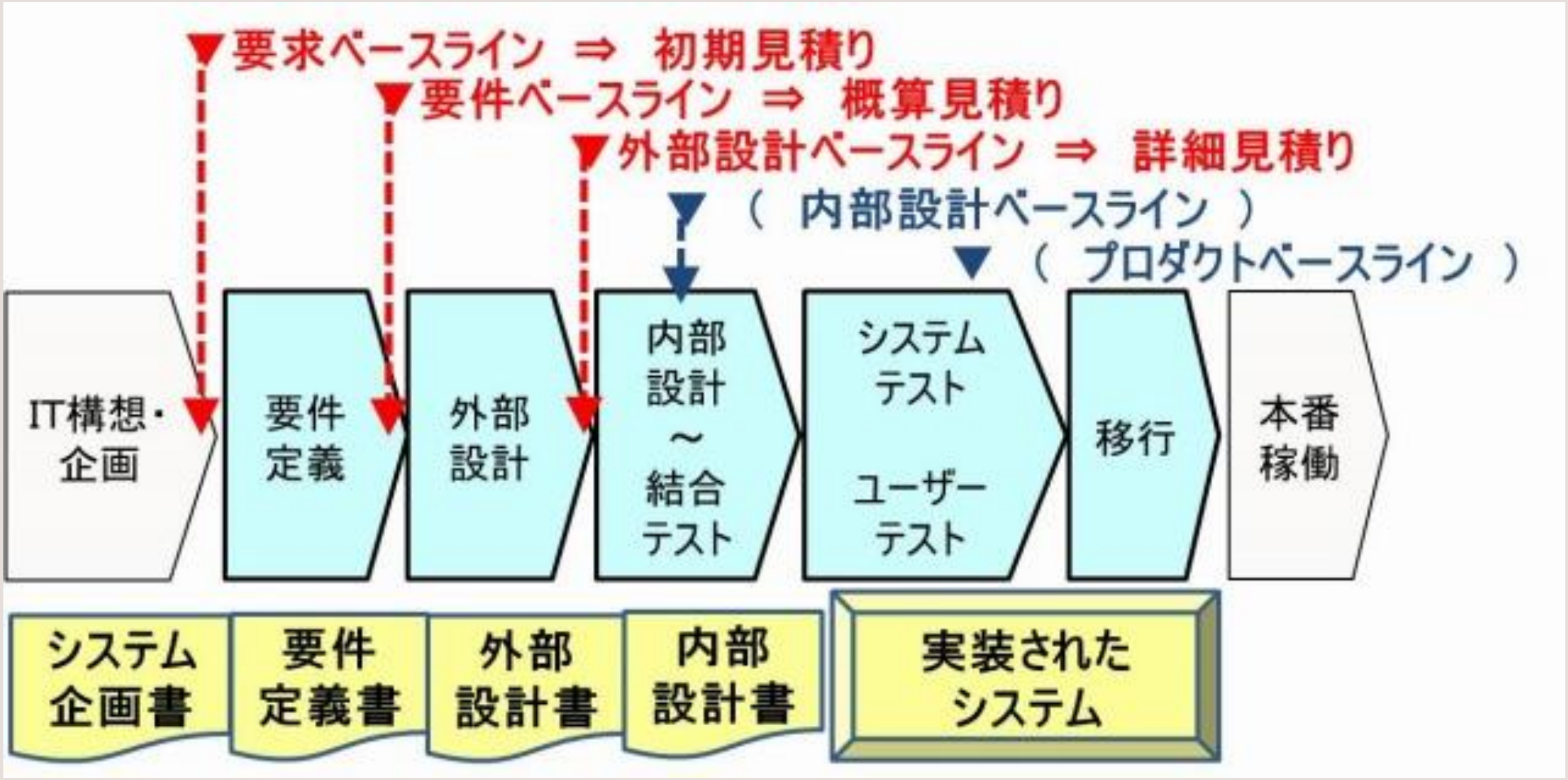
システム設計実践演習
第１０回

仕事の流れについて


要求ベースライン ⇒ 初期見積り
要件ベースライン ⇒ 概算見積り
外部設計ベースライン ⇒ 詳細見積り
（ 内部設計ベースライン ）
（ プロダクトベースライン ）
IT構想・企画
要件定義
外部設計
内部設計～結合テスト
システムテスト
ユーザーテスト
移行
本番稼働
システム企画書
要件定義書
外部設計書
内部設計書
実装されたシステム

テスト工程

テストの必要性

システムに不備があると思わぬ動作（誤作動）を起こします。**ソフトウェアが誤作動を起こすと、時間や金銭を失う不便な事態**になります。そのため、開発者は「誤作動を起こさないソフトウェア」を開発する必要があります。

さらに、ユーザーのニーズ(要求)を満たしているかも重要です。

「誤作動を起こさなければ良いシステム」とは限りません。

システムが使いづらければユーザーの満足度が下がります。

テストの必要性

つまりテストとは、**欠陥を取り除いて、ユーザーの要求を満たす、品質の良いソフトウェアを作る**大切な工程です。

質の良いテストを行うには、設計書や仕様書に書かれている文面を自分の理解に留めず、設計者や開発者の意図を理解する必要があります。

これを怠ると、エラーでシステムが落ちることはないが、

思わぬ動きをするシステムになってしまいます。

欠陥とは

誤作動の原因がソフトウェアにあることが特定されたもの。
バグや不具合とも呼ばれる。

欠陥は、ソフトウェアを作る際に人間のミスによって作りこまれる。**テストの主な役割は、ソフトウェアを搭載した製品やサービスを市場に出す前に、欠陥がないかをチェックすることです。**

ソフトウェアの進化と開発現場

ソフトウェアはここ数十年で大きな進化を遂げました。

例えば、携帯電話には通話機能しかありませんでした。数年後にはメール機能やスケジュール機能が追加され、今ではカメラ、ウェブ閲覧、電子マネー機能などが追加されています。

このように、**ソフトウェアを搭載した製品は、多機能化する傾向**にあります。

ソフトウェアの進化と開発現場

また、**開発期間の短縮**もエンジニアの負担として重くのしかかっています。

以前は数年かけて開発していた規模のシステムと同程度のものを、数か月で開発しなければならない場合もあります。

このような状況では、十分な余裕を持つことが出来ないので、欠陥が作りこまれてしまう一因となっています。

テストで問題に気づく

開発したソフトウェアが正しく動作するか確認する作業を**テスト**と言います。ソフトウェアは工業製品のように単体で使用されるものでは無く、複雑に絡み合った要因を想定して開発しなければなりません。

正しいデータや操作を正常に処理できることはもちろんのこと、**誤ったデータや操作に対しても異常終了することなく、適切な処理を継続して実行できることを確認**しなければなりません。

テスト計画書

抜けや漏れが無いようにテストは、事前に内容を考える必要があります。テスト開始前には**テスト計画書**を作成し、どのような目的で何をテストするのか？、その具体的な方法やテストの終了基準、スケジュールなどを書いておきます。

また、テスト計画書に沿って、テストデータの準備も必要です。

エビデンス

テストを実施した時には、入力や操作に対して得られた結果のハードコピー(画面キャプチャや出力された帳票)を取得して、証拠として残すことがあります。このような**検証結果の証拠をエビデンスと言います**。

開発後に不具合が発生すると、その原因を追究するためにいつ、どのような内容でテストを実施したのか？説明を求められます。エビデンスを残すことで、ベンダーが発注者に説明するときの文書として提示が出来ます。

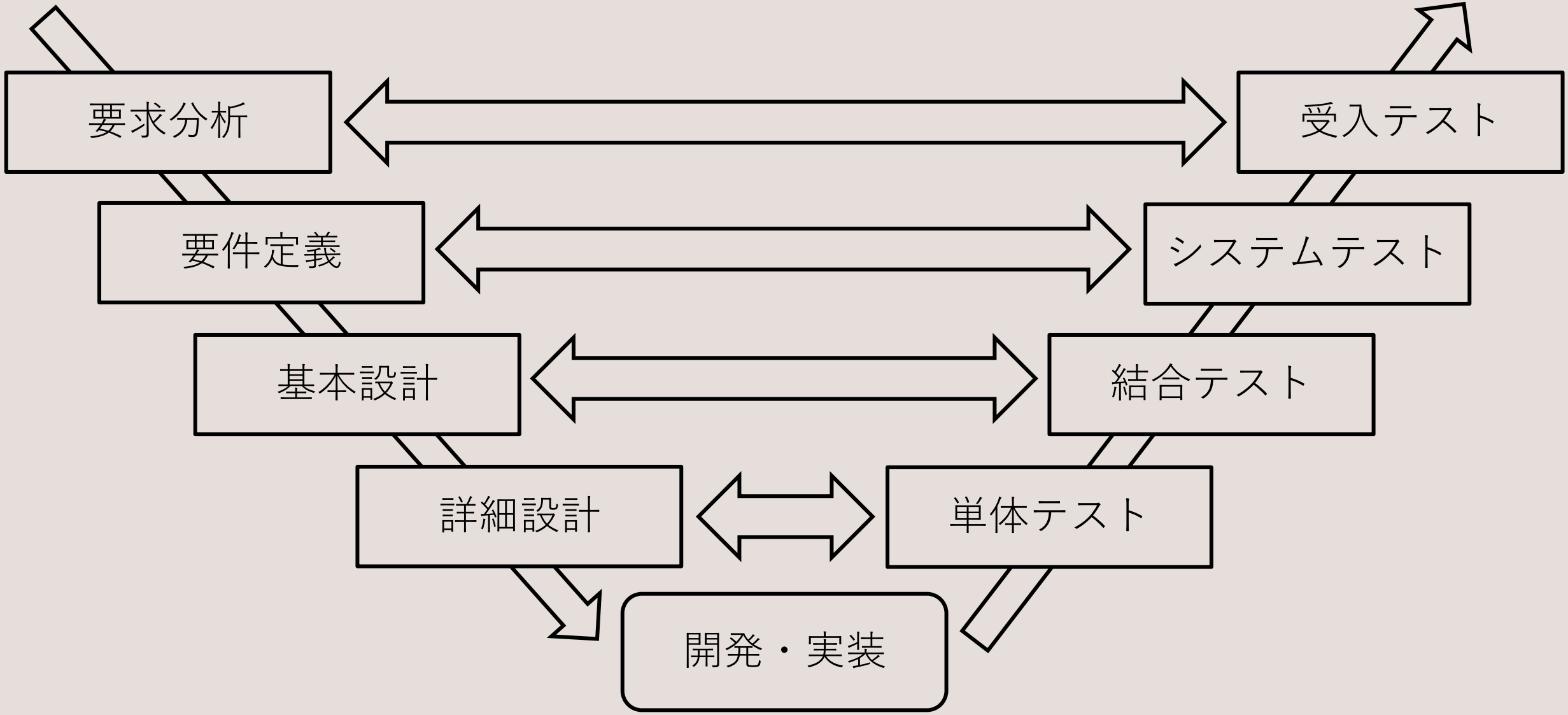
テストの分類

ウォータフォールモデルで開発を行う場合、「要求分析」「要件定義」「基本設計」「詳細設計」というそれぞれの工程を上流から下流に工程が進んでいきます。

テストでは下流から上流に順に各工程の内容が正しく実装されているかチェックを行います。

この流れがＶ字を描く形となるので、Ｖ字モデルといいます。

V字モデル



テストケース

テストをどのように実施するのかを書いた文書を**テスト仕様書**といい、テストする内容を**テストケース**といいます。テストの前提となる条件に加えて、テストデータとして与えられるもの、実際の動作、そして欲しい結果が書かれています。

テスト仕様書例

単体テスト仕様書						
システム名		VoiceOperation				
機能名		音声入力画面				
No.	テスト項目	条件	想定結果	担当者	チェック	日付
0	画面起動時		テキストボックスが表示されている	良原	OK	8月11日
1	画面起動時		テキストボックスが表示されている			
2			実行ボタンが表示されている			
3			出力ラベルが表示されている			
4			出力表示テキストが空文字で表示されている			
5			音声操作ボタンが表示されている			
6	実行ボタン押下時	テキストが入力されている場合	出力表示テキストに入力した内容が 表示されている			
7		テキストが入力されていない場合	出力表示テキストが空文字で 表示されている			
8	音声操作ボタン押下時		音声受付ダイアログが表示されている			
9	音声認識処理時 ※音声受付ダイアログ	「～を調べて」と発言した場合	ブラウザが起動して、 発言した内容が検索されている			
10		「～に行きたい」と発言した場合	GoogleMapが起動して、発言した場所に ナビゲーションが開始されている			

ブラックボックステスト

プログラムの入出力だけに注目し、動作が仕様通りかを判定する方法を**ブラックボックステスト**と言います。ソフトウェア開発においては仕様が定められているため、仕様に沿ってテストケースを設定し、それぞれ正しい結果が得られるかを検証します。

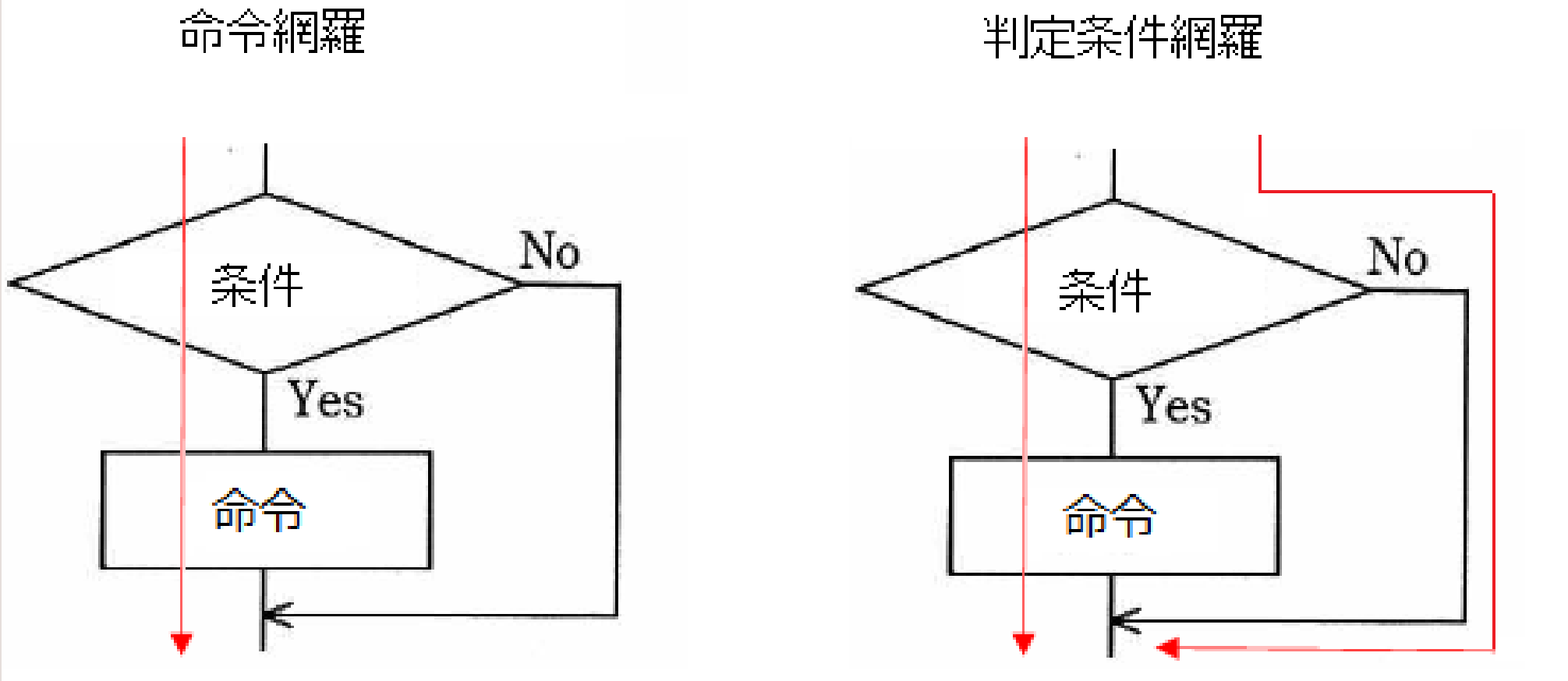
単体テスト、結合テスト、システムテスト、受入テストのすべての工程で使われます。

ホワイトボックステスト

ソースコードの中身をみて、各処理に使われている命令や分岐、条件などを、テストケースがどれぐらい網羅しているかを確認しつつテストする方法を**ホワイトボックステスト**と言います。

ホワイトボックステストのチェック指標として網羅率（カバレッジ）があり、命令網羅、分岐網羅、条件網羅などが使われます。

命令網羅と条件網羅



ブラック・ホワイトボックステストの使い分け

ホワイトボックスは通過するルートを調べることはできますがその条件自体の記述ミスはわかりません。このようなバグはブラックボックステストが有効です。

このため、基本的にはブラックボックステストを実施して、ホワイトボックステストで補完する手法の使い分けが必要です。

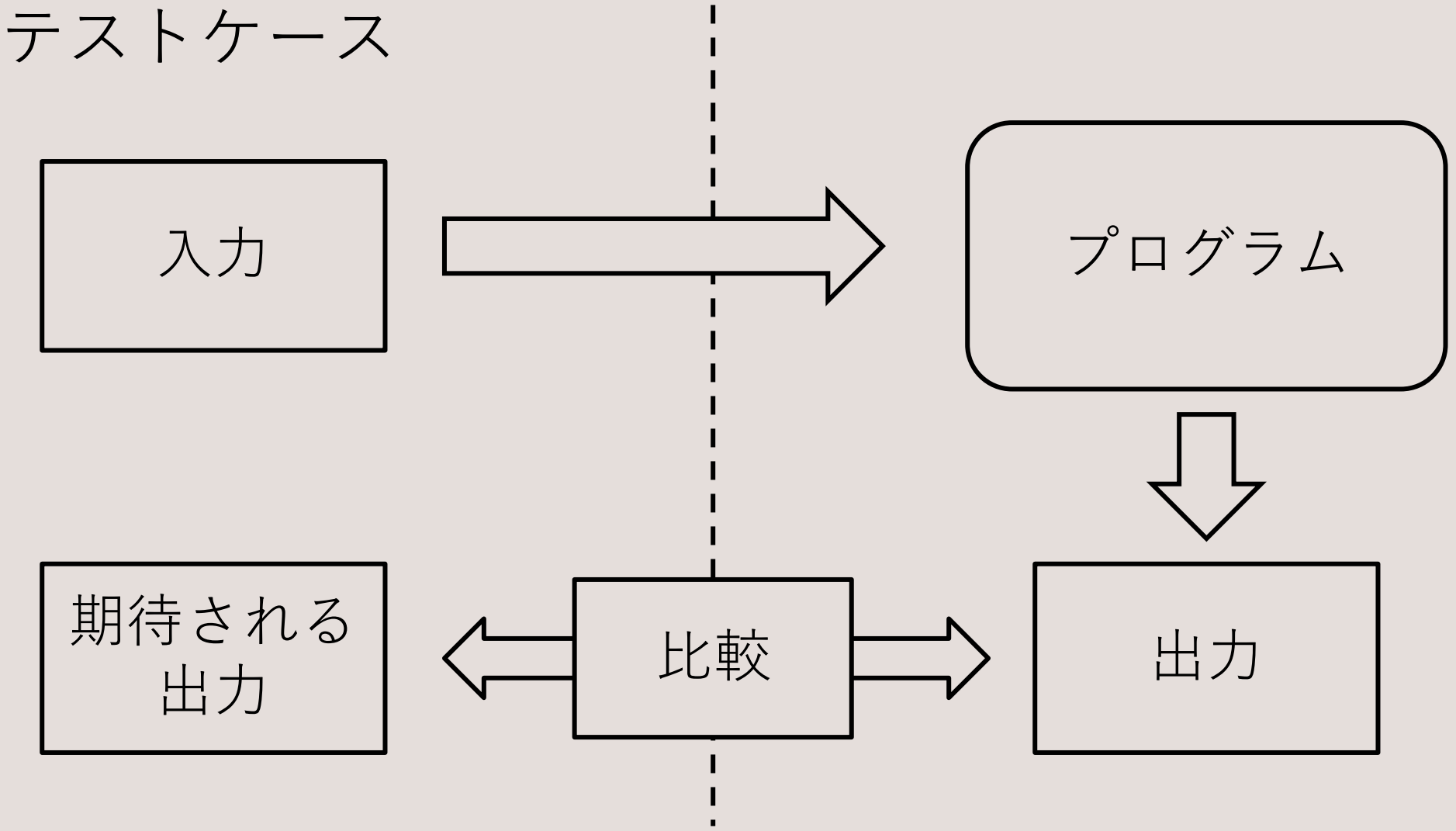
単体テスト

プログラムの個々の部分が問題なく実装されていることを確認するために行われるのが、**単体テスト**(ユニットテスト)です。

ある入力が与えられたときに想定している出力と同じ値が出力されるかを確認するなどの方法が使われます。

詳細設計で決めた内容を満たしているかを確認するもので、それぞれのソースコードを見ないとテストできないので、基本的には開発者がテストを行います。

単体テスト



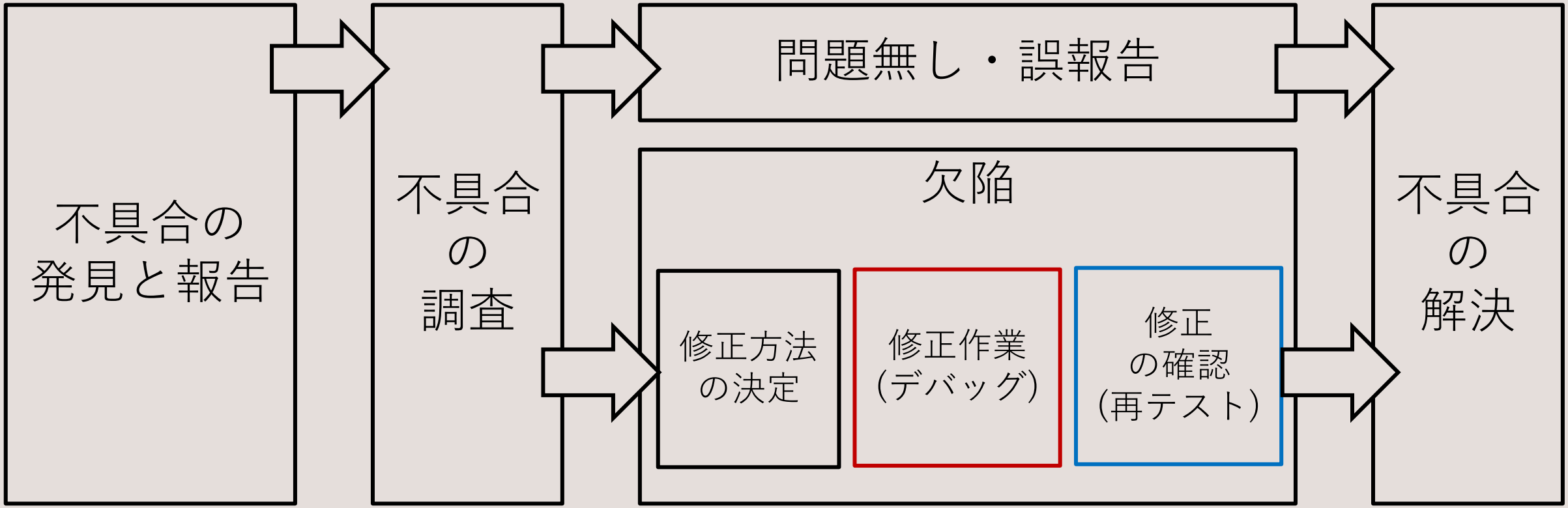
デバッグ

テストを進める中で欠陥が見つかると、それを取り除かなければなりません。このような欠陥を取り除く作業を**デバッグ**といいます。なお、デバッグには欠陥がないか探す作業が含まれることもあります。

テスト	正しく動くことを確認する
デバッグ	正しく動かない部分を探して修正する

テストをして問題が見つかったときにはデバッグを行い、修正されたことをテストで再度確認するといった流れが一般的です。

不具合の発覚から解決までの手順



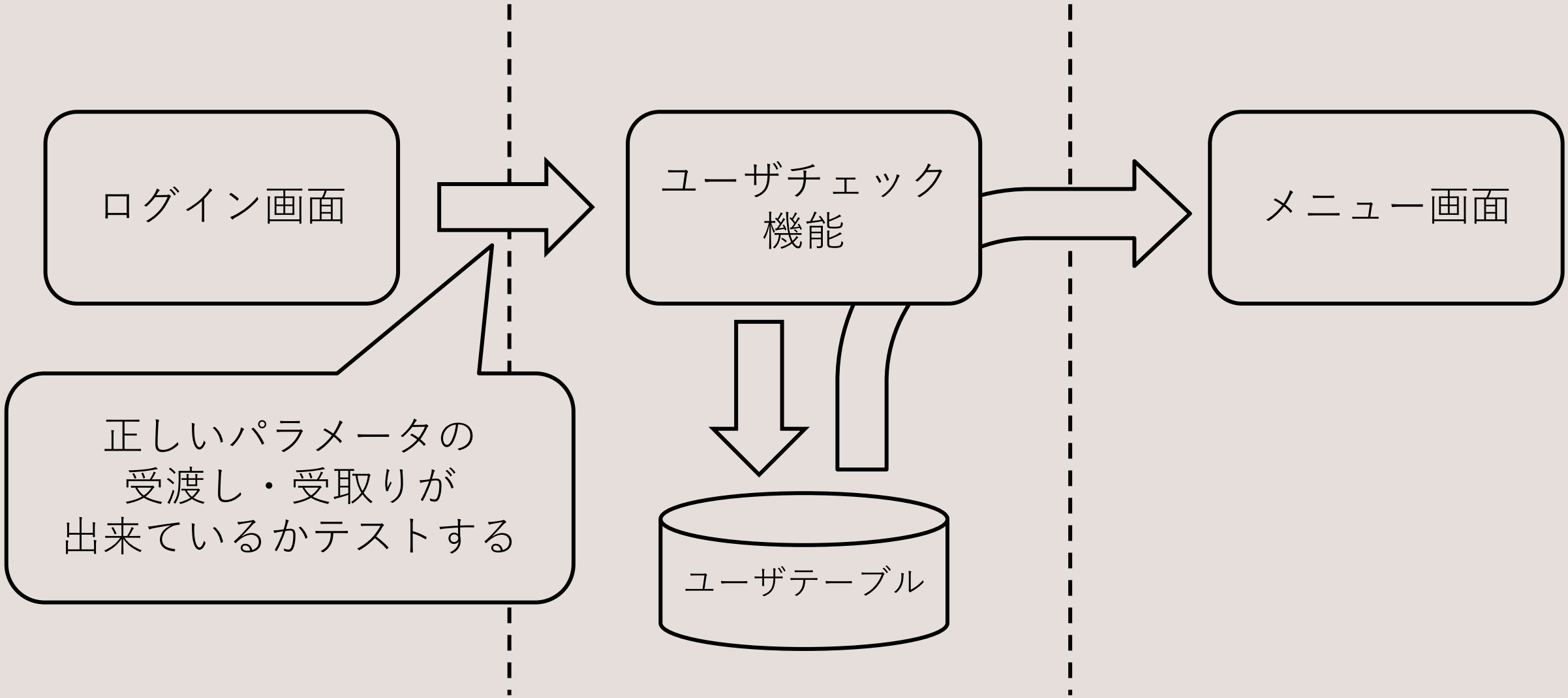
結合テスト

一般的にプログラムは、複数のモジュールと呼ばれる部品を組み合わせて構築されます。複数のモジュールを結合して行うテストを**結合テスト**(インテグレーションテスト)と言います。

単体テストが完了していることを前提として、**それぞれのデータをやり取りするインターフェースが一致しているかを確認**するためのテストです。

基本設計で決めた内容を満たしているかを確認するもので、最終的にまとめられたテスト結果は発注者が見ても理解しやすいです。

結合テスト


ログイン画面
正しいパラメータの
受渡し・受取りが
出来ているかテストする
ユーザチェック
機能
ユーザテーブル
メニュー画面

結合テストの手法

最初からすべてのモジュールを結合してテストすると、うまく動かなかったときにどこが問題なのかわかりません。

結合テストの行い方には、下位モジュールから順に上位に向けて結合する**ボトムアップテスト**と、上位のモジュールから順に下位に向けて結合する**トップダウンテスト**があります。

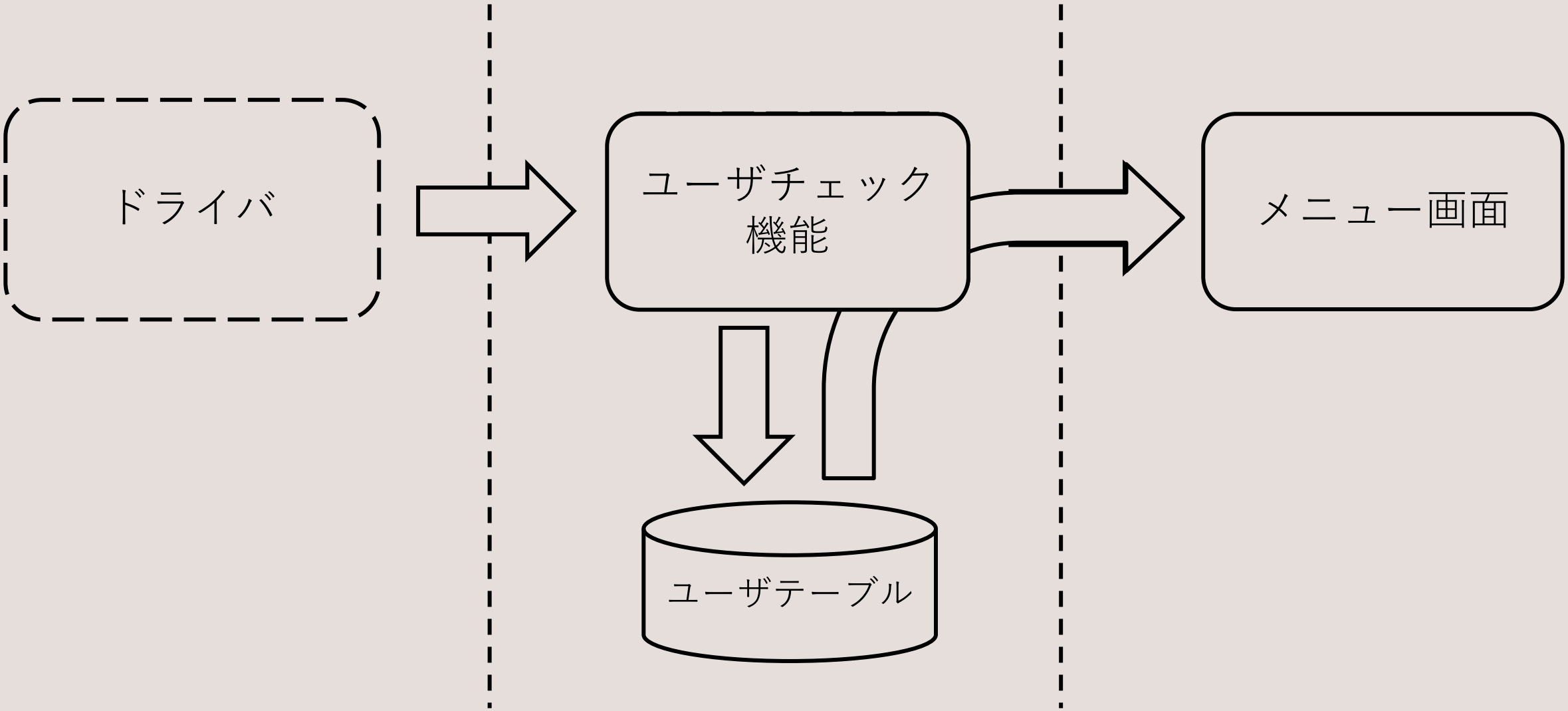
ボトムアップテスト

ボトムアップテストでは、まずテスト対象のモジュールを組み合わせて呼び出す「**ドライバ**」というプログラムを作成します。

ドライバは、上位モジュールの代わりになるもので、下位モジュールの処理を呼び出したときに正しい結果が得られるかを確認するために作られます。

テスト対象が問題なければ、ドライバを本来のモジュールに変更して、次はその上のドライバを用意して順にテストを行います。

ボトムアップテストイメージ

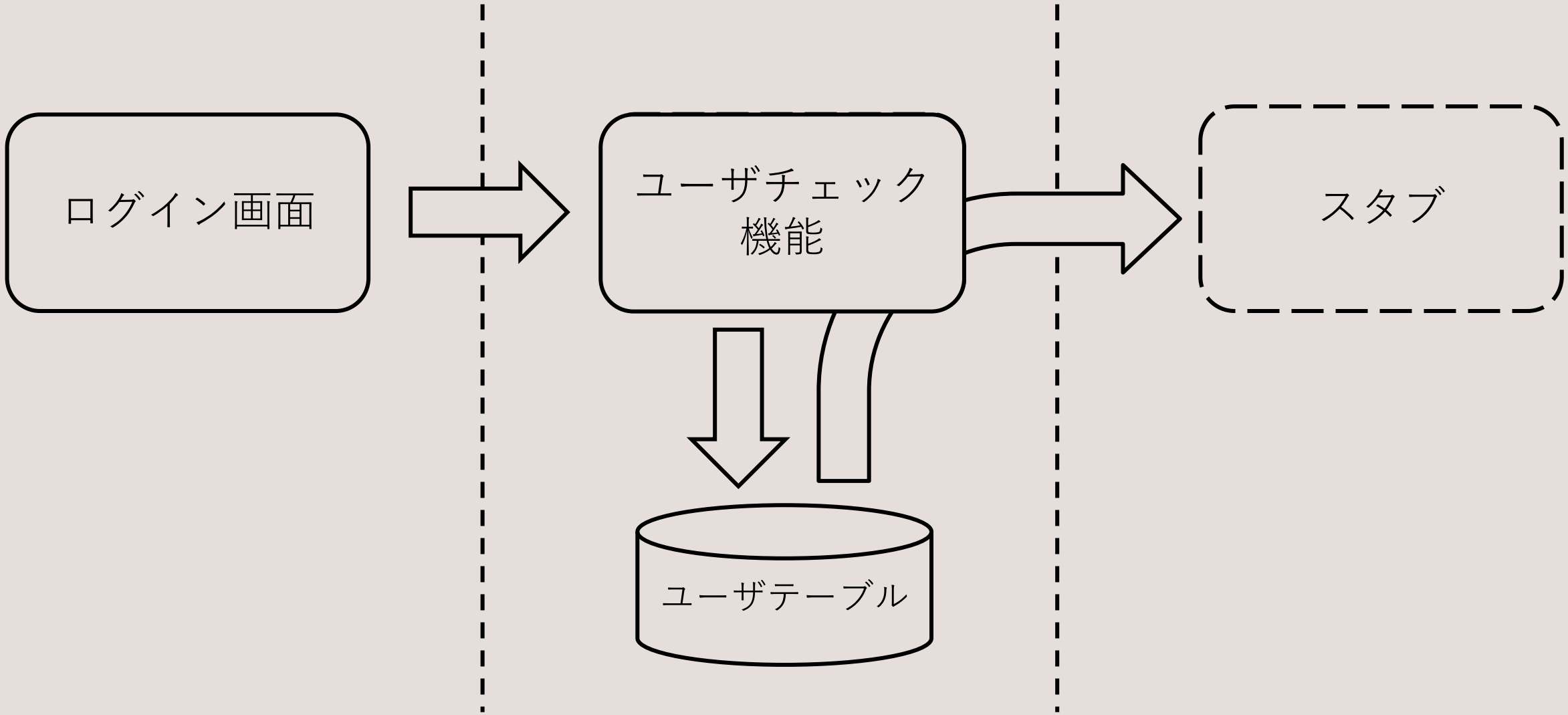


トップダウンテスト

トップダウンテストでは、最上位のモジュールから下位に向かってテストを進めます。最初の段階では下位のモジュールが出来ていないため、ダミーのモジュールを作成します。

これを「**スタブ**」といいます。

トップダウンテストイメージ



単体・結合は開発者側のテスト

単体テストや結合テストが終わると、開発者の環境では設計の内容に沿ったプログラムが問題なく動いていることを確認できます。

システムテスト

単体・結合テストが終わると、基本的にはシステムとして問題なく動くはずです。しかし、それは開発者の環境でテストしたときの話で、利用者が使う環境でも問題なく動くかはわかりません。

そこで、実際に使われるハードウェアなどを用いてシステム全体のテストを行うことを、**システムテスト**(総合テスト)と言います。開発者が行う最終的なテストでここで問題なければ発注者側にシステムが引き渡されます。

システムテスト

システムテストは、要件定義で定めた機能要件が実装されているかを確認するだけでなく、想定した時間内に処理できるか、セキュリティに不備はないか、システムの負荷に問題ないか、といった**非機能要件も確認**します。

また、運用についても考慮し、何らかの障害が発生した場合にどのように復旧するかや、ログの閲覧方法などについても確認します。

テスト観点の一覧

大分類	中分類	テスト観点
機能	正常系	・基本機能 ・遷移（状態、画面） ・設定（保持、変更、反映） ・アップロード／ダウンロード ・インストール ・表示 ・データ登録、変更、削除、反映 ・ユーザーインターフェース ・セキュリティ ・外部接続
	異常系	・異常値入力 ・エラー検知 ・エラーメッセージ確認 ・エラー復旧 ・異常データ ・記憶装置異常 ・異常状態 ・異常環境 ・異常操作（ボタン連打など） ・エラーログ確認
	組合せ	・同時動作 ・割込動作 ・排他処理（同時アクセス） ・互換性 ・構成 ・設定 ・互換運用性 ・オプション／付属品
非機能		・処理速度 ・負荷 ・大容量 ・連続動作 ・通信帯域不足 ・リソース不足
ユーザ		・出力品質（印刷など） ・ユーザビリティ、操作性 ・魅力性 ・業務シナリオ ・導入、保守

受入テスト

システムテストが終わると、今度は発注者側での確認に移ります。これを**受入テスト**といいます。発注者が希望した機能のうち、要求分析を経て実現すると決まったものが、システムで実現できているか確認します。

ただし、発注者側にシステムの専門知識がなくテストを実施できない場合は、別の事業者に受入テストを委託することもあります。

受入テストで問題なければ検収となります。通常はこの段階で、契約時に定められた金額の支払いが行われます。