ユニットテスト200X

つくりながら考えて、使いながらつくったpyunit

2006/06/30: オブジェクト倶楽部　夏イベント

渋川＠本田技術研究所

次回予告

「打倒リーンソフトウェア開発」
～ホンダフィロソフィ＋ソフトウェア開発～

あと３年以内には・・・

今回はテストのお話

- 最近ユニットテストってあんまり変化ないんじゃない？
  - XP 本出てから大分経つけど
  - モックオブジェクトぐらい？
  - BDD も、TDD と視点は変わるけど、できることはあんまり変わらないし

テスティングフレームワークの可能性

- Python 用に新しいテスティングフレームワークを作ってみました
- 使いながら、作りながらユニットテストの可能性を考えてみました
  - 理論よりも、現場、現物、現実

NUnit の@ほにゃほにゃという書き方が
羨ましかったのが一番の理由なんだけどね (^_^;

その名も

PyUnitX

なぜXが付くのか？

- 本家との差別化のため
- Xが最後についているのってかっこいいものが多い気がする

Acura/Honda NSX

Acura RDX プロトタイプ

写真は www.honda.co.jp より引用。引用にあたっては以下のルールに従って自己責任でやってます。
http://www.honda.co.jp/customer/website/pictures/homepage/index.html

とりあえずのチャレンジ目標

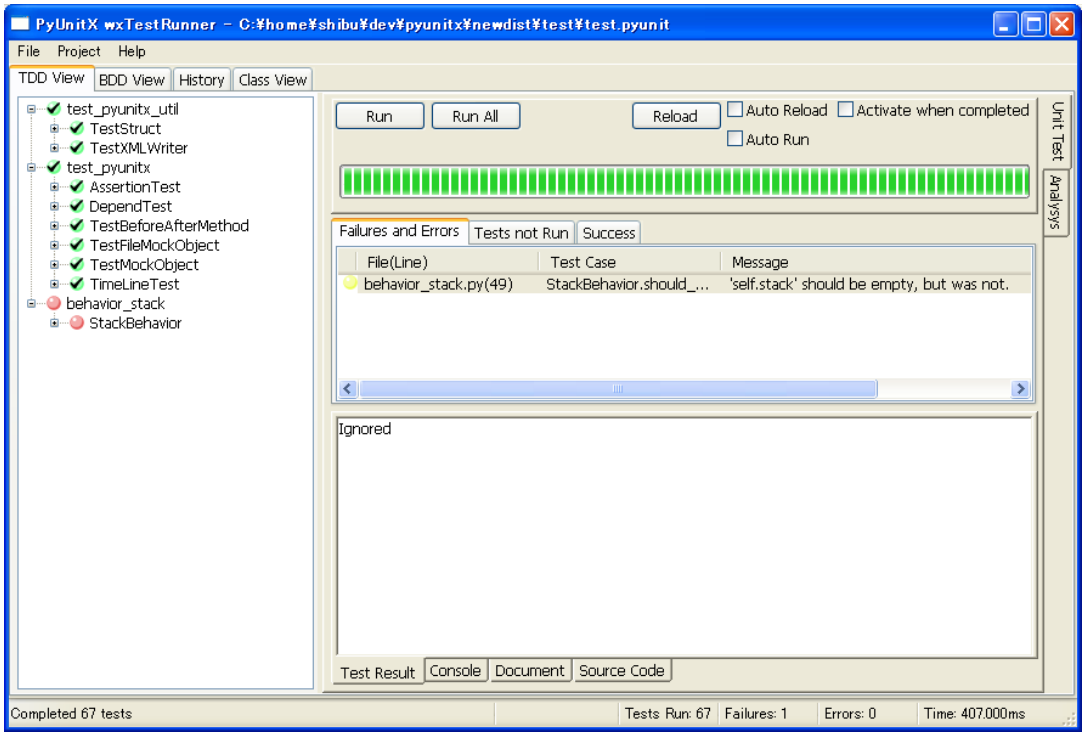
- 紙 (MS Office) で書くドキュメントをどれだけ減らせるのか？
- Python でメタプログラミングをいっぱいする

機能 (1)

テスト実行環境

GUI テストランナー

- wxPython で作ってみました



CUI 版もあるけど、

テスト実行のみ

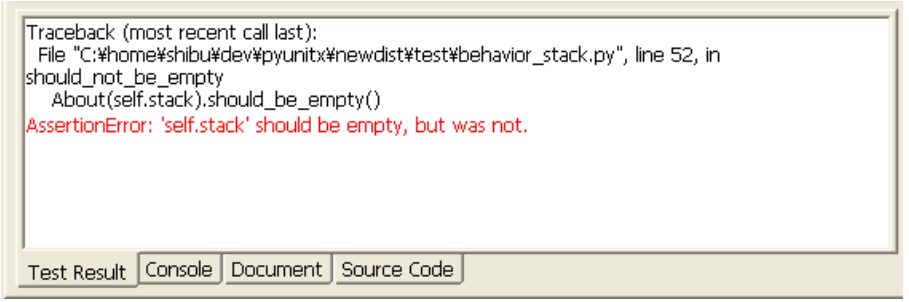
オートリロード機能

- メモ帳が IDE になる画期的な機能
- ファイルが変更されたら、関連するモジュールも含め自動的にリロード
- 自動実行して、成功 / 失敗にダイアログをアクティブ化するオプションもあり

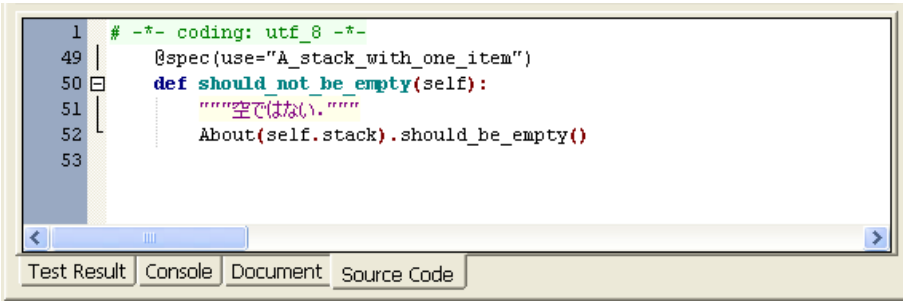
- でも、なかなかバグが一掃できません

テスト結果詳細タブ

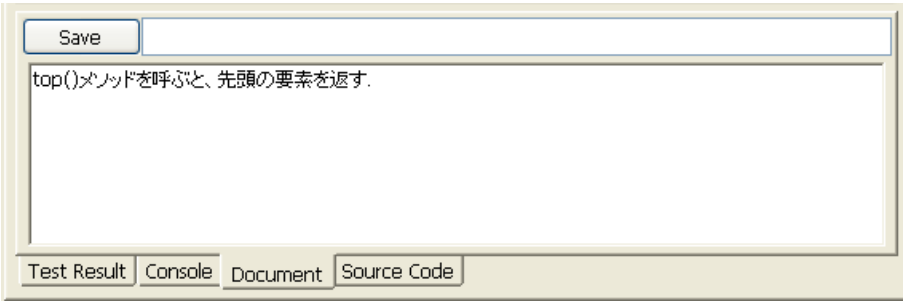
エラー詳細 (スタック情報付き)



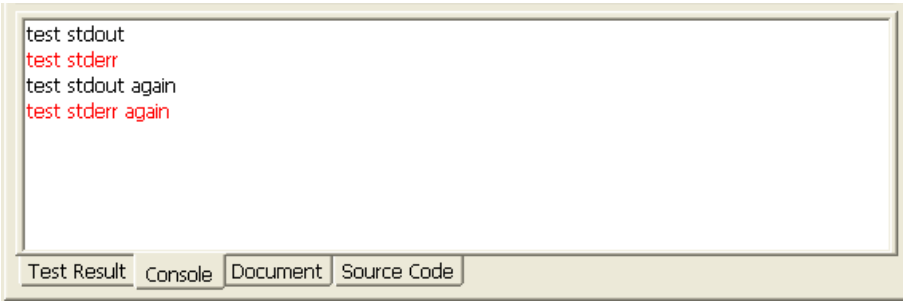
テストコードビューア (色付き)



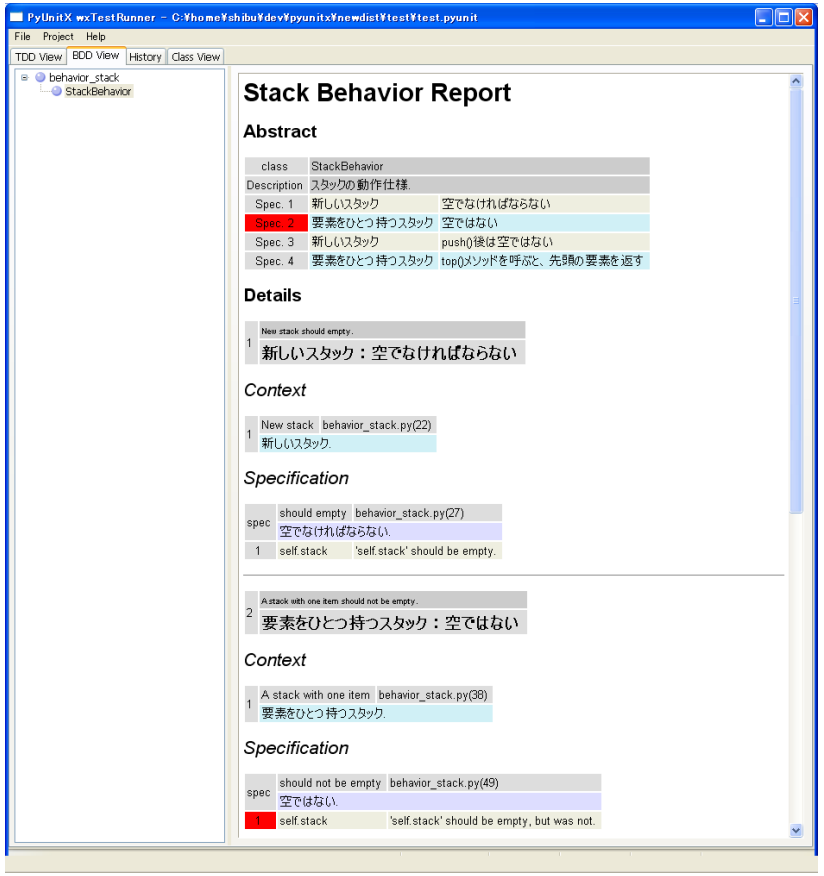
docstring(コメント) 表示



コンソール出力結果



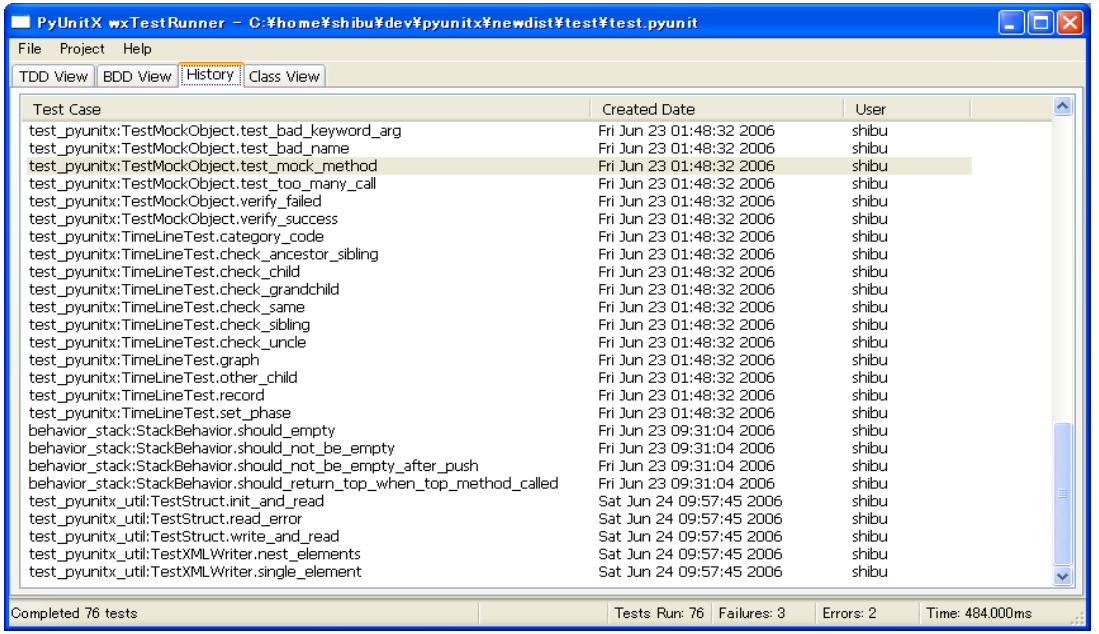
BDD レポート機能



- 動的にドキュメントを作成
  - メソッド名
  - docstring
  - 検証メソッド

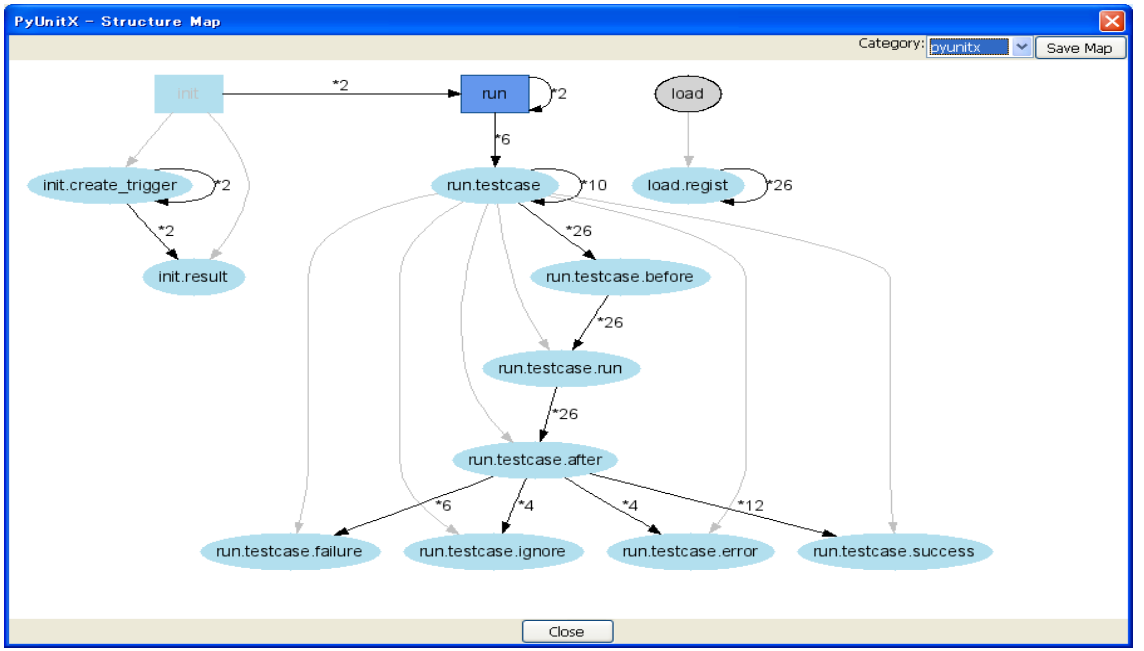
ヒストリー機能

- テストコードがどういう順序で構築されたのかが分かるように



ソフトウェア地図機能

- 機能的な全体像を表示



- フェーズ名を指定するコードは追加必要

ソフトウェア構造地図機能

- 木構造を記述可能
  - init.dialog.load_xml
- 今は見るだけだが、マインドマップで事前に作成して、比較検証することで構造のテストが可能になるかも・・・
  - JUDE でマインドマップ出力 API が整備されたら・・・・ね

機能 (2)

基本のテスト機能

テストの書き方

- デコレータというメソッド修飾構文を利用
- 継承はしない

```
class StackTest:
  @test
  def create_empty_stack(self)
    stack = Stack()
    Assert.are_equals(0, stack.size())
```

こんなのも真似して実装 (1)

- @before, @after
  - テストごとの準備と片づけ
- @beforeclass, @afterclass
  - １度だけ実行される準備と片づけ
- @test(expected= 例外クラス)
  - 例外が飛んでくるのを期待

こんなのも真似して実装 (2)

- @ignore
  - テスト実行をスキップ
- @test(timeout= タイムアウト時間)
  - 今はないけど、実装したい
  - そんなに使う予定もないけど

BDD 風オリジナルデコレータ

- @context
  - @before とほぼ一緒
- @spec(use=”コンテキスト名”)
  - ビヘイビアを記述
  - まぁ、内部の動作は 90% 以上 @test と一緒ですけど。

Assert の書き方

- Assert.are_equals(期待値、実際値)
- Assert.is_true(式)

とか、よく使いそうなものを一通り実装

BDD 風の書き方

- About(実際値).should_equal(期待値)
- About(式).should_be_true()

RSpec には負けるけど、基本の言語機能の範疇で、自分なりにがんばったつもり
→ .Net 方面や C++ の人も真似できます

モックオブジェクト

- 簡易モックオブジェクトも実装しました
  - 回数指定みたいな高度な機能はなし

```
mock = MockObject()
mock.add(1, 2).result(3)
mock.end_record()

About(mock.add(1, 2)).are_equals(3)
```

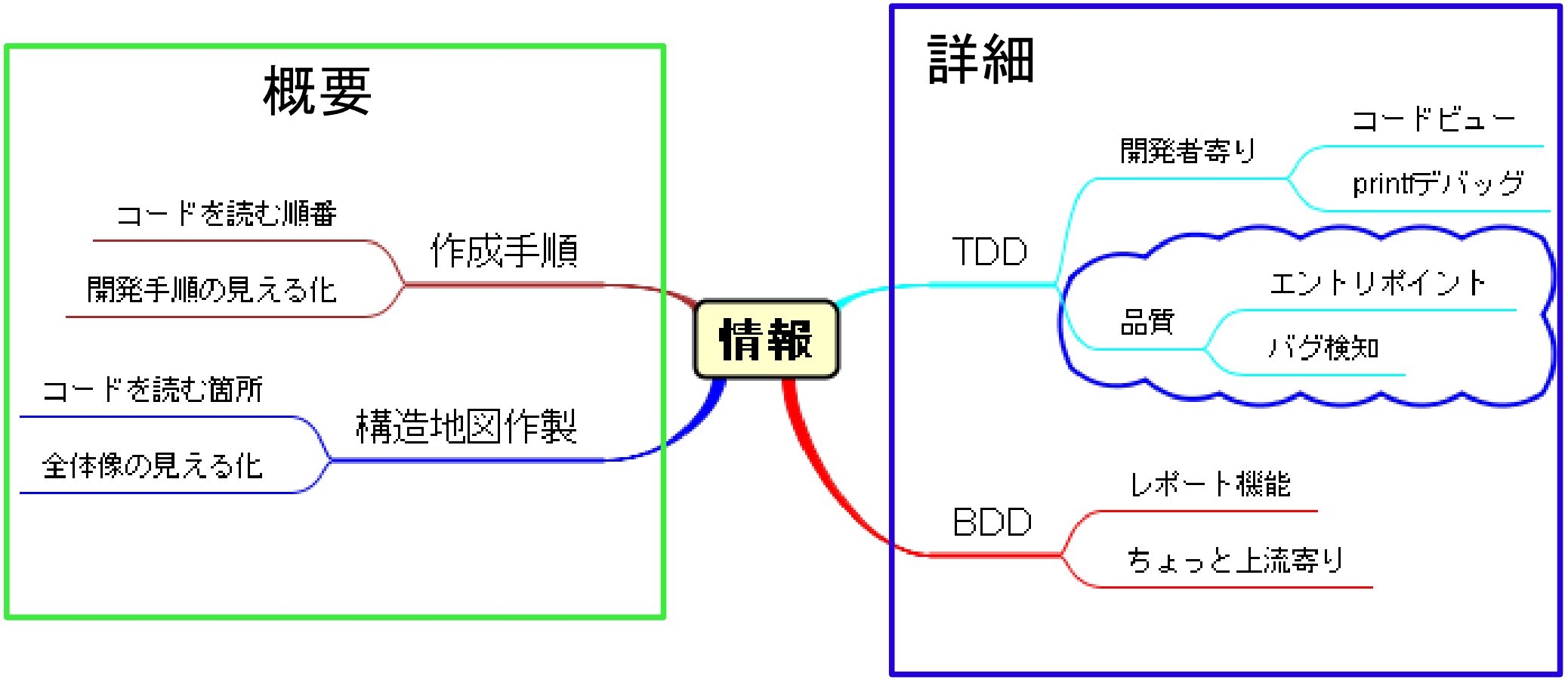
呼び方を記録

記録終了

試し実行

PyUnitX の扱う情報

扱える情報


概要
コードを読む順番
作成手順
開発手順の見える化
コードを読む箇所
構造地図作製
全体像の見える化
情報
詳細
TDD
開発者寄り
コードビュー
printfデバッグ
品質
エントリポイント
バグ検知
BDD
レポート機能
ちょっと上流寄り

今までのユニットテスト

- 動く仕様書とは言っていたけど・・・
  - 読むには情報が細切れで、詳細すぎ
  - 1000 個テストケースがあったとして、どこから読めばいいの？

読むという視点の TestRunner を利用することで、読めるユニットテストへ

改善余地はまだまだ

- ヒストリ
  - 見せ方を工夫したい
- 構造地図
  - 自動取得できる情報も扱って精度向上
- docstring 表示
  - マークアップ言語利用とか、Graphviz 連携とかを考えたい

他には？

ここから先はアイディアのみです

近日追加予定：複雑さをテスト

- リファクタリングしたら？を通知
  - if/for/while が X 個以上ネスト
  - メソッド行数が XX 行以上
  - ファイルの行数が XXXX 行以上
  - クラス、操作、属性の命名規則が変

QoEL のために (1)

- テスト実行可能時間帯を設定

```
EnvironmentError:  テスト実行時間エラー
    夜9時を過ぎました。早くお家に帰りましょう。
```

QoEL のために (2)

- 作業時間の管理のために

EnvironmentError:　休憩時間エラー

3時間連続で作業をしています。休憩が必要です
水分を取り、ストレッチをしてください。

EnvironmentError:　テスト連続実行エラー

今日は 10 時間以上テストが実行されています
残業は作業効率を下げます

開発プロセスを守るために

- プラクティスをテスト

```
Refactoring:  リファクタリングエラー
      既存のコードが1週間変更されていません
```

医療情報と連動

- 健康こそ QoEL の基礎

HealthError: 運動不足エラー
今日の開発は空気イスで行う必要があります。

会計システム等と連動して・・・

- お金から見える情報もテスト

HealthError:　お菓子買いすぎエラー
　今月は脂っこいお菓子を買いすぎです！

EnvironmentError:　地球温暖化エラー
　社用車をシビックハイブリッドに買い換えましょう

- 2006/07/03 に追加
  - スライドはここでおしまいです
  - ここまで来る前に銅鑼がなるだろう、という見込みで、まとめとか、そういう気の利いたものは作りませんでした
  - 本番では予定通り銅鑼がなりました