TDDってどんな感じ?
FizzBuzzを作ってみる

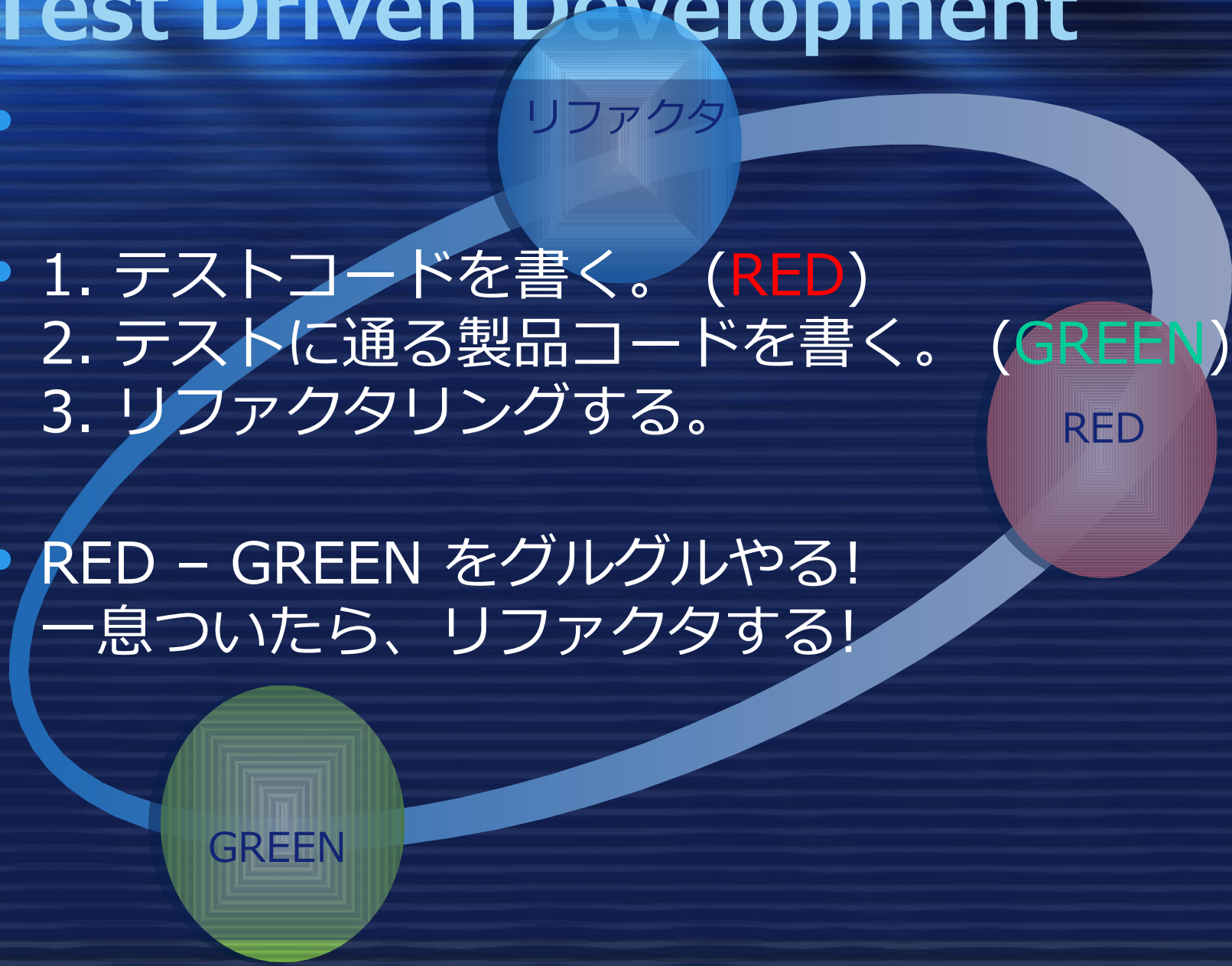
2010/01/22 biac

自己紹介

- 山本康彦 / **biac**
  - いまだにプログラムを書いてる 52歳
    名古屋生まれの名古屋育ち
  - http://**tdd-net.jp**/
    http://bluewatersoft.cocolog-nifty.com/
    ※ ハンドルでぐぐってもらえば見つかる(経済産業諮問委員会じゃないほう)
- コミュニティ
  わんくま同盟に出没
- もとは機械の設計屋さん
  - ものごとの見方・考え方が、きっとズレてる

Test Driven Development

-



- 1. テストコードを書く。 (RED)
  2. テストに通る製品コードを書く。 (GREEN)
  3. リファクタリングする。

- RED – GREEN をグルグルやる!
  一息ついたら、リファクタする!

TDD 三原則

- 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。
- 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。
- 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。

by Robert C. Martin (UncleBob)

http://www.butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd

FizzBuzz

- 実際には、どんなふうにやってるの?

  ライブコーディングはムリだけど、雰囲気だけでも…

- FizzBuzz ゲーム

  最初のプレイヤーは「1」と言う。

  次のプレイヤーは直前のプレイヤーの次の数字を発言していく。

  ただし、**3 で割り切れる場合は「Fizz」、5 で割り切れる場合は「Buzz」、両者で割り切れる場合は「Fizz Buzz」**と言う。

  1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, Fizz Buzz, 16, 17, ...

準備

- 1. 考える
- 2. コードを書く準備

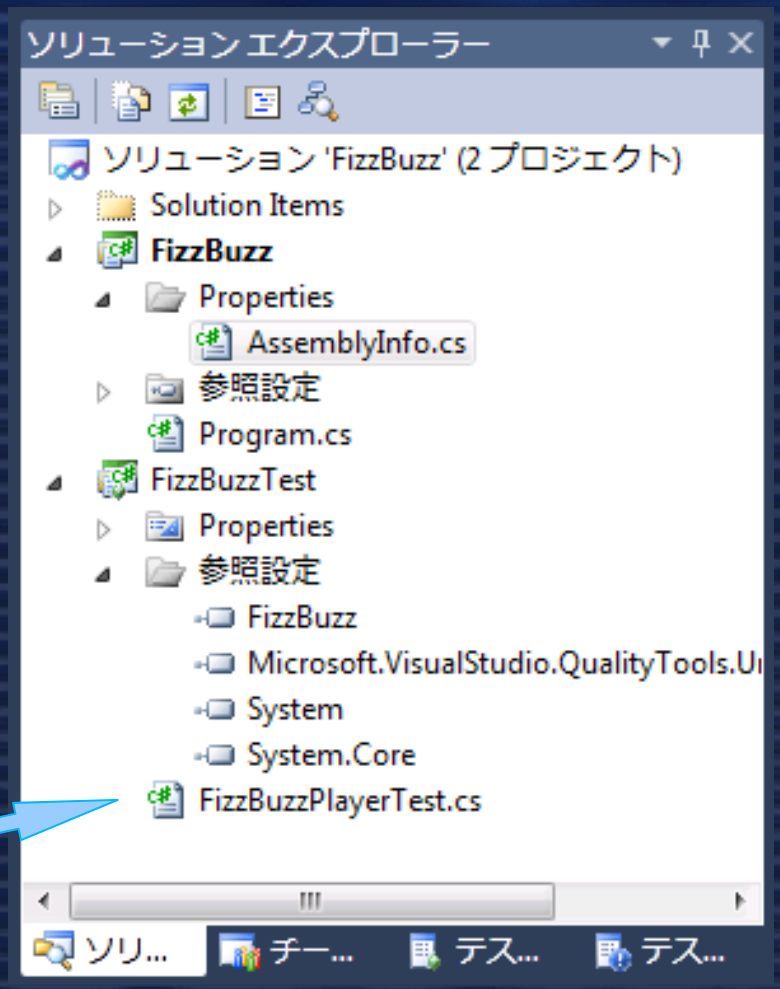
  Visual Studio なら

  2つプロジェクトを作り、

  テストコードのソースファ

  イルを作る。

  FizzBuzzPlayerTest.cs



TDD 開始: 1 ⇒ “1” が返る

```
[TestMethod]
public void TestSay1()
{
    FizzBuzzPlayer p = new FizzBuzzPlayer();
    Assert.AreEqual<string>("1", p.Say(1));
}
```

⇒ **コンパイルエラー! (RED)**
→ 製品コードのソースファイル FizzBuzzPlayer.cs を作る

```
internal class FizzBuzzPlayer
{
    internal string Say(int number)
    {
        return "1";
    }
}
```

"1" が返れば
いいのよ～ !!

2 ⇒ “2” が返る

```
[TestMethod]
public void TestSay2()
{
    FizzBuzzPlayer p = new FizzBuzzPlayer();
    Assert.AreEqual<string>("2", p.Say(2));
}
```

※ テスト追加時には
そのテストだけ実行

⇒ **テスト失敗! (RED)**
→ 2つのテストを同時に通すには? (**三角測量**)

```
internal class FizzBuzzPlayer
{
    internal string Say(int number)
    {
        //return "1";
        return number.ToString();
    }
}
```

※ 製品コード修正時には
全てのテストを実施

ちょっとテストをリファクタ

```
private FizzBuzzPlayer _testTarget;

[TestInitialize]
public void TestInitialize() {
    this._testTarget = new FizzBuzzPlayer();
}

[TestMethod]
public void TestSay1() {
    Assert.AreEqual<string>("1", this._testTarget.Say(1));
}

[TestMethod]
public void TestSay2() {
    Assert.AreEqual<string>("2", this._testTarget.Say(2));
}
```

安全索が無いので
慎重に!

コンストラクタは 1つだけ。

将来、増えることも無さそう。

3 ⇒ “Fizz” が返る

```
[TestMethod]
public void TestSay3Fizz()
{
    Assert.AreEqual<string>("Fizz", this._testTarget.Say(3));
}
```

```
internal class FizzBuzzPlayer
{
    internal string Say(int number)
    {
        if (number == 3)
            return "Fizz";

        return number.ToString();
    }
}
```

3 だったら "Fizz" を返せばいいのよ～ !!

4 ⇒ “4” が返る

- …でも、このテストは失敗しない (はず) なので、TDD 三原則により、テストを書かない。

5 ⇒ “Buzz” が返る

```
[TestMethod]
public void TestSay5Buzz()
{
  Assert.AreEqual<string>("Buzz", this._testTarget.Say(5));
}
```

```
internal class FizzBuzzPlayer
{
    internal string Say(int number) {
        if (number == 3)
            return "Fizz";

        if (number == 5)
            return "Buzz";

        return number.ToString();
    }
}
```

5 だったら "Buzz" を返せばいいのよ～ !!

6 ⇒ “Fizz” が返る

```
[TestMethod]
public void TestSay6Fizz() {
  Assert.AreEqual<string>("Fizz", this._testTarget.Say(6));
}
```

```
internal class FizzBuzzPlayer
{
    internal string Say(int number) {
        //if (number == 3)
        if (number % 3 == 0)
            return "Fizz";

        if (number == 5)
            return "Buzz";
        return number.ToString();
    }
}
```

3 の倍数だったら "Fizz"

※ 三角測量

もう分かってるけど、
テストを通すのに関係無
いから、こっちは我慢！

n = 7, 8, 9

- …でも、このテストも失敗しない (はず) なので、テストを書かない。

10 ⇒ “Buzz” が返る

```
[TestMethod]
public void TestSay10Buzz() {
  Assert.AreEqual<string>("Buzz", this._testTarget.Say(10));
}
```

```
internal class FizzBuzzPlayer
{
    internal string Say(int number) {
        if (number % 3 == 0)
            return "Fizz";

        //if (number == 5)
        if (number % 5 == 0)
            return "Buzz";

        return number.ToString();
    }
}
```

5 の倍数なら "Buzz"

※ 三角測量

n = 11, 12, 13, 14

- …でも、このテストも (以下略

15 ⇒ “Fizz Buzz” が返る

```
[TestMethod]
public void TestSay15FizzBuzz() {
  Assert.AreEqual<string>("Fizz Buzz", this._testTarget.Say(15));
}
```

```
internal class FizzBuzzPlayer
{
    internal string Say(int number) {
        if ((number % 3 == 0) && (number % 5 == 0))
            return "Fizz Buzz";

        if (number % 3 == 0)
            return "Fizz";
        if (number % 5 == 0)
            return "Buzz";
        return number.ToString();
    }
}
```

いきなり書いちゃったけど。

難しいときは三角測量!

ちゃんと動いた

- これ以上、失敗するテストは思いつかない
  = テストファースト終了 !!
  最後にリファクタリング

リファクタリング

```
internal string Say(int number)
{
    bool isMultipleOf3 = (number % 3 == 0);
    bool isMultipleOf5 = (number % 5 == 0);

    //if ((number % 3 == 0) && (number % 5 == 0))
    if (isMultipleOf3 && isMultipleOf5)
        return "Fizz Buzz";

    //if (number % 3 == 0)
    if (isMultipleOf3)
        return "Fizz";

    //if (number % 5 == 0)
    if (isMultipleOf5)
        return "Buzz";

    return number.ToString();
}
```

重複を排除するため

キャッシュ変数を導入

完成！

- 最後にオールグリーンを確認して、チェックイン!

```
internal string Say(int number)
{
    bool isMultipleOf3 = (number % 3 == 0);
    bool isMultipleOf5 = (number % 5 == 0);

    if (isMultipleOf3 && isMultipleOf5)
        return "Fizz Buzz";

    if (isMultipleOf3)
        return "Fizz";

    if (isMultipleOf5)
        return "Buzz";

    return number.ToString();
}
```

TDD しよう!

- プログラミングを覚えたときと同じ

TDD も 練習あるのみ！

ありがとうございました

バグは付きもの

- しばらくして…
  偉いシト 「biac くん、いつぞやのアレだけど。 FizzBuzz って 1 から始めるもんだろう? 間違って 0 とか突っ込んだらエラーにしてくれないと困るよ!」

  … **orz**

バグ対応

- TDD 的には、

バグ = テストケースの不備

- 先にテストを直してから、
  それに通るように製品コードを修正する。
  バグ対応も、RED ⇒ GREEN！

0 ⇒ 例外が出る

```
[TestMethod]
[ExpectedException(typeof(ArgumentOutOfRangeException))]
public void TestSay0Error() {
  string result = this._testTarget.Say(0);
  Assert.Fail("例外が発生せず、'{0}'が返りました。", result);
}
```

```
internal string Say(int number)
{
  if (number < 1)
    throw new ArgumentOutOfRangeException("number", number,
                                          "FizzBuzz ゲームは 1 から始まります。");

  bool isMultipleOf3 = (number % 3 == 0);
  bool isMultipleOf5 = (number % 5 == 0);

  if (isMultipleOf3 && isMultipleOf5)
    return "Fizz Buzz";

  // ...
```

- ありがとうございました