テストパターン検討会
「テストファーストにおけるテストパターンの活用」

XPJUG　スタッフ
小井土 亨

アジェンダ

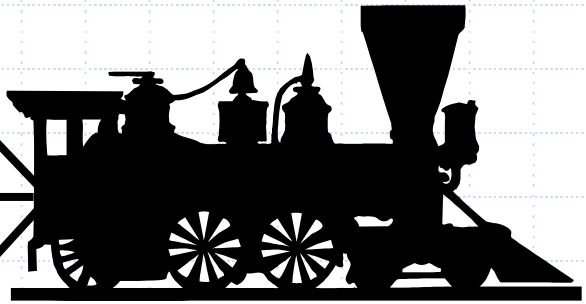
- 良いテストとは
- テストの問題点と解決策
- テストパターンとは
- Kent Beckのテストに関するパターン
- テストとテスト対象の複合パターン

テストは重要ですね

継続的なリリース
継続的なテストが必要

リファクタリング
自動テストによる確認が必須

高い品質
テストによる確認が必要



テストが開発を牽引

良いテストの条件

- テスト目的が明確である
  - 何をテストするか明確である
  - その目的も明確で、更にひとつであること
- テストの判定が正しい
  - テストの成功、失敗が正しく判断されている
- テストを独立して実行することができる
  - テストが他のテストに依存することなく、独立している
- 繰り返し実行することができる
  - 何度でも繰り返して、テストを実行することができる
- テストを実行しても、状態が変化しない
  - テストを実行し、成功した場合でも失敗した場合でも、テストを実行する前と後で何も変わらない

良いテストの効果

- メソッドが一つの機能を実現している
  - メソッドが明確な一つの機能だけを提供する
- メソッドの結果を提供する
  - 外部に対して、メソッドの処理結果を判断できるような何らかの方法を提供する
- クラスやメソッドの独立度が高いこと
  - クラスやメソッドが、他のクラスやメソッドとの依存が低い
- 特定の環境への依存度が低い
  - 特定のファイルやデータベース構造などに対する依存度が低い

重要なポイント

- シンプル
  - テスト、クラス、メソッドなど、全てがシンプルであることが最善である
  - 常にシンプルになっているか確認し、改善（リファクタリング）して、シンプルを追求する
- 低い依存度
  - テスト、クラス、メソッドなど、全てが他のテストやクラスやメソッドに対して、依存している部分を低くすることが重要である
  - 依存度を低くして、高い独立性を追求する

テストの問題点と解決策

- 問題点1
  - 計画が遅れて、テスト期間が確保できない
  - 解決策 → Test Driven Development
- 問題点2
  - 手作業による人海戦術
  - 解決策 → テストフレームワークの導入
- 問題点3
  - 品質の良いテストが書けない
  - 解決策 → テストパターン

テストパターンとは

- テストパターンの目的
  - テストファーストにおいて効率的なテストを行うための知識を共有する
- 対象とするテスト
  - 自動化することができるテスト
  - 単体テスト (単体クラスレベルのテスト)
  - 総合テスト (シナリオレベルのテスト)
  - 受入れテスト (システムの外部機能レベルのテスト)
- テストパターンの範囲
  - テストのパターン
  - テスト及びテスト対象のクラスから構成されるパターン

Kent Beck
テストに関するパターン

Kent Beckの著書「テスト駆動開発入門」からテストに関するいくつかのパターンを紹介

下位のテスト

- Q 大きすぎるテストケースをどのように動作させるのか
- A 大きなテストケースで失敗する部分を抜き出して、小さなテストケースを作成する
- 大きすぎるテストケースを作成してしまった場合
  - なぜ大きいか考える
  - どうすすれば、小さくなるか考える
  - 1つのテストで多くのことをやろうとしている
  - 小さいテストに分離する

モックオブジェクト

- Q 高価または複雑なリソースに依存するオブジェクトのテストはどのように行うのか
- A 定数を返す仮バージョンのリソースを作成する
- データベーステストでモックオブジェクトを導入
  - データベースのように動作するが実際にはメモリに存在するだけのオブジェクトを用いる
  - モックオブジェクトと実オブジェクトの振る舞いの違いがリスクとして発生する
    - テストが実際のオブジェクトで動作するとリスクが減少する

自己接続

Q オブジェクトが他のオブジェクトと正しく通信していることをどのようにテストするのか

A テスト対象のオブジェクトを本来通信すべきオブジェクトではなく、テストケースと通信させる

- 自己接続オブジェクトの効果と必要項目
  - 可読性が高くなる
  - 実装対象のインターフェイス抽出が必要
  - 限られインターフェイスが必要となる

ログ文字列

Q メッセージが呼びだされる順番が正しいことをどのようにテストするのか

A 文字列にログを維持し、メッセージが呼び出されるたびにその文字列を追加する

- ログ文字列導入の効果
  - テストの可読性が向上する
  - 順番にこだわらない場合は、文字列の集合にする
  - ログ文字列と自己接続を組み合わせる

クラッシュテストダミー

Q 呼びだされそうにないエラーコードをどのようにテストするのか

A 実際の処理は行わず、例外を起こす特別なオブジェクトで、そのエラーコードを呼び出す

- クラッシュテストダミー
  - 必要なエラーをシミュレーションするために、適切なメソッドだけを動作させないようにする
  - 必要なメソッドだけをテストケースでオーバライドする

テストとテスト対象の複合パターン

ヌルオブジェクト

Q オブジェクトが存在しないので、テストが正しく失敗しない

A 正しく失敗させるために、本来の実装の前にヌルオブジェクトを実装する

- 利用場面
  - オブジェクトを返すメソッドのテスト
    - クラスが提供するメソッドが処理結果としてインスタンスを返すようなメソッドをテストする場合に、正しくテストを失敗させる
  - テストの最初の実行（レッドバー）
    - テストの最初の実行時に、レッドバーと理由を正しく表示させる

インフォメーションセンター

Q 環境に依存するプログラムをテストしたい

A 事前にテスト用に環境を構築し、実行時に環境を切り替えてテストを可能とする
環境に依存する設定部分を一元管理する

- 利用場面
  - データベースの状態に依存するテストの実行
    - データベースの状態に依存するテストを行う場合に、状態を再現したデータベースに環境を変更してテストを実行する
  - 設定ファイルやファイルに依存するテストの実行
    - ファイルの内容に依存するテストを行う場合に、様々な内容のファイルを用意し、対象を変更してテストを実行する

ビューステート

Q ユーザーインターフェイスをテストしたい

A ユーザーインターフェイスの状態やロジックを別クラスに分離し、自動テストを行う

- 利用場面
  - ユーザーインターフェイスのテスト
    - ユーザーインターフェイスの状態やロジックを自動テストする
  - シナリオテスト
    - 複数のユーザーインターフェイスに対応したビューステートクラスを作成し、様々なシナリオテストを行う

オーダーシート

Q結合度の高い複数のクラスをテストしたい

Aクラス間の依存関係を無くして、テストを実行する

- 利用場面
  - 処理のバリエーション実行テスト
    - オーダーシートクラスに様々な設定を行い、カバレッジの高いテストの実行を行う
  - シナリオテスト
    - 複数のビューステートクラスにオーダーシートクラスを渡す方法で、シナリオ型のユーザーインターフェイステストを行う

ロールバック

Q 確かなデータベースのテストを確実に行いたい

A テストクラスとテスト対象クラスを同じトランザクションに含めることで、テスト実行後データベースをロールバックする

- 利用場面
  - データベースの更新テスト
    - データベースの状態を変えることなく、データベーステストの実行を行う