1

テスト駆動開発のプロセス解析

JaSST'11 Tokyo
2011/01/25

中山裕貴, 大月美佳
(佐賀大学)

2

発表の流れ

- 研究の背景と目的
  - TDD
  - TDD初心者教育の問題
- プロセスデータを記録するEclipse用Plugin
- データ収集対象プログラム
- プロセス解析
- 解析自動化の検討
- 今後の予定

3

研究の背景

- テスト駆動開発(TDD)
  - ソフトウェア開発手法の一つ
  - Red/Green/Refactorを繰り返す
- TDD初心者教育の問題
  - TDDに関する知識不足
  - 仕様を理解しないまま実装
  - テスト項目の選び方、適用順番
- 経験者の技術の吸収は上達方法の一つ
  - 経験者は指導の時間がない
  - 経験者は少数で初心者は圧倒的多数＝指導が困難

本研究の対象

4

研究の目的

- 個々の開発プロセスを記録・解析
  - 開発現場を見なくても実装順序が把握可能
  - 開発プロセスに対して個々に指摘(フィードバック)を行える
- 解析、フィードバックの自動化 最終目的
  - 初心者が多くても対応可能
  - 経験者に負担がかからない

•開発プロセスを記録するEclipse用Plugin
•プロセス解析方法と解析結果
•解析の自動化について

5

開発プロセスの記録

- 開発中の画面を記録
  - 記録ミスの可能性
  - データサイズの肥大化
- バージョン管理システム(VCS)
  - コミット忘れ
  - VCS初心者には教育が必要⇒コスト
- 開発環境にプロセスを自動記録させる
  - EclipseのPlugin開発は比較的容易
    ⇒プロセスを記録するEclispe用Pluginの開発

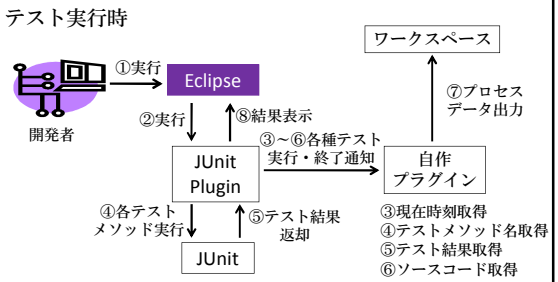
6

作成したPluginの仕様

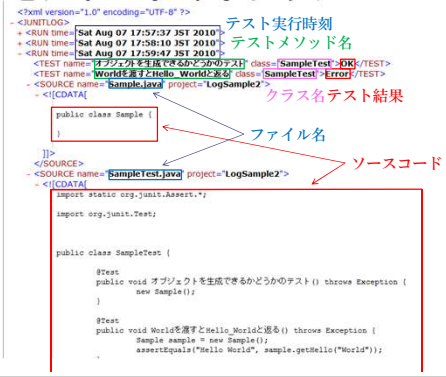
- 以下の項目をXMLファイルに記録
  - テスト開始時刻
  - 実行したテストメソッド名と所属クラス名
  - テスト結果
  - テストが属するプロジェクト下のソースコード
- 記録はJUnitによるテスト実行時に自動で記録
  - 時刻：テスト実行開始時
  - テスト結果：各テストケース実行後
  - ソースコード：全テストケース終了後

作成したPluginの動作

テスト実行時



プロセスデータのサンプル



作成したPluginの動作検証

- 簡単な仕様のプログラムをTDDで実装、記録
- TDDの開発サイクルに沿った開発を確認
  ⇒プロセス解析のためのデータとして利用可能

IDE	Eclipse(3.5以降)
必須Plugin	JUnit Plugin 3.5.2以降
JDK	Jdk1.6.0-21
OS	Windows XP, Vista, 7 Mac OS X 10.6 Ubuntu10.04LTS Fedora 11

表１：作成したPluginの動作検証環境

記録対象プログラム

- 10名程度の経験者に依頼
  - 職業プログラマ
  - TDD Boot Camp参加者
- 課題は以下の4種類
  - Hello Worldメソッド
  - 割り算メソッド
  - うるう年判定に関するクラス
  - ファイル入出力に関するクラス

プロセス解析方法

- 今回はすべて手作業で解析
- プロセスデータを項目ごとに色分け
- テスト回数や間隔、コード変更について解析

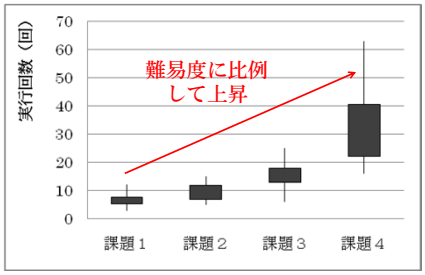
色分け対象項目	色
テスト実行時刻	青
テスト成功	緑
成功以外のテスト結果	赤
コードの追加・変更部分	赤
コード削除部分 (コメント付加)	橙

表２：プロセスデータに記録された項目と対応する色

プロセスデータの項目識別

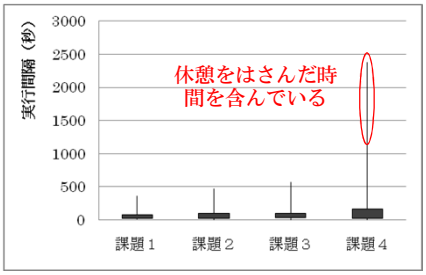
13

課題別テスト実行回数



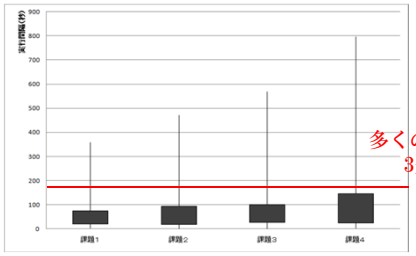
14

課題別テスト実行間隔



15

課題別テスト実行間隔（例外除外版）



16

解析結果

- ほとんど課題で具体的実装順序が得られた
- 課題4の一部クラスに関しては得られなかった
  - ➢実装が様々だった
  - ➢データ収集をやり直す必要あり
- 解析結果が一般的なTDDの傾向と同じ
  - ➢解析結果が信頼できる
  - ➢フィードバックに利用可能と判断
- 手作業では解析にコストがかかる
  ⇒解析の自動化

17

解析自動化の検討

- 手作業解析における各作業をすべて自動化
  - 時間、人員コストの削減
  - 解析における作業ミスの排除
- 自動化したい作業
  - テスト実行間隔の計算
  - 各課題におけるテスト実行回数の集計
  - ソースコードの差分取得
  - 各テスト実行時の作業内容判断
  - 各テスト実行時の粒度判断

構文解析＋diffを利用

18

なぜ構文解析とdiffを用いるか

例：下記プログラムのdiffを取って比較

- 変更前

```
Class Sample {
        public void sample() {
                System.out.println();
        }
}
```

- 変更後

```
Class Sample {
        public void sample(String text) {
                System.out.println(text);
        }
}
```

•引数を追加
•渡された文字列の出力を追加

19

なぜ構文解析とdiffを用いるか

- diffのみでの差分取得(例：行単位)

```
Class Sample {
        public void sample() {
        public void sample(String text) {
                System.out.println();
                System.out.println(text);
        }
}
```

赤：追加
橙：削除

行の追加・削除で変更を判別
⇒どのような変更か自分で考える必要がある
⇒自動での解析には不向き

20

なぜ構文解析とdiffを用いるか

- 構文解析とdiffをあわせた差分取得

```
Class Sample {
        public void sample(String text) {
                System.out.println(text);
        }
}
```

赤：追加
橙：削除

トークン単位で追加・削除を判別
⇒どのような変更かがより簡単・正確に把握可能
⇒自動での解析に向いている

21

今後の予定

- フィードバック項目の決定
- プロセス解析の自動化
  - 手作業の解析手順をもとに自動化
  - フィードバック内容に必要なデータ項目
- フィードバックデータの自動生成
  - 解析からフィードバックまでを自動化できる
  - プロセス解析結果を利用
- フィードバック閲覧ページの作成