オープンソース・ツールで楽しむ ソフトウェアテスト改善

キャッツ株式会社
ソフトウェア事業部
穴田 啓樹

本日の発表内容

1. テストの改善とは何か
2. 事例説明
3. ツール活用のポイント
4. 改善効果
5. まとめ

1. テスト改善とは何か

■ 目の前の **不便を意識** することから始める

職場から一定周期のリズムを刻む
音が聞こえてきませんか？

・テストは繰り返しが多い

・3回おなじことを繰り返したら、やり方を考え直せ！

■ 日々の **小さな工夫** を積み重ねる

1.1 何を、なぜ改善するのか

作成 → 実行 → 検証

■ **テストシーケンス作成**
- ドキュメント作成を楽にしたい！
- テストドライバ作成を楽にしたい！
- 両者を一度に作成したい！

■ **テスト結果検証**
- 多量のログ解析作業を楽にしたい！
- ミスを防ぎたい！

1.2 利用したオープンソースツール

■ **スクリプト言語**

http://www.ruby-lang.org/ja/
http://rubyinstaller.rubyforge.org/wiki/wiki.pl

- **Ruby**
  まつもと ゆきひろ氏が開発しているオブジェクト指向スクリプト言語。
  インタープリタ型言語で、気軽にプログラミングできる。
- **Ruby/Tk**
  RubyからTcl/TkのToolkitを利用できるようにした環境。
  手軽にGUI を持たせることができる。

■ **特に有効なケース**
- “簡単に” 複雑な検索をしたい
- 文字列を整形したい
- 文字列を置き換えたい

→
- ログ解析
- 規約チェック
- ドキュメント整形

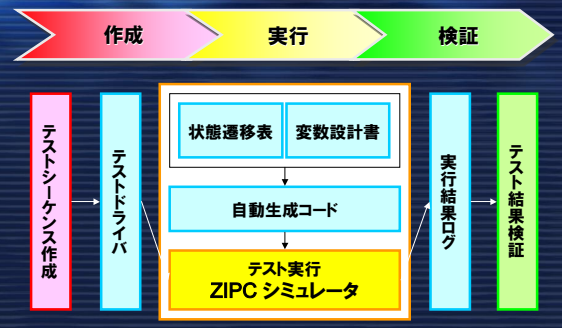
2. 事例説明

■ **CDプレーヤー**
- 状態遷移表をもとにテストする
- テストシーケンスを記述した仕様書を作成する
- テストドライバを作成する
- 実行環境は ZIPC（※）シミュレータを利用する
- テストを通して変数が設計の範囲内で変化したか確認する

「電源ON-OFFテスト」
1. 電源ON-OFFによるトレーの状態を確認する
1）POWER イベントを発生させる
2）TRAY_OPEN 状態か確認する
3）POWER イベントを発生させる
4）OFF 状態か確認する

```
char Test(void)
{
  /* POWER イベントを発生させる*/
  EXEC_POWER;
  /* TRAY_OPEN 状態か確認する */
  CHECK_TRAY_OPEN;
```


2.1 システム構成



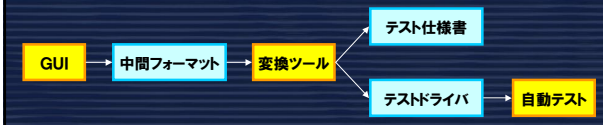
2.2 事例「テストシーケンス作成」

デモ

■ テスト仕様書とテストドライバを作成する

定型、同単語が多数出現 ⇒ 入力の手間を減らしたい

それぞれ作ると手間 ⇒ 両者を一度に作りたい



テストシーケンス作成支援ツール

2.2 表現の工夫（中間フォーマット）

■ テストシーケンスの構成要素を分類する

⇒ データを再利用したい

■**事前準備** このテストを始める前に整えておくこと
■**事前条件** このテストを始める前にあるべき状態
■**イベント** このテストで発生させる事象
■**事後条件** このテストを終えた後にあるべき状態
■**事後処理** このテストを終える前に後始末しておくこと

■「<タグ>データ」でシーケンスを構成する

⇒ 追跡支援の手がかりにしたい

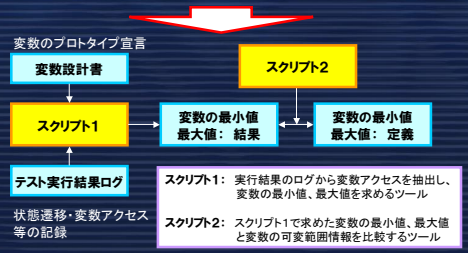
例）
ある状態遷移表のセルの遷移先が変更されたら、「事前条件」「イベント」で特定されるテストシーケンスの事後条件を修正する。

2.3 事例「テスト結果検証」

デモ

■ 変数の可変範囲が設計通りか確認する

毎回のログ解析作業が手間 ⇒ 自動化したい



スクリプト1： 実行結果のログから変数アクセスを抽出し、変数の最小値、最大値を求めるツール

スクリプト2： スクリプト1で求めた変数の最小値、最大値と変数の可変範囲情報を比較するツール

ログ解析支援ツール

2.4 ログ解析の基本（1）

■ 基本構造

```
while line = gets          # 1行読み込む→繰り返す
  if /正規表現/ =~line     # マッチする行を探す
    処理                   # 必要な処理を行う
  end
end
```

```
# sample.rb
while line = gets
  if /test/ =~line
    print line
  end
end
```

```
#test.txt
これはテストです。
test test
```

```
C:¥Develop>ruby sample.rb < test.txt
#test.txt
test test
C:¥Develop>
```


2.4 ログ解析の基本（2）

■ ポイントは正規表現

文字列の中から特定パターンを探すときのルールを表現する方法。文字とメタ文字で構成する。

例1）
文字列「aaaaa test bbbbb」において
正規表現 /¥s¥w+¥s/
（空白で囲まれた1文字以上の英数字）
がマッチするのは「 test 」である。

例2）
文字列「aaaaa test bbbbb」において
「空白で囲まれた1文字以上の英数字に続く文字列を得たい。
正規表現 /¥s¥w+¥s(¥w)/ ⇒「bbbbb」

代表的なメタ文字

¥w	英数字
¥W	非英数字
¥s	空白文字
¥S	非空白文字
¥d	数字
¥D	非数字
*	直前の表現の0回以上の繰り返し
+	直前の表現の1回以上の繰り返し
()	グループ化

2.4 ログ解析の基本（3）

■ 2.3の事例での解析例

「シミュレーションログから変数毎の最小値、最大値を求める」

ログの例：

28 SME 67000000 RUN “CD” “CD” “ubSpeed” “1” “2“

※変数「ubSpeed」：「1」→「2」へ変化

1） ログから、変数名「ubSpeed」と変数値「2」を抜き出したい

/¥d+¥s¥w+¥s¥d+¥s¥w+¥s”¥w+”¥s“¥w+”¥s“(¥w+)”¥s“¥d+”¥s“(¥d+)”/

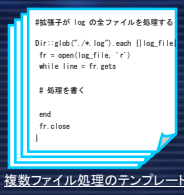
2） にマッチさせると、$1に変数名、$2に変数値（の文字列）が入る

3） これらを利用して、変数毎の最小値、最大値を求める

3. ツール活用のポイント

■ テンプレートを準備しておく

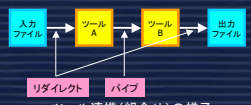
- 不便を感じたときに、すぐ作れるように
  - テキスト処理の基本
  - 複数ファイル処理



複数ファイル処理のテンプレート

■ 小さなツールを組み合わせる

- 再利用性を高めるため
- デバッグに時間を費やさないため
- 保守性を高めるため



ツール連携（組合せ）の様子

4. 改善効果

定量的	定性的
<テストシーケンスの作成支援> 改善後： 約1．5日 改善前： 約3日 ※あるテストにおいて	<テストシーケンスの作成支援> 文章のゆらぎ 文章の曖昧さ
<テスト結果検証> ツール： 約3秒 手作業： 約15分 ※あるテストにおいて	<開発者の意識向上> 工夫するのって「楽しい」！ こんなに早くなるなんて「すごい！」 楽(ラク)したい！

5. まとめ

目の前の不便を意識しよう！

スクリプト言語を活用しよう！

小さな工夫を積み上げよう！