単体テストのすすめ

A recommendation of unit testing

KOMATSU Seiji (comutt)
atWare, Inc.
September 14, 2012

skomatsu [at] atware.co.jp
facebook.com/comutt
@comutt

About

- 名前: 小松 聖司
- 職業: システムエンジニア
- 職歴: 株式会社アットウェアにて、
  Web アプリ開発に携わること4年目
- 言語: 日本語, 英語, Java, Python, etc

Purpose

単体テストを
おすすめ・布教する

Timetable

- 19:00 - 19:30
  単体テストについて説明
- 19:30 - 20:00
  単体テストを実際に書いてみよう
- 20:00 - 20:15
  自動テスト（CI）について説明
- 20:15 - 20:30
  自動テスト（CI）の実演
- 20:30 - 20:40
  まとめ
- 20:40 - 21:00
  質疑応答、撤収

Target

- 対象
  - Lv.0 単体テストを知らない方
  - Lv.1 単体テストの書き方を知らない方
  - Lv.2 単体テストの書き方は知ってるけど、書くメリットがわからない方
- 非対象
  - Lv.3 単体テストを書くメリットは理解してるけど、面倒な方
  - Lv.4 ばりばり単体テストを書いてる方

Unit Test?

- ユニット（モジュール、クラス、メソッド）単位でテストするからユニットテストといいます
  （日本語訳: 単体テスト）
- アプリケーションを横断的・全体的にテストする結合テストに比べて、非常に小さい、軽量なテストです

Unit Test?

モジュールの数だけ、
テストコードを書きます

Unit Test?

パターンの数だけ、
テストコードを書きます

Example 1

コード

```
int sum(int n, int m) {
    return n + m;
}
```

テストパターン

- 1 + 1 = 2 // 正常系1
- -1 + -1 = -2 // 正常系2

Example 2

コード

```
// n を m で割った値を返す
int divide(int n, int m) {
    // 0除算防止
    if (m == 0)
        return 2147483647;

    return n / m;
}
```

テストパターン

- 9/ 2 = 4 // 正常系1
- -5 / 3 = -1 // 正常系2
- 1 / 0 = 2147483647 // 異常系

Point

- テストコードは正常系だけではなく、異常系も書く
  - アプリが「予期せずエラーを吐いて終了」そんな経験ありませんか？
    - テストコードを書けば、その可能性をぐんと減らせます

Point

- テストコードは、適切な粒度で書く
  - 実は何気に難しい
  - パターンを網羅することを意識しすぎると、冗長すぎるテストコードが完成します
  - テストコードがないことに比べれば1000倍良いですが、保守性を低下させるなどの弊害が生じます

Principle

テストコードは、

実装ではなく

仕様に基づいて書く

Reason

- 実装に基づいてテストコードを書いてしまうと、仕様違反を検出できません

Merit

- ソースコードレベルで仕様が明確になります
  - テストコードは仕様に基づいて書くため
  - 仕様書と辻褄の合わないソースコード、経験ありませんか？

Merit

- 機能追加やリファクタリングが容易になります
  - 機能追加・リファクタリングしたら、動作が変わってしまう（デグレ）かもしれない、という不安・リスクから解放されます

Optional Merit

- 引き継ぎのコストが下がります
  - テストコードで仕様が分かる
  - ソースコードを修正するリスクが低い

Principle

テストコードは
こまめに書く

Reason

- 間違っても、「アプリケーション組み上げたらまとめて書く」なんて考えてはいけません
  - テストによって見つけられるはずの早期不具合発見ができなくなる
  - 実装を忘れてしまうリスクがある
  - 往々にして、一気にテストコードを書くのは「苦痛」

More

- できる限り、関数・メソッド単位で
  テストコードを追加していきましょう
- つきつめると、
  TDD (Test Driven Development)、
  ないしはそれに近い形になります

Principle

テストコードは、
常にアップデートする

Reason

- 「仕様に基づいて書く」のだから、
  仕様がアップデートされるたびに、
  テストコードがアップデートされなければ
  なりません
  - そうしないと、
    テストコードが「腐る」
  - 自動テスト（CI）を活用する

1 Minute Break