現場ですぐ使えるソフトウェアテストツール

2011年7月19日
株式会社NTTデータ
町田　欣史

変える力を、ともに生み出す。
NTT DATAグループ

INDEX

Copyright © 2011 NTT DATA CORPORATION

自己紹介

町田　欣史（まちだ　よしのぶ）

所属

- 株式会社NTTデータ 技術開発本部 プロアクティブ・テスティングCOE
  - テスト・品質保証に関する技術支援、研究開発

活動

- JSTQB(テスト技術者資格認定)技術委員会
- SQuBOK(ソフトウェア品質知識体系)策定部会
- 執筆・講演歴
  - ソフトウェア・テスト PRESS Vol.1～Vol.4, Vol.6, Vol.10, 総集編
  - JSTQB教科書 JSTQB認定テスト技術者 Foundation Level試験
  - ソフトウェア品質知識体系ガイドーSQuBOK Guideー
  - 現場で使えるソフトウェアテスト Java編
  - @IT「Eclipseで使えるテストツールカタログ」
  - ITPro EXPO 2008 (X-over Development Conference)
  - ソフトウェアテストシンポジウム (JaSST) 2009 東京、2011 東京
  - 第19回 ソフトウェア開発環境展 (SODEC)　など

注意事項

1. 講演資料に使用される用語は基本的に以下に準拠します。

- ISTQBテスト技術者資格制度 Foundation Level シラバス 日本語版 Version 2011.J01
- ISTQBテスト技術者資格制度 Advanced Level シラバス 日本語版 Version 2007.J02
- ソフトウェアテスト標準用語集 日本語版 Version 2.0.J02

http://www.jstqb.jp/syllabus.html

2. 講演で取り上げるテストツールは、「Java」「Webアプリケーション」に対応したものが中心になります。

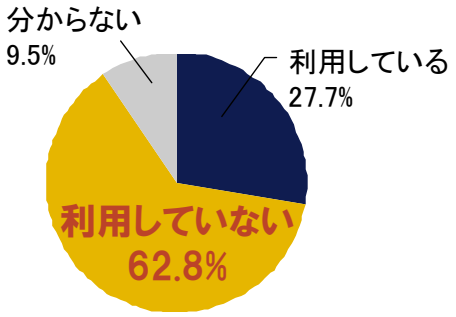
1. テストツールの導入

1.1 テストツールの利用状況

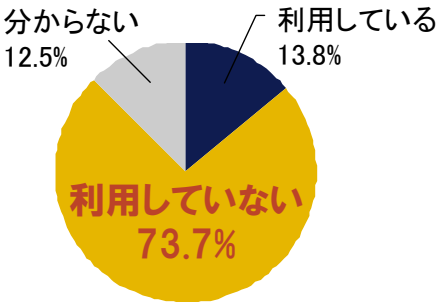
変える力を、ともに生み出す。
NTT DATAグループ

テストツールの利用率は非常に低い

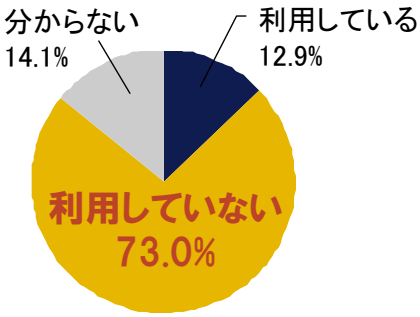
テスト工程別のツール利用率



単体テストのツール利用率



結合テストのツール利用率



システムテストのツール利用率

（注）この調査における定義
単体テスト　・・・　プログラム単体の静的/動的テスト
結合テスト　・・・　複数のプログラムを結合させた機能テスト
システムテスト・・・性能や信頼性、運用性などを確認するテスト

日経SYSTEMS 2011年6月号「特集3 なぜテストツールは使われないのか---2011開発支援ツール徹底調査」
（有効回答1648）よりp.48図1の一部を日経BP社の許可を得て転載

1.2 テストツールが利用されない理由

変える力を、ともに生み出す。 NTT DATAグループ

テストツールは認知度が低く、利用者の満足度も低い

■ テストツールが利用されない主な理由

本日はすべてフリーのツールを取り上げます

- コストがかかる
  - ツールのコスト → フリーのツールでも有用なものが多数あります
  - 学習コスト → Webや雑誌の情報を参考に習得しましょう
- 効果が出ない（手動テストのほうが早い） → すぐには効果は出ません 粘り強く使い続けましょう
- ツールを知らない → 情報発信します！
- 機能が足りない → 複数のツールを組み合わせましょう ツールのカスタマイズを検討しましょう
- メンテナンスが大変 → テスト対象が変更になればメンテナンスは必要 工夫次第で再利用も可能

1.3 テストツール導入の必要性

変える力を、ともに生み出す。 NTT DATAグループ

テストツールを使うことはテスト技術者の使命

■ 効果が出ないからといって、すぐにあきらめますか？

- テストツール導入の効果
  - 作業時間の短縮
  - 繰り返し作業の自動化
  - 正確性の向上

■ テストツールを使わないことは「自己否定」になる

- IT化は「業務を効率する」「世の中を便利にする」ためのもの
- ソフトウェア開発もIT化によって効率化されるべき
- ツールの導入が必須！

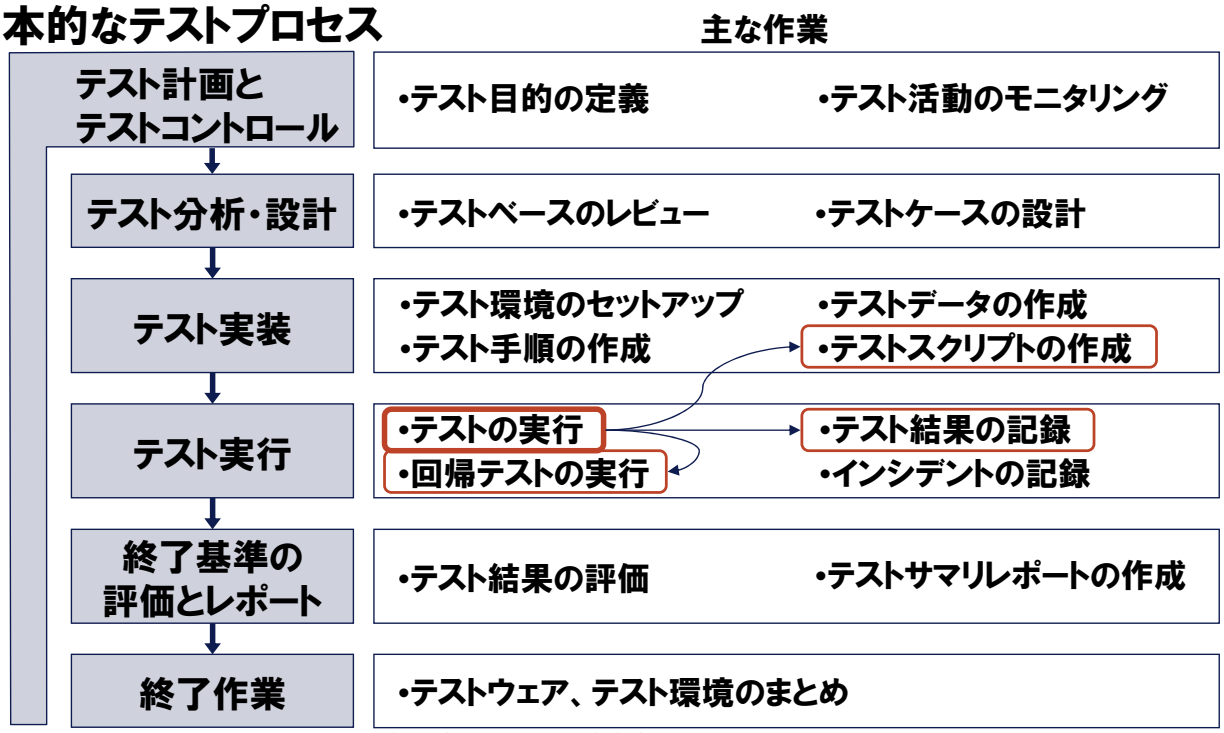
2. テスト実行の自動化

2.1 テストツール導入のステップ1

変える力を、ともに生み出す。
NTT DATAグループ

「テストの実行」と関連する作業を自動化

■ 基本的なテストプロセス



（参考）ISTQBテスト技術者資格制度 Foundation Level シラバス 日本語版 Version 2011.J01

Copyright © 2011 NTT DATA CORPORATION

2.2 テスト実行の自動化の対象

変える力を、ともに生み出す。 NTT DATAグループ

テストの種類に応じたテストツールがあります

■ 動的テスト

テストレベル	テストタイプ	テストツール
・コンポーネントテスト（ユニットテスト）	機能テスト	ユニットテストフレームワーク（2.3）
・統合テスト ・システムテスト	機能テスト	キャプチャプレイバックツール（2.4）
	性能テスト ロードテスト ストレステスト	性能テスト／ロードテスト／ ストレステストツール（2.5）

※その他のテストタイプ（セキュリティ、ユーザビリティなど）は今回は取り上げません

■ 静的テスト

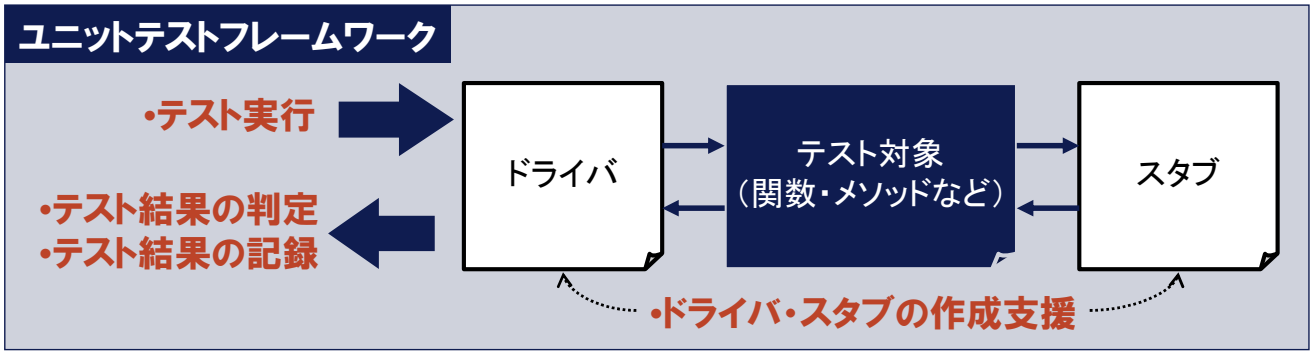
- 静的解析ツール（2.6）

2.3 ユニットテストフレームワーク

変える力を、ともに生み出す。 NTT DATAグループ

コンポーネントテストの機能テストを自動化

■ 概要



■ 代表的なツール

- プログラミング言語ごとにフレームワークが提供されています。
  - JUnit　Java用
  - CppUnit　C++用
  - NUnit　.NET対応言語用

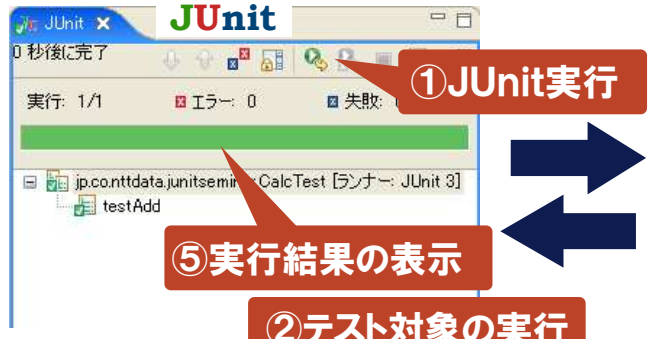
（参考） http://ja.wikipedia.org/wiki/XUnit/

2.3.1 JUnit

変える力を、ともに生み出す。 NTT DATAグループ

Java単体テストのデファクトスタンダード

■ 主な機能
- テストコード(ドライバ)作成支援
- テストの実行と結果の表示



①JUnit実行

⑤実行結果の表示

②テスト対象の実行

```
public class Calc {
    public int add(int num1,int num2) {
        return num1+num2;
    }
}
```

テスト対象クラス

③実行結果の返却

テストコード

```
package jp.co.shoeisha.sample;

import junit.framework.TestCase;
import org.junit.Test;

public class CalcTest extends TestCase {

    @Test
    public void testSum() throws Exception {
        int input1 = 1;
        int input2 = 2;
        Calc calc = new Calc();
        int result = calc.sum(input1, input2);
        assertEquals(3, result);
    }
}
```

④実行結果の判定

2.3.2 JUnitのメリット・デメリット

変える力を、ともに生み出す。 NTT DATAグループ

JUnitには多くのメリットがあり、デメリットも軽減可能

■ メリット （★ ユニットテストフレームワーク共通のメリット）
- テスト方法の統一 ★
- テスト仕様・証跡の記録 ★
- 回帰テストの自動化 ★
- 他ツールとの連携 （Eclipse、Antなど）
- テストコードと実装コードの分離 ★

■ デメリット （すべてユニットテストフレームワーク共通のデメリット）
- ツールの学習コスト ➡ Webや書籍などに多くの情報があり、習得は比較的容易
- テストコード作成の作業負荷
- テストコードメンテナンスの必要性

➡ JUnitのテストコードはシンプルなので、コード量ほどの作成工数はかからない

➡ JUnitをサポートするツールを使えば テストコード作成作業をさらに軽減

2.3.3 JUnitのテストコード作成(1／4)

Eclipse標準機能ではテンプレートしか作成されない

テスト対象クラス

Calc.java

```
1  public class Calc {
2
3      public int add(int num1,int num2) {
4          return num1+num2;
5      }
6
7  }
8
```

Eclipseのウィザードを使ってテストコードを作成

テストコード

Calc.java　CalcTest.java

```
1  import static org.junit.Assert.*;
2
3  import org.junit.Test;
4
5
6  public class CalcTest {
7
8      @Test
9      public void testAdd() {
10         fail("まだ実装されていません");
11     }
12
13 }
14
```

テストメソッドの中身はすべて実装しなければならない

JUnit
http://www.junit.org/

Eclipse
http://www.eclipse.org/

2.3.3 JUnitのテストコード作成(2／4)

JUnit Helperを使うとテストメソッドの一部を自動生成

テスト対象クラス

Calc.java

```
1  public class Calc {
2
3      public int add(int num1,int num2) {
4          return num1+num2;
5      }
6
7  }
8
```

JUnit Helperを使ってテストコードを作成

テストコード

Calc.java　CalcTest.java

```
1  import Calc.*;
2  import junit.framework.TestCase;
3
4  public class CalcTest extends TestCase {
5
6      public void test_type() throws Exception {
7          // TODO auto-generated by JUnit Helper.
8          assertNotNull(Calc.class);
9      }
10
11     public void test_instantiation() throws Exception {
12         // TODO auto-generated by JUnit Helper.
13         Calc target = new Calc();
14         assertNotNull(target);
15     }
16
17     public void test_add_A$int$int() throws Exception {
18         // TODO auto-generated by JUnit Helper.
19         Calc target = new Calc();
20         int num1 = 0;
21         int num2 = 0;
22         int actual = target.add(num1, num2);
23         int expected = 0;
24         assertEquals(expected, actual);
25     }
26
27 }
```

入力値の設定や期待結果の検証部分を自動生成

JUnit Helper
http://code.google.com/p/junithelper/

2.3.3 JUnitのテストコード作成(3／4)

CodePro Analytixを使うと入力値・期待結果も自動生成

テスト対象クラス

```
Calc.java
1  public class Calc {
2
3      public int add(int num1,int num2) {
4          return num1+num2;
5      }
6
7  }
8
```

CodePro Analytixを使ってテストコードを作成

テストコード

```
Calc.java  CalcTest.java
1  import org.junit.*;
3
4  /**
5   * The class <code>CalcTest</code> contains test
6   *
7   * @generatedBy CodePro at 11/07/17 11:28
8   * @author machidays
9   * @version $Revision: 1.0 $
10  */
    lic class CalcTest {
     /**
      * Run the int add(int,int) method test.
      *
      * @throws Exception
      *
      * @generatedBy CodePro at 11/07/17 11:28
      */
     @Test
     public void testAdd_1()
         throws Exception {
         Calc fixture = new Calc();
         int num1 = 1;
         int num2 = 1;

         int result = fixture.add(num1, num2);

         // add additional test code here
         assertEquals(2, result);
```

CodePro Analytix
http://code.google.com/javadevtools/codepro/doc/

入力値とその実行結果(仮の期待結果)を自動生成

2.3.3 JUnitのテストコード作成(4／4)

CodePro Analytixでは入力値・期待結果を表で記述可能

テストコード

```
Calc.java
10  */
11 public class CalcTest {
12     /**
13      * Run the int add(int,int) method test.
14      *
15      * @throws Exception
16      *
17      * @generatedBy CodePro at 11/07/17 11:28
18      */
19     @Test
20     public void testAdd_1()
21         throws Exception {
22         Calc fixture = new Calc();
23         int num1 = 1;
24         int num2 = 1;
25
26         int result = fixture.add(num1, num2);
27
28         // add additional test code here
29         assertEquals(2, result);
30     }
31
32     @Test
33     public void testAdd_2()
34         throws Exception {
35         Calc calc = new Calc();
36         int num1 = -1;
37         int num2 = 1;
38         int result = calc.add(num1, num2);
39         assertEquals(0, result);
40     }
41
```

テストケース表

* Calc.add(int, int)

		引数			アー...ン
	テスト・メソッド	fixture	num1	num2	result
1	testAdd_1	new Calc()	1	1	2
2	testAdd_2	new Calc()	-1	1	0

表とテストコードが同期

2.3.4 JUnitをサポートするツール

変える力を、ともに生み出す。 NTT DATAグループ

JUnitだけでは実行困難なテストでは他のツールと連携

■ モック(スタブ)を容易に作りたい
- テスト対象から呼び出されるクラスを擬似的に(動的に)生成する
  - Mockito、EasyMock 、jMock 、djUnit　　など

■ Webアプリケーションをテストしたい
- サーブレットコンテナを必要とするコードをテストする
  - Cactus　　など

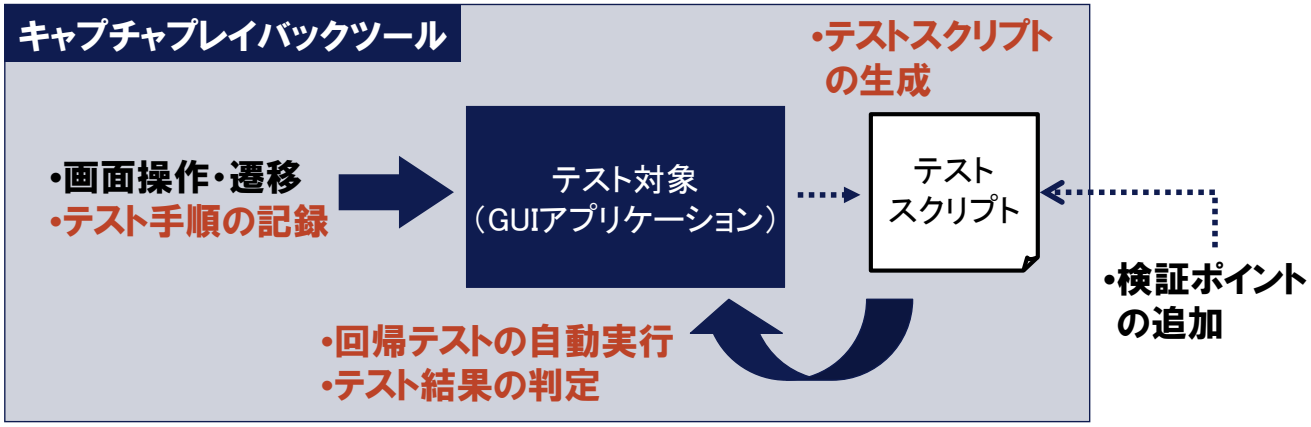
■ DBアプリケーションをテストしたい
- DBとの接続やDBに対する操作を行うテストをする
  - DbUnit　　など

2.4 キャプチャプレイバックツール

変える力を、ともに生み出す。 NTT DATAグループ

GUIアプリケーションの画面操作を記録・再生

■ 概要



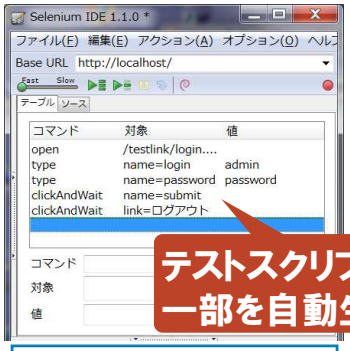
■ 代表的なツール
- Selemium
  - Webブラウザベースのアプリケーションの自動テスト

2.4.1 Selenium

Webブラウザで動作するアプリケーションのテストの自動化

画面操作の記録

手動で画面操作

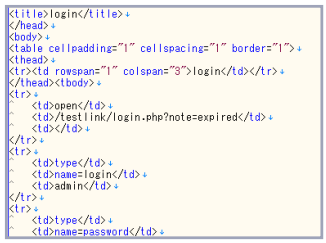


テストスクリプトの一部を自動生成

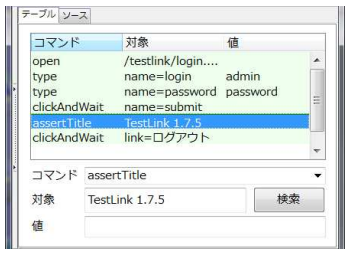
Selenium IDEで記録

テストスクリプトの生成

スクリプト(HTML)を直接編集

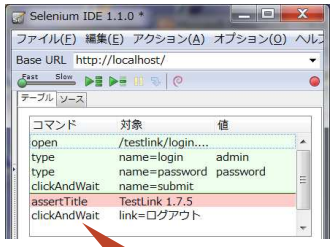


Selenium IDEでコマンド挿入



テストの実行

Selenium IDEでテストの自動実行と結果の確認



エラーがあった場合は赤色で表示

Selenium
http://seleniumhq.org/

2.4.2 Seleniumのメリット・デメリット

Seleniumで効果を出すにはプログラミングスキルが必要

■ メリット （★ キャプチャプレイバックツール共通のメリット）

- 回帰テストの自動化による工数の削減 ★
- テスト手順や結果の確認の誤りの減少 ★
- テストスクリプトがHTML形式で簡単（Selenium0.x、1.x） → Selenium2.0.0ではJava、C#などのプログラミング言語による記述が必要

■ デメリット （★ 一部のキャプチャプレイバックツールに共通のデメリット）

- 初回のテストは手動実行が必要 ★ → テストスクリプトのプログラミングで対応可能
- 画面操作の記録はFirefoxのみ → Selenium RCでFirefox以外にも対応可能
- 異なる環境では記録を取り直す ★ → Selenium Gridで同一テストを複数環境で実行可能
- 検証機能の不足（DBの検証など）★
- 証跡取得機能がない ★

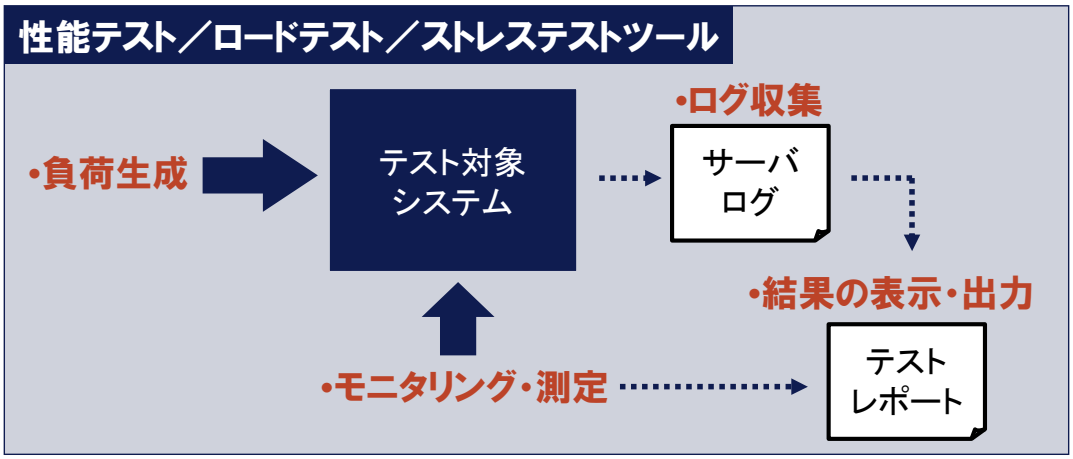
→ ツールのカスタマイズが必要

2.5 性能テスト／ロードテスト／ストレステストツール

変える力を、ともに生み出す。 NTT DATAグループ

実行時の性能測定や高い負荷をかけたテストを自動化

■ 概要



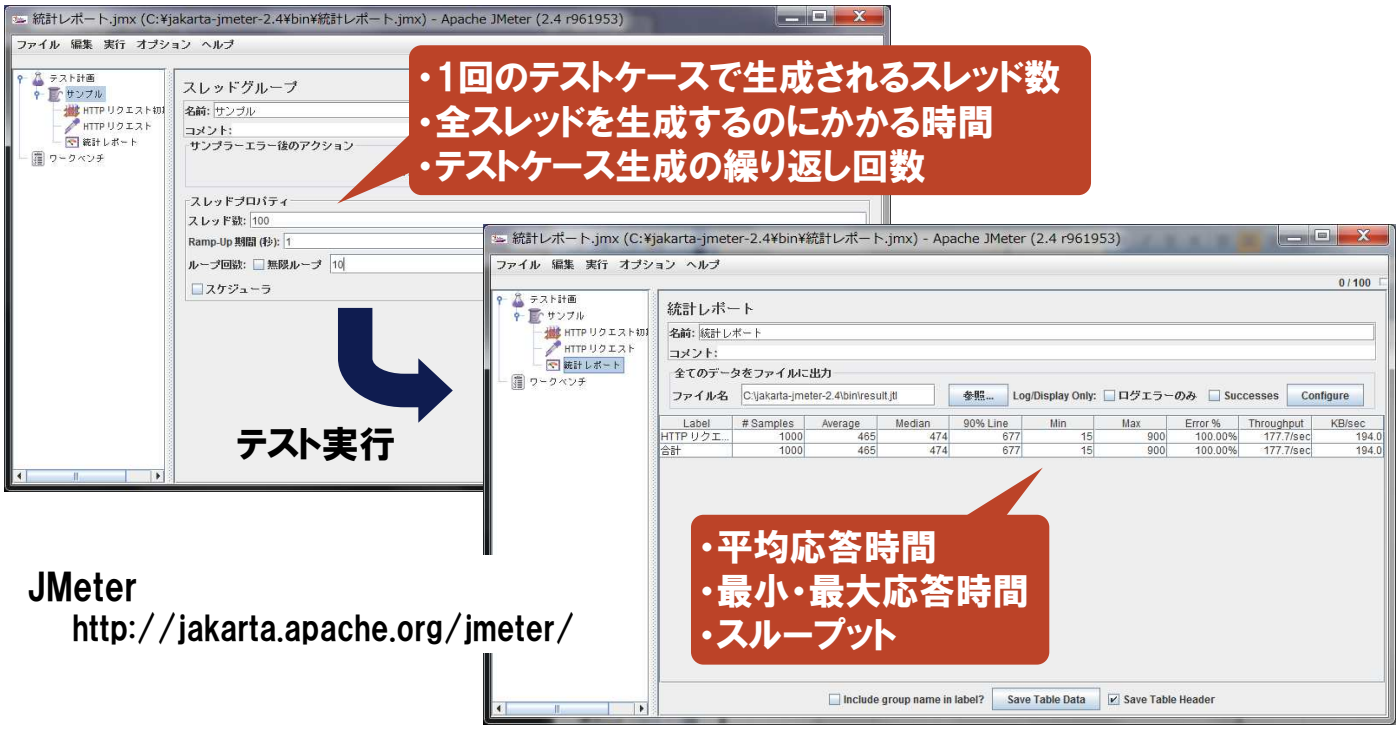
■ 代表的なツール

- JMeter

2.5.1 JMeter

変える力を、ともに生み出す。 NTT DATAグループ

多数のスレッド生成やテスト結果の測定・集計を自動化



JMeter
http://jakarta.apache.org/jmeter/

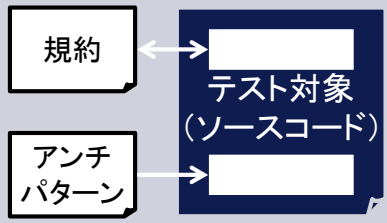
2.6 静的解析ツール

ソースコードの自動解析によるレビューの自動化

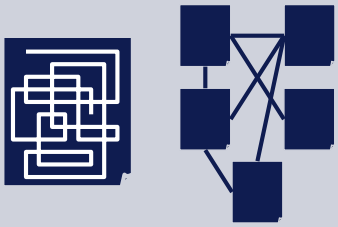
■ 概要

静的解析ツール

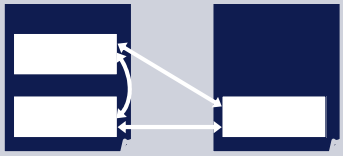
•コーディング規約違反の検出
•欠陥(の可能性)の検出



•構造や依存関係の分析



•重複・類似コード
(コードクローン)の検出

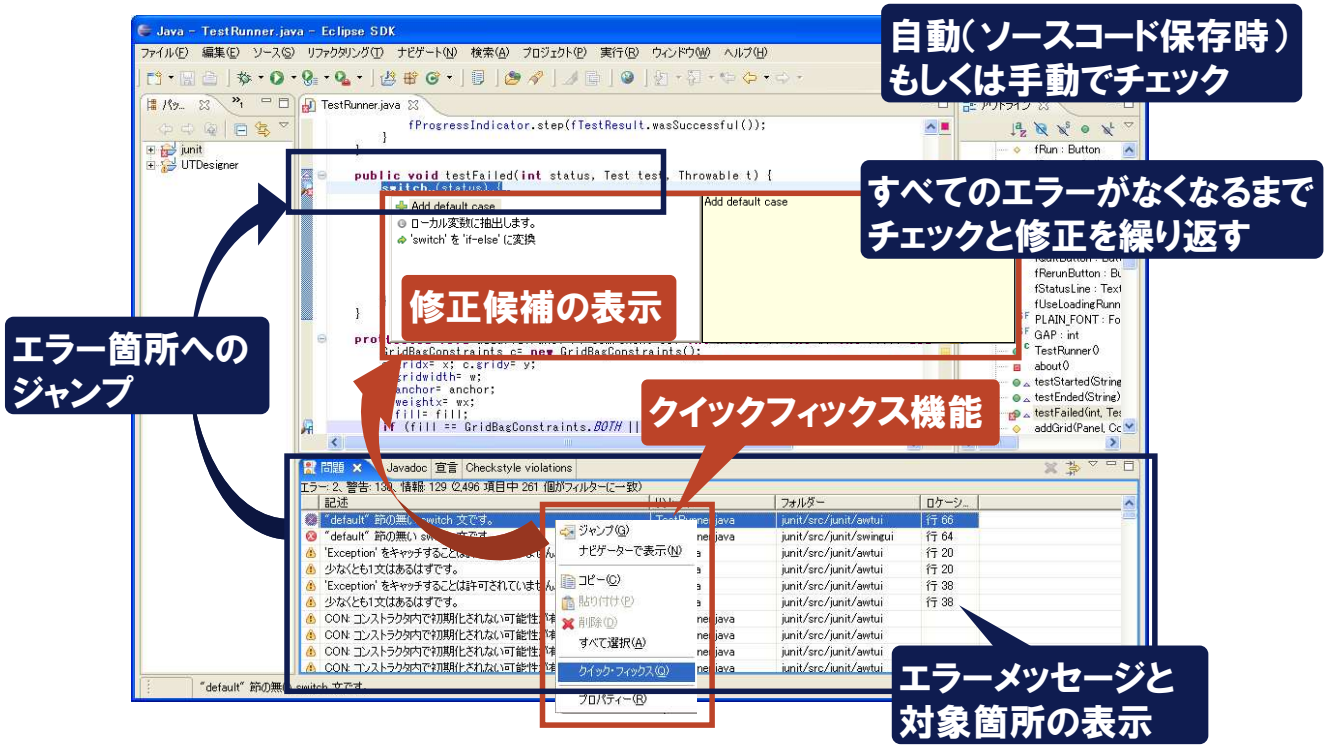


■ 代表的なツール

- ➢ 規約違反・欠陥検出
  - Checkstyle
  - FindBugs
  - PMD
- ➢ 構造・依存関係分析
  - EclipseMetrics
  - JDepend
- ➢ 重複・類似コード検出
  - CCFinderX
  - PMD (CPD)

2.6.1 Checkstyle, FindBugs

Eclipseのプラグインとして容易に導入・実行可能



2.6.2 Checkstyle, FindBugsのメリット・デメリット

変える力を、ともに生み出す。 NTT DATAグループ

多くチェックできることはメリット・デメリットの両面

■ メリット
- 多数のチェックルールを装備
- 独自のチェックルールの作成が可能
- ソースコードの自動修正機能や修正候補の表示機能
- チェックルールをGUIで設定可能
- チェックルール設定をインポート・エクスポート可能

■ デメリット
- チェックルールを選択しすぎると、エラーが大量に検出
  - ➡ ルールの絞込みが必要
- どのエラーから修正すればよいか分からない
  - ➡ 優先度づけが必要

【参考】HAREL

変える力を、ともに生み出す。 NTT DATAグループ

HTMLのアクセシビリティチェックサイト

HAREL http://harel.nttdata.co.jp/

アクセシビリティの観点に限定したオンラインによる静的解析

参考までに、JaSST'11四国のサイトをチェックすると・・・

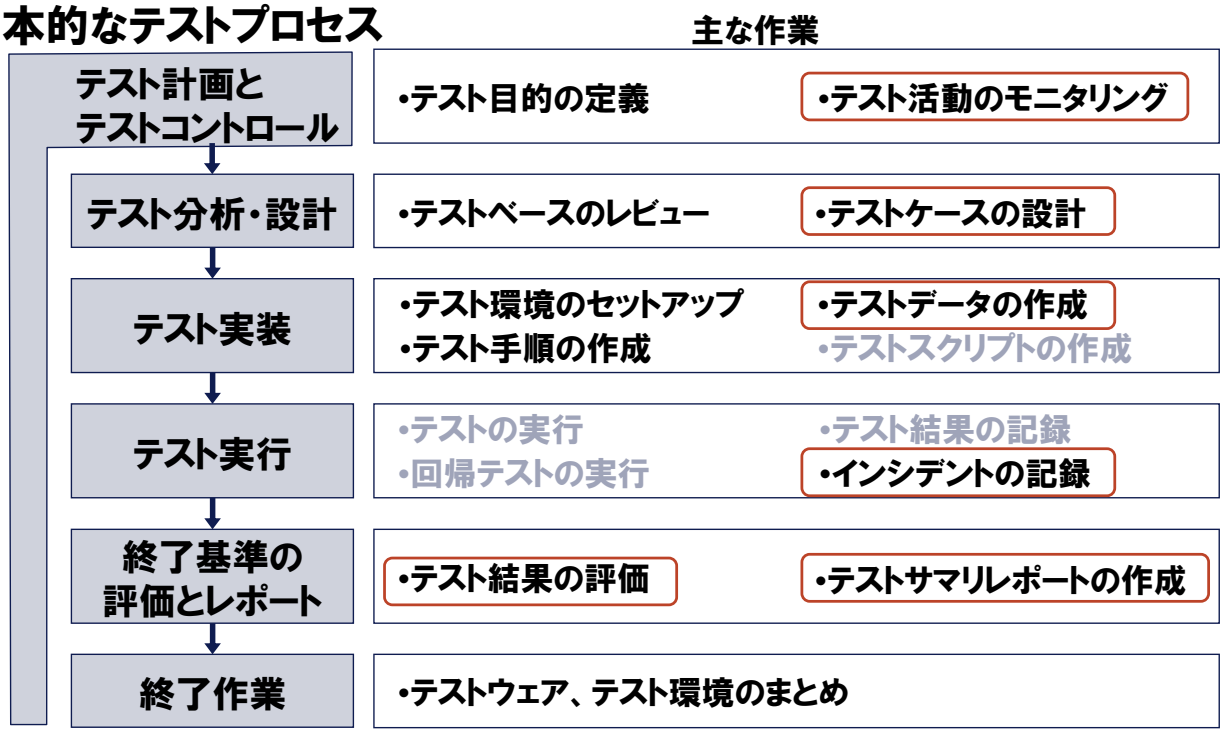
URL／ファイル	種別	晴れポイント	雨ポイント	曇りポイント
入力テキスト	HTML	9	13	7

3. 他のテスト作業の自動化

3.1 テストツール導入のステップ2

「テストの実行」以外の作業を自動化

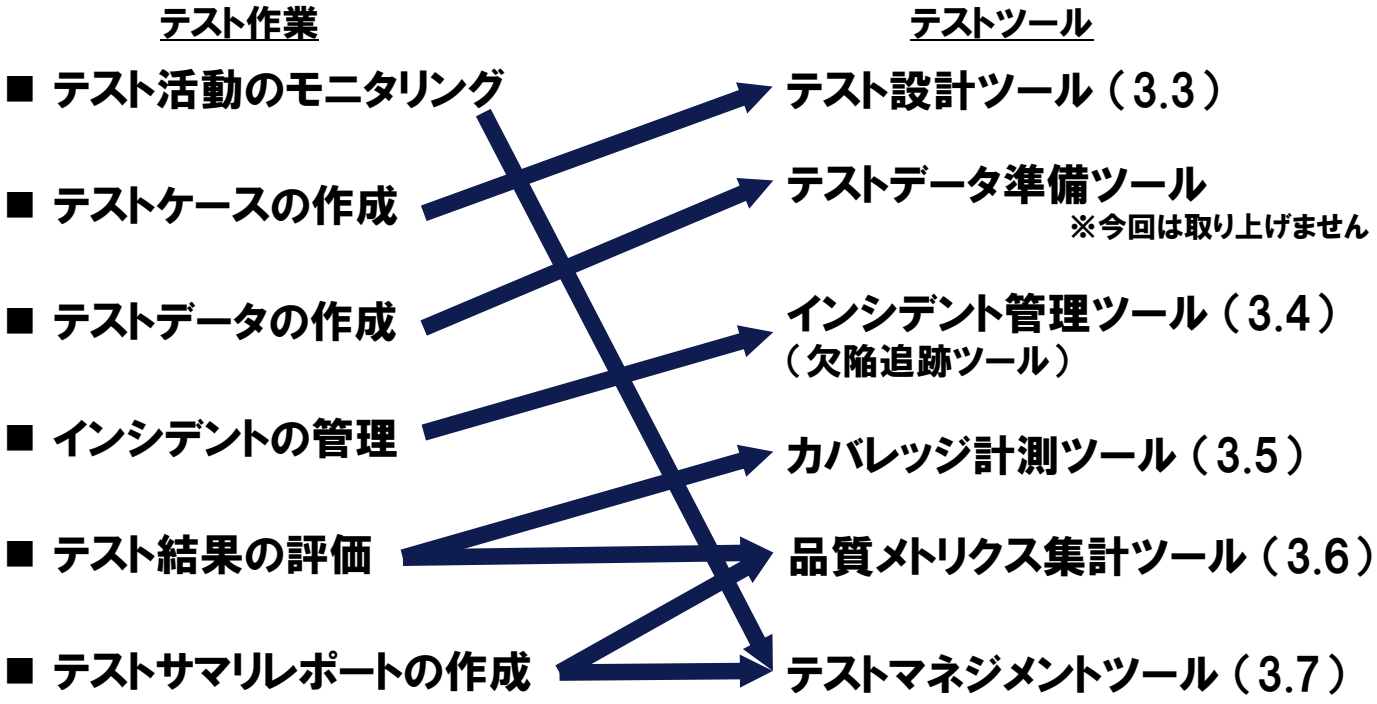
■ 基本的なテストプロセス



3.2 テスト実行以外のテストツール

変える力を、ともに生み出す。 NTT DATAグループ

「テストの実行」以外の作業にもテストツールが存在

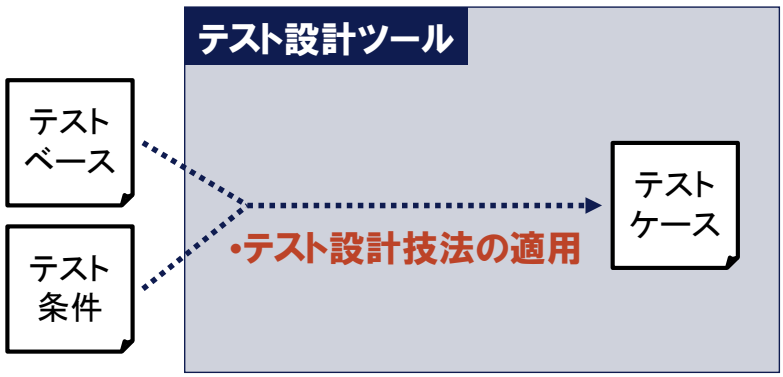


3.3 テスト設計ツール

変える力を、ともに生み出す。 NTT DATAグループ

テスト設計技法を適用してテストケースを作成

■ 概要



■ 代表的なツール

	適用するテスト設計技法
➢ PictMaster	All-Pair法（ペアワイズ法）
➢ CEGTest	原因結果グラフ、デシジョンテーブルテスト
➢ State Matrix	状態遷移テスト

3.3.1 PictMaster（1／2）

変える力を、ともに生み出す。 NTT DATAグループ

All-Pair法による組合せを容易に作成

パラメータと取りうる値を設定

パラメータ	値の並び
出発地	東京，大阪，札幌，広島，福岡
到着地	高松，松山，高知，徳島
交通手段	飛行機，電車，バス，船

PictMaster実行

組合せ生成

No.	出発地	到着地	交通手段
1	広島	高松	船
2	広島	高知	バス
3	広島	松山	電車
4	広島	徳島	飛行機
5	札幌	高松	バス
6	札幌	高知	飛行機
7	札幌	松山	電車
8	札幌	徳島	船
9	大阪	高松	飛行機
10	大阪	高知	電車
11	大阪	松山	バス
12	大阪	徳島	船
13	東京	高松	電車
14	東京	高知	飛行機
15	東京	松山	船
16	東京	徳島	バス
17	福岡	高松	バス
18	福岡	高知	船
19	福岡	松山	飛行機
20	福岡	徳島	電車

PictMaster
http://sourceforge.jp/projects/pictmaster/

3.3.1 PictMaster（2／2）

変える力を、ともに生み出す。 NTT DATAグループ

あり得ない組合せを除いた組合せを生成可能

パラメータと取りうる値を設定

パラメータ	値の並び
出発地	東京，大阪，札幌，広島，福岡
到着地	高松，松山，高知，徳島
交通手段	飛行機，電車，バス，船

PictMaster実行

組合せ生成

No.	出発地	到着地	交通手段
1	広島	高松	船
2	広島	高知	電車
3	広島	松山	バス
4	広島	徳島	船
5	札幌	高松	電車
6	札幌	高知	飛行機
7	札幌	松山	電車
8	札幌	徳島	飛行機
9	大阪	高松	バス
10	大阪	高知	飛行機
11	大阪	松山	船
12	大阪	徳島	電車
13	東京	高松	電車
14	東京	高知	バス
15	東京	松山	飛行機
16	東京	徳島	バス
17	福岡	高松	飛行機
18	福岡	高知	船
19	福岡	松山	バス
20	福岡	徳島	電車

あり得ない組合せを除外するための設定

制約表

パラメータ	制約１	制約２	制約３	制約４	制約５
出発地	東京，札幌	札幌	広島	大阪	福岡
到着地				高松，徳島	松山，高知
交通手段	#船	#バス	#飛行機	#飛行機	#飛行機

注．あくまで例であり、実際に存在しない交通手段を表したものではありません

（例）福岡から松山・高知へは飛行機で移動不可

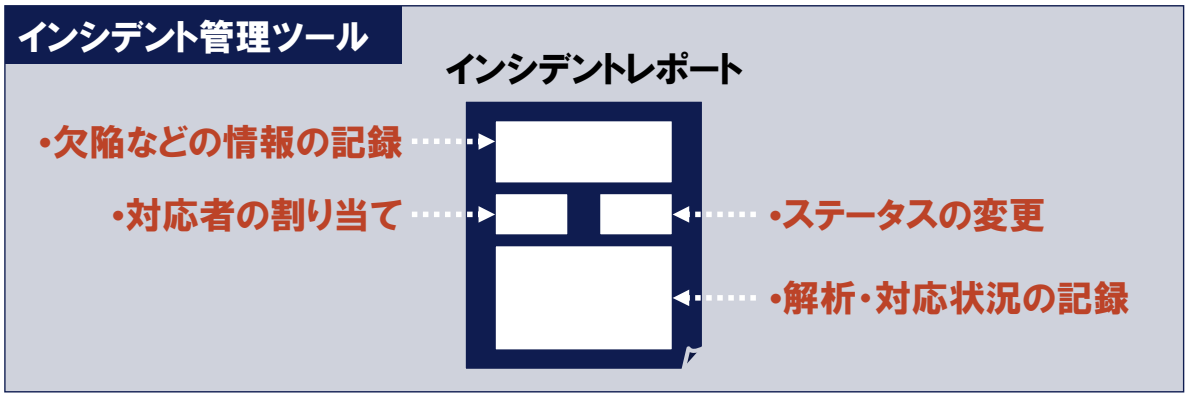
Copyright © 2011 NTT DATA CORPORATION

3.4 インシデント管理ツール（欠陥追跡ツール）

変える力を、ともに生み出す。 NTT DATAグループ

インシデントレポートの格納と管理により対応状況を追跡

■ 概要



■ 代表的なツール

- Bugzilla
- Trac
- Mantis
- 影舞
- Redmine

3.4.1 Mantis（1／2）

変える力を、ともに生み出す。 NTT DATAグループ

インシデントレポートの格納と管理により対応状況を追跡

Mantisのインシデント(チケット)登録画面

担当者の割り当て

解決状況(実装済み、差し戻し、保留など)

チケットの内容(簡易) [コメント] [リマ ー送信] [詳細] [履歴] [印刷]

ID	カテゴリ	重要度	再現性	登録日	更新日
0000007	[サンプル ロジェクト1]	微調整	不定	2011-07-18 10:59	2011-07-18 11:43
レポーター	administrator	公開	公開		
担当者	aoki				
優先度	低	解決状況	実装済		
ステータス	解決済				
要約	0000007: バグサンプル				
詳細	バグレポートのサンプルです。				
追加情報					
タグ	設定されていません。				
タグの設定	(','で区切ってください)	登録済タグ	登録		
添付ファイル					

優先度(高・中・低など)

ステータス(新規・確認済み・完了など

Mantis　http://www.alles.or.jp/~sogabe/mantis/

Copyright © 2011 NTT DATA CORPORATION

3.4.1 Mantis（2／2）

変える力を、ともに生み出す。 NTT DATAグループ

インシデントレポートの対応状況を集計して表示

Mantisのサマリ画面（抜粋）

ステータス	新規	実装済	完了	合計
新規	3	-	-	3
内容確認済	1	-	-	1
再現済	1	-	-	1
担当者決定	3	-	-	3
解決済	-	2	-	2
完了	-	-	1	1

解決状況	新規	実装済	完了	合計
不明	3	0	0	3
実装済	1	1	1	3
再現不可	2	0	0	2
修正不要	0	1	0	1
保留	1	0	0	1
後回し	1	0	0	1

重要度	新規	実装済	完了	合計
要望	1	0	0	1
些細な	0	0	1	1
表示	1	0	0	1
微調整	0	1	0	1
マイナー	1	1	0	2
メジャー	3	0	0	3
クラッシュ	1	0	0	1
システム停止	1	0	0	1

優先度	新規	実装済	完了	合計
未定	0	0	1	1
低	2	1	0	3
中	3	1	0	4
高	1	0	0	1
緊急	1	0	0	1
即時	1	0	0	1

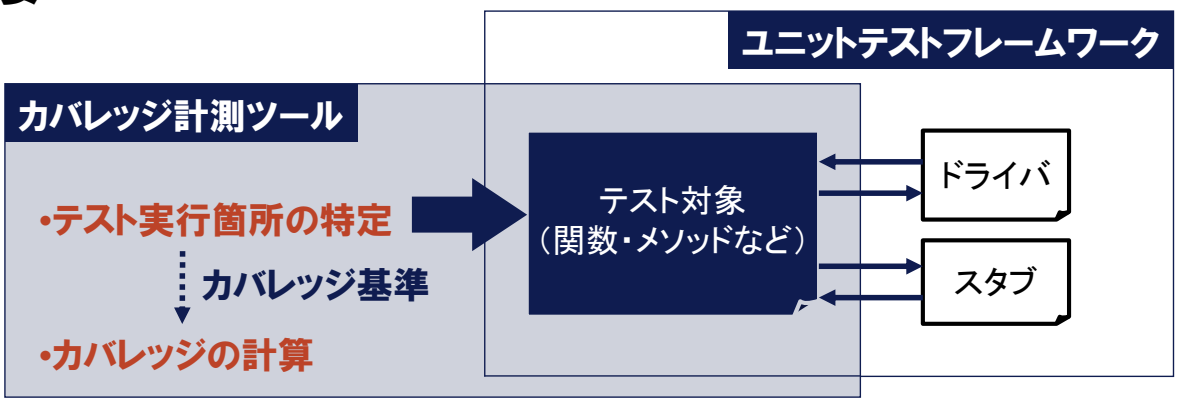
開発者の解決状況	不明	実装済	差戻し	再現不可	修正不可	二重登録	修正不要	保留	後回し	% 実装済
suzuki	0	1	0	0	0	0	0	0	1	100%
matsui	0	0	0	2	0	0	0	1	0	0%
aoki	0	2	0	0	0	0	1	0	0	100%

3.5 カバレッジ計測ツール

変える力を、ともに生み出す。 NTT DATAグループ

テスト実行をモニタリングし、コードカバレッジを測定

■ 概要



※他のテスト実行ツールに対応するものもあります

■ 代表的なツール

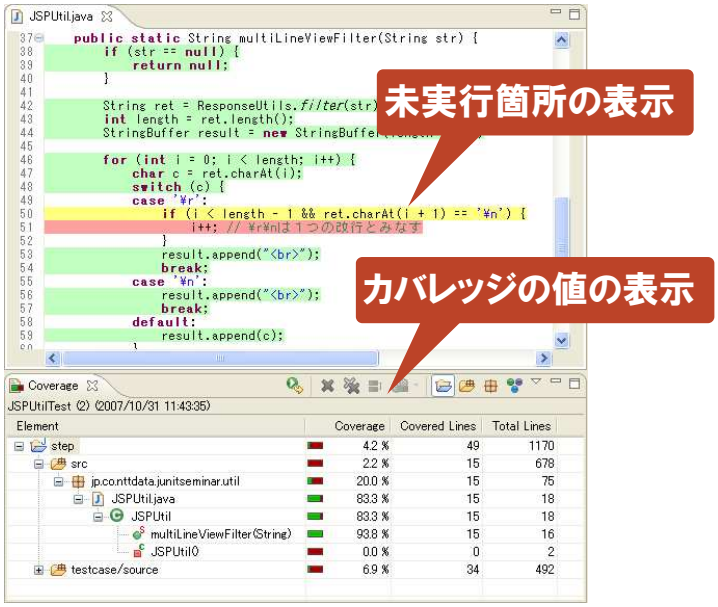
- Cobertura
- djUnit
- EMMA

3.5.1 EclEmma

変える力を、ともに生み出す。 NTT DATAグループ

JUnitにより実行された行を記録し、カバレッジを測定

EclEmmaの実行結果例



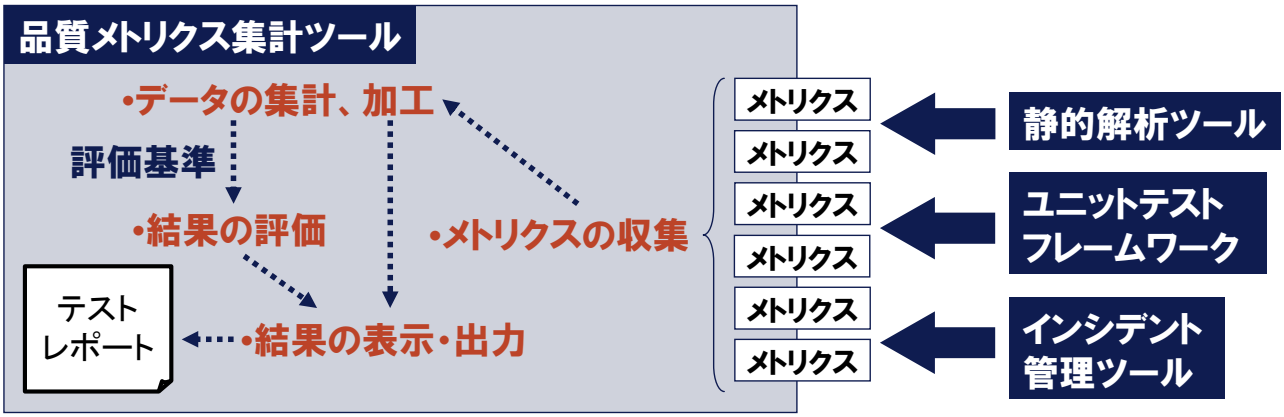
EclEmma　http://www.eclemma.org/　※EMMAのEclipseプラグイン版

3.6 品質メトリクス集計ツール

変える力を、ともに生み出す。 NTT DATAグループ

テストにおける各種メトリクスを一元化してレポート

■ 概要



※他のテストツールに対応するものもあります

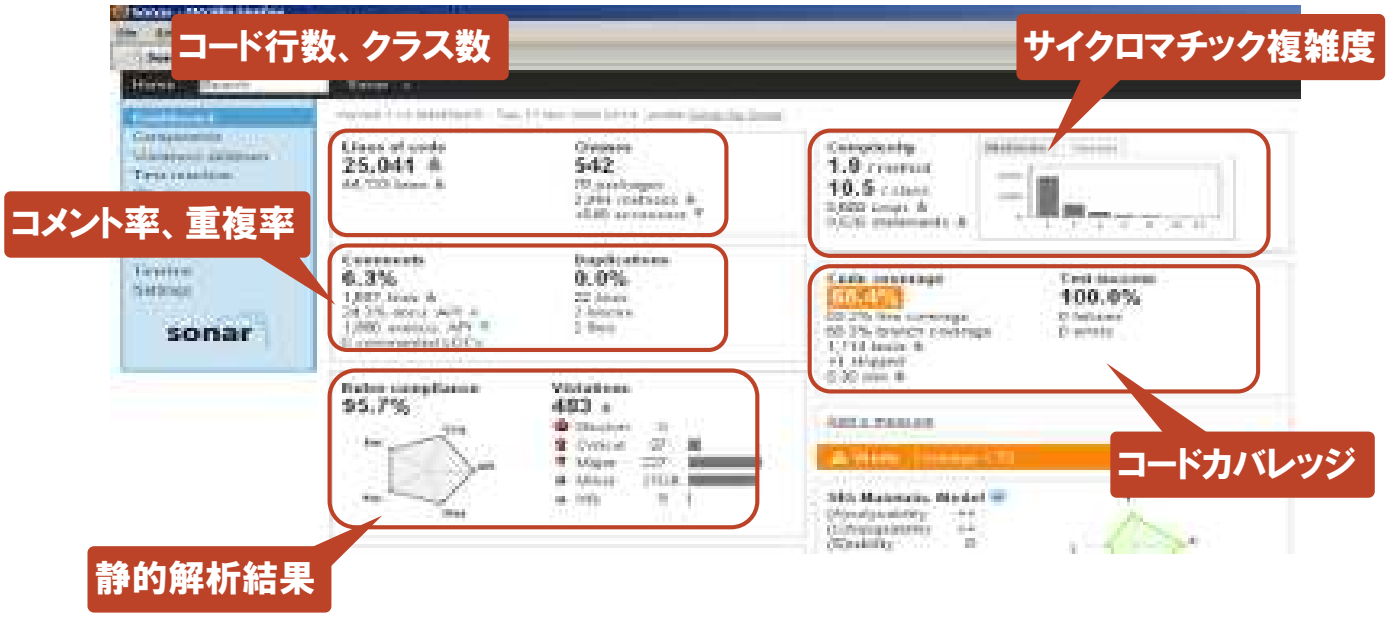
■ 代表的なツール

- QALab
- Sonar

3.6.1 Sonar

静的解析やユニットテストの結果をまとめて表示

Sonarの画面　※ http://www.sonarsource.org/ の画像を転載

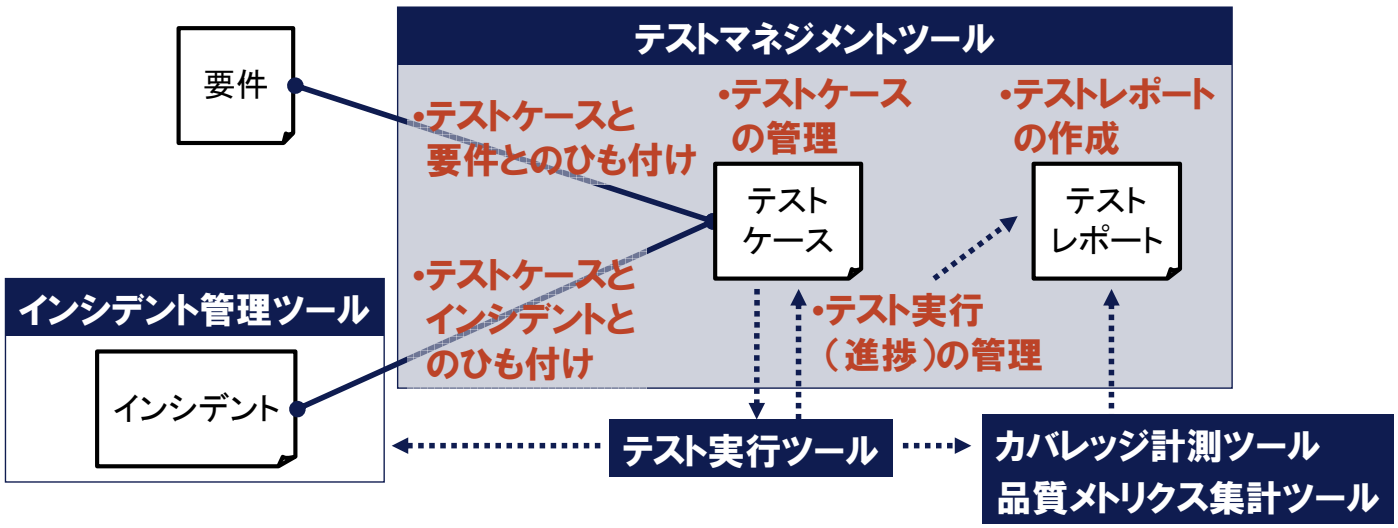


Sonar　http://www.sonarsource.org/

3.7 テストマネジメントツール

テストにおける各種作業を関連付けて一元管理

■ 概要



■ 代表的なツール

- TestLink

3.7.1 TestLink（1／7）

変える力を、ともに生み出す。 NTT DATAグループ

様々なテスト管理の作業ができるオープンソースツール

■ 主な機能

- メンバと役割の管理
- テストケースの作成・管理
- 要件仕様との対応づけ
- テスト結果の記録
- インシデント管理ツールとの連携
- レポートの出力

TestLink
http://testlink.sourceforge.net/docs/testLink.php

TestLink日本語化プロジェクト
http://testlinkjp.org/

3.7.1 TestLink（2／7）

変える力を、ともに生み出す。 NTT DATAグループ

メンバの役割設定や担当するテストケースの割り当て

メンバと役割の管理

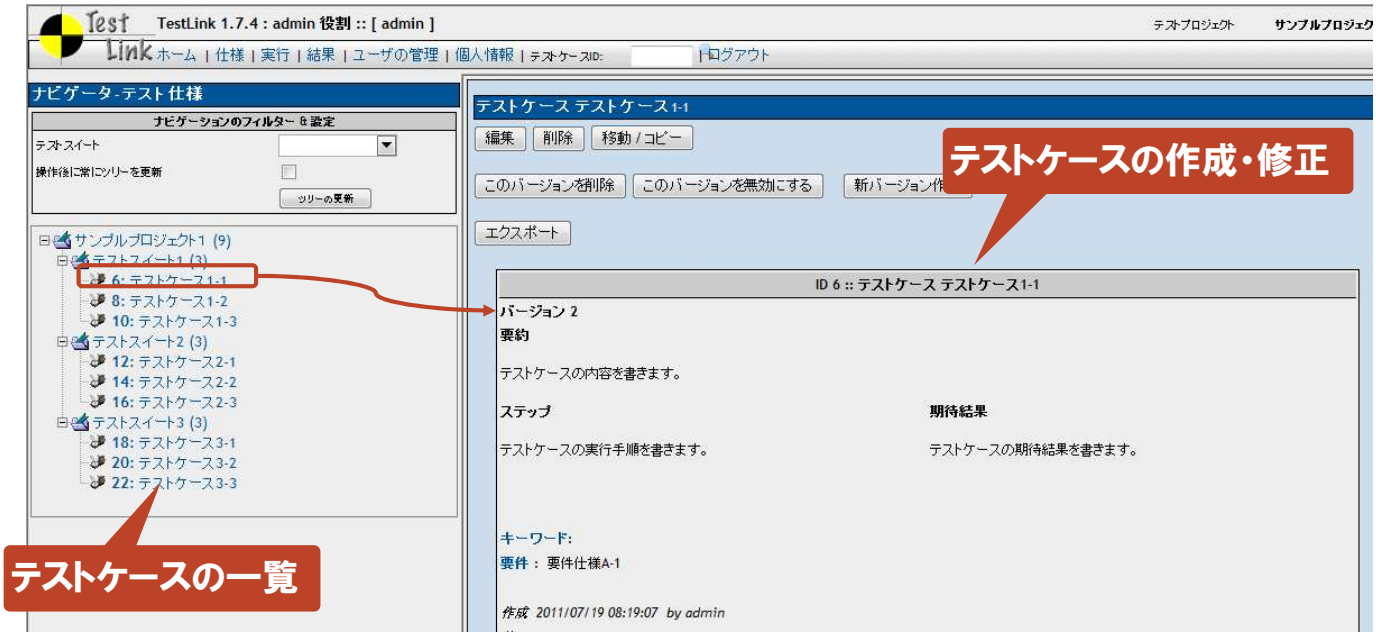
ログイン名	名	姓	Eメール	役割	言語	有効	削除
admin	Testlink	Administrator	xxx@sample.com	admin	Japanese	はい	
aoki	宣親	青木	xxx@sample.com	test designer	Japanese	はい	
darvish	有	ダルビッシュ	xxx@sample.com	tester	Japanese	はい	
hara	辰徳	原	xxx@sample.com	leader	Japanese	はい	
kiyohara	和博	清原	xxx@sample.com	senior tester	Japanese	はい	
kuwata	真澄	桑田	xxx@sample.com	senior tester	Japanese	はい	
matsui	秀喜	松井	xxx@sample.com	test designer	Japanese	はい	
matsuzaka	大輔	松坂	xxx@sample.com	tester	Japanese	はい	
suzuki	一朗	鈴木	xxx@sample.com	test designer	Japanese	はい	
tanaka	将大	田中	xxx@sample.com	tester	Japanese	はい	

3.7.1 TestLink（3／7）

変える力を、ともに生み出す。 NTT DATAグループ

テストケースの作成やバージョン管理が可能

テストケースの作成と管理



3.7.1 TestLink（4／7）

変える力を、ともに生み出す。 NTT DATAグループ

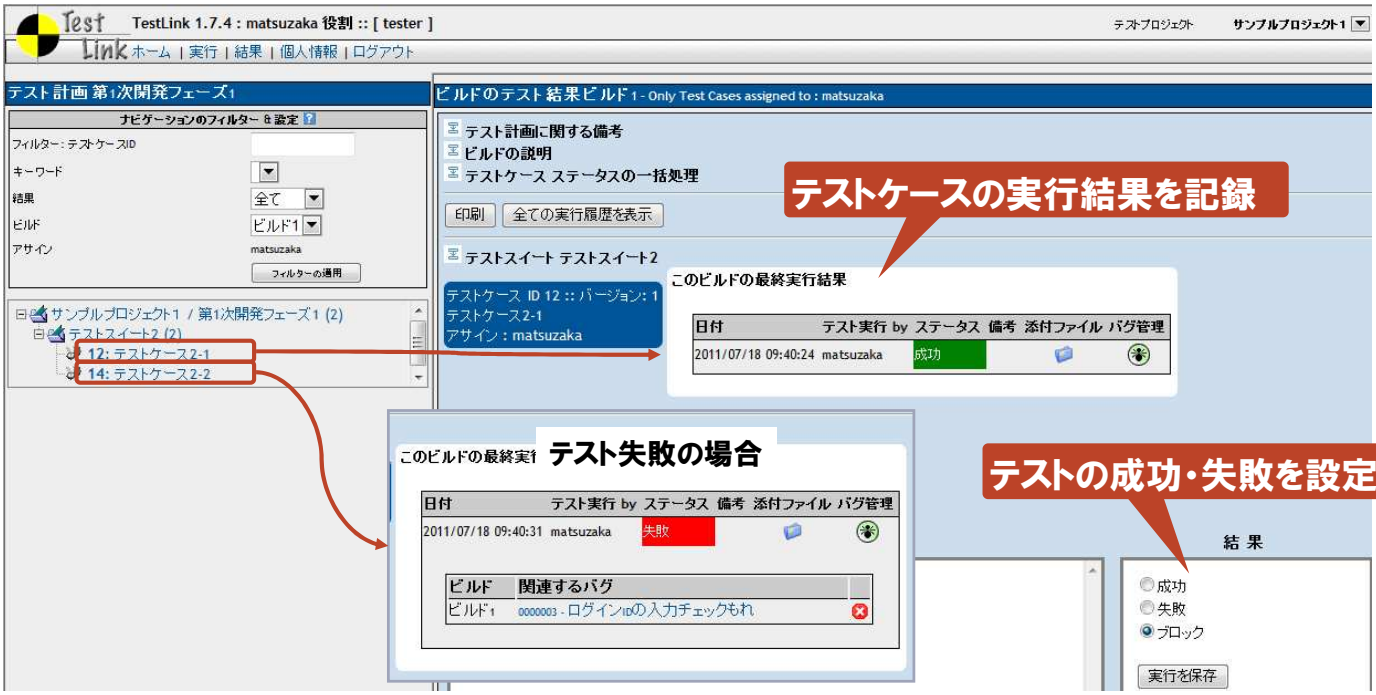
テストケースと要件仕様を対応づけて、テスト漏れを防止

要件仕様との対応づけ

3.7.1 TestLink（5／7）

テストケースと要件仕様を対応づけて、テスト漏れを防止

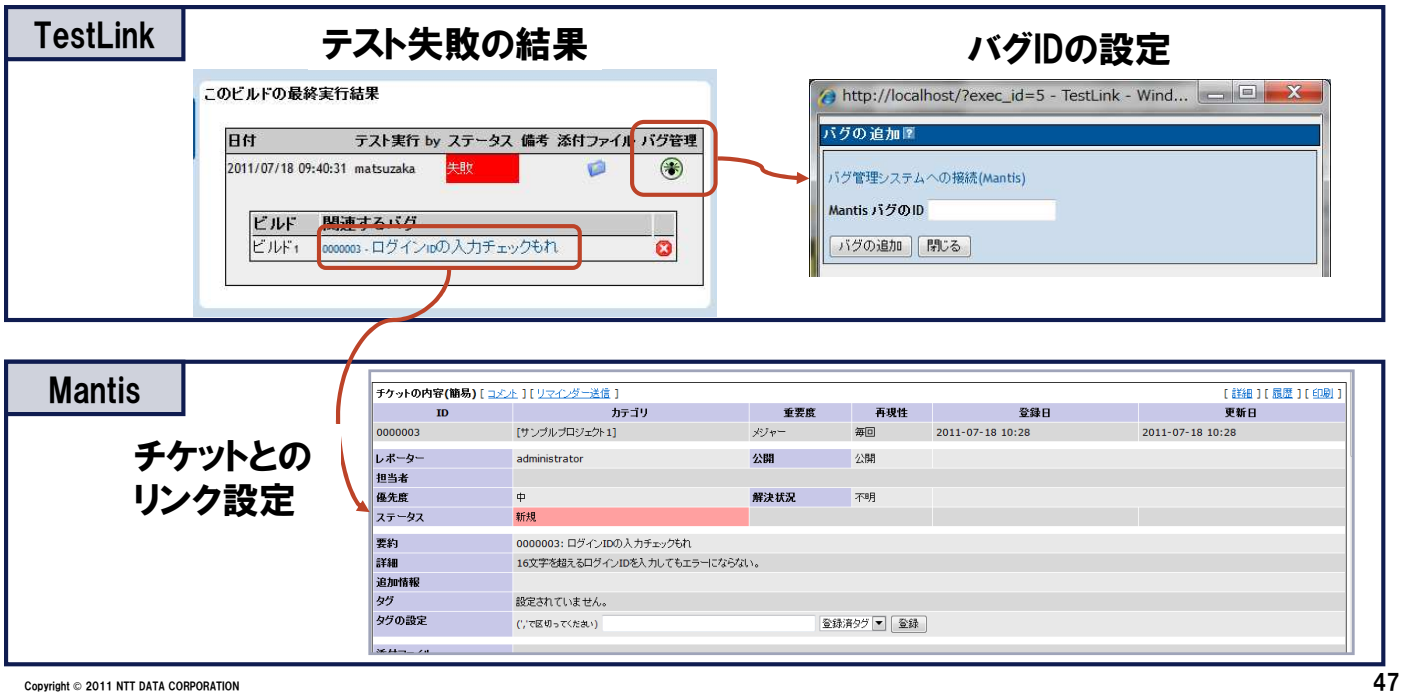
テスト結果の記録



3.7.1 TestLink（6／7）

テスト失敗の結果からインシデントレポートにリンク可能

インシデント管理ツールとの連携



3.7.1 TestLink（7／7）

変える力を、ともに生み出す。
NTT DATAグループ

テストの実行状況をまとめて表示

レポートの出力

変える力を、ともに生み出す。
NTT DATAグループ

4. まとめ

4.1 テストツール導入可能な作業

変える力を、ともに生み出す。 NTT DATAグループ

テスト作業の大部分はテストツールで自動化可能

■ 基本的なテストプロセス

4.2 テストツール導入のポイント

変える力を、ともに生み出す。 NTT DATAグループ

テストツールの導入でソフトウェア開発を高度化

■ 一気にたくさんのことを自動化しようとしない

- ターゲットを絞って、段階的に導入
- ツールの特性を見極め
  - できること、できないこと
  - 信用していい部分と確認が必要な部分
- ツール導入で増える作業があることを認識
  - まずは増加作業を認めた上で、次のステップでその作業の効率化にチャレンジする

■ ツールを組み合わせることで可能性が広がる

- 継続的インテグレーション（CI）の導入によって自動化を促進
- 上流工程のツールとテストツールの連携
- 開発だけでなく計画や運用も含めたライフサイクル全体をツールにより管理
  - ➡ ALM（Application Lifecycle Manaegment）

変える力を、ともに生み出す。
NTT DATAグループ

変える力を、ともに生み出す。
NTT DATAグループ

記載されている商品名、サービス名、会社名等は、
各社の商標または登録商標です。