ClearCode

テスティングフレームワークに必要なもの

書きやすさとデバッグのしやすさ

須藤功平
株式会社クリアコード

2009/11/26

流れ

- 自己紹介
- テストは当たり前
  - ディスカッション
- テスティングフレームワーク
  - ディスカッション
- ~~実装~~
- ディスカッション

自己紹介

みんなが！

テンプレート:

- ✓ 名前
- ✓ 開発中のフリーソフトウェア
- ✓ テストに関して一言

口を開く

> 最初に喋らなかったら、ずっと何も喋らなくてもいいという暗黙の了解を得たと思ってしまう。

［「アジャイルレトロスペクティブズ」より引用］

自己紹介

- プライベート
  - フリーソフトウェア開発者
- 仕事
  - 代表取締役
  - （フリーソフトウェア）開発者

フリーソフトウェア: メイン

- Cutter
  - C/C++用xUnit
- test-unit 2.x
  - Ruby用xUnit
- GaUnit, Pikzie, Rabbit, rcairo, Ruby-GNOME2, ActiveLdap, milter manager, ...

フリーソフトウェア: 参加

- Ruby
  - RSSライブラリのところ
- Subversion
  - Rubyバインディングのところ
- groonga: 全文検索エンジン
  - テストやRubyバインディング

テストは当たり前

- ✓ 自己紹介
- ✓ テストは当たり前
  - ✓ ディスカッション
- ✓ テスティングフレームワーク
  - ✓ ディスカッション
- ✓ ~~実装~~
- ✓ ディスカッション

ここでのテスト

- ✓ 開発者が書くテスト
- ✓ 自動化されたテスト
- ✓ （主に）単体テスト

質問(1)

参加したプロジェクトにテストが

- なかった！
  - そんなもん？ありえない？
- あった！
  - 優秀だね？普通だね？

質問(2)

自分のソフトウェアにテストが

- ✓ ある
- ✓ ない
- ✓ あるといいよねぇ

差は何か

壁

どうしてこうなった

壁を越えるには

- ✓ きっかけ
- ✓ 方法
- ✓ どうして越えられないか

壁を越えるきっかけ

テスト？

世間では
どういう扱い？

テストはストレス？

> ソフトウェア開発プロジェクトにおいてテストは極めてストレスに満ちています。

[「テンプレートから学ぶ　受注する開発者のためのテスト仕様書（1/3）：CodeZine」より引用]

バグ探しは不愉快？

> 「テストとは作った成果物に誤りがあるかどうかを見つける作業だ」という本質的に不愉快な活動であること

[「テンプレートから学ぶ　受注する開発者のためのテスト仕様書（1/3）：CodeZine」より引用]

テストはきつい作業？

> プロジェクトの終わりにさしかかって時間も逼迫しているのに仕様変更を受けて再テストなどという、体力的にも精神的にもきつい作業であるからです。

[「テンプレートから学ぶ　受注する開発者のためのテスト仕様書（1/3）：CodeZine」より引用]

とてもネガティブ

- ✓ バグ探しは不愉快
- ✓ テスト実行はきつい

ネガティブなものなの？

フリーソフトウェアでは

- ✓ バグレポートは感謝
- ✓ アイディアの提案は歓迎
- ✓ テストは自動化
  （あるいはあまりしてない）

完璧？

自分が完璧だという
前提に立っていないか

> 「テストとは作った成果物に誤りがあるかどうかを見つける作業だ」という本質的に不愉快な活動であること

[「テンプレートから学ぶ　受注する開発者のためのテスト仕様書（1/3）：CodeZine」より引用]

現実と向き合う

完璧ではない自分が
よいソフトウェアを書くには
どうしたらよいか

壁を越えるきっかけ

ClearCode

現実と向き合う

壁を越える方法

テスト


自動化

さわると壊れるコード

- ✓ 頭では影響範囲を網羅できない
- ✓ バグ修正が新たなバグを誘発
- ✓ 変更するたび手動で再テスト
  - ✓ ストレス・人為的ミス

実装完了 → 手動テスト開始

- 実装時
  - たんたんと実装
- 手動テスト時
  - バグ出し→デバッグ
  - 修正→再テスト
  - 人為的ミス

デバッグのタイミング

- 実装してすぐ
  - 覚えているから直しやすい
- 後でまとめて
  - 思い出すのが大変

すぐに直す

- すぐにバグを見つける
  - 頻繁にテストする（大変）
- 常にリリースできる状態を維持

テストを自動化

- 実行コストが低い
  - 頻繁にテストできる
  - 人為的ミスを防げる
- テスト開発コストがかかる

自動化のコスト

継続するなら
割に合う

壁を越える方法

ClearCode

テストを自動化

どうして
壁を
越えられ
ないか

すばらしい世界

- ✓ ×: テストを書けば…
- ✓ ○: テストを活かせば…

目的

- ✓ ×: テストを書くこと
- ✓ ○: よいソフトウェアを書くこと

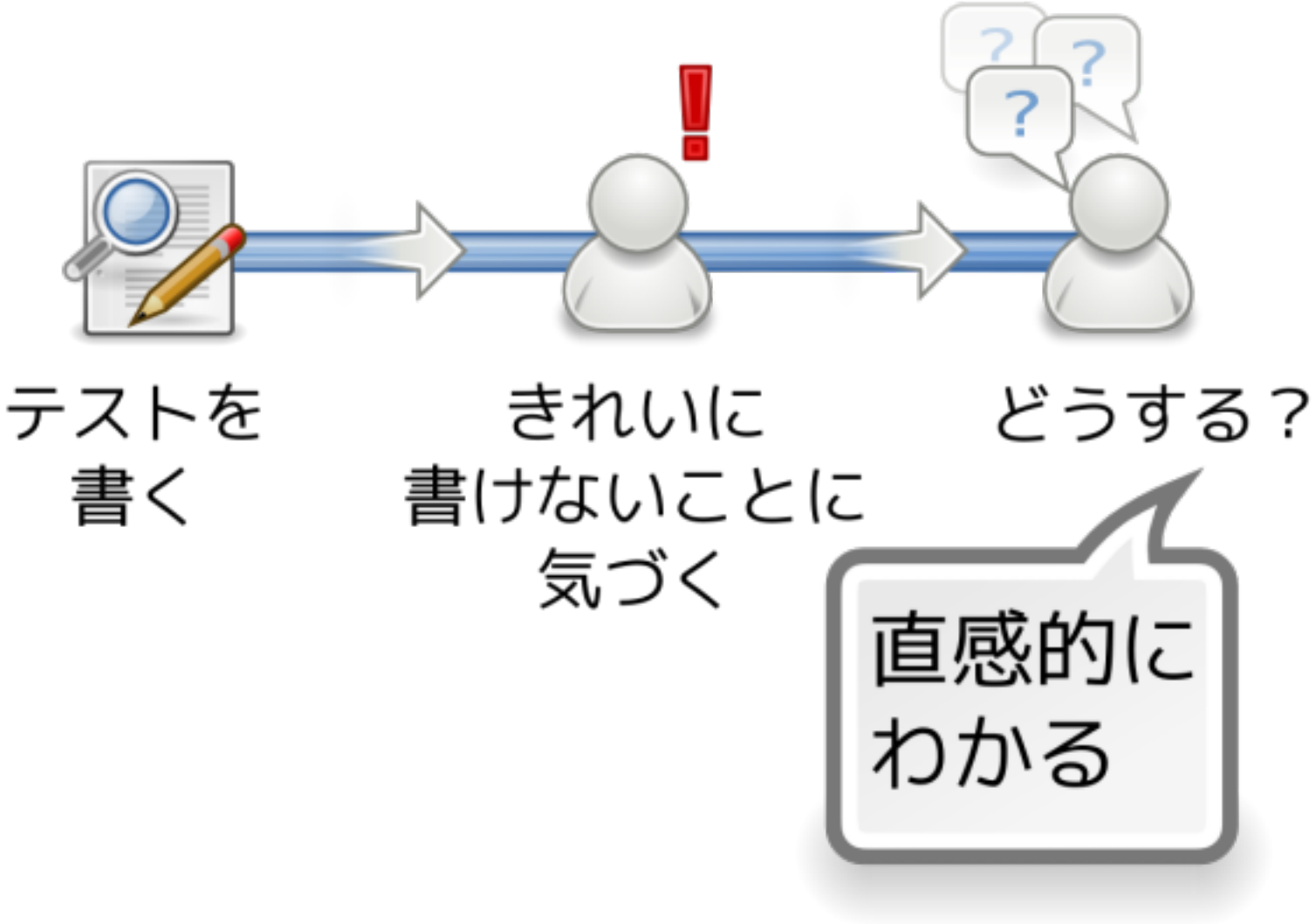
TDDですべて解決？

- ✓ TDDは身につけられる習慣
- ✓ ×: 使えば必ずうまくいく
- ✓ ○: よいソフトウェアが目的
  - ✓ どうしてTDDしているかを忘れない

よいソフトウェア？

- ✓ 必要な機能を提供している
- ✓ 余計なことをしていない
- ✓ 問題発生時に原因がわかる
- ✓ その他
  - ✓ オブジェクト指向: 適度な結合度

テストを活かす



どうして壁を越えられないか

- ✓ テストを書くことが目的
- ✓ テストを活かしていない
  - ✓ テストで問題に気づかない
  - ✓ 解決法が直感的にわからない

壁のまとめ

- 越えるきっかけ
  - 現実と向き合う
- 越える方法
  - テストを自動化
- 越えられない場合
  - テストを活かしていない

ディスカッション

- ✓ 壁を越えていますか？
- ✓ どうやって越えましたか？
- ✓ 越えたプロジェクトは？

ClearCode

テスティング フレームワーク

- 自己紹介
- テストは当たり前
  - ディスカッション
- テスティングフレームワーク
  - ディスカッション
- 実装
- ディスカッション

質問

テスティングフレームワーク…

- ✓ 使ってる？
- ✓ 何を使ってる？
- ✓ 満足してる？
- ✓ よいところは？悪いところは？

何を支援するか

テストを
活かすこと

一番大事なこと

無理をしない

無理をしない

- ○: 新しいコードから始める
- ○: 変更するコードから始める
- ×: テストを書くだけの作業
  - テストを書くことが目的になる

続けることが大事

- 完璧を目指さない
  - 続けられるペースをキープ
  - テストをストレスにしない
- 最初に自動化
  - 継続するなら割に合う

必要な機能は何か

- ✓ 書きやすさ
  - ✓ テストを作りやすい
- ✓ デバッグのしやすさ
  - ✓ テストを活かしやすい

書きやすさの目的

ClearCode

無理せず
続けるため

書きやすさ？

- テストだけ書けば動く
- 面倒なことも簡単に書ける
  - 高レベルな便利機能とか
- いつも通り書ける
  - 覚えることが少ない
- キレイなテストが書ける

必要な機能

フィクスチャ

- ✓ setup/teardown
- ✓ 初期化・終了処理を共有
- ✓ テストコンテキストを共有
  - ✓ テスト内容のパラメータ化

フィクスチャ: 実装

- ✓ だいたいある
- ✓ gauche.testにはない
- ✓ 使いこなせてないケースも多々
- ✓ 基本…でもない？

フィクスチャ: 変数

- 継承ベース
  - インスタンス変数
  - test-unit (Ruby), unittest (Python), CppUnit (C++), gtest (C++)
- モジュールベース
  - モジュール変数
  - Pikzie (Python), Cutter (C/C++), GaUnit (Gauche)

書くだけでテスト定義

- スクリプトだと当たり前
  - そうじゃないのもあるけど
- Cなどでは登録が必要
  - そうじゃないのもあるけど

nose (Python)

```
from nose.tools import *

def test_add():
    assert_equals(5, 2 + 3)
```

```
% nosetests test_add.py
```

Pikzie (Python)

```
import pikzie

def test_add():
    assert_equal(5, 2 + 3)
```

```
% python test_add.py
```

gtest (C++)

```
#include <gtest/gtest.h>

TEST(CalcTest, Add) {
  ASSERT_EQ(5, 2 + 3);
}
```

Cutter (C/C++)

```c
#include <cutter.h>

void
test_add(void)
{
  cut_assert_equal_int(5, 2 + 3);
}
```

データ駆動テスト

- テストデータのパラメータ化
- 動的にメソッド定義
  - 動的なスクリプト言語
  - RSpec (Ruby)
- データ登録
  - NUnit (C#), gtest (C++), Cutter (C/C++), UxU (JavaScript)

UxU (JavaScript)

```
testAdd.parameters = {
  plus: { x: 1, y: 2, expected: 3 },
  minus: { x: 1, y: -2, expected: -1 }
};

function testAdd(aParameter) {
  assert.equals(aParameter.expected,
                aParameter.x + aParameter.y);
}
```

ファイル操作

- 一時ディレクトリの作成・削除
  - きれいな環境を作るために便利
- スクリプト言語では豊富
- 言語標準でない場合
  - フレームワークで提供
  - 便利ライブラリをオススメ

StringIO

- フレームワークのテストに必要
  - Ruby, Python - StringIO
  - C++ - std::ostringstream
  - Cutter (C, GLib) - GIOChannel
- テストしやすいコードへ
  - IOオブジェクトをパラメータ化

外部プロセス

- ✓ コマンドのテストで便利
- ✓ 出力を文字列でとれると便利
- ✓ 入出力をやりとりできると便利
- ✓ 終わったら自動で強制終了
- ✓ 注: 遅い

イメージ

```
spawn("echo") do |process|
  process.write("hello\n")
  assert_equal("hello\n", process.gets)
end
```

マルチスレッド

- 同じテストを同時実行
- サブプロセスでやるのが安全
  - スレッドで壊れても影響を受けない
  - タイムアウトを設定しやすい
- 結果はプロセス間通信で渡す
  - CutterはXML

マルチプロセス

- 同じテストを多重実行
- サブプロセスでやるのが安全
  - SEGVっても影響を受けない
  - タイムアウトを設定しやすい
- 結果はプロセス間通信で渡す
  - CutterはXML

メモリ管理

- GC
  - スクリプト言語は標準搭載
- スコープは決まっている
  - テスト内のみ
  - 解放を自動化できる

Cutter (C)

```
void
test_strndup (void)
{
    const char *actual;
    actual = cut_take_string(strndup("abcdef", 3));
    cut_assert_equal_string("abc", actual);
}
```

書きやすさのまとめ

- 目的は無理せず続けるため
  - テストをイヤにならないように
- よく使う機能は標準で
- 面倒な操作用機能は標準で
  - 細かくエラー処理してあるとよい

ディスカッション

- ✓ 書きやすさのための機能は？
- ✓ こんな機能があるといいのに！

デバッグのしやすさの目的

無理せず
続けるため

いつデバッグ？

- 開発はデバッグの連続
  - テストは失敗するもの
- そんなことはない？
  - あなたが完璧！
  - 意味のないテスト？

テストは味方

- ✓ ストレスじゃない
- ✓ 負担をかけるものじゃない
- ✓ イヤイヤやるものじゃない

デバッグのしやすさ？

- テストの活かしやすさ
  ↑の方が適切だった
- 問題が何か見つけやすい
- よいコードに導く
  - 使い勝手の悪さに気づく
  - （ほんとは）
    アプリケーションを書いた方がよい

必要な機能

特定のテストだけ実行

- テスト名で指定
  - 完全一致や正規表現など
- テストケース名で指定
- 複数条件で絞り込めるとよい
  - テストケース名 → テスト名

テストを間引く

- ✓ 多くのテスト → 長い実行時間
- ✓ 興味のあるとこだけ実行したい
- ✓ 指定するのは面倒

間引き方

- 最近更新されたファイル周辺
  - 決まりが必要（Rails）
- 前に失敗したやつを実行
  - 成功するまで他に手をつけない！
- ランダムに実行
  - どこに影響があるかわからない
- ↑の合わせ技

途中で終了

- 1つでも失敗したら止めたい
  - すぐに確認したい
- C-cやキャンセルボタン
  - 対応: RSpec, test-unit, Cutter
  - 未対応: minitest

何が悪かった？

- 期待値と実測値の違いは？
  - 縦に並べる
  - diff
- なかったことを示すのは難しい
  - 正規表現がマッチしなかった
  - モックで呼び出されなかったのは？
  - CSSセレクタがマッチしなかった

違いは縦に並べる

```
<111011> expected but was <110111>

expected: <111011>
  actual: <110111>
```

diff

```
-(kou@delian.clear-code.com)-<09/10/19 13:32>---(...e/kou/work/c/cutter.clean)-
-[64935]% test/run-test.sh -n /equal_string_diff/      [~/work/c/cutter.clean]
FF

1) Failure: test_equal_string_diff_ascii
<"abcde" == "abCDe">
expected: <abcde>
  actual: <abCDe>

diff:
? abcde
?   CD

test/cutter/test-cut-readable-differ.c:29: test_equal_string_diff_ascii(): cut_a
ssert_equal_string("abcde", "abCDe", )

2) Failure: test_equal_string_diff_japanese
<"あいうえお" == "あいウエお">
expected: <あいうえお>
  actual: <あいウエお>

diff:
? あいうえお
?     ウエ

test/cutter/test-cut-readable-differ.c:23: test_equal_string_diff_japanese(): cu
t_assert_equal_string("あいうえお", "あいウエお", )

Finished in 0.018022 seconds (total: 0.001666 seconds)

2 test(s), 0 assertion(s), 2 failure(s), 0 error(s), 0 pending(s), 0 omission(s)
, 0 notification(s)
0% passed
-(kou@delian.clear-code.com)-<09/10/19 13:32>---(...e/kou/work/c/cutter.clean)-
-[64935]%                                              [~/work/c/cutter.clean]
```

どうして悪くなった？

- ✓ バックトレース
- ✓ assert失敗→ブレークポイント
- ✓ C/C++: マクロをさける
  - ✓ ステップ実行しづらい

ソースへジャンプ

- Emacs:

```
test/test-stack.c:10: assert_equal(...)
```

- Visual Studio:

```
test\test-stack.c(10): assert_equal(...)
```

特定用途向けassert

- ✓ 必要な情報を出すため

```
ok(File.exist?(path))
↓
assert_path_exist(path)
```

- ✓ pathはなに？

ディスカッション

- ✓ テストを活かすための機能は？
- ✓ こんな機能があるといいのに！

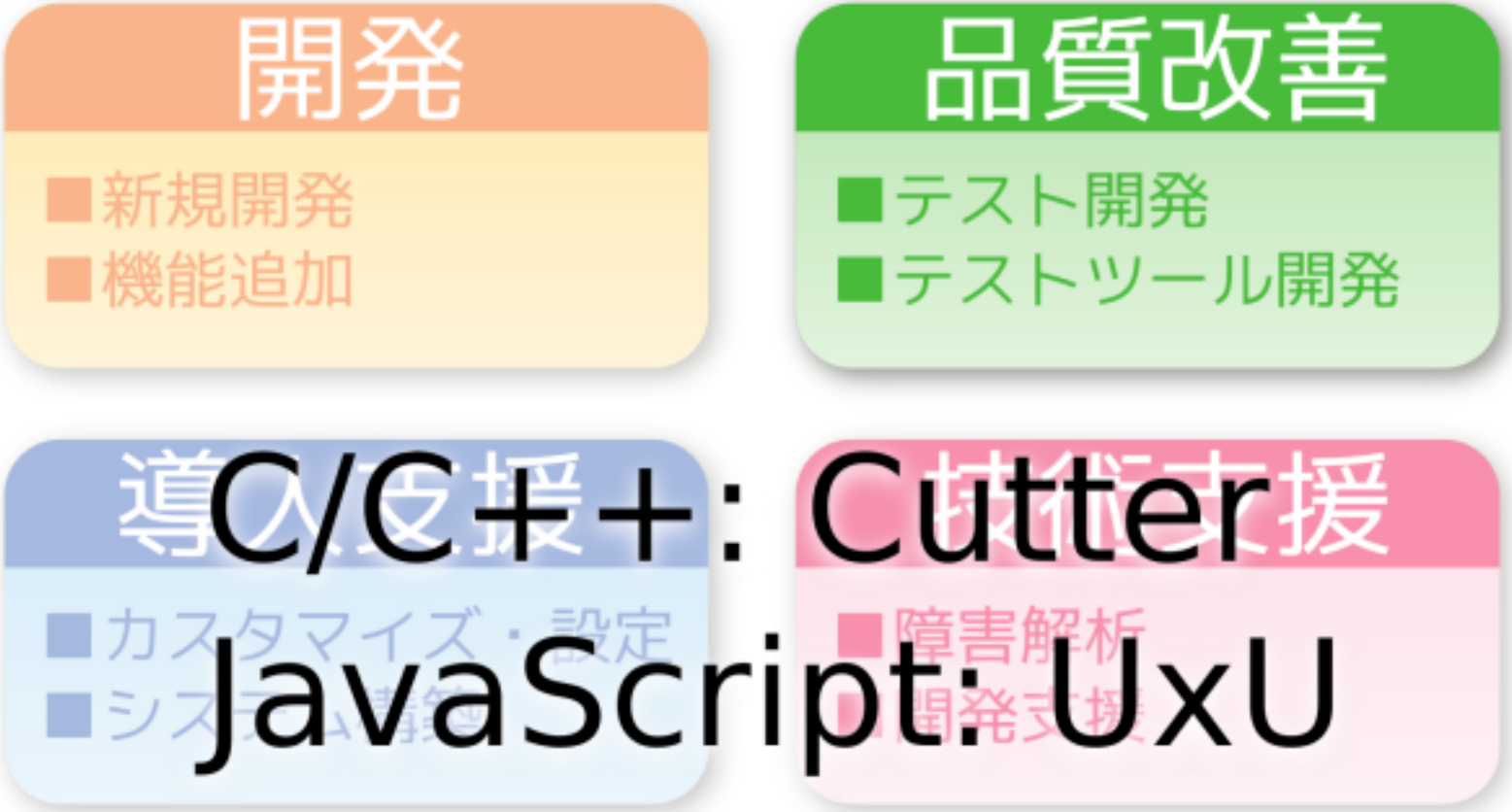
まとめ

- ✓ テストは活用するもの
- ✓ 無理をしない
- ✓ 無理せずテストを続けられる
フレームワークを使おう

明日からはじめる人へ

- テストを書くタイミング
  - バグ報告を受けたとき
  - 新機能を追加するとき
- いきなり無理をしない
  - はじめから完璧なテストは書けない
- まず自動化しておく

テストのことなら
クリアコードへ



お問い合わせ先: http://www.clear-code.com/contact/

ディスカッション

- ✓ 壁を越えるきっかけになった？
- ✓ テストは味方になった？