TDD道場
～場外乱闘編～

TDD流派対決

biac vs guicheng

Twitter: @biac @guicheng
http://www.tdd-net.jp/
http://blogs.wankuma.com/rudicast/

自己紹介: biac

- 1957年 名古屋生まれ (宇宙時代以前)
- 大学の専攻は航空工学
- 卒業後、本田技術研究所で**機械設計**

- 1994年 プログラマーに転身
- 2001年 Windows XP のリリースと前後して XP と TDD を知る
- この9月から自宅警備員w

自己紹介: guicheng

- 学生時代の専攻
  - 分析化学(水溶液中金属の超微量分析)
- 趣味
  - 宇宙工学(特に、輸送系というかエンジン)
- 現職
  - プログラマ(AutoCADのカスタマイズ)
    - この業界に舞い戻って(出戻って)きました。

Autodesk
Authorized Developer

おしながき

- 前振り
  - 簡単にTDDのおさらい
- ディスカッション
  - biac & guicheng & みなさん
  - **TDDには様々な流儀が!**
    どっちの流儀がいいか、会場の皆さんも一緒に考えてみましょう。

前振り

TDDって？

【前振り】TDDって？(1/5)

- TDD: **T**est **D**riven **D**evelopment
  テスト駆動開発
  ※「開発」と言ってるけど実装だけの話

- **TDD = テストファースト + リファクタリング**
  biac: 「外部設計→内部設計」　　t_wada: 「黄金の回転」

※ t_wada図: デブサミ2011の資料 http://www.slideshare.net/t_wada/the-only-one-big-thing-every-programmer-should-know より

【前振り】TDDって？(2/5)

- テストファースト
  ※「テストケース ファースト」と言った方がいいかも。
  - 1. 失敗するテスト (=未実装) を書く (RED)
  - 2. そのテストを通すだけの実装を書く (GREEN)
  - 3. メソッドが完成するまで 1. に戻る

※ 参照: TDD三原則 http://www.tdd-net.jp/2009/08/tdd-9534.html

【前振り】TDDって？ (3/5)

- **テストファーストの原理**

メソッドの外部設計 → 内部設計

※ 外部設計を決めてから、内部設計を考える。

※ アプリケーションの品質保証テストを実装中にやるわけじゃない。

- **メソッドの外部設計** (機械設計では「部品のスペック決め」)
  昔: シグネチャ定義、入出力表、状態遷移表 etc.
  TDD: 以上を自動実行可能なテストケースで表現!

- **メソッドの内部設計** (機械設計では「部品の設計・製図」)
  昔: フローチャート、PAD図 etc.
  TDD: 外部設計を満たすコードを直接書いちゃえ!
  ※ 後で、リファクタリングして改善するよ。

※ 環境の変化も見逃せない: 紙の上で悩んでいるより、IDE上でコードを書いて・ビルドして・自動テストで試してみるほうが、今や圧倒的に早い! (低コスト・短時間になった)

【前振り】TDDって？ (4/5)

- **リファクタリング**

  外部設計を保ったまま、内部設計を改善する

  - **1. テストが成功することを確認する (GREEN)**
  - **2. 実装をちょっと直す**
  - **3. テストが成功することを確認する (GREEN)**
    必要なだけ 2. に戻る

※ 参照: Martin Fowler's Bliki
リファクタリングの誤用 http://capsctrl.que.jp/kdmsnr/wiki/bliki/?RefactoringMalapropism
リファクタリングの境界線 http://capsctrl.que.jp/kdmsnr/wiki/bliki/?RefactoringBoundary

【前振り】TDDって？(5/5)

- **リファクタリングの原理**

  外部設計が変わらないなら、
  内部設計を変えても良いじゃなイカ!?

  - **昔: 動いているコードは触るな!**
    コードを変更したら、テストしなければならない。
    テストには人手と時間が掛かる。

  - **TDD: テストファーストした部分は改善してもOK!**
    外部設計に影響を及ぼしていないことは、
    自動化されたテストですぐに証明できる。
    ※ 自動化されたテストの存在が前提 (大切なことなので2回!)


それでは…

ディスカッション

TDD 流派対決

ルール

- それぞれのお題について、
  どっちが良いか、まず会場のみなさんに
  挙手してもらいます。
- biacとguichengが、それぞれの立場から、
  とりあえず喋ります。
  ※ 少数派だった方から
- 会場のみなさんも、
  どんどん手を上げて、言いたい事・ツッ
  コミたいことを喋ってください。

お題

- 他に何かありませんか!?
- どれからやりましょうか?
  - 1: **リファクタリングのタイミング**は、RED→GREENの都度? vs 一段落ついてから?
  - 2: **実装のスケルトンは、自動生成?** vs 先に手書き?
  - 3: **最初のテストケース**は、メインルート? vs ガード句?
  - 4: テストケースを書いたとき、その**REDの確認は、全テスト?** vs そのテストケースだけ?
  - 5: テストケースを書かずに **Enum の 0** を定義しちゃう? vs しない?
  - 6: 似た**テストケースを書くのにコピペ**は、OK? vs NG?
  - 7: 子メソッドが必要だと気付いた時、今やってるテストケースは、**コメントアウト?** vs **ignore(Explicit)属性?**
  - 8: **自動実装プロパティ**をテストする? or しない?
  - 9: 状態を持っている**シングルトンオブジェクト**は?

お題（続き）

- (当日会場から出たお題)
  -

お題1: リファクタリングのタイミング

- 1: リファクタリングを行うタイミングは、
  RED→GREENの都度?
  　　vs
  メソッドがひととおり完成するなど、
  一段落ついてから?
- [memo]
  –

お題2: 実装のスケルトン作成

- 2: 実装メソッドのスケルトンは…
  テストコードから自動生成?
  　　vs
  手書きで、テストコードより先に作る?
  (実装のスケルトンからテストコード生成)
- [memo]
  –

お題3: 最初のテストケース

- 3: 最初に書くテストケースは、実装の…
  メインルート (デフォルト処理) ?
  vs
  ガード句?
- [memo]
  - デフォルト処理: FizzBuzzなら、3でも5でもない時。一般に、末尾に書かれる。(return n.ToString();)
  - ガード句: 例外的な条件のときに実行する処理。メソッドの先頭付近に書かれる。

お題4: RED確認のテスト

- 4: テストケースを書いたとき、そのREDの確認のために実行するのは、
  全テスト?
  　　vs
  そのテストケースだけ?
- [memo]
  –

お題5: 使わない Enum の 0

- 5: Enum の 0 を使うテストケースを書かない(必要が無い)ときに、それでも…
  定義しちゃう?
  　　vs
  定義しない?
- [memo]
  –

お題6: テストコードのコピペ

- 6: 似たテストケースを書くとき、コピペは、OK?
  vs
  NG?
- [memo]
  -

お題7: 一時的に不要になったテスト

- 7: しばらく不要になったREDになるテストケースは…
  コメントアウト?
  　　vs
  ignore (Explicit) 属性を付ける?
- [memo]
  - この状況は、たとえば今作っているメソッドが複雑で、子のメソッドが必要だと気付いた時など。今作っているメソッドのテストケースを中途で放置して、子のメソッドの作成に移ることになる。

お題8: 自動実装プロパティ

- 8: 自動実装プロパティをテストする?
  vs
  しない?
- [memo]
  - 自動実装プロパティの場合、setしたものをgetできて当然なので、テストは不要という考え方がある。でも、プロパティの中身を実装する可能性があるのでテストしておくという考え方もある。

お題9: 状態を持つシングルトン

- 9:状態を持っているシングルトンオブジェクト

- [memo]
  - テストケースを実行すると中身(状態)が変わってしまい、テストの実行順に依存してしまう。
    どうやったらテストファーストで実装できるのか?

お題10:（追加用）

- 10:
  vs
- [memo]
  -


終了～

以下、附録

テストファースト: TDD三原則

1. **失敗するユニットテストを成功させるためにしか、プロダクトコードを書いてはならない。**

2. **失敗させるためにしか、ユニットテストを書いてはならない。**(コンパイルエラーは失敗に数える。)

3. **ユニットテストを1つだけ成功させる以上に、プロダクトコードを書いてはならない。**

※ 出典: Robert C Martin (UncleBob) (翻訳 安井力)
http://www.butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd
http://twitter.com/unclebobmartin
http://twitter.com/yattom

宣伝: TDD.NET

- http://www.**tdd-net.jp/**
  - 「**VB2010** Express + NUnit 2.5 で、**初めてのTDD Step by Step**」(PDF版,72ページ)
    http://www.tdd-net.jp/vb2010ee-nunit25-tdd-stepbystep.html

VB.NET で
TDDの実際を
詳しく解説

おねがい

- ぼくと契約して**翻訳家**になってよ!!

NUnitマニュアル翻訳プロジェクト

http://www45.atwiki.jp/nunitjp/

NUnit3(いつだよ? w)を待って、活動開始予定!