consulting & education

Practical Object-Oriented Development

TDDライブ

リズムを感じて

永和システムマネジメント
天野 勝　m-amano@esm.co.jp
安井 力

概要

- テスト駆動開発(TDD：Test-Driven Development)は、実行可能なテストスクリプトを書きながら、プログラミングを進めるという、開発スタイルです。
- TDDでポイントとなるのは、「リズム」と呼ばれる規律に則った開発プロセスのまわし方です。アジャイル開発の文脈で語られることが多いことから、アジャイル開発特有の開発スタイルと思われることもありますが、アジャイル開発以外にも適用できる幅広さを備えています。
- 本セッションでは、TDDによる開発を、ライブで見ていただくことで、このリズムとはどのようなものか感じ取っていただき、メリット・デメリット、適用可能・不可能を考える材料を提供します。

本日の予定

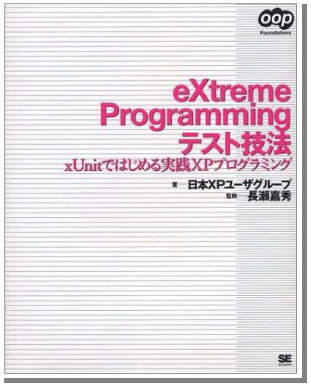
- TDDについての説明とデモ
  - TDDの説明をデモを交えながら行います
  - 随時、質問を受け付けます
- Q&A
  - 会場からの質問やリクエストにお答えします
  - # できることは限られていますが・・・

会社紹介

- 株式会社 永和システムマネジメント
  - 本社は福井県福井市
  - 金融、医療、オブジェクト指向を使ったシステム開発
  - 1980年創業、2002年東京事務所開設

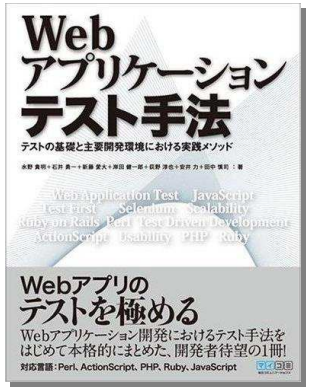
講演者紹介

- 天野 勝(あまの まさる)
  - オブジェクト指向、アジャイル開発、開発現場の活性化をテーマに、ファシリテーションを活用したコンサルティング、セミナーに従事。
  - 2002年にKent BeckからTDDを教わる
  - 著書:『eXtreme Programmingテスト技法』
  - 訳書:『リーン開発の本質』『アジャイルソフトウェア開発スクラム』



講演者紹介

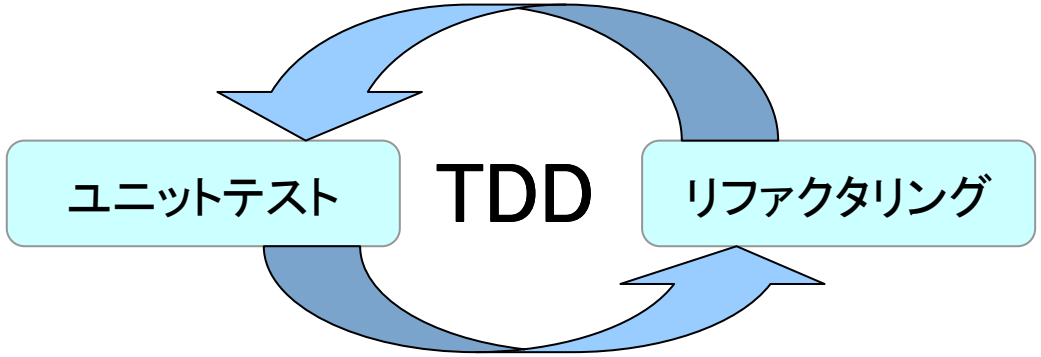
- 安井 力(やすい つとむ)
  - 「人間の力を活かすことがよい結果を生む」という信念で、アジャイルとファシリテーションを武器に開発現場の支援に従事。ユーザ不在の現場を危惧し、アジャイルがユーザと技術者をつなぐ鍵になると考えている。
  - 著書:『Web2.0ビギナーズバイブル』『Webアプリケーションテスト手法』
  - 訳書:『アジャイルな見積りと計画づくり』



TDDの説明

テスト駆動開発 (TDD：Test-Driven Development)

- ユニットテストを作り、ユニットテストで動作を確認しながら、開発をすすめるという、開発手法
  - Extreme Programming（XP）のプラクティスでテストファーストと呼ばれていた開発手法の進化系
  - 品質保証の手法ではない、開発者のための設計手法
  - 開発のフィードバックサイクルを小さくする技術
  - ユニットテスト、リファクタリングと相互補完する



POINT テストが開発を駆動していきます

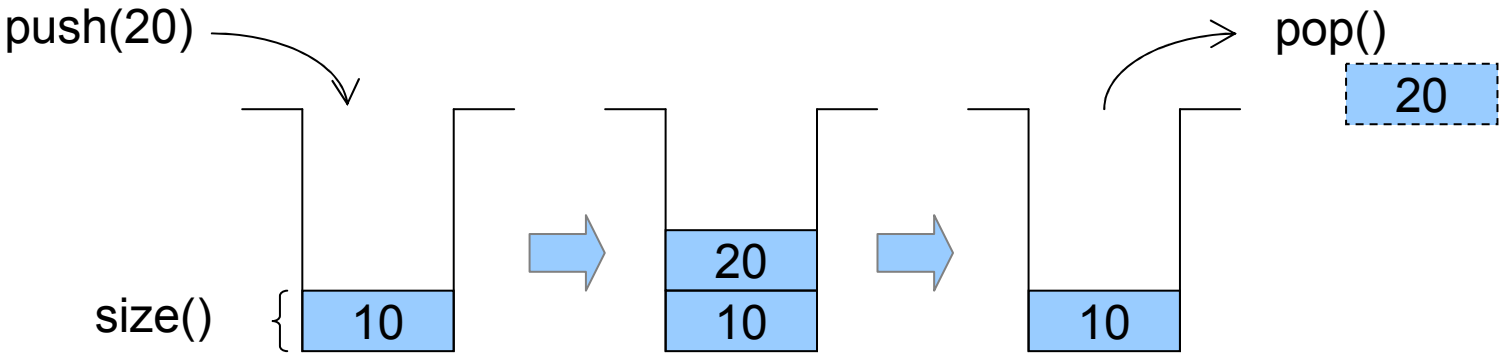
TDDの手順

1. テストコード(ユニットテスト)を書く
2. コンパイルが通る最小のコードを記述する
3. テストに失敗することを確認する （赤）
4. テストが通る最小のコードを記述する
5. テストが通ることを確認する （緑）
6. 不吉な臭いがしたら、リファクタリングする
7. テストが通ることを確認する （緑）
8. 仕様が充分に満たされるまで1～7を繰り返す

POINT 赤、緑、リファクタリングで開発のリズムをつくります

TDDデモ：スタックの作成

- スタックの仕様
  - size()でスタックのサイズを取得する
    - int size()
  - push()で引数の値をスタックの一番上に積む
    - void push(int value)
  - pop()でスタックの一番上の値を返し、取り除く
    - int pop()
    - スタックが空の場合、EmptyExceptionが発生する



TDD(テストファースト)効果

- 仕様を把握できる
  - 仕様を知らなければテストコードは作成できない
    - テストコード＝仕様、サンプルコード
  - 最初に目的地をきっちり決める
- 無駄なコードが無くなる
  - 小さな目標を決め、そこに向かってプログラミングする
    - 自動の受入テスト(大きな目標)があるのがベター
  - 着実に目的地の方向に進む
- テストしやすい設計になる
  - 事前にテストのことを考慮する
  - 標識(チェックポイント)が沢山ある

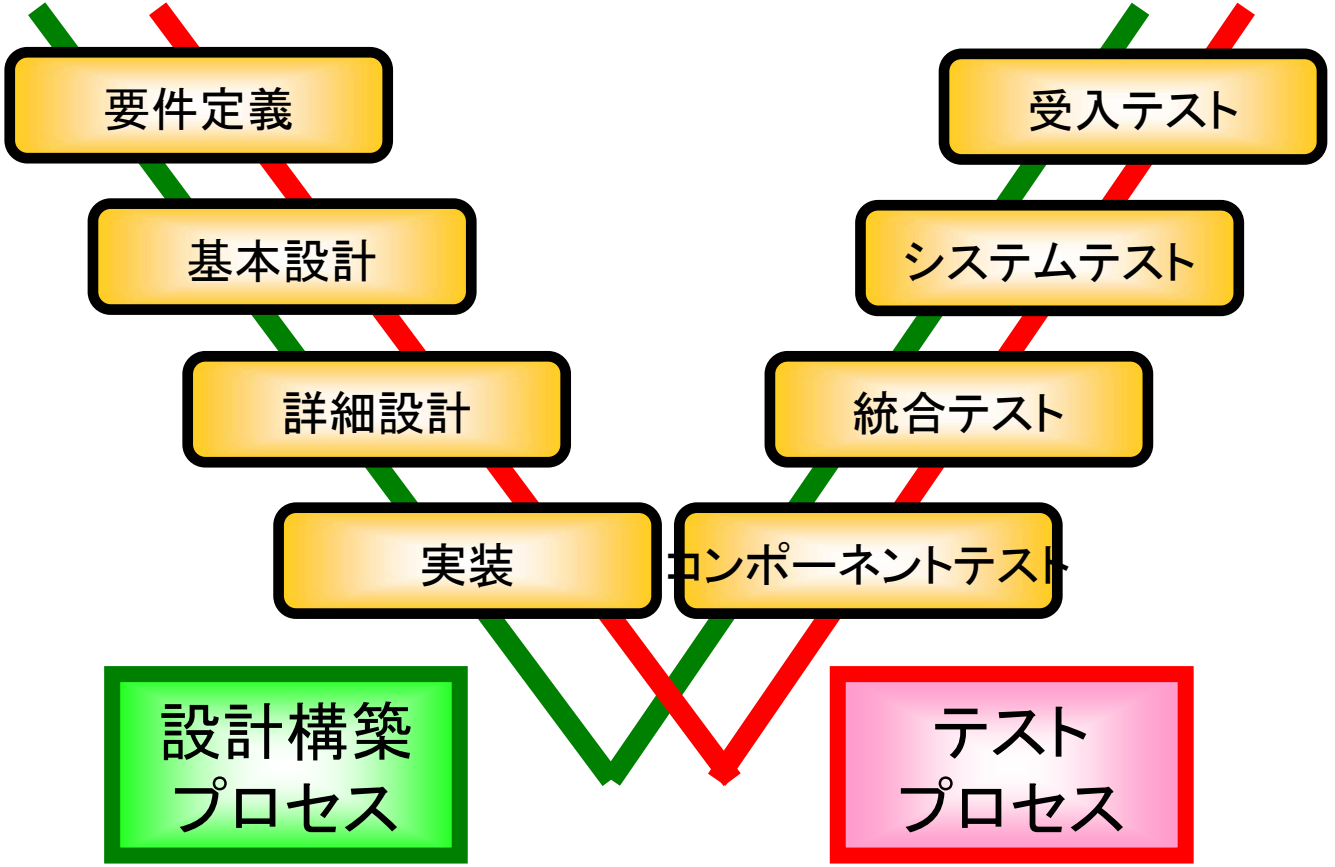
POINT 品質と、生産性の向上が期待できます

TDD中に行っていること

- 仕様の理解
- テスト(ケース)設計
- プログラム(モジュール)設計
- 自動テストケース作成
- プログラム作成
- テストケースの妥当性確認
- プログラムの妥当性確認

POINT 開発者の多能工化が求められます

ダブルV字モデル



出典「ISTQB」

ここでのユニットテストの定義

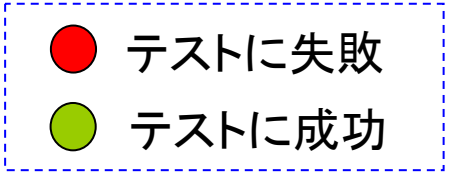
- モジュール単位のテスト
  - クラスや、メソッドの単位で、仕様どおりに動作するかを確認する
- プログラマ自身が行うテスト
  - プログラマ自身が、自分の作ったプログラムをテストする
    - テストが甘く(不十分)なりやすい
- ホワイトボックステスト
  - プログラムの内部を念頭に置いて、テストする
    - プログラムどおりに動作するかをテストしてしまいがち
- 動的テスト
  - 実際にプログラムを動作させてテストする

POINT プログラマが行う、小さい単位のテストです

ユニットテストを行う理由

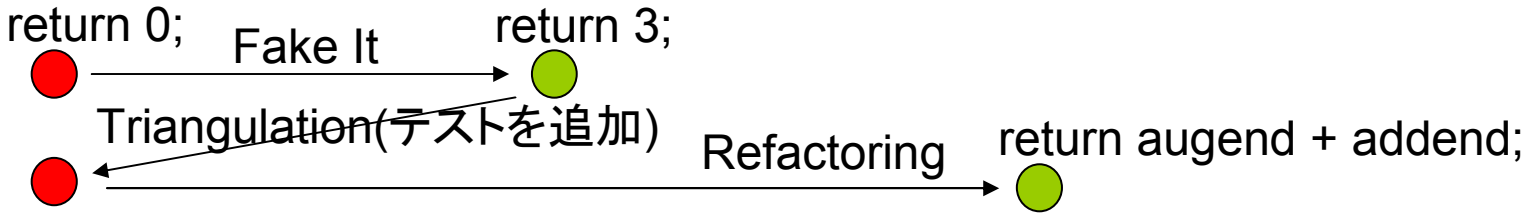
- 比較的容易かつ低コストで効果の高いテストできる
  - 小さく単純なユニットを検証
    - 巨大で複雑なシステムも、小さく単純なユニットから構成されている
- 欠陥の多くはユニットレベルで検出できる
  - ユニットに誤りがあればシステム全体に影響
  - 誤りの「大きさとその結果生じる問題の間には何の関係もない」
    - 出典：ジェラルド・M・ワインバーグ「プログラミングの心理学」

TDDの主な3つのアプローチ

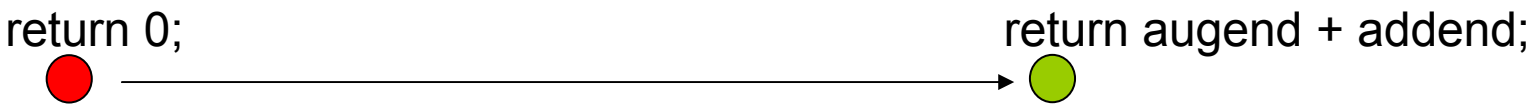


- Fake It → Refactoring

- Triangulation（三角測量）



- Obvious Implementation(明白な実装）



POINT 赤になってはじめて製品コードが書けます

ここまでのまとめ

- テストコードは仕様、サンプルコードの役割を持つ
- 小さい目標を立て、そこに向かって作業を進める
- テストを赤くしなくては、製品コードを書いてはならない
- 赤、緑、リファクタリングのリズムで進める
  - Fail It → Fake It → Refactoring
  - Fail It → Fake It → Triangulation → Refactoring
  - Fail It → Obvious Implementation