ソフトウェアテストシンポジウム2005 (JaSST2005, 2005年1月24日)

受け入れテストフレームワーク FIT/FitNesseによる高生産性開発

鷲崎 弘宜
情報・システム機構 国立情報学研究所
http://www.washizaki.net/

FIT (Framework for Integrated Test)

- 受け入れテストに使用可能なテストフレームワーク
  - オープンソース
  - HTML表によるテスト記述
  - テスト実行の自動化
  - 様々なプログラミング言語をサポート: Java, C#/.NET, Python, Lisp, Perl, Ruby (C++は鋭意対応中?)
  - XPとの相性が良い
- 開発者: Ward Cunningham
  - Kent Beckと共にパターンランゲージをソフトウェア開発に持ち込んだ人
  - eXtreme Programming (XP) の生みの親
- 入手先: http://fit.c2.com/

FIT: まずはさわってみる

- 最初の例: 四則演算プログラムのテスト arithmetic.html
- 何が起きるのか
  - HTML表の変化
    - 緑色/黄色/赤色: テスト成功/警告/失敗
  - HTML表以外の無変化

FitNesse: FIT+Wikiサーバ

- HTMLを編集するのは面倒・・・
- そこで、FitNesse
  - http://sourceforge.net/projects/fitnesse/
  - FITが組み込まれたWikiサーバ
  - オープンソース
  - インストールがとても簡単
  - Wikiによるテスト記述
  - Excelシートからの貼り付け
  - フィットネス (fitness) : 健康維持

こんな経験、ありませんか?

- 納期が迫ってきて、実装優先。テスト後回し。
- 後になって機能不足や欠陥 (バグ) が見つかる。
  - 顧客「ここ、私がお願いしたものと違いますよ」
  - 開発者 (・・・ 最初からきちんとテストしておけば良かった ・・・)
- 徹夜で修正作業。。。

では、どうすれば良いのか?

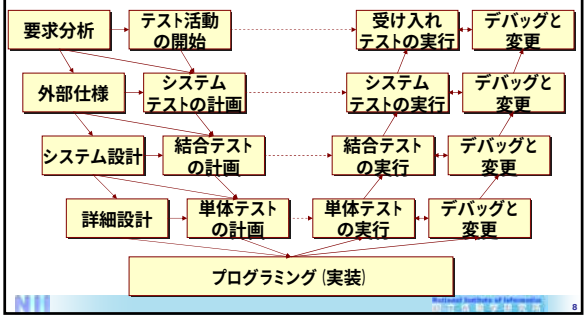
- プログラム開発の目的: 顧客が望む事柄をプログラムに実装すること
- ギャップ:
  - 顧客が望む事柄
  - プログラマが考えている「顧客が望む事柄」
- 受け入れテストが、そのギャップを埋める
  - 受け入れテストをしっかりやる→自動化
  - 「テストに通るように作る」→ 受け入れテスト駆動開発
  - 結果として生産性が向上

受け入れテストと構造テスト

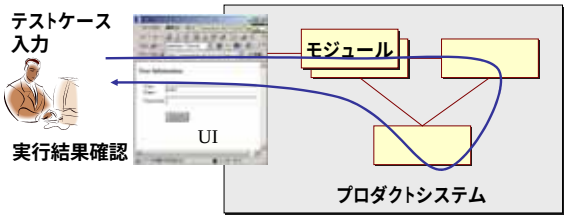
- 受け入れテスト
  - 開発しているプロダクトプログラムが、顧客の期待する内容を実装していることを、顧客が主体となって確認する作業
    - リリース時受け入れテスト
    - 配布時受け入れテスト
  - ブラックボックステスト: どのように実装されているかは考えない
  - フレームワーク: Canoo, Fit/FitNesseなど
- 構造テスト
  - 内部的な実装を、開発者が主体となって確認
  - ホワイトボックステスト: どのように実装されているかを検証する
  - フレームワーク: JUnitなど

VからWへ

- A. Spillner: *From V-Model to W-Model – Establishing the Whole Test Process*, 4th Conference on Quality Engineering in Software Technology, 2000

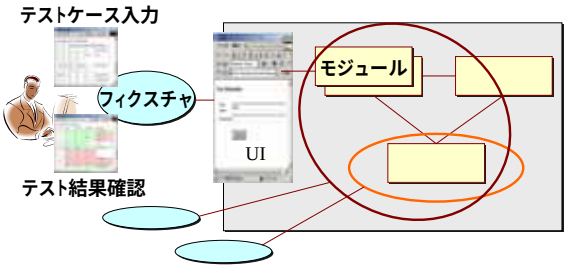


UI確認による受け入れテスト



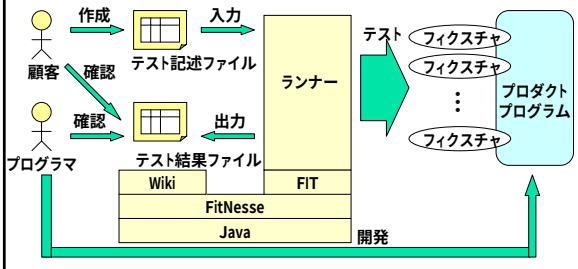
- 繰り返しの実行が困難
- 確認作業の信頼性が低い
- UI経由の大量のテストケースが必要

FITによる受け入れテスト



- 顧客が確認したい機能の集中的テスト
- 繰り返し自動実行

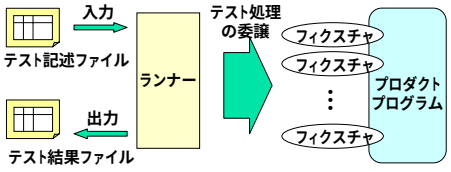
FitNesseの仕組み



- 顧客は、HTML表を含むテスト記述ファイルを作成
- プログラマは、プロダクトプログラムを開発
- プログラマは、フィクスチャを開発
- 顧客（またはプログラマ）は、テスト記述ファイルをランナーに入力し、テスト結果をテスト結果ファイルで確認

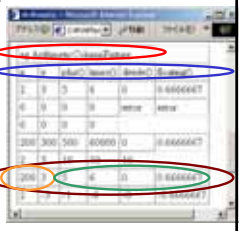
ランナー

- HTMLファイルのフィルタプログラム
  - テスト記述ファイルの<table>～</table>をHTML表として認識
  - HTML表で指定のフィクスチャに、HTML表の解釈処理を委譲
  - フィクスチャによる処理結果をテスト結果ファイルに書き出し
- 種類
  - fit.FileRunner: HTML表の扱い
  - fit.WikiRunner: Wikiタグで囲まれたHTML表の扱い



テスト記述ファイル

- 受け入れテスト内容をHTML表として持つWikiページ
  - テストケースを「例」として記述
- HTML表
  - 1つ目のセルでフィクスチャ指定
  - 2行目以降で、項目指定とテストケース記述
    - フィクスチャへの入力値
    - (実行結果としての) 期待値
- デフォルトで記述可能な型
  - 文字、文字列、数値リテラル、日付表現 (Date)



テスト結果ファイル

- 受け入れテストの結果が書き込まれたHTML表を持つ、自動生成されたWikiページ
  - HTML表以外はテスト記述ファイルと同じ
  - HTML表は「信号」
- セルの背景色
  - 緑色: 成功
  - 黄色: 警告 (例外の理由)
  - 赤色: 失敗 (期待値、実際の値)

フィクスチャ

- テストインタフェース: テスト記述ファイル内のHTML表と、プロダクトプログラムの結びつけ
  - どの列 (セル) の値が、入力値なのか
  - どの列 (セル) の値が、どの処理を行った結果の期待値なのか
- 作成方法
  - 既に用意されたFixtureクラスをそのまま利用
  - 継承して作成／利用
- 用意されているフィクスチャ
  - fit.Summary : 要約表示
  - fit.ColumnFixture: プロダクトへのまとまった入力と結果検証
  - fit.RowFixture : プロダクト中のデータ集合の検証
  - fit.ActionFixture : 他のアクタフィクスチャへの連続した入力と結果検証

などなど

受け入れテスト駆動

- 要求仕様をテスト記述ファイルとして記述し、テストに通るようにプロダクトを開発する
- 必ず受け入れテストを通るプロダクトが得られる！

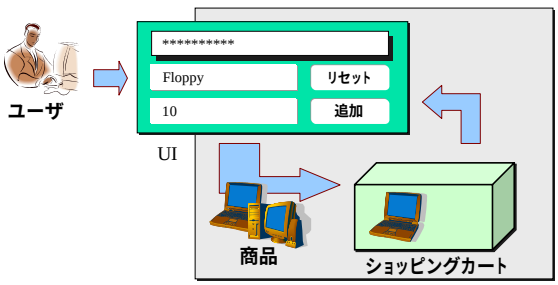
1. 顧客は、プログラムに求める機能をストーリーとして記述
2. 顧客は、ストーリーの具体的な様子をHTML表を含むテスト記述ファイルとして記述
3. 顧客は、受け入れテストを実行 → 失敗
4. プログラマは、HTML表を解釈するフィクスチャを開発
5. プログラマは、テストを実行 → 失敗
6. プログラマは、単体テスト駆動でプロダクトの開発
7. プログラマは、テストを実行 → 成功
8. 顧客は、受け入れテストを実行 → 成功

FitNesseの設計方針

- HTML表を含むWikiページが受け入れテスト単位
  - 顧客自身で作成可能 → 顧客を「巻き込む」
  - 表は、本質的に分かりやすい
  - 表以外の部分は自由に記述 → そのままドキュメント
- フィクスチャの用意
  - テストしたい機能・個所を集中的にテスト可能
  - 手間はかかる
- 受け入れテストの自動化
  - 繰り返し実行可能
  - 回帰テストの実現

実践例: ショッピングカート

- PC関連商品を、ショッピングカート式に選択／購入可能なシステム



ストーリーの作成

- 顧客が、必要な機能を自然言語で記述
- 優先順位付け

今回のターゲット

(1) カートへの商品の追加／確認

商品名と個数を指定して商品をカートに追加することができる。
カート内には、入力された商品がそれぞれ、入力時に
指定した個数の合計分存在する。

(2) グラフィカルな操作

ユーザは、全ての操作をグラフィカルに行うことができる。

表の作成

- 顧客が、機能が働く具体的な様子を表として記述

商品追加表

itemName	itemQuantity	valid
Floppy	10	true
Keybord	2	true
Mouse	0	false
Floppy	-10	false
Floppy	1	true

0以下の個数を
入力したら、
追加に失敗すること

追加後の、カート内の商品確認表

itemName	itemQuantity
Floppy	11
Keybord	2

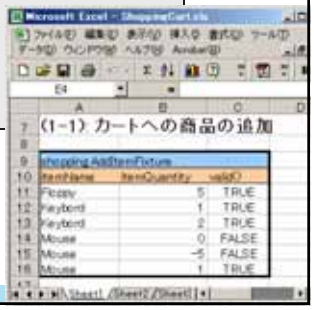
Wikiルールに従って記述／Excelシート

- Wiki

```
(1-1): カートへの商品の追加

|itemName|itemQuantity|valid()|
|Floppy|5|true|
|Keybord|1|true|
|Keybord|2|true|
|Mouse|0|false|
|Mouse|-5|false|
|Mouse|1|true|
```

- Excel

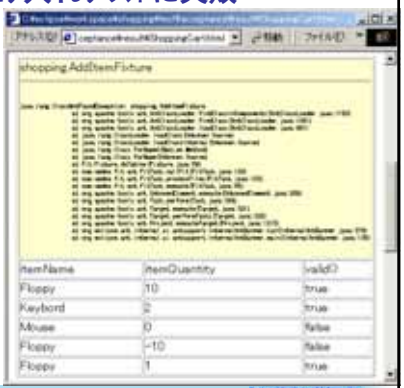


フィクスチャの指定

- プログラマが、
フィクスチャ指定の追加
  - 商品追加表
  AddItemFixture
  - カート確認表
  DisplayCartFixture
- 顧客が、
補足事項の追加

まずは受け入れテストに失敗

- フィクスチャが
存在しない
- そもそも、
プロダクトが
存在しない

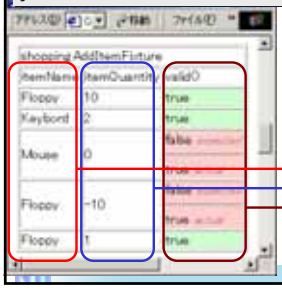


商品追加表のフィクスチャ実装

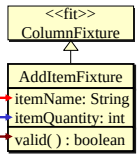
- 商品名 (itemName) と個数 (itemQuantity) を追加入力して、
カート内の状態変化 (valid()) を問い合わせる
- fit.ColumnFixtureを継承して作成

AddItemFixtureの空実装

```
package shopping;
public class AddItemFixture extends fit.ColumnFixture {
      public String itemName;
      public int itemQuantity;
      public boolean valid() { return true; }
}
```



- まずはフィクスチャの空実装
- テストに失敗



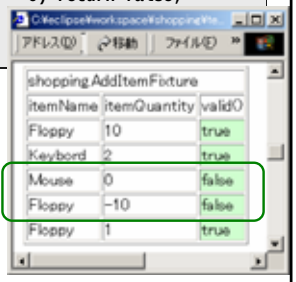
フィールドへの値入力

メソッドの実行、戻り値と期待値の比較

Fake It（まずは、テストに通す）

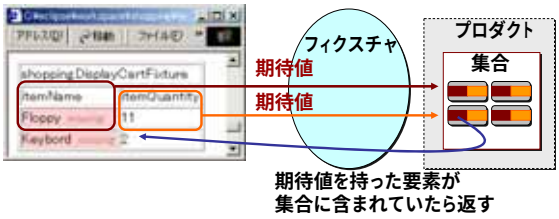
```
public class AddItemFixture extends fit.ColumnFixture {
      public String itemName;
      public int itemQuantity;
      public boolean valid() {
            if(itemQuantity <= 0) return false;
            return true;
      }
}
```

- テストケースのテストに成功する最も単純な実装
- 商品確認表のテストに（一応）成功
- カート確認表のテストに依然として失敗



カート内確認表のフィクスチャ実装

- 欲しいフィクスチャ
  - カート内の商品集合中に、表で指定した商品が指定した個数だけ存在するかどうかを問い合わせる。状態変更は不要。
  - fit.RowFixtureを継承して作成
- fit.RowFixture
  - 指定したフィールド値／メソッド戻り値を持つインスタンスが集合中に存在するかどうかを検索・検証

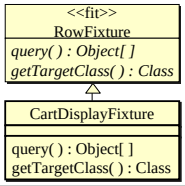


期待値を持った要素が集合に含まれていたら返す

DisplayCartFixtureの空実装

```
public class DisplayCartFixture extends fit.RowFixture {
      public Object[] query() throws Exception {
            return null;
      }
      public Class getTargetClass() { return null; }
}
```

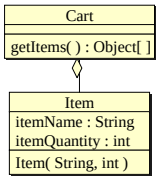
- カート確認表のテストに依然として失敗



メソッド	機能
Object[] query()	フィクスチャが参照する集合全体を配列として返す
Class getTargetClass()	参照する集合の構成クラスを返す

プロダクトプログラムの構想

- プロダクトプログラムの構成要素
  - カートを表すCartクラス
  - 商品を表すItemクラス
- 両クラスの関連
  - CartインスタンスはItemインスタンスの集合を持つ
  - Cartが集約するItemインスタンスの集合は、Cartクラスのget Items()メソッドによって取得できる



- DisplayCartFixtureの修正

```
public class DisplayCartFixture extends fit.RowFixture {
      static Cart cart = new Cart();
      public Object[] query() throws Exception {
            return cart.getItems();
      }
      public Class getTargetClass() {
            return Item.class;
      }
}
```


プロダクトプログラムの実装

```
public class Item {
      public String itemName;
      public int itemQuantity;
      public Item(String itemName, int itemQuantity) {
            this.itemName = itemName;
            this.itemQuantity = itemQuantity;
      }
}
```

```
public class Cart {
      public Object[] getItems() {
            return new Object[0];
      }
}
```

- しかし、テストに失敗

AddItemFixtureの再修正

- カート確認表のテストの失敗原因は、そもそも、商品追加表における操作によって、カートに商品が追加されていないこと
- Cartクラスに商品追加メソッド addItem() の追加

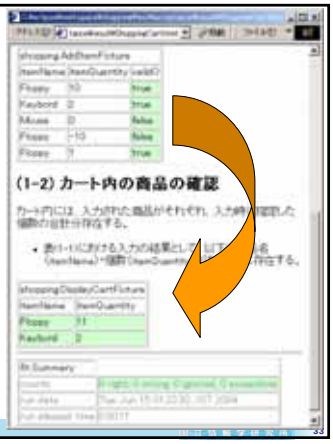
```
public class AddItemFixture extends fit.ColumnFixture {
      private Cart cart = DisplayCartFixture.cart;
      public String itemName;
      public int itemQuantity;
      public boolean valid() {
            return cart.addItem(itemName, itemQuantity);
      }
}
```


Cartの修正

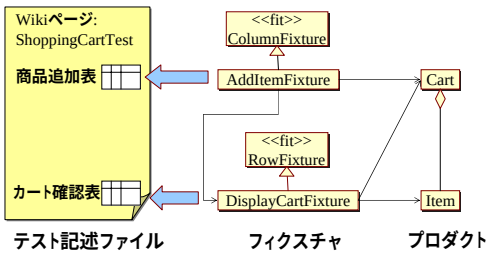
```
public class Cart {
  private HashMap itemMap; // 商品名と個数の対を保持するマップ
  public Cart() { itemMap = new HashMap(); } // マップの初期化
  public boolean addItem(String itemName, int newQty) {
    if(newQty <= 0) return false;
    Integer oldQty = (Integer) itemMap.remove(itemName);
    itemMap.put(itemName,
      new Integer((oldQty == null) ?
        newQty : oldQty.intValue() + newQty)
    );
    return true;
  }
  public Object[] getItems() {
    ArrayList items = new ArrayList();
    for(Iterator it=itemMap.keySet().iterator();it.hasNext();) {
      String key = (String) it.next();
      int value = ((Integer) itemMap.get(key)).intValue();
      items.add(new Item(key, value));
    }
    return items.toArray();
  }
}
```

全テストに成功！

- 商品追加表における追加の結果が、カート確認表のテスト時点で残っている



得られた設計

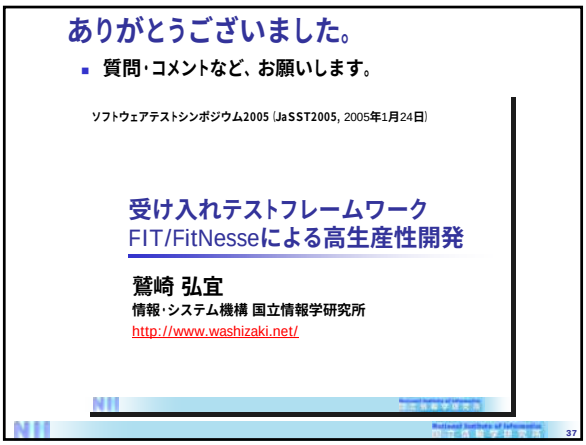


現実的な問題

- FITは、要求通りのプログラムを得るための魔法ではない
  - フィクスチャのテストが必要
  - FITは例外の扱いに弱い
  - プロダクトプログラムのユニットテストも忘れずに！
  - プロダクト側が持つべき機能を、フィクスチャに持たせないように注意
- GUIの受け入れテストは？
  - サーブレットコンテナにアクセスするような機能をFIT/FitNesseは持たない
  - MVCパターンなどを利用して、テスト対象をModelに集中させる
    - テスト容易性の向上
  - JWebUnit/HTTPUnit用のフィクスチャJWebUnitFixtureの活用
- DB接続の受け入れテストは？
  - DBの代わりとして振舞う機能をFIT/FitNesseは持たない
  - テスト時用のデータソースクラスを作成する、など

まとめ

- FIT/FitNesseで、顧客参加型の受け入れテストを実現
- xUnitがそうであったように、FIT/FitNesseはこれからのソフトウェア開発に大きな影響を与える可能性がある
- 情報源
  - 『最新テスティングフレームワークFITに迫る』(北野弘治氏、JavaPress、Vol.34、2004、技術評論社)
  - 『オープンソースJavaプロダクツ』(2004、技術評論社)
  - 『受け入れテストもEclipse』(鷲崎弘宜、Eclipseパーフェクトマニュアル、Vol.4、2004、技術評論社)


ありがとうございました。
質問・コメントなど、お願いします。
ソフトウェアテストシンポジウム2005 (JaSST2005, 2005年1月24日)
受け入れテストフレームワーク
FIT/FitNesseによる高生産性開発
鷲崎 弘宜
情報・システム機構 国立情報学研究所
http://www.washizaki.net/
37