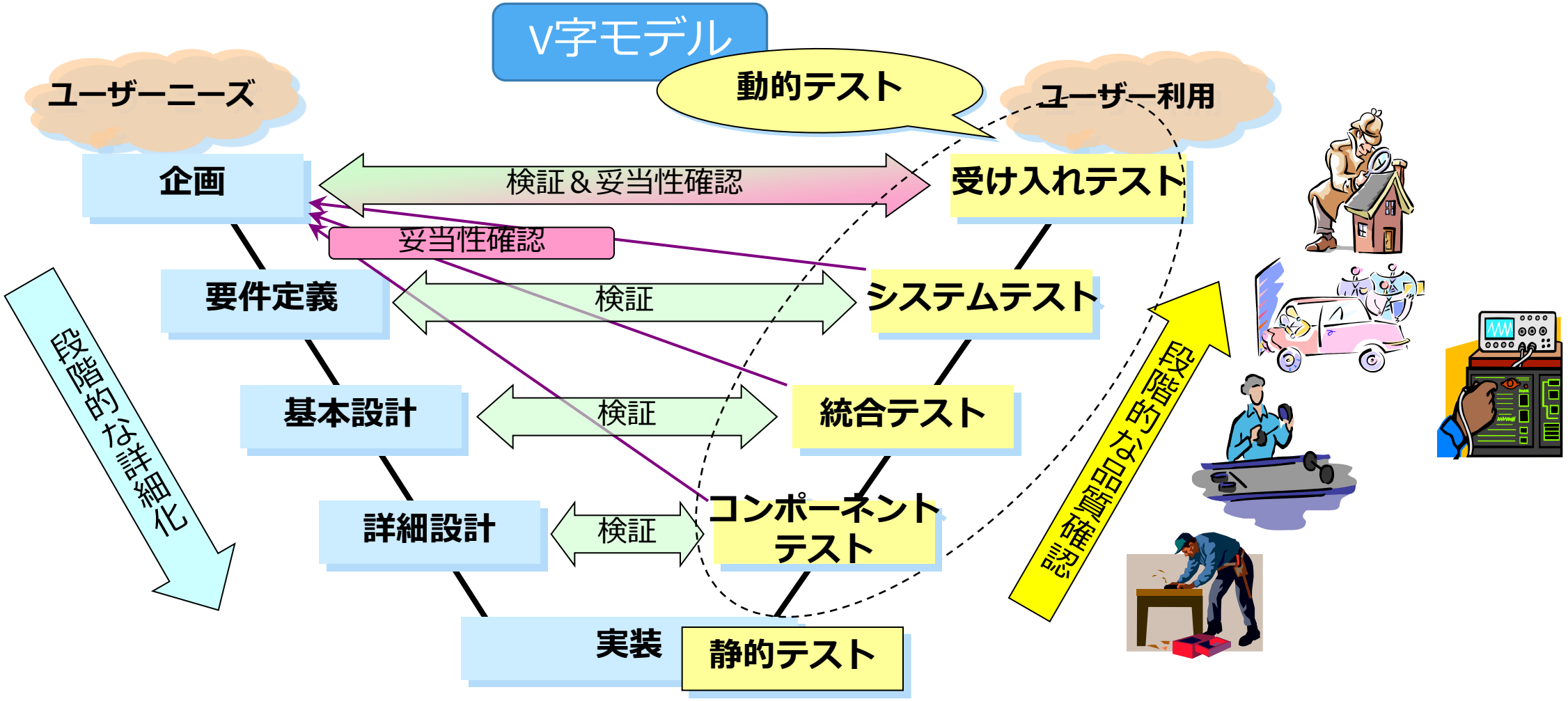
1件の取引を貸方と借方の両方に記録して突き合せる**複式簿記**と同じ

© YTTELAB LTD 2026

テスト全体像のための3つの重要な概念

© YTTELAB LTD 2026-

ソフトウェアテストといってもいろいろある

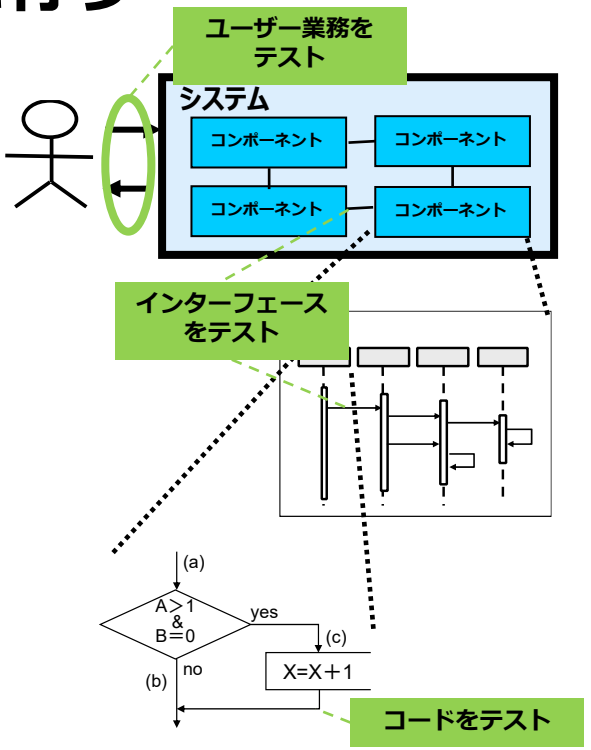


© YTTELAB LTD 2026-

テストレベル（段階的な品質確認）

テストは、テスト対象の詳細レベルに合わせて段階的に行うことが必要となる：これを<u>テストレベル</u>という

・コードをテストする(コンポーネントテスト)
・インターフェースをテストする（統合テスト）
・ユーザー業務が遂行できるかをテストする（システムテスト）



テストレベルを考慮することで<u>テスト対象を効果的、効率的にテストする</u>ことができる

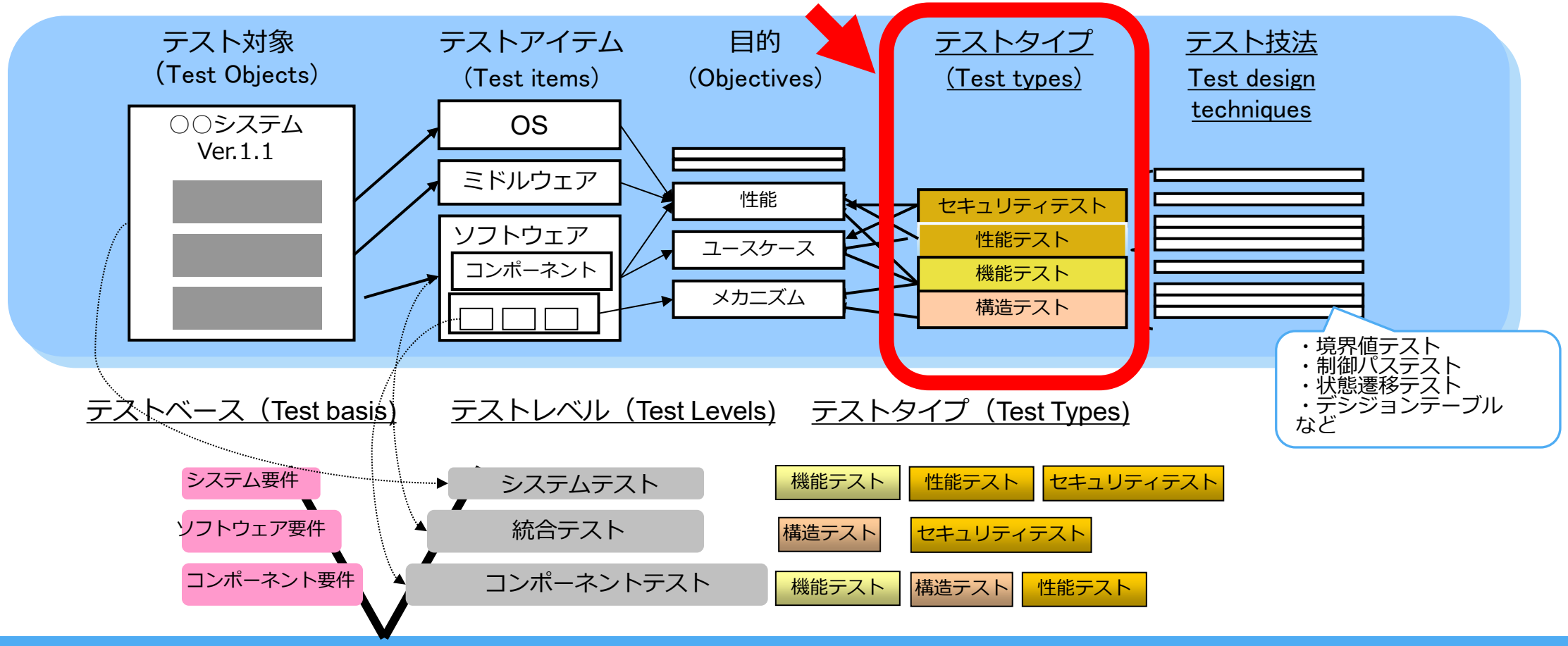
・<u>網羅するための母数を混ぜると危険</u>
- ユーザー業務が遂行できたかをテストする際に、コードのパスを母数にするとテスト量が膨大になる

・<u>テストの仕方が非効率になる</u>
- 開発環境でコードのif文通過のテストでユーザー業務遂行可能か判断するのは難しい
- 本番相当の大規模なテスト環境で、多くの人が関与してユーザー業務が遂行できたかをテストする際に、コードのif文通過を確認するテストを行うのは難しい

© YTTELAB LTD 2026-

テストタイプ

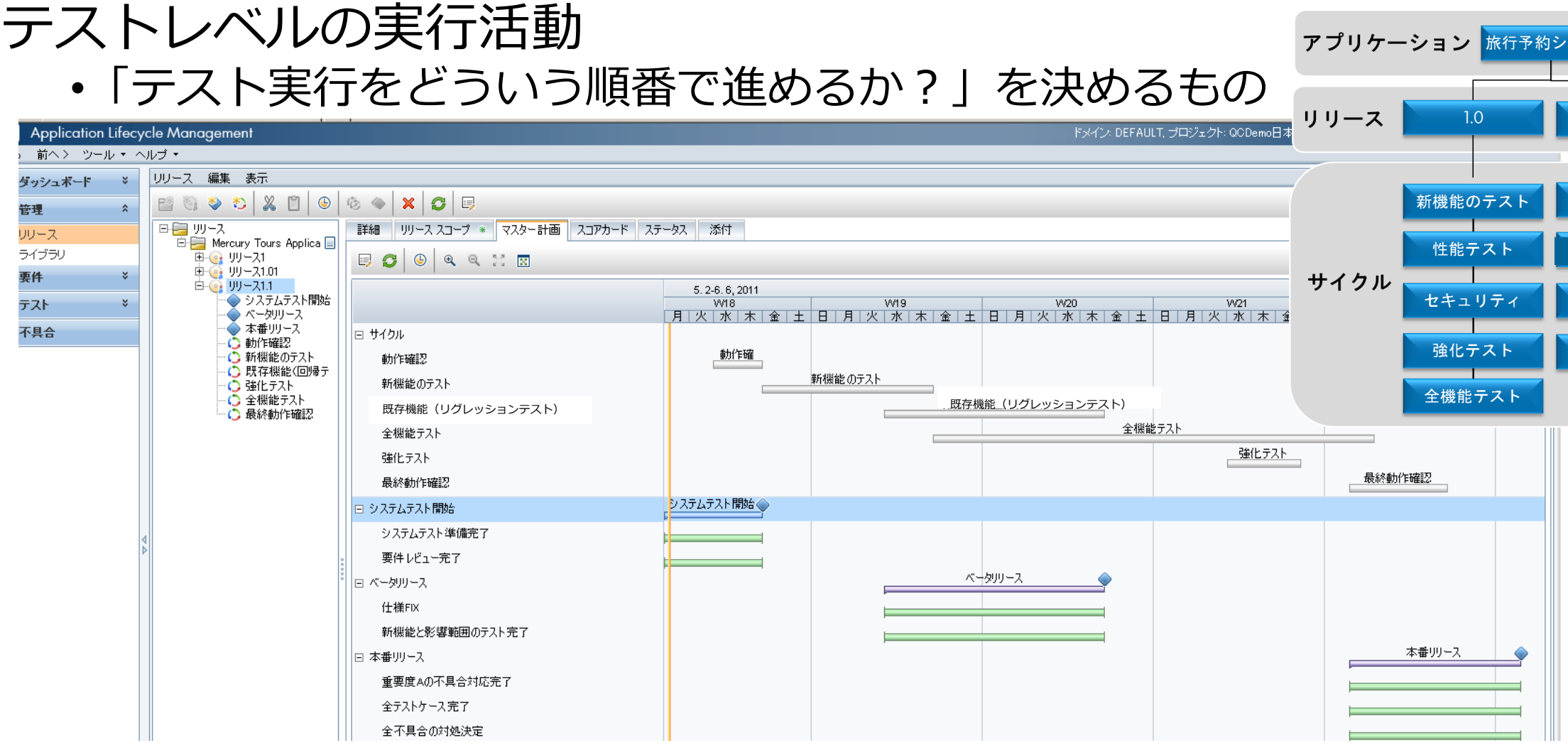
- 特定のテストの目的（機能、性能、セキュリティなど）にフォーカスしたテスト（テストケース群）の分類
  - 各テストタイプのテストは、一つもしくは複数のテストレベルで行われる



© YTTELAB LTD 2026-

テストフェーズ(テストサイクル)

- テスト活動をプロジェクト中でマネジメントしやすくまとめた、あるテストレベルの実行活動
  - 「テスト実行をどういう順番で進めるか？」を決めるもの

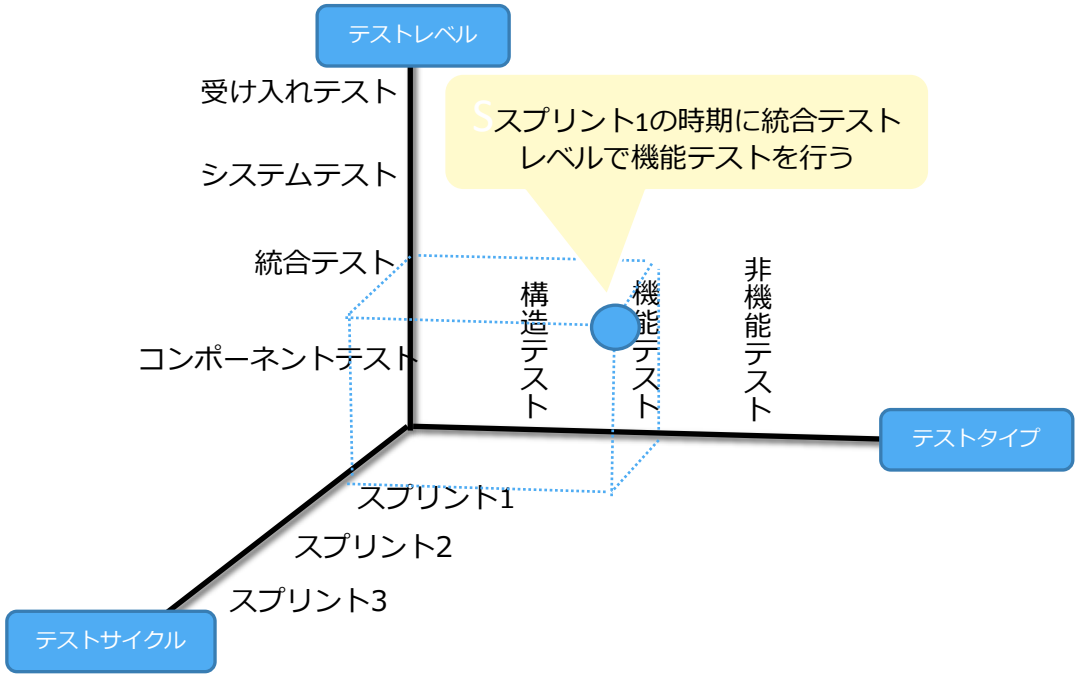


© YTTELAB LTD 2026-

[参考]テストアプローチを明確にする

各テストレベルをいつどのように行うか（テストアプローチ）を明確にする

開発プロセスに合わせて、テストレベル、テストタイプ、テストサイクルの組み合わせ、どのように確認結果を積み上げていくかを計画する



アプローチ立案のための３つの重要な概念

テストレベル	テストサイクル	テストタイプ
対象	時期	目的
・バイナリ/コード ・ドキュメント ・実行環境	・開始ポイント ・終了基準 ・プロジェクトリスク	・プロダクトリスク ・テスト設計技法 ・テストツール

© YTTELAB LTD 2026-

[参考]テストアプローチ策定

- テストアプローチ策定のための３つの重要な概念を組み合わせる

プロジェクトで定義したテストサイクル

プロジェクトで定義したテストレベル

- 複合システム
- 単一システム
- 部品の統合
- 単一部品

プロジェクトで定義したテストタイプ

- 使用量増加時の応答時間
- 複数利用者の同時利用
- 出力結果
- 分岐条件の動作結果

テストアプローチ

	サイクル名	テスト目的	テストレベル	テストタイプ
サイクル1	コンポーネントテスト	単一部品の仕様との合致	単一部品	構造テスト（分岐条件）
サイクル2	統合テスト	処理Aの仕様との合致	部品の統合	機能テスト（出力結果）
サイクル3	リグレッションテスト	既存機能への副作用がないこと	単一システム	機能テスト（出力結果）
サイクル4	システム統合テスト	連結して問題ないことを確認	複合システム	機能テスト（複数同時）
サイクル5	UAT	業務利用できることを確認	複合システム	性能テスト（応答時間）

© YTTELAB LTD 2026-

テストケースを作る過程を
再現可能にするために必要なのが
プロセス

© YTTELAB LTD 2026-

活動を具体的にあきらかにしてチームで仕事を分け合ってできるようにするのがプロセス

・テスト実行だけでは、何故その行為をするのか？が不明で目的達成ができない

© ytteLab Ltd 2026-

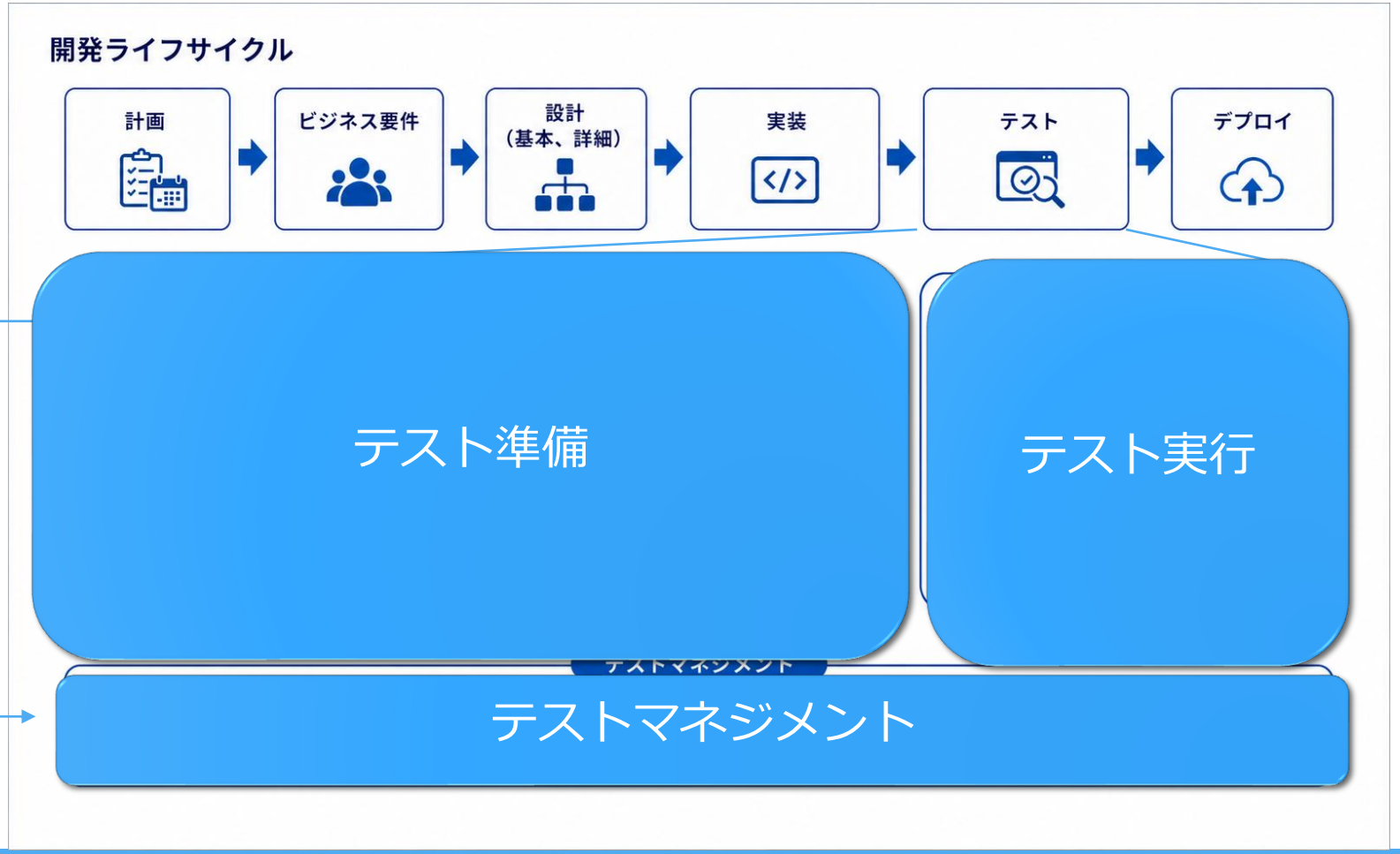
活動を具体的にあきらかにしてチームで仕事を分け合ってできるようにするのがプロセス

実行前に準備する（目的を明らかにし、具体的にしていく）
マネジメントする（うまくできるように仕掛けを入れ込む）

© ytteLab Ltd 2026-

テストプロセス

・事前に仕組みや仕掛けを入れ込んでおかないと、テスト実行でさえままならない
・活動をもっと明らかにすると、開発初期からテスト活動が必要だと認識できる



© YTTELAB LTD 2026-

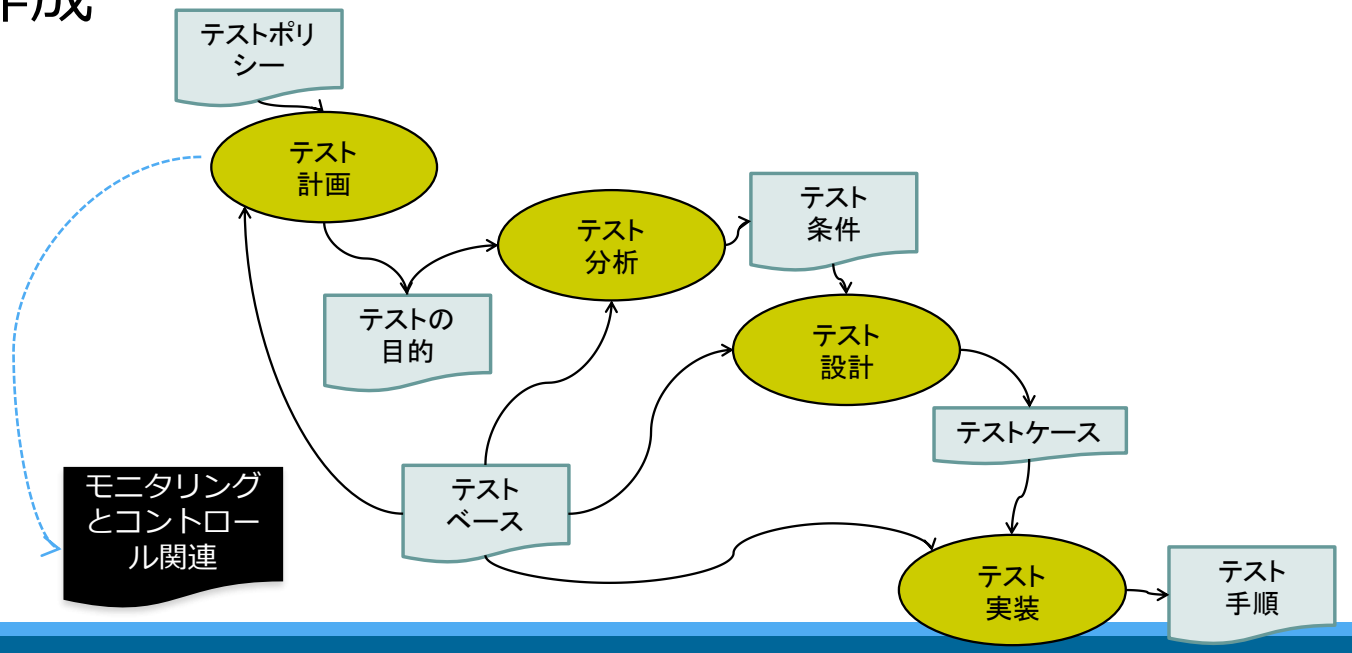
テストプロセス

- 事前に仕組みや仕掛けを入れ込んでおかないと、テスト実行でさえままならない
- 活動をもっと明らかにすると、開発初期からテスト活動が必要だと認識できる

© YTTELAB LTD 2026-

[参考]テスト開発プロセス（JSTQBの定義）

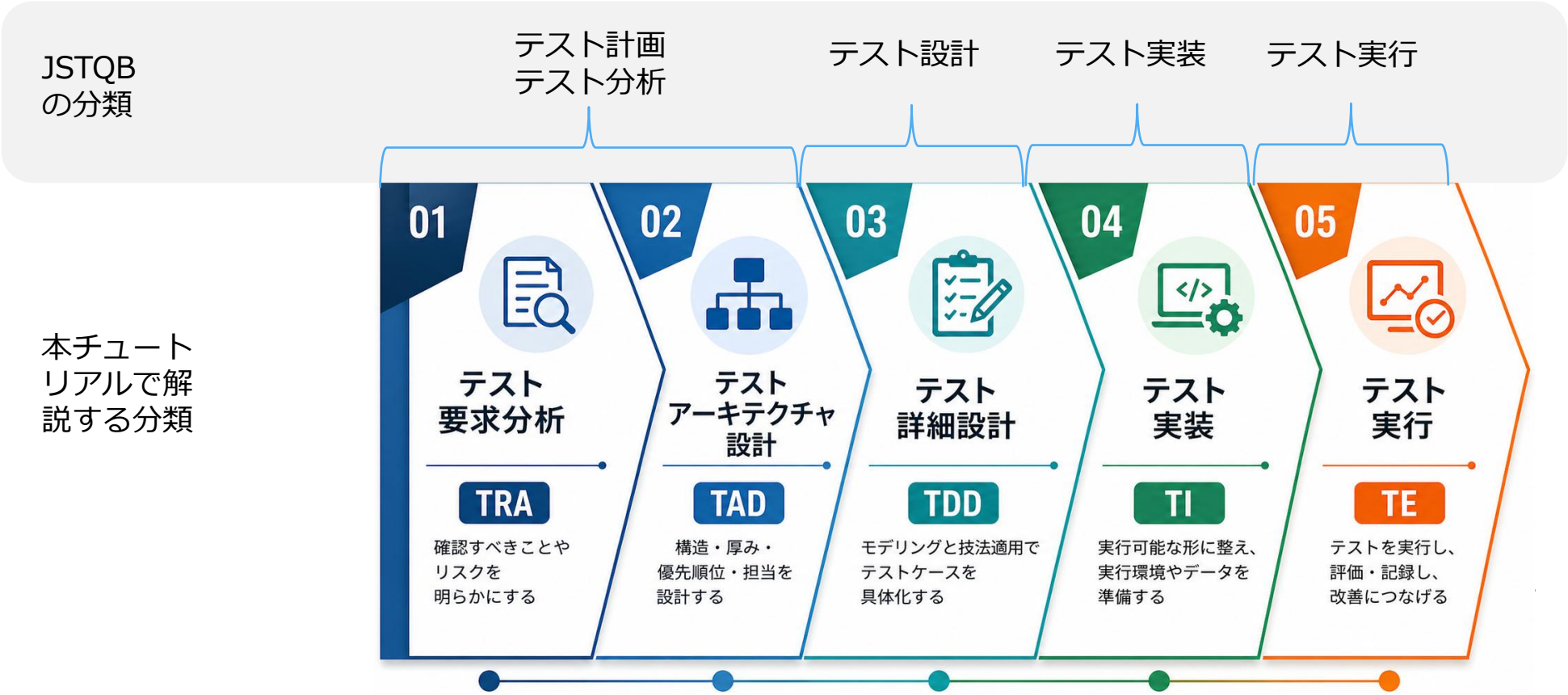
- テストの目的やテスト対象を明らかにする（計画する）
- テスト対象を理解するために分解、整理する（分析する）
- 目的に沿ったテストになるように作る（設計する）
- テストを実行できるようにする（実装する）
  - 手動であれば実行順序を作る、自動であればスクリプトを作る
  - 環境構築、テストデータ作成



© YTTELAB LTD 2026-

本チュートリアルで解説するテスト開発プロセス

・(JSTQBではテスト計画で行う）テストマネジメント向けの活動を省いている
・(JSTQBではあまり明確に語られていないが、）本当は重要な分析と設計の架け橋になる部分をテストアーキテクチャー設計として明示している

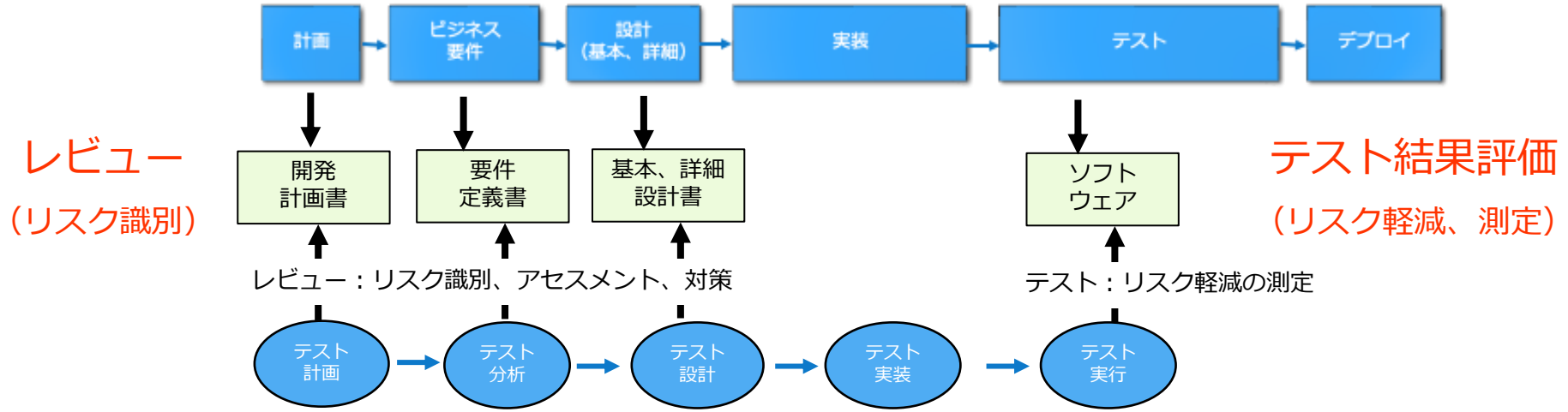


© YTTELAB LTD 2026-

テスト開発プロセスの中でテストを効果的にする

テスト開発プロセスで品質リスクを早期に見極める

•有限なリソースでより効果的にテストするためにはテストの優先度付けが必要（テストをリスクマネジメントの手段と位置づける）



「どれだけ多くテストするか」から「何をテストするか」へ
→テストへの適正な投資と最大限の効果をもたらす

© YTTELAB LTD 2026-

リスクとは？

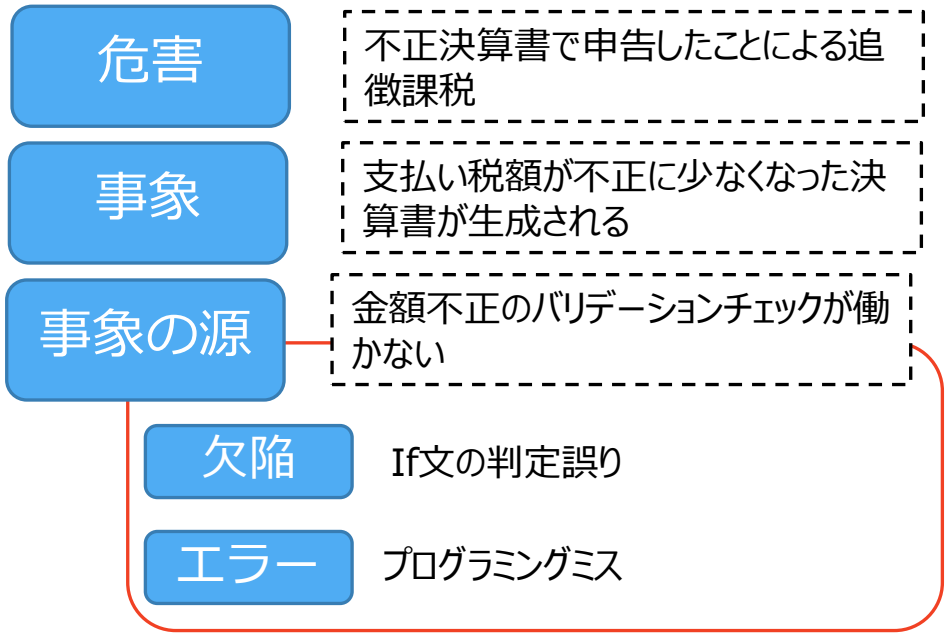
リスクとは、顧客、ユーザー、開発関係者にとって望ましくない結果をもたらす可能性（潜在的な問題）のこと

- プロダクトリスク (または**品質リスク**ともよぶ)
  - 潜在的な問題の主な影響がリリースした後の製品の品質に及ぶ場合を指す
    - リリース後の致命的バグで全品回収かもしれない
- プロジェクトリスク (または計画リスクともよぶ)
  - 潜在的な問題の主な影響がプロジェクト成功（コスト、納期）におよぶ場合を指す
    - 計画通りの日程で製品がリリースできないかもしれない

© YTTELAB LTD 2026-

ソフトウェアテストで扱う品質リスク

「危害」が起きる潜在的な問題をリスクとして扱う



リスクアイテム
・支払い税額不正の決算書生成

リスクアセスメント
危害がひどく、広範囲になる
×
危害が起きる可能性が高い
＝高リスク

リスクアセスメントの結果からテスト内容を決めていく方法を
リスクベースドテストと呼ぶ

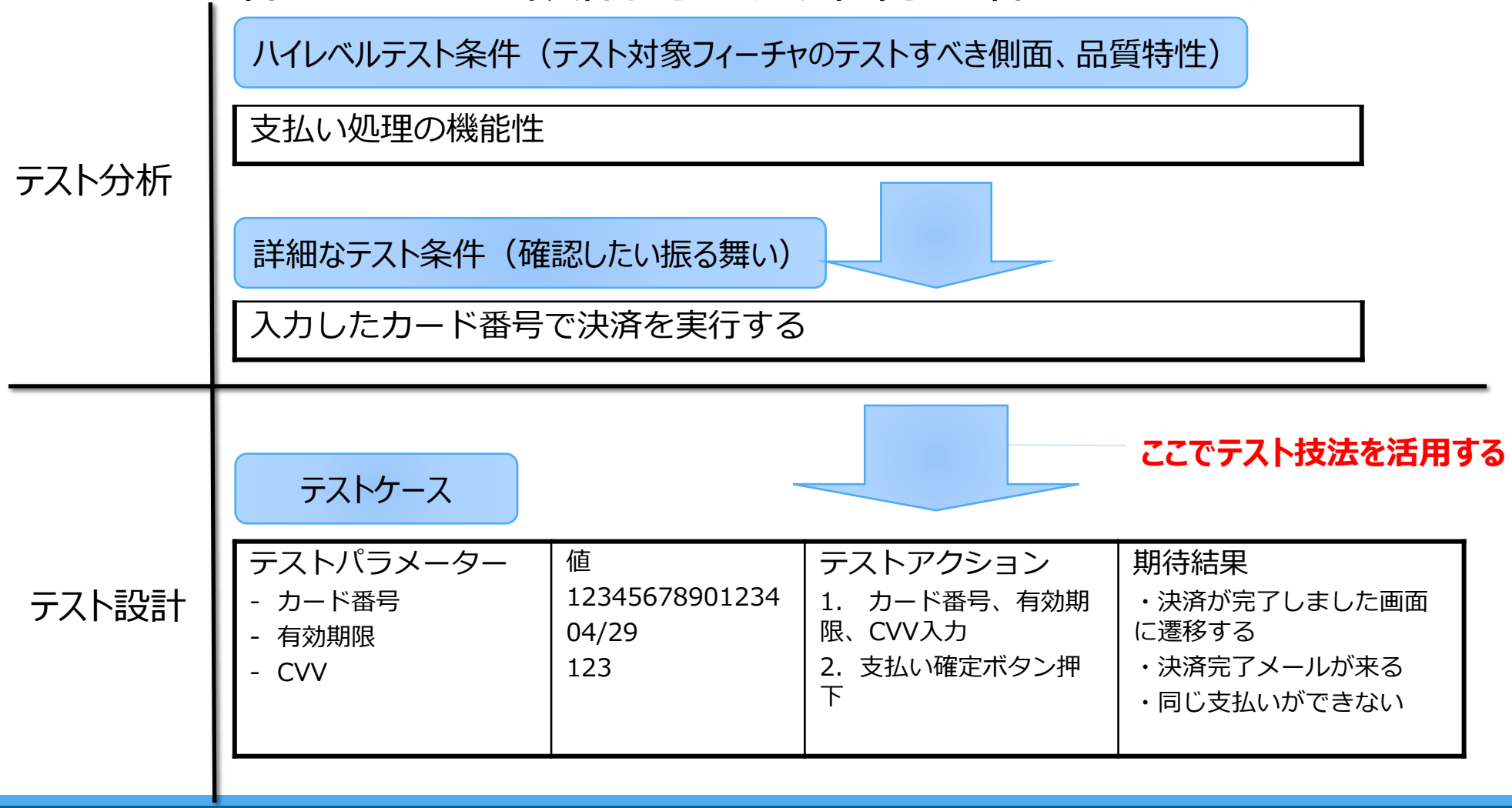
© YTTELAB LTD 2026-

テスト観点の注意点

© YTTELAB LTD 2026-

[参考]JSTQBでのテスト開発プロセスの中の成果物

- テストケースを作るまでに段階的に成果物を作っていく



© YTTELAB LTD 2026-

テスト観点という言葉の意味

何をテストするのか（テストケースの意図）のみを端的に示すもの

テスト観点の例

- 支払い処理の機能性
- 1桁短いカード番号を拒否する
- 期限切れのカード番号を拒否する
- 二重登録させない

カード番号桁数 **ボタン二度押し** **有効期限内と外**

JSTQBの用語

- **ハイレベルなテスト条件**
- **詳細なテスト条件（画面入力制御）**
- **詳細なテスト条件（内部判定ロジック）**
- **詳細なテスト条件（状態管理）**
- **テストパラメーター/テストカバレッジアイテム**

テストケースを書く前のアウトプットは**全部テスト観点**
テスト開発プロセスのどこでどう使うかが異なるので注意する

© YTTELAB LTD 2026-

テスト開発プロセス

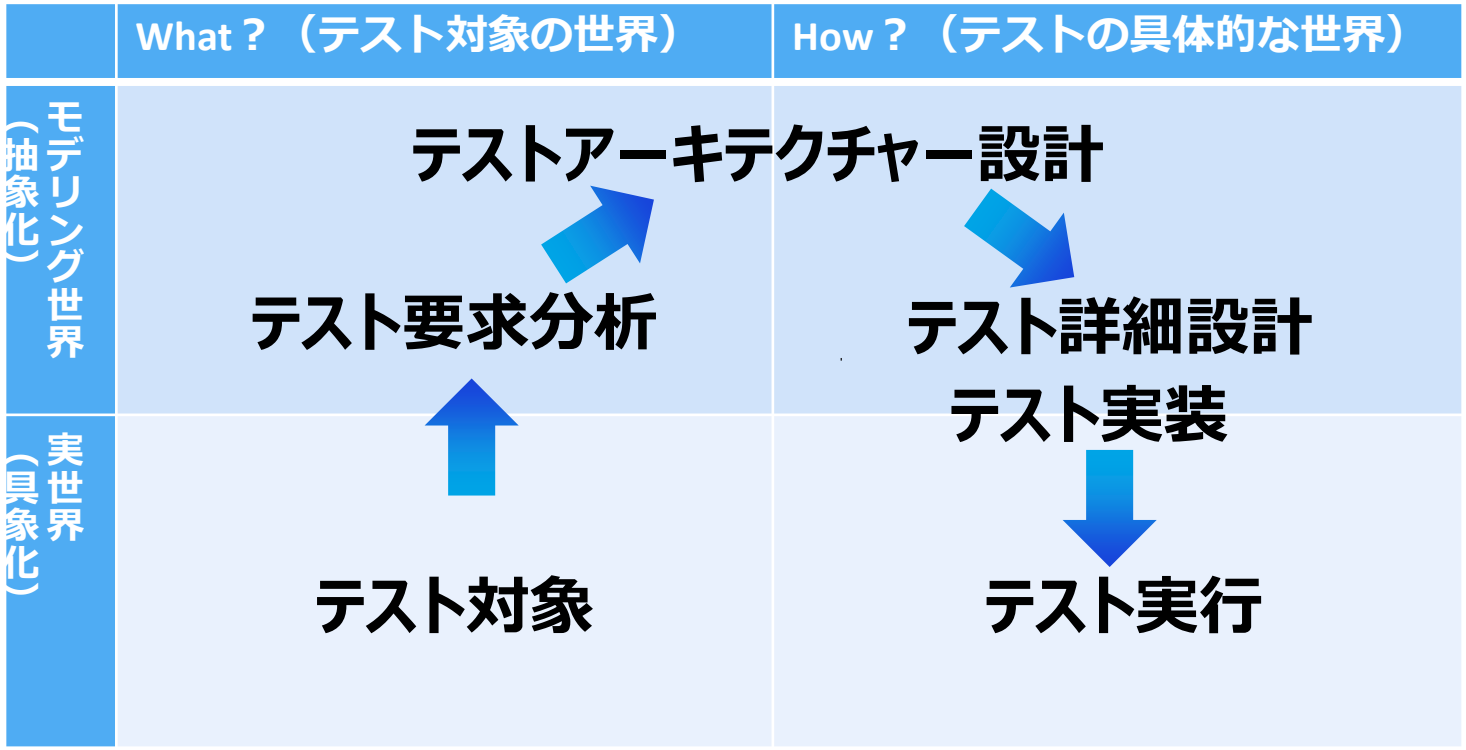
ここでの説明ではプロジェクトリスクやリソース確保、スケジュールなど、マネジメント制約については一切触れません。

あくまでテスト開発プロセスにフォーカスします。

© YTTELAB LTD 2026-

納得できるテストを作るためのテスト開発プロセス

・複雑なテスト対象を直接的にテスト実行に対応づけるのは非常に難しいので
　モデリングをすることが重要になる
「抽象化⇆具象化」という道具（つまり、モデル）をテスト開発に活用する



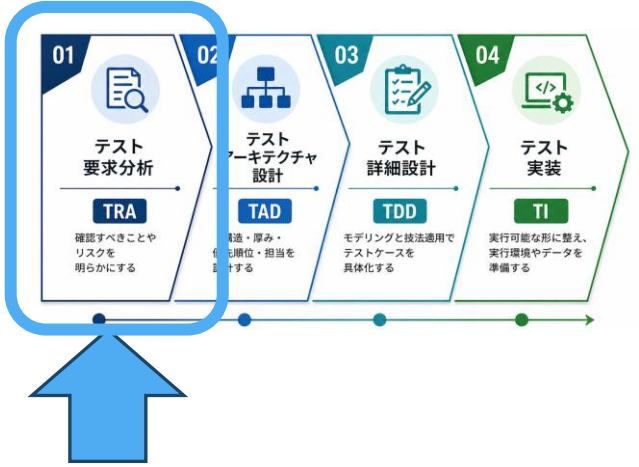
© YTTELAB LTD 2026-

テスト開発プロセスについての解説の注意点

テスト開発プロセスの具体的な活動をこれから解説していくが、順序やグルーピングは**流派によって異なる**ことがある。

- 本チュートリアルの解説はあくまで具体例の一つとして理解すること
- 大事なのは、テスト開発プロセスの**どこかで必ずやること**が何かを知ること
- 具体的な手法や技法は解説しないので、知りたい場合は、過去動画や過去作品を参照すること

© YTTELAB LTD 2026-

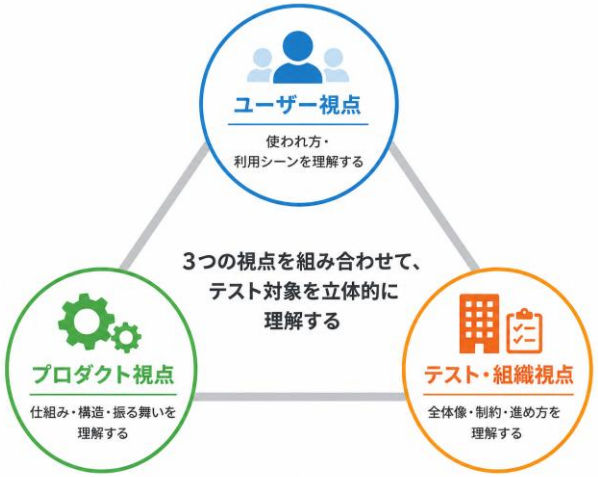


テスト要求分析（TRA）

© YTTELAB LTD 2026-

テスト要求分析（TRA）でやること

1. 3つの視点を組み合わせてテスト対象を理解する
    1. テスト対象がどう使われるか理解する **ユーザー視点**
    2. テスト対象そのものを理解する **プロダクト視点**
    3. テストの全体像を作る **テスト、組織視点**



2. プロダクトリスクを識別、評価する

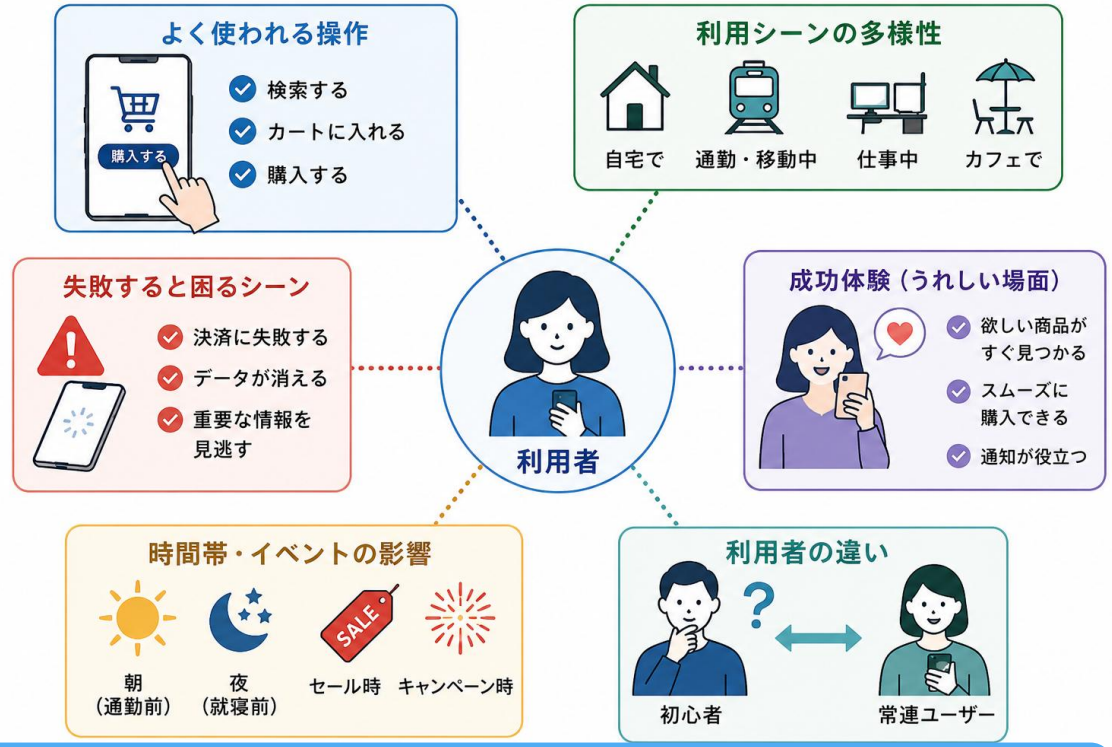
3. 詳細なテスト条件とテストパラメーターを識別、整理する

© YTTELAB LTD 2026-

TRA①-1テスト対象がどう使われるか理解する

仕様書に書かれたことだけでなく、利用者がどのような場面で、どのように使うかを理解する。使われ方がわかることで、重要な機能や優先して確認すべきことが見えてくる。

- よく使われる操作は何か
- どんな時に「これいいね」って思うか
- 失敗すると困る利用シーンは何か
- 時間帯やイベントで使われ方が変わるか
- 初心者と常連で使い方が違うか



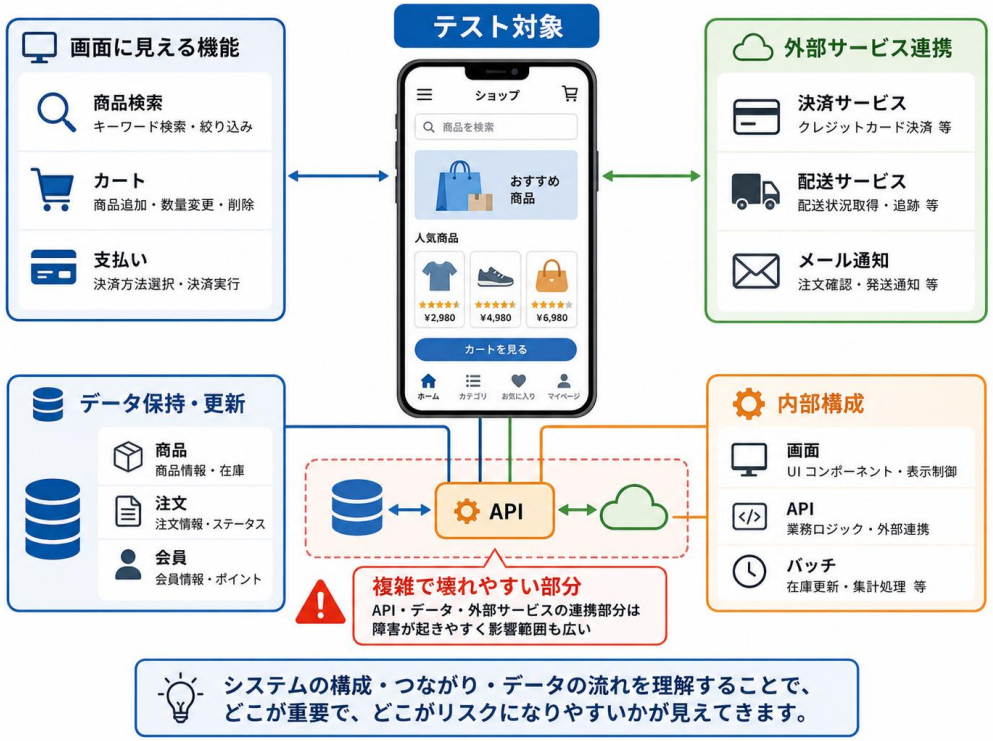
使われ方を知ることで、本当に必要なテストが見えてくる

© YTTELAB LTD 2026-

TRA①-2テスト対象そのものを理解する

使われ方だけでなく、テスト対象がどのような仕組みで動いているかを理解する。
画面に見える機能だけでなく、内部の構成（ソフトウェア設計上の責務）や外部サービスとの連携も把握することが重要。

- どの機能で構成されているか
- どこが他システムとつながっているか
- データはどこで保持・更新されるか
- 障害が起きやすそうな複雑な部分はどこか



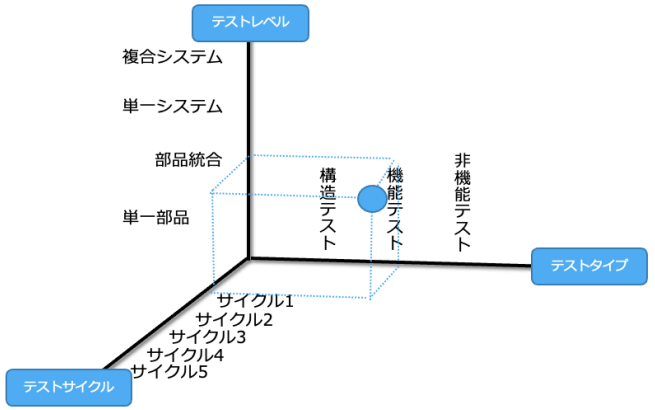
仕組みを知ると、壊れやすい場所や重要な確認点が見えてくる

© YTTELAB LTD 2026-

TRA①-3テストの全体像を整理する

詳細なテストケースを考え始める前に、「どこで」「何を」「いつ」確認するかの全体像を整理する。組織の状況に応じた**テスト方針**に基づいたテストアプローチで定めたスコープ内で確認が必要なハイレベルテスト条件を明らかにする。

- テストアプローチ
  - テストレベル：どの粒度で確認するか
    - 例：コンポーネント / 統合 / システム
  - テストタイプ：何を目的に確認するか
    - 例：機能性 / 性能 / セキュリティ
      →ハイレベルテスト条件**（フィーチャのテストすべき側面）**
  - テストサイクル：いつ、どのまとまりで進めるか
- 責務分担：誰がどこまで確認するか
  - 別テストレベル、別会社で開発、変更対象外など



	サイクル名	テスト目的	テストレベル	テストタイプ	担当
サイクル1	コンポーネント機能	単一部品の仕様との合致	単一部品	構造テスト（分岐条件）	開発
サイクル2	システム新機能	処理Aの仕様との合致	部品の統合	機能テスト（出力結果）	開発
サイクル3	リグレッションテスト	既存機能への副作用がないこと	単一システム	機能テスト（出力結果）	QA
サイクル4	システム全体	連結して問題ないことを確認	複合システム	機能テスト（複数同時）	QA
サイクル5	ユーザー操作	業務利用できることを確認	複合システム	性能テスト（応答時間）	PdM

組織の状況に応じた、テスト全体の地図を作ることが重要

© YTTELAB LTD 2026-

[参考]現代のソフトウェア開発で有効なテスト方針

- **段階的にテストを行う**
  - 開発プロジェクトの状況とマッチしたテストレベルを定義する
    - 各テストレベルのテスト結果を積み上げることで、システムの品質を確保する
- **複数のテストを最適化する**
  - e.g.テストベースレビューを行う（レビューも組み合わせてテスト量を決める）
  - e.g.ライフサイクル全体でテストに関わる関係者のテストから得られる情報をコーディネートする
    - 開発者自身で行うテスト/開発ベンダーとして行うテスト /エンドユーザーの立場で行うテスト

- **優先度の高い事柄から先にテストする**
  - 分析的：e.g. プロダクトリスク分析
  - 体系的：e.g. 品質特性などの体系的なモデルをベースに優先度の高い品質を分析
  - 動的かつ経験則的：e.g. テスト中に見つかった不具合の分析をベースに優先度を判断
- **リグレッションテストを戦略的に行う**
  - e.g.テスト自動化、影響度分析

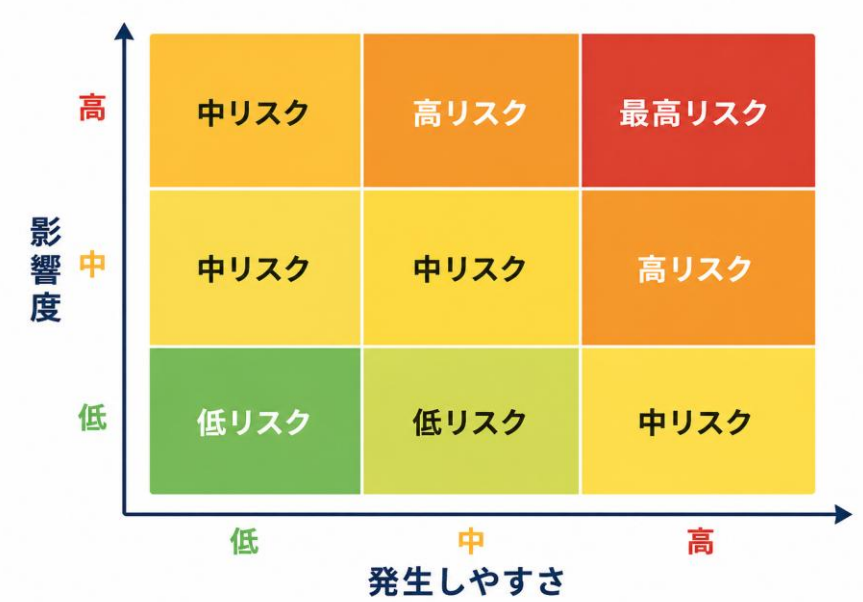
無駄に多くテストをしないように全体像を作るための大前提

© YTTELAB LTD 2026-

TRA②プロダクトリスクの識別と評価をする

3つの視点を組み合わせて、品質上の問題が起きると利用者の作業や業務にどのような影響があるか、その問題が起こりやすいかどうかを考える。考えた結果を踏まえて、テストで優先して確認すべき対象を明らかにする。

- プロダクトリスクの見方
  - 影響度
    - 最も重要な機能か、よく使われる機能か(ユーザーにとって大事か)
  - 発生しやすさ
    - 今回の変更が大きいか、仕様が複雑か(バグが出そうか)
- リスク評価の使い方
  - テストの厚みを決める
  - テスト実行の順序を決める



危ないところから先に、厚く見ることが重要

© YTTELAB LTD 2026-

TRA③詳細なテスト条件とテストパラメーターを識別し、整理する

ハイレベルテスト条件だけでは、具体的なテストケースを設計できない。確認したいことを、詳細なテスト条件に分解し、テストアーキテクチャー設計やテスト詳細設計のインプットにする。

- 詳細なテスト条件とテストパラメーターの識別
  - テスト条件（確認したい振る舞い）
    - 事前条件や入力値に加え、期待結果、事後状態がわかる
      「1桁短いカード番号を拒否する」

アーキテクチャー設計時の要素に割り付ける

  - テストパラメーター（振る舞いを変化させる要素）
    - 事前条件、入力値がわかる
      「カード番号桁数」

テスト詳細設計にて最適になるよう工夫する

支払い処理の機能性

テスト条件

・1桁短いカード番号を拒否する
・期限切れのカード番号を拒否する
・二重登録させない

テストパラメーター

・カード番号桁数
・有効期限
・ブランド

現時点に理解できていることを書くにとどめる。不明点は確認後にTDDで追加できるので今は気にしない。

詳細に識別することが、仕様の不備や不明点の気づきになる

© YTTELAB LTD 2026-

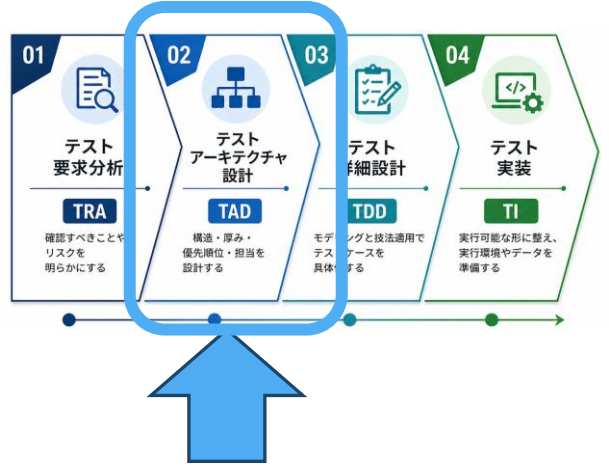
[参考]テスト条件=テスト分析のアウトプット

テストベースから見つけた「テストしたほうがよいこと」

- テストレベルがコンポーネントテストであれば…
  - テストベース=ソースコード
  - テスト条件=判定部分
- テストレベルがシステムテストであれば…
  - テストベース=要件が書かれたドキュメント
  - テスト条件=　テストすべき要件/仕様
    - （～な場合に○○をすると××となること）
- テスト条件にはテストケースの要素が含まれている
  - テスト条件に対して一つ以上のテストケースが作られる=テスト設計

テストベース
コンポーネント要件やシステム要件を推測できる全てのドキュメントであり、これらのドキュメントがテストケースのベースとなる

© YTTELAB LTD 2026-



テストアーキテクチャー設計（TAD）

© YTTELAB LTD 2026-

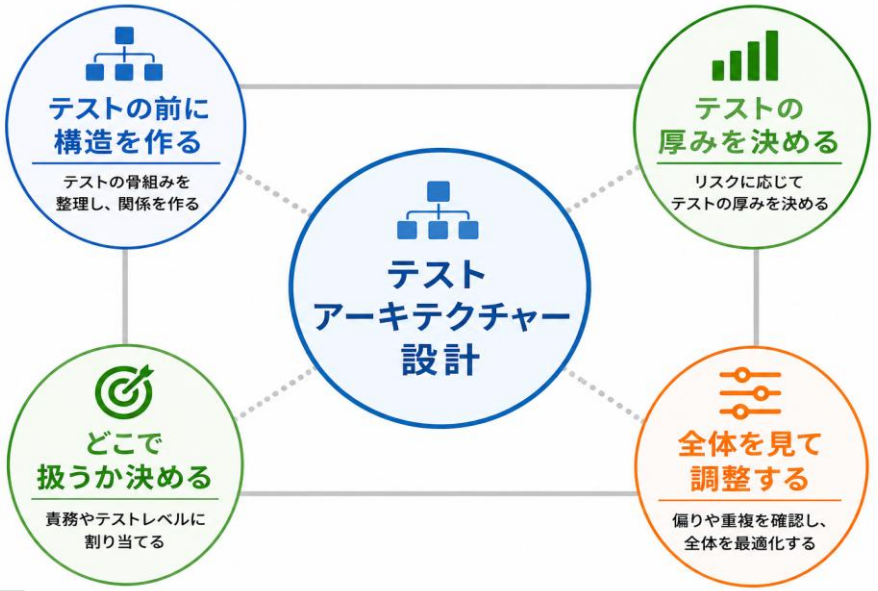
テストアーキテクチャー設計でやること

1. 要素の関係を明らかにする

2. リスクや責務分担で要素の厚み
   やバランスを明らかにする

3. 「詳細なテスト条件」の担当を決める

4. テスト全体の厚みやバランスでテスト条件を調整する

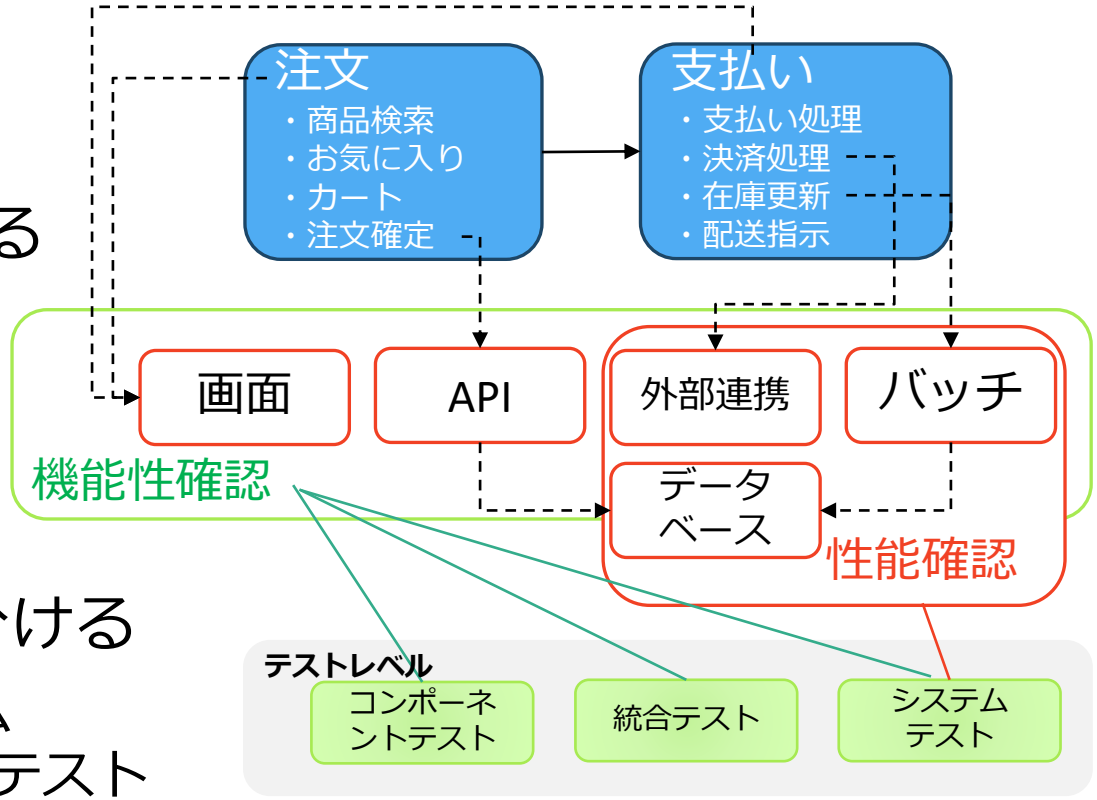


© YTTELAB LTD 2026-

TAD①要素の関係を明らかにする

テストはバラバラのケース集合ではなく、構造が必要。テストケースを個別に考え始める前に、テスト全体を意味のある単位の要素でモデリングする。

- 機能ごと要素に分ける
  - 注文 / 支払い / 配送 / 会員管理 /お気に入り
- 品質特性（テストタイプ）ごと要素に分ける
  - 機能性 / 性能 / セキュリティ / ユーザビリティ
- ソフトウェア設計の責務ごと要素に分ける
  - 画面 / API / バッチ / 外部連携/ データベース
- テストレベル/テストサイクルごと要素に分ける
  - テストレベル：コンポーネント / 統合 / システム
  - テストサイクル：新機能テスト/ リグレッションテスト



まずは整理棚を作る（どこに置き、どうつながるかを決める）

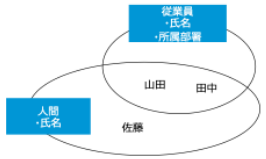
© YTTELAB LTD 2026-

[参考]モデリングで行うこと

関係を整理する（構造化）

（更に）特に大事な2つのことに特化して説明する

グループにまとめる



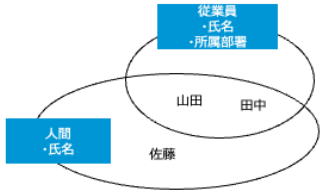
グループを更にまとめるグループを作る（階層化）

グループにまとめる

本質的で共通の特徴を抽出してグループにする

特徴によって同じ要素（人やモノ）でも別のグループになることがある

- 例：山田太郎さんと田中華子さん、そして佐藤一郎さんは「人間」=3人が同じグループ
  - 氏名で識別できる
- 例：山田太郎さんと田中華子さんは同じ会社の社員なので「従業員」=2人が同じグループ
  - 氏名と所属部署で識別できる



グループ化する共通の特徴の数が多いほど、属するものの数が減る
共通の特徴の数は「知りたいこと」を基準に増減させる

グループを更にまとめるグループを作る（階層化）

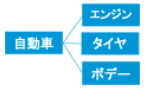
階層関係を明らかにしていくのは2つの考え方がある

共通しているものから、特に意識したい特徴を抽出して分けていく

- 共通の特徴を発見したら、共通の特徴でグループ化する
  - 従業員は、若手、中堅、ベテラン社員に分けられる
  - 他の例：取扱商品には、食品、書籍、家具がある

構成要素で分けていく

- 各要素が何かを形成している一部であればグループ化する
  - 従業員は、組織構成として平社員、中間管理職、役員に分けられる
  - 他の例：自動車の部品には、エンジン、タイヤ、ボデーがある



グループをまとめるグループによって簡潔に説明ができるようになる

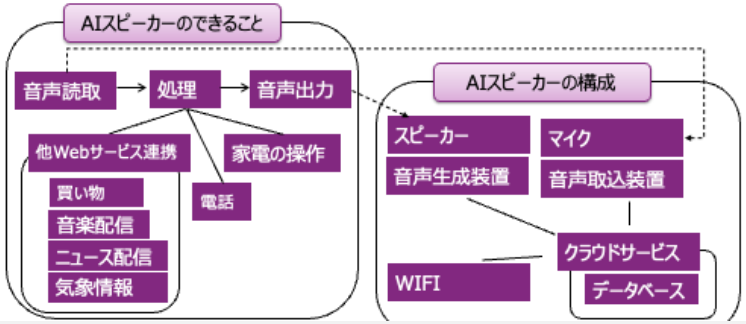
関係するものを繋ぐ

内容の整理を更にする

- 配置を見直したり、グループを作ったりする

関係を考える

- 利用するものに「⇢」をひく、処理順序に「⟶」をひく
- とりあえず関係ありそうな場合は線だけ引いておき、後で意味を追記する



「納得できるテストをするための
モデリング入門」から引用（5/14公開）
https://note.com/yumotsuyo/m/m517f13fe4101

© YTTELAB LTD 2026-

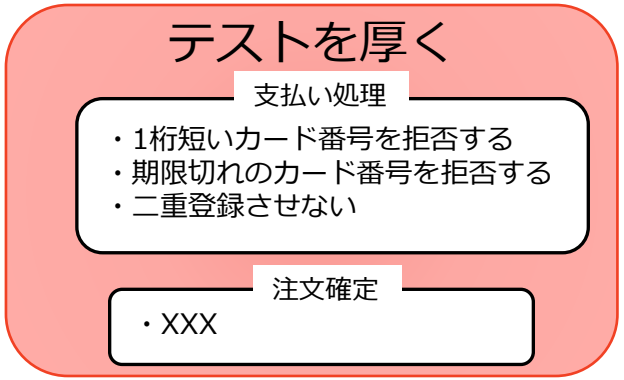
どういう切り口で、どのようなグループにして関係付けるかは、設計者の工夫次第

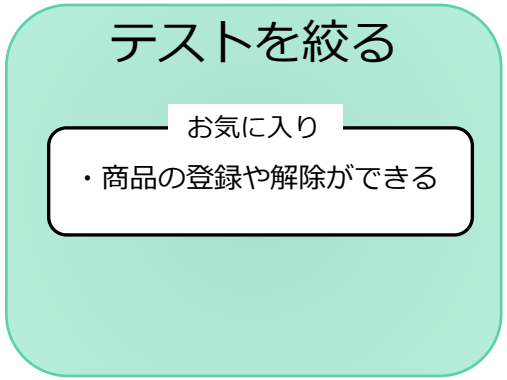
© YTTELAB LTD 2026-

TAD②厚みやバランスを決める

整理棚で分けたものを、全部同じ深さでテストする必要はない。限られた時間と人員の中では、すべてを同じ量だけテストすることはできないので、重要な部分は厚く、影響の小さい部分は絞る。

- 重要機能にパワーを集中する
- 全体として漏れを防ぐ
- 過剰品質を避ける
  - そのためには識別したリスクを考慮する
    - ユーザーへ提供する価値を損なうか？
    - バグが出そうか？





機能	リスク	テストの厚み
支払い処理	高	厚く
注文確定	高	厚く
商品検索	中	標準
お気に入り	低	絞る

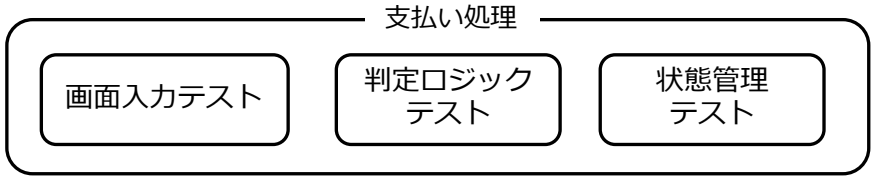
すべて同じテスト量を行うのではなく、適切に配分すること

© YTTELAB LTD 2026-

TAD③「詳細なテスト条件」の担当を決める

テスト要求分析で洗い出した「詳細なテスト条件」を、テストアーキテクチャーのどこで確認するか（＝担当グループ）を明らかにする。これにより、漏れや重複を防ぎ、各テストの役割を明確にできる。

- 同じテスト条件を複数箇所で見ない
- 誰も見ていないテスト条件をなくす
- 扱う場所（担当グループ）を決めると
  - 分担しやすくなる
  - トレーサビリティが作りやすくなる



テスト条件	担当グループ	機能	リスク	厚み
16桁超を拒否 →	画面入力テスト	支払い処理	高	厚く
14-16桁はOK→				
14桁未満拒否→				
期限内はOK→	判定ロジックテスト			
期限切れを拒否 →				
二重押下防止 →	状態管理テスト			

テスト条件を「どこで扱うか」決めて初めて設計できる

© YTTELAB LTD 2026-

どういう切り口で、どのようなグループにして関係付けるかは、設計者の工夫次第

© YTTELAB LTD 2026-

TAD④割り当てたテスト条件を調整する

詳細なテスト条件の担当グループを決めた後も、テスト全体の厚みやバランスを見ながら、テスト条件の確認の深さや粒度を調整してTDDにインプットする。

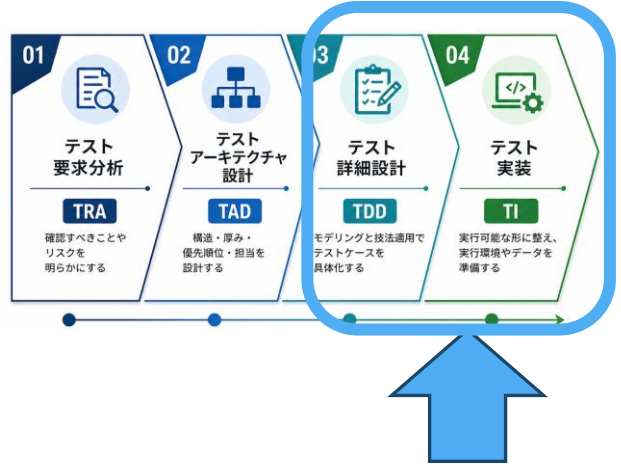
- 顧客に与える悪影響
- 他機能に与える影響
- 利用頻度
- コードベース複雑度合い
- 開発変更量
- 責務分担

調整内容がTDDでのカバレッジアイテム、つまり何をどこまで網羅するかを決めるための方針となる。

テストの調整内容	理由	テスト条件	
境界値まで詳細確認	決済ができないと目的が果たせない。なのでユーザー影響が大きい	カード番号桁数判定（入力したカード番号で決済を実行する）	16桁超を拒否→
			14桁はOK→
			16桁はOK→
			14桁未満拒否→
代表条件のみ確認	ユーザー影響が比較的小さい	商品検索（検索条件に合致した商品を検索する）	Or 条件一致→
			Or 条件不一致→
API単体テストへ委譲 代表条件のみ確認	コード変更の影響なし、かつ下位レベルのテストで担保可能	販売時税計算（明細に合わせた税区分を自動判定する）	食品の場合8%→
			預かり金は0%→

TRAでの深い理解によって、適切な調整ができる

© YTTELAB LTD 2026-



テスト詳細設計、実装（TDD、TI）

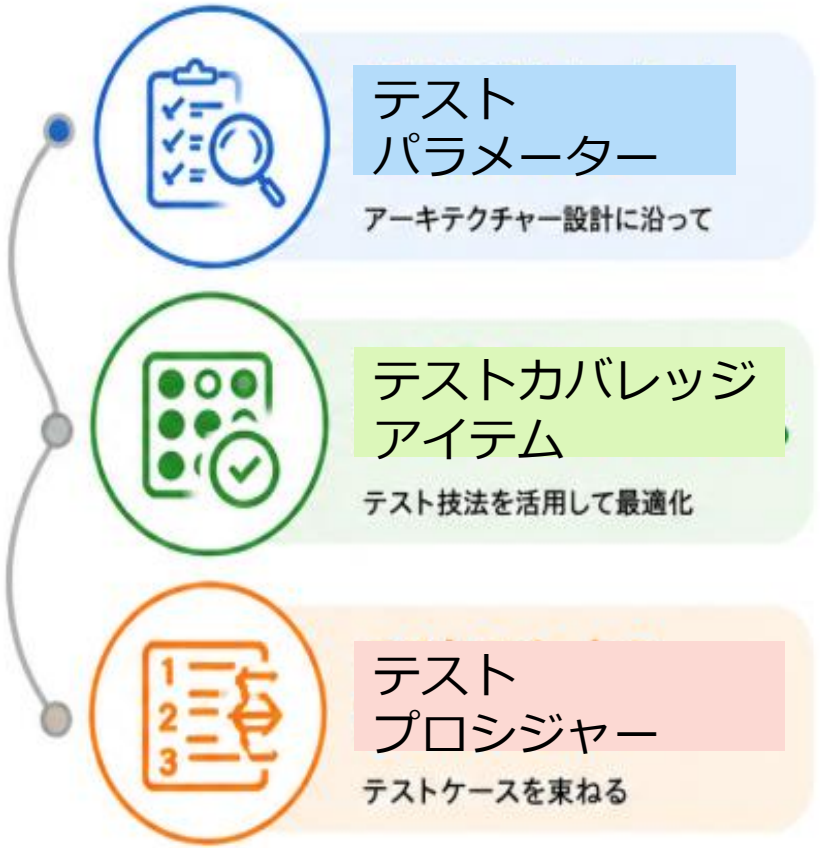
© YTTELAB LTD 2026-

テスト詳細設計、テスト実装でやること

1. テストアーキテクチャー設計に沿って
   テストパラメーターと値を仮決めする

2. パターンに抜け漏れがなく、数が爆発
   しないようカバレッジアイテムを決める
   （そのためにテスト技法を活用する）

3. テストケースを束ねたテストスイート
   の実行順序を作る



© YTTELAB LTD 2026-

TDD ①パラメーターと値を仮決めする

テスト条件を十分に確認するために、**テスト要求分析をベース**に入力値、状態、環境など、変化させる要素（パラメーター）と確認に使う具体的な値候補を選ぶ。

- その際、**テストアーキテクチャー設計で決めた厚みや優先度** に沿って、既出のパラメーターや値候補に対して、追加や修正をする。
  - 重点的に見る場所は厚め

機能	詳細なテスト条件	テストパラメーター	値候補
支払い処理	入力したカード番号で決済を実行する （1桁短いカード番号を拒否する）	桁数 有効期限 ブランド 操作繰り返し	14 / 15 / 16 / 17 有効 / 当月 / 期限切れ VISA / Master / JCB / AMEX/Diners 1回（なし）/ 2回/ 3回

  - そうでない場所は絞る

機能	詳細なテスト条件	テストパラメーター	値候補
お気に入り登録	商品の登録や解除ができる	操作	登録/ 解除

仕様に書いてなくてもバグが出そうなパラメーターを必要に応じて追加する

何を変えて試すかを決めることが詳細設計の出発点

© YTTELAB LTD 2026-

TDD ②パターンの組み合わせを賢く絞る

複数のパラメーターをそのまま組み合わせると、テストケース数は急激に増える。

- テスト技法を使うなどして必要な組み合わせをモデリングして、テストカバレッジアイテムとして決定する。
  - テストカバレッジアイテム（一部抜粋）

機能	詳細なテスト条件	テストカバレッジアイテム	期待結果
支払い処理	入力したカード番号で決済を実行する（1桁短いカード番号を拒否する）	桁数:13,有効期限:有効,ブランド:Diners,操作:1回	1桁短いカード番号拒否
		桁数:14,有効期限:有効,ブランド:AMEX,操作:3回	1桁短いカード番号拒否
		桁数:15,有効期限:有効,ブランド:VISA,操作:2回	1桁短いカード番号拒否
		桁数:16,有効期限:有効,ブランド:JCB,操作:1回	決済成功

  - 単純に組み合わせると4*3*5*3=180通り
    - 技法を使うと…　ペアワイズテストで20通り(制約を考慮するとこうはならないが単純に技法を適用した場合)
    - 現実的には、仕様やソフトウェア構造を理解して組み合わせて、カバレッジアイテムを適切な数にする
      - Dinersは14桁、AMEXは15桁、その他は16桁なので、値は13を追加しないといけない
      - 桁数判定だけを目的にする場合、有効期限は「有効」なものだけで固定して組み合わせない

多く作るより、必要十分に絞ることが重要

© YTTELAB LTD 2026-

[参考]テストケース=詳細設計のアウトプット

テスト条件をカバーする「ケース（場合）」

- テスト条件にはテストケースの要素が含まれている
  - テストケースになるようテスト条件を整理する
  - パラメータの組み合わせパターン分のケースが作れる
- テスト条件に対して一つ以上のテストケースが作られる=テスト設計
  - より少ないケース数でテスト条件を網羅
  - パラメーターの選択とパラメーターの組み合わせを工夫する
  - テストケースでは、工夫して明らかになったカバレッジアイテムを網羅する
- 先人の知恵が詰まった工夫のベストプラクティス=**テスト技法**

テストケースの構成要素
・パラメーター（～な場合に）
・アクション（○○をすると）
・期待結果（××となる）

なんでもかんでもやらない
視点を絞る

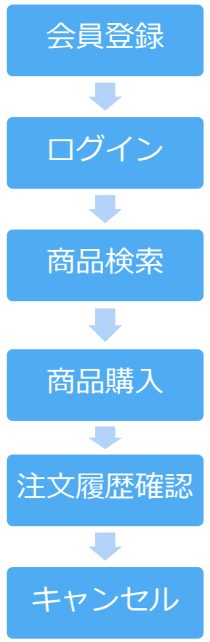
© YTTELAB LTD 2026-

TI ①実行しやすい順序に並べる

テストケースを実行効率や前後関係を考えて並べ、テストスイートや手順（テストプロシジャー）としてまとめる。実行しやすい形に実装できると、テストの価値はさらに高まる。

- 並べる時の観点
  - 前提条件を満たす順番か
  - 同じ準備データでまとめて実行できるか
  - 環境切替を減らせるか
  - 重要なケースを先に実行できるか
  - 異常終了時に影響が少ない順か

テスト実装モデリングの例



NGな例
- 購入前に注文履歴確認
- ログアウト後に購入
- 毎回環境初期化が必要な順番

手動テストでは手順書、自動テストでは実行シナリオとなる

© YTTELAB LTD 2026-

その他考慮してほしいこと

これから解説することは、テスト設計コンテストで考慮してほしいことです。

しかし、現場でテスト設計をする際にも、とても大事なことです！

© YTTELAB LTD 2026-

文書点

優れた内容でも、読みづらい資料では相手に伝わらない。テストは一人でやるものではないので、成果物は、読み手が理解しやすい文書であることが重要。

- 観点
  - 文書体系を示すこと
  - 情報量をコントロールすること
  - 見出しや表現が統一されていること
  - 更新しやすい構成であること
  - レビューしやすいこと
    - （文字がツールで読み取れるのは大事！）

NGな例
- 情報が1ページに詰め込みすぎ
- 用語がページごとにぶれる
- どこに何が書いてあるかわからない

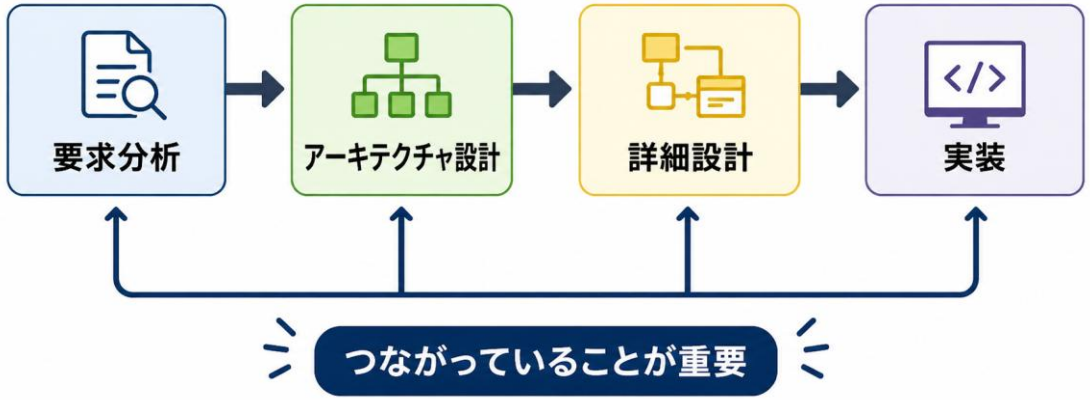
良い成果物は、読む人への配慮をした設計がされている

© YTTELAB LTD 2026-

工程一貫性点

各活動の成果物がつながっていることが重要。前の活動で考えたことが、次で活かされてこそプロセスとなり、再現性のあるテスト設計になる。

- 観点
  - 前の活動の成果物が次の成果物で使われていることがわかる
  - 一つの成果物だけが肥大化しない
  - トレーサビリティ番号が一意である



NGな例
- リスク分析の結果がアーキテクチャー設計に反映されない
- アーキテクチャー設計の意図がテストケースでは無視されている

良い成果物は、単一の活動ではなく全体で評価される

© YTTELAB LTD 2026-

総合点

基本ができたうえで、さらにテスト設計の価値を高めるのが独自性、新規性、技術的工夫である。奇抜さだけではなく、基本の品質があってこそ価値がある。

- 観点
  - 新規性
    - 既存の手法（例：VSTeP、ゆもつよメソッド など）と別の新しい考え方や表現がある
  - 独自性
    - 他のやりかたにはない、自分たちらしい切り口や構造化がある
  - 技術レベル
    - 既存でも新規でもよいが、モデリング、手法、技法適用などが適切に行われている
    - AIエージェントの活用はOKだが、説明責任を果たせないアウトプットは技術レベルが高いと言えない
  - 発想力
    - 課題に対して柔軟で面白い着眼点やアプローチがある（テスト設計では実績がないAI技術を使うなど）

基本を押さえた上での光る工夫が総合点につながる

© YTTELAB LTD 2026-

本日のまとめ

- テストの役割と全体像
  - テストは「大丈夫か？」を確認するエンジニアリング活動である
  - テストレベル、テストタイプ、テストサイクルで全体像を捉える
  - 無駄に多くテストしないために、テストの全体像を作るようにしよう
- テスト開発プロセス
  - テスト要求分析で、確認すべきこととプロダクトリスクを明らかにする
  - テストアーキテクチャー設計で、構造、厚み、役割分担を決める
  - テスト詳細設計と実装で、実行可能なテストケースと手順にする
- その他考慮して欲しいこと
  - 読み手が理解しやすい文書にする
  - 各活動の成果物をつなげ、一貫性を持たせる
  - 基本を押さえた上で、光る工夫を加える

© YTTELAB LTD 2026-

最後に

- テスト設計の技術力とは、**思いついた観点**を並べることではない!
- 本当に問われるのは、整理し、構造化し、的を射たテストを作る力!

- そこで思いついたテスト観点は…
  - テスト条件なのか？
  - テストカバレッジアイテムなのか？
  - リスクなのか？
  - モデルの要素なのか？
  - テストケースの実行順序なのか？

テスト設計コンテストを活用して
技術力を高め合っていきましょう！

© YTTELAB LTD 2026-

ありがとう
ございました！

© YTTELAB LTD 2026-