テスト駆動開発(Test Driven Development)

TDDはテスト手法ではない。TDDは分析技法であり、設計術法であり、実際には解析の全てのアクティビティを構造化する技法である。 **TDDの目的:「動作するきれいなコード**
当たり前だが、"動作しない、きれいなコード" より "汚い、動作するコード" の方が価値がある。
TDDでは動くコードを作成した後、きれいなコードを目指していく。（アーキテクチャ駆動とは反対）

TDDのサイクル & メリット

TDDのサイクル

以下のサイクルを短い時間(10分程度)で回す。

> 1. テストを書き
> 2. そのテストを実行して失敗させ(`Red`)
> 3. 目的のコードを書き
> 4. 1で書いたテストを成功させ(`Green`)
> 5. テストが通った状態を維持しながらリファクタリングする(`Refactor`)
> 6. 1 ~ 5 を繰り返す

t-wadaさんのスライドから引用

最初に必要な機能のTODOリストを作成し、これから何を実装するのかを検討。
一部のテストコードを書き、失敗させる。
テストコードが通る最速の実際コードを書き、成功させる。
コードをきれいな形にリファクタリングし、次のTODOリストを行う。

メリット

> TDDで大切なことは、細かいステップを踏むことではなく、踏み続け蹴られることだ。

テスト駆動開発 から引用

サイクルを細かく回すメリットは「いつ、どこで失敗したのかわかること」だと思う。自分の実装手順でどこに問題があるのかを早い段階で知ることができる。また、テストを書くことでコードのきれいさに惑わされず、コードがもつの責務に集中することができる。そのコードを使う側の気持ちになることもできる。

テスト評価手法

テスト駆動開発 で紹介されていた手法に補足する形書く。

- ステートメントカバレッジ: プログラム内のコード（命令文）のうち、テストを実行した割合を示す。テスト駆動開発で開発を行うとカバレッジは100%になる。（ただし、100%を追求しても品質は高くならないのでは？という議論もある)
  松岡さんのYoutube(TDDオンライン勉強会 #1) で出てきた、"カバレッジは「足りないこと」はわかるが「十分であること」の根拠には使えない」は名言すぎる。

- 欠陥挿入(defect insertion): プロダクトコードの任意の行の意味合いを変えたらテストは失敗しなければならない。

他に伝えたいこと

- テスト間の依存関係は悪である。
    - 1つのテストコードが失敗したら後続のテストが失敗すること。
    - テストA → テストBの順番ならテストが成功するが、テストB → テストAの順番だと失敗する。
- 途中で開発をやめるうまいタイミングは、「テストを失敗した状態にすること」
    - テストが失敗しているので、どこでやめたかわかりやすい。自分が何を考えていたかを思い出しやすい。