


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

渋谷@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
-

テストイングフレームワークの可能性

- Python 用に新しいテストイ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**

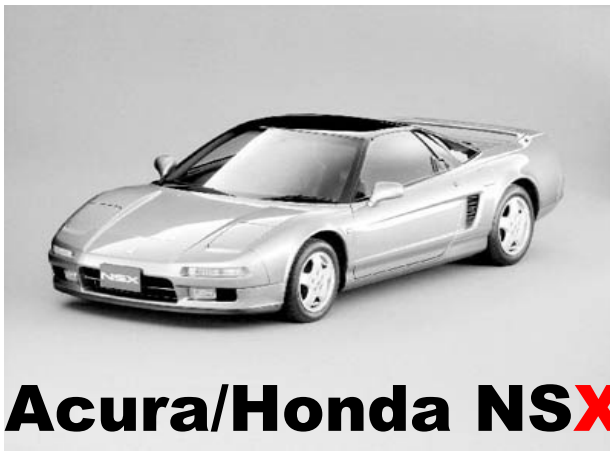
NUnit の@ほにゃほにゃという書き方が羨ましかったのが一番の理由なんだけどね (^_^;

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

とりあえずのチャレンジ目標

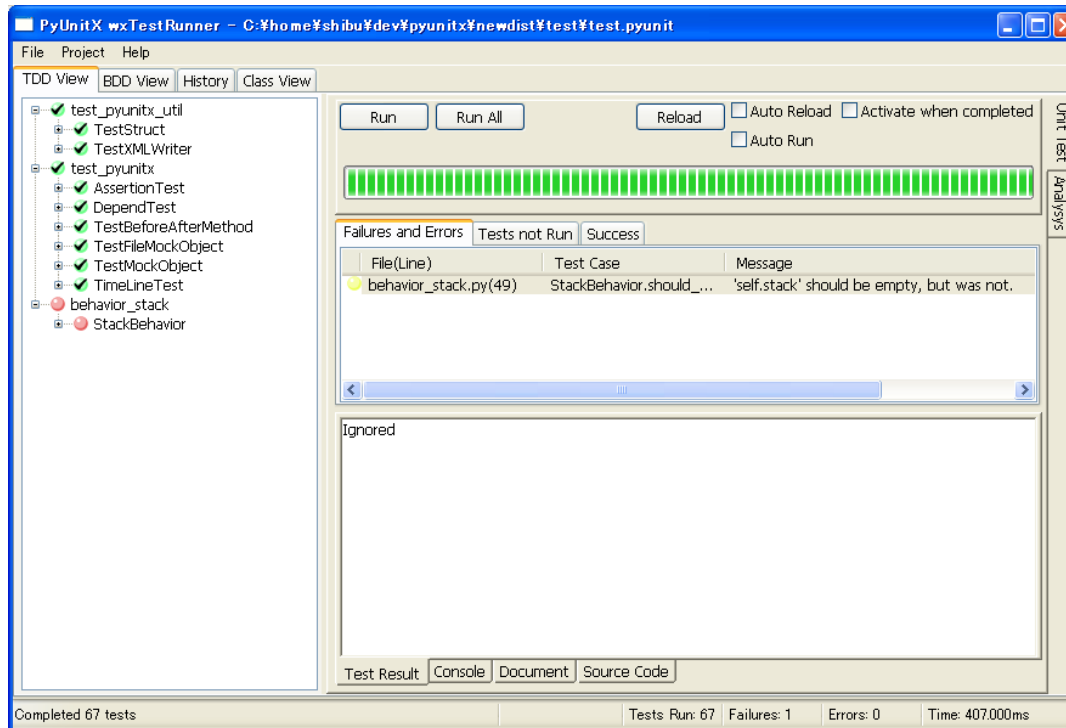
- 紙 (MS Office) で書くドキュメントをどれだけ減らせるのか？
 - Python でメタプログラミングをいっばいする
-

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)

```
Traceback (most recent call last):
  File "C:\home\shibu\dev\pyunitx\newdist\test\behavior_stack.py", line 52, in
  should_not_be_empty
    About(self.stack).should_be_empty()
AssertionError: 'self.stack' should be empty, but was not.
```

Test Result Console Document Source Code

テストコードビューア (色付き)

```
1  # -*- coding: utf_8 -*-
49 | @spec(use="A_stack_with_one_item")
50 | def should_not_be_empty(self):
51 |     """空ではない."""
52 |     About(self.stack).should_be_empty()
53 |
```

Test Result Console Document Source Code

docstring(コメント) 表示

Save

```
top()メソッドを呼ぶと、先頭の要素を返す。
```

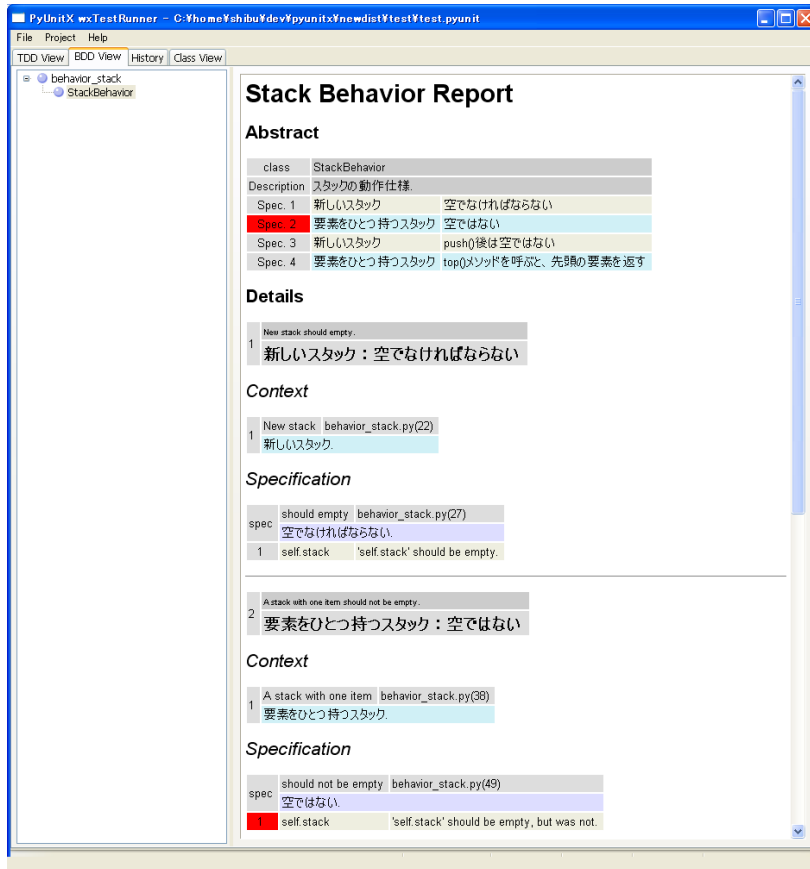
Test Result Console Document Source Code

コンソール出力結果

```
test stdout
test stderr
test stdout again
test stderr again
```

Test Result Console Document Source Code

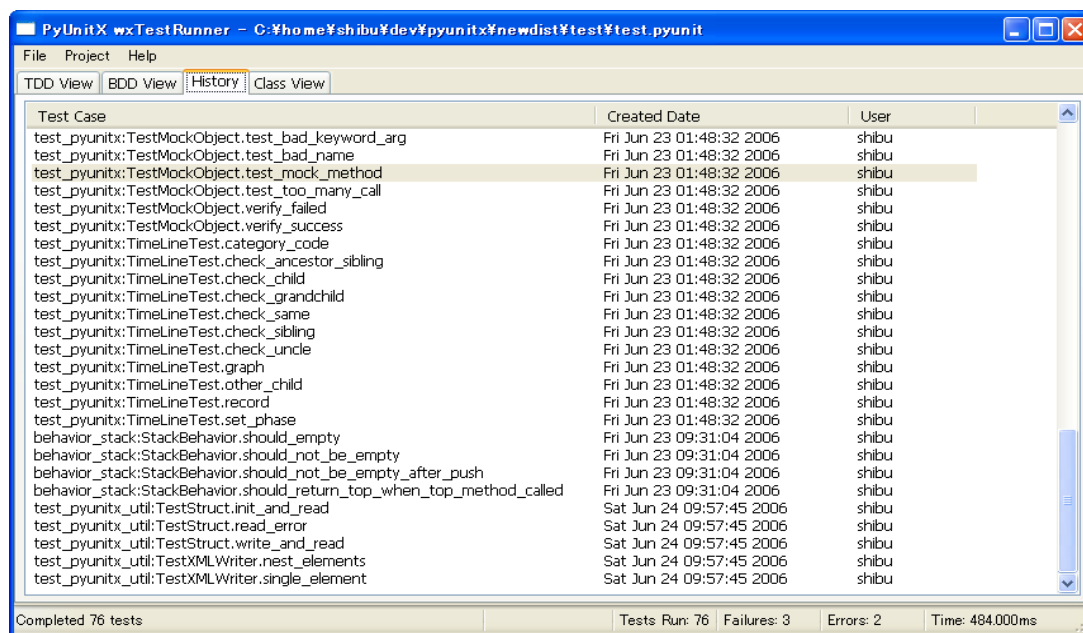
BDD レポート機能



- 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

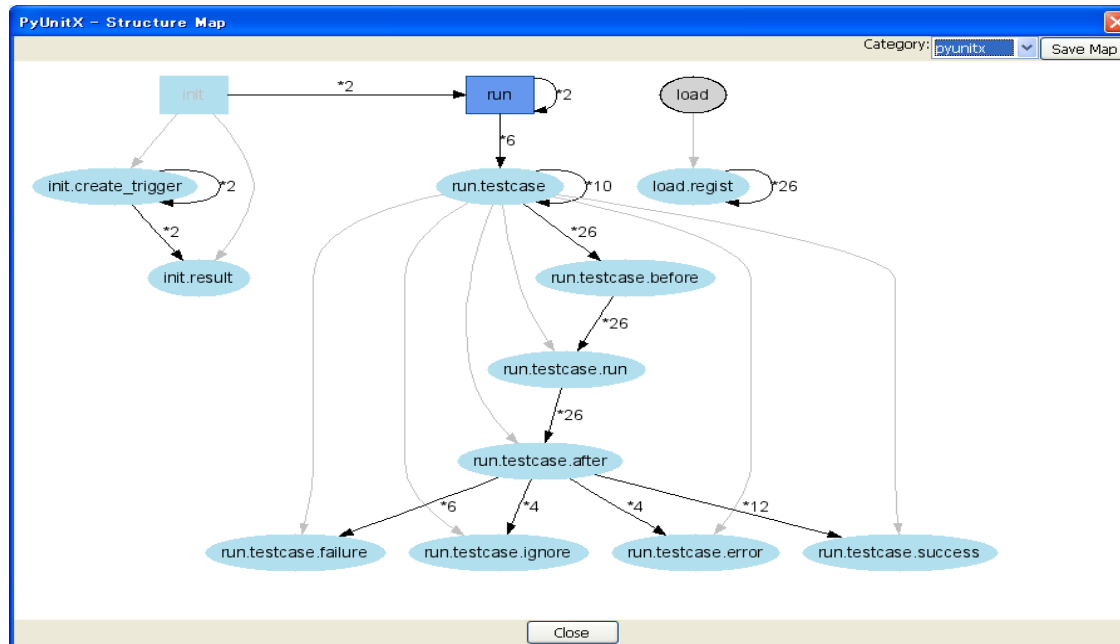


The screenshot shows the PyUnitX wxTestRunner application window. The title bar reads "PyUnitX wxTestRunner - C:\home\shibu\dev\pyunitx\newdist\test\test.pyunit". The menu bar includes "File", "Project", and "Help". Below the menu bar are tabs for "TDD View", "BDD View", "History", and "Class View", with "History" selected. The main area displays a table of test cases with columns for "Test Case", "Created Date", and "User". The table lists 30 test cases, including various tests for TestMockObject, TimeLineTest, and StackBehavior. At the bottom of the window, a status bar shows "Completed 76 tests", "Tests Run: 76", "Failures: 3", "Errors: 2", and "Time: 484.000ms".

Test Case	Created Date	User
test_pyunitx:TestMockObject.test_bad_keyword_arg	Fri Jun 23 01:48:32 2006	shibu
test_pyunitx:TestMockObject.test_bad_name	Fri Jun 23 01:48:32 2006	shibu
test_pyunitx:TestMockObject.test_mock_method	Fri Jun 23 01:48:32 2006	shibu
test_pyunitx:TestMockObject.test_too_many_call	Fri Jun 23 01:48:32 2006	shibu
test_pyunitx:TestMockObject.verify_failed	Fri Jun 23 01:48:32 2006	shibu
test_pyunitx:TestMockObject.verify_success	Fri Jun 23 01:48:32 2006	shibu
test_pyunitx:TimeLineTest.category_code	Fri Jun 23 01:48:32 2006	shibu
test_pyunitx:TimeLineTest.check_ancestor_sibling	Fri Jun 23 01:48:32 2006	shibu
test_pyunitx:TimeLineTest.check_child	Fri Jun 23 01:48:32 2006	shibu
test_pyunitx:TimeLineTest.check_grandchild	Fri Jun 23 01:48:32 2006	shibu
test_pyunitx:TimeLineTest.check_same	Fri Jun 23 01:48:32 2006	shibu
test_pyunitx:TimeLineTest.check_sibling	Fri Jun 23 01:48:32 2006	shibu
test_pyunitx:TimeLineTest.check_unde	Fri Jun 23 01:48:32 2006	shibu
test_pyunitx:TimeLineTest.graph	Fri Jun 23 01:48:32 2006	shibu
test_pyunitx:TimeLineTest.other_child	Fri Jun 23 01:48:32 2006	shibu
test_pyunitx:TimeLineTest.record	Fri Jun 23 01:48:32 2006	shibu
test_pyunitx:TimeLineTest.set_phase	Fri Jun 23 01:48:32 2006	shibu
behavior_stack:StackBehavior.should_empty	Fri Jun 23 09:31:04 2006	shibu
behavior_stack:StackBehavior.should_not_be_empty	Fri Jun 23 09:31:04 2006	shibu
behavior_stack:StackBehavior.should_not_be_empty_after_push	Fri Jun 23 09:31:04 2006	shibu
behavior_stack:StackBehavior.should_return_top_when_top_method_called	Fri Jun 23 09:31:04 2006	shibu
test_pyunitx_util:TestStruct.init_and_read	Sat Jun 24 09:57:45 2006	shibu
test_pyunitx_util:TestStruct.read_error	Sat Jun 24 09:57:45 2006	shibu
test_pyunitx_util:TestStruct.write_and_read	Sat Jun 24 09:57:45 2006	shibu
test_pyunitx_util:TestXMLWriter.nest_elements	Sat Jun 24 09:57:45 2006	shibu
test_pyunitx_util:TestXMLWriter.single_element	Sat Jun 24 09:57:45 2006	shibu

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
 - まあ、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()
```

呼び方を記録

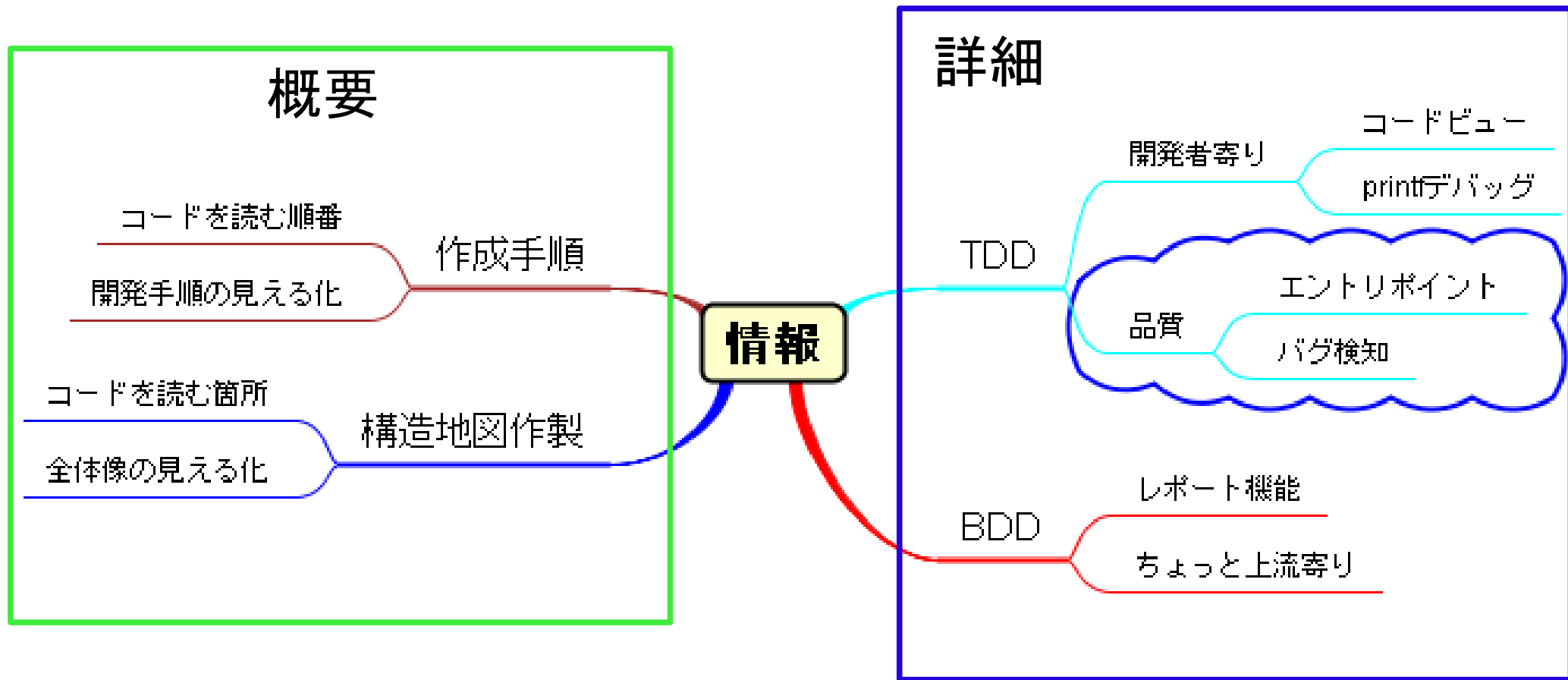
記録終了

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

試し実行

PyUnitX の扱う情報

扱える情報



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
