FUJITSU

テスティングフレームワーク「JUnit」が開発現場にもたらした8つの効果と3つの戦い

ソフトウェアテストシンポジウム２００５

2005年1月24日

和田　憲明（富士通株式会社）

wadan@jp.fujitsu.com

テストが嫌いでした

JaSST'05

Excelでテスト仕様書を書くという苦痛

セル結合された設計書の内容をコピーペーストしようとして
「結合されたセルの一部を変更することはできません」

テスト仕様書の値をDOS窓から入力する日々、そして目視確認

ほとほと疲れます。さらに、仕様追加で再テスト。

そんな時出会ったのが．．．

IBM DevelopersWorks「AntとJUnitを用いた漸進的開発」

JUnitってナンダ!?

JaSST'05

JUnitは、たった119キロバイトのjarファイルからなるテストツール。
しかし、このツールが開発現場にもたらした影響は、大きい。

初めて見たJUnitテストケース：

```
public void testSayHello() {
    HelloWorld world = new HelloWorld();
    assert(world != null);
    assertEquals("Hello World", world.sayHello());
}
```

※IBM DevelopersWorks「AntとJUnitを用いた漸進的開発」より

JUnitってナンダ!?

JaSST'05

DeveloperWorksが教えてくれたこと：

自分が作成したものの品質に自信を持てるという安心感は快感

↓

安心感を与えてくれるのは、テストケースとテストデータしかない

↓

我々の究極の道具である「プログラム」でテストを効率化しよう

↓

テストプロセスの根本的な改善

私のこれまでの経歴

JaSST'05

1990年代：統合ＣＡＳＥツールの開発と適用
- 設計情報の一元管理による設計品質、影響検索精度の向上
- 設計書からの実行コード自動生成による品質向上

2000年前後：大規模ＥＪＢプロジェクトにて開発支援
- EJBという仕組みの活用
- 設計書からJavaソースの自動生成による生産性向上

最近：小規模、中規模Java/EJBプロジェクトで開発作業
- 業務設計、方式設計、コーディング、テスト
- プロジェクト管理、品質管理、リスク管理

→　開発現場が直面している問題がわかってきた

企業システム開発での品質低下要因

JaSST'05

№	問題点	品質への影響
1	テスト仕様書の記述精度と実施精度の不徹底	テスト作業の品質低下
2	度重なる仕様変更の発生	変更したソースへの再テストが不十分で品質維持できない
3	厳しい納期のため単体テスト工程を軽視	単体テスト完了基準が不明確になり品質低下
4	開発者へのテスト教育不足	テスト実施レベルが不統一で品質のバラツキ発生

企業システム開発での品質低下要因

№	問題点	品質への影響
5	過密スケジュールなどで開発者への負担増加	開発者のモチベーション低下による品質低下
6	開発者が日々の作業に追われ、過去の反省や工夫を行なう時間がない	反省を生かせず品質低下を改善できない
7	開発者のスキル不足	低スキル開発者が作成したモジュールに対しケアできないため品質低下
8	ビジネスの変化が速い	仕様変更に備えた開発手法でないためプロジェクトがあたふたし品質低下

課題へのアプローチ

私が「JUnit」を開発現場で実際に使用したところ、このツールは上記の「品質低下要因」の全てに対し驚くべき改善効果をもたらした。

以降、各改善効果について具体例とともに説明する。

①. Improve Quality 品質向上

№	問題点	品質への影響
1	テスト仕様書の記述精度と実施精度の不徹底	テスト作業の品質低下

JUnit適用

JUnitで書くテストケースは実行可能だから正確に書く必要あり

正確に書くのは大変だが、自動実行できて、うれしい

テストケース・テストデータの品質が向上する。
そのテスト対象であるプログラムも品質が向上する。

あいまいなテスト仕様書 (1)

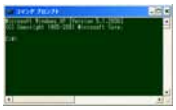
	入力内容
テストケースA	存在しない日付

「存在しない日付」とは？ 2/30、13/32、0A/0B、、、、

テスト実施者によって入力値（観点）が異なる。

直接入力する実施方法では、入力値は記録に残らない。

再テスト時にはもう一度考えないといけない。



あいまいなテスト仕様書 (2)

	入力内容	予測値
テストケースB	Null or 空文字	ABCRuntimeException

「Null or 空文字」について、実際にどちらを入力したのか不明。

JUnitでNullと空文字の両方を再テストしたところ、

Nullの場合はNullPointerException発生！！

（結果欄は ○ となっていた）

あいまいなテスト実施

年齢計算機能に関する異常系のテストケース

No.	入力内容	予測値	結果
001	8桁。数値以外	ABCRuntimeException	○
002	10桁。/のみ	ABCRuntimeException	○
003	8桁。生年月日より前	ABCRuntimeException	○
004	10桁。生年月日より前	ABCRuntimeException	○
005	11桁以上	ABCRuntimeException	○

実は java.lang.RuntimeException 発生！！

担当者に確認したら、DOS画面を目視確認時に見間違えたらしい。

②. Keep Quality　品質維持

№	問題点	品質への影響
2	度重なる仕様変更の発生	変更したソースへの再テストが不十分で品質維持できない

JUnit適用

JUnitで書かれたテストケースはコストをかけず簡単に実行できる

何回でも（毎日でも）全てのテストケースを再テストすることができる

障害修正時、仕様変更時には十分な範囲の
リグレッションテスト実施が可能となり、品質が維持される。

テスト仕様書に関する大事なこと

テスト仕様書を陳腐化させないこと

仕様変更時に修正漏れ → ソースとテスト仕様書が乖離

のちのテスト実施で、ひとつでも乖離したものが発見されると．．．

どれが正しいテスト仕様書なのかわからなくなってしまうため
乖離したテスト仕様書はもちろんのこと、テスト仕様書全体の
信頼性が低下する

苦労して作ったものが、結局使われない物となってしまう。

Excel仕様書とJUnitの記述量

という説明を上司に行なうと、こういう指摘が来ます。

（JUnitでテストケースを書くと）
仕様が変わったときにJUnitを修正するのが大変じゃないの？

？？

Excel仕様書でも、JUnitテストソースでも、テストケース数は同じ。
仕様変更時に修正が必要なテストケース数も同じ。

ただし

継承などの仕組みを活用して、テストケースのコーディング量を減らす必要がある。（Excelで記述する場合と同等くらいまで）

記述量を減らす工夫（1）

継承の仕組みを活用し、最下層のテストケースソースには業務的なテスト処理のみを記述できるように工夫する。

TestCase　junit.framework.TestCase

FXXTestCase　会社共通のテスト関連処理（DB汎用アクセス処理、ログ制御処理など）

ABTestCase　ABプロジェクト共通のテスト関連処理（エラー解釈処理、EJB呼び出し処理、共通情報設定処理、などなど）

ABTest_XYEJBHandler_xxxMasterToroku　対象メソッドのテスト処理のみ！

記述量を減らす工夫（2）

JUnit ↔ 業務処理（テスト対象） ↔ 9000 | 和田 | 憲明 | ．．．（テストセット001）

```
/**
*  【観点】引数の従業員番号に該当するレコードを取得する。
*        比較項目：従業員番号#姓#名#所属名
*  【テストデータ番号】テストセット001
*/
public void test002 {
  assertEquals("9000#和田#憲明#ＬＭＣ技術部", call("9000"));
}
```

電文の内容を一度に検証できる工夫（インターネット上で発見）

工夫した実例

以下のメソッドについてJUnitでテストケースを書く場合

```
/**
 * チェックディジットのチェックを行ないます。
 * @param hpr 依頼項目、定数、除数
 * @return HashMap 返却コード、余り
 */
public HashMap check(HashMap hpr) {
    .......
}
```

引数、返却値ともHashMapなので、テストケースには、HashMapに設定する処理、取り出す処理をたくさん記述しないといけない。

工夫した例 ？

うちの若手が書いたJUnitテストケース

```
/**
 *【観点】依頼項目、定数が一桁で、余りが0でない
 */
public void testLengh_001() {
    setParam("1", "9", "8");
    invoke();
    assertCheckResult(1);
}
```

さて、何を検証しているのだろう．．．

記述量の削減に走りすぎ、テスト仕様書として成立していない。

工夫した実例

```
/**
 *【観点】依頼項目、定数が一桁で、余りが0でない
 */
public void testLengh_001() {
    setParam("1", "9", "8");
    invoke();
    assertEquals(1, getAmari());
    assertEquals(Constant.ErrorEnd, getReturnCode());
}
```

本来テスト仕様書に記述すべき内容はキッチリ書くこと。

③. Ease Of Management テスト完了基準の明確化

№	問題点	品質への影響
3	厳しい納期のため 単体テスト工程を軽視	単体テスト完了基準が不明確に なり品質低下

JUnit適用

テストの進捗状況を、テストケース作成数とテストケース成功数という定量化された数値で常に管理することができる

リーダ自身がいつでも全テストケースを実行して結果を確認することができる。

リーダの心の平安を維持

③. Ease Of Management テスト完了基準の明確化

これまでは．．．

開発者をひとりひとり呼んで、
「今日の進捗は？」→ あいまいで大変

JUnitなら．．．

CVSから資産を取り出して全件テスト実行するだけで、
(現在の) テストケース数、成功数を把握することができる。

④. Education Of Test テストの教育

№	問題点	品質への影響
4	開発者への テスト教育不足	テスト実施レベルが不統一で 品質のバラツキ発生

JUnit適用

JUnitは自動実行可能なので、「テスト実施担当者」は不要

従来なら、スキル不足のため「テスト実施担当者」としてプロジェクトに参加していた人が、テスト設計～テストケース作成～テストデータ作成というテスト作業全体を担当することができる

実際に効果がありました

私は、あるプロジェクトで「テスト実施担当者」として参画した2名（Tさん、Yさん）に協力していただき、JUnitを使用したテスト作業を試行した。

具体的には、2人のテスト担当者に、SEが作成したユーザインタフェース設計書と、EJB開発者が作成したJavaDoc（機能詳細設計書）を読んでもらい、EJBメソッドに対し、JUnitによるテストケース作成、テストデータ作成、テスト実行を実施してもらった。

その結果、2名ともテスト教育効果が現れた。

テスト作業全体を経験することで、
「テストするとはどういうことか」を理解し、
テスト全般に対する知識が養われた。

⑤. Motivation 働く意欲

№	問題点	品質への影響
5	過密スケジュールなどで開発者への負担増加	開発者のモチベーション低下による品質低下

JUnit適用

Javaでテストケースを作成し、プログラムとして実行するため、テスト作業の一環としてプログラムを作成することができる

プログラミングは楽しいから、テストやるぞ！

⑤. Motivation 働く意欲

JUnitでのテスト作業なら、Javaでテストケースを作成し、プログラムとして実行するため、テスト作業の一環としてプログラムを作成することができ、楽しい。

実際に、JUnitでのテストを実施した上述のTさんから

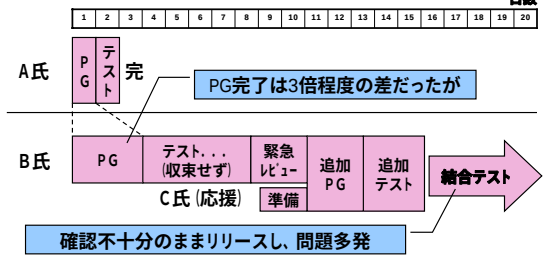
「大変だけどやりがいがあって楽しい。」

というコメントを聞くことができた。

品質と生産性は、開発者のモチベーションに大きく左右される。

品質と生産性は、開発者のスキルとモチベーションに大きく左右される

スキルがありモチベーションが高い開発者A氏
スキルが不足していてモチベーションも低い開発者B氏



モチベーションの高い状態とは

あるプロジェクトで、ＥＪＢ側の業務メソッドに対するテストをJUnitで作成することにした。

EJBプログラマーであったＮ氏は、JUnitを使用して、参照系、更新系、CSV入出力系のメソッドに対するテストケースを作り上げた。

後日、彼に「モジュールをテストチームにリリースした時の品質はどうだったか」と聞いた。彼はこう答えた。

「文句あっか、こんにゃろ〜」
という感じで（モジュールを）出しましたから。

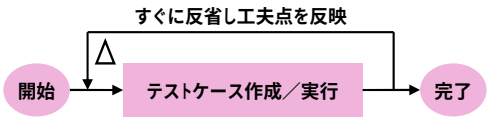
この状態になってこそ、品質を計測する意味がある！！

⑥. Versatility 知恵と工夫

№	問題点	品質への影響
6	開発者が日々の作業に追われ、過去の反省や工夫を行なう時間がない	反省を生かせず品質低下を改善できない

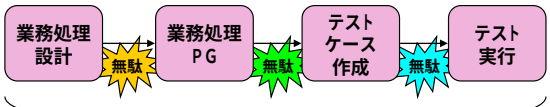
JUnit適用

「テストケースを作成／実行」を次々に繰り返すなかで、反省や工夫を次の作業にすぐに反映することができる。



⑥. Versatility 知恵と工夫

工程を越えて担当することによるメリット



ひとりで担当することによって、複数人で分担していたのではわからなかった無駄を発見できる。

トヨタ生産方式の考えと同じ

トヨタ生産方式

NHKスペシャル （2001年5月12日(土) 放送）
「常識の壁を打ち破れ ～脱･大量生産の工場改革～」

鳥取三洋電機の生産工場をトヨタ生産方式で改革する話

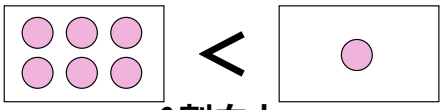
1980年代では、完全分業による大量生産が日本の大量消費時代を支えてきた。しかし最近では、消費者の傾向は多様化し、多品種少量生産になった。そのため、単能工による完全分業制では、多品種に対応できずコスト競争に負けてしまうため、生産拠点が中国へ移ってしまった。

コスト競争に勝つため、鳥取三洋電機では工場改革を実施した。ひとりで複数の仕事をこなす「多能工」を導入し、生産性を向上させる試みである。この改革での根本の考え方は以下の点である。

人間は、知恵と工夫と働く意欲で自ら生産性を向上することができる。能力を止める管理をしてはいけない。

トヨタ生産方式

この番組で紹介されたエピソードでは、従来6名で分業していた携帯電話の生産工程に対して、ある女性従業員が6名分の工程をひとりで担当した。そして試行開始からわずか3日後には、その女性の1台の組み立て時間が、従来の6名分業による組み立て時間(1台あたり207秒)よりも短くなり、生産性を2割も向上させた。



2割向上

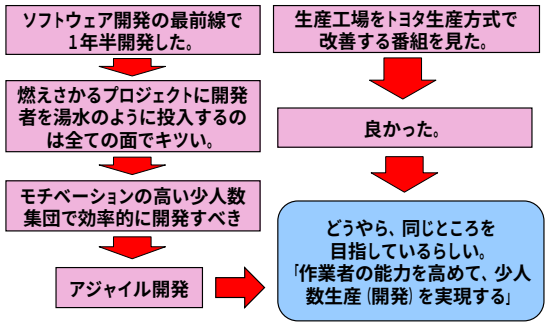
トヨタ生産方式

私は、トヨタ生産方式の根幹は以下の2点と解釈している。

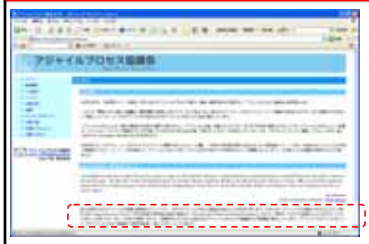
- 徹底的な無駄の排除
- 現場の知恵と工夫で品質と効率を向上させる

この考え方は、ソフトウェア開発現場でも同じ改善効果をもたらす。
→ 「リーンソフトウェア開発」などが注目されている。

トヨタ生産方式とアジャイル



これは僕の個人的な意見ではない



訳：日本においてアジャイルの活動が行われることは、1986年1月の Harvard Business Review で竹内弘高氏と野中郁次郎氏が発表した「The New New Product Development Game」というセミナー資料が スクラムの起源となった点からも、とてもふさわしいと思います。

また、トヨタの大野耐一氏によって実践されたかんばん生産方式はアジャイルプロセスの知識面および実用面の源となっています。 Ken Schwaber 氏のコメント

モチベーションの維持

大量投入を前提としたこれまでの開発手法（管理手法）は、全ての開発者に対して、いかにして平均的に働いてもらうかに主眼をおいている。→ やる気のある人、スキルのある人にとってはマイナスの方向に働く管理。

やる気のない人が作成したプログラムは品質面で問題あり。
→ 後のテスト工程で品質面において問題となり、工数を浪費し、即座にプロジェクトの赤字要因となる。

いかにして開発者のモチベーションを維持するか、が重要

⑦. Leverage 基礎

№	問題点	品質への影響
7	開発者のスキル不足	低スキル開発者が作成したモジュールに対しケアできないため品質低下

JUnit適用

JUnitで作成するテストケースは、少ない種類の命令で記述することができる

JUnitは、プログラマに必要な基礎力を効率良く学習できる。

⑦. Leverage 基礎

私は、過去のプロジェクト経験にて、企業システム構築に必要不可欠で、かつJavaの入門書であまり触れられていない文法の代表格が、まさに「データクラスの配列」やHashMapであることに気がついた。JUnitによるテスト作業は、これらの基本文法を身につけるのに最適なトレーニングである。

データクラスの配列

HashMap

⑧. Embrace Change 開発手法の変革

№	問題点	品質への影響
8	ビジネスの変化が速い	仕様変更に備えた開発手法でないためプロジェクトがあたふたし品質低下

JUnit適用

JUnitでの自動テストを導入すると、実際の現場で多発する仕様変更に備えることができる。

JUnitは、テスト工程だけでなく、開発プロセス全般に変革を促した。

Outbreak Of Fight 戦いの勃発

JUnitは、上記にあげた改善効果をもたらす一方、従来の管理方法や開発支援ツールの欠点を明らかにする。そのため、従来の開発手法をベースとした管理者やツール開発者との間で戦いが勃発する。

品質管理指標

「バグ検出率による品質管理」
（キロステップあたりN件のバグを発見しないとダメという指標）

JUnitやEclipseを利用して開発する場合は、プログラム作成とテスト実施を並行して進めたり、テストケースを先に作成してからプログラムを作成するなどの開発方法が一般的である。

プログラミングと単体テストの境がないJavaの開発では計測不可能

JUnitをベースとした新しい品質管理指標を検討する必要がある。

Javaでテスト仕様書を書くということ

JUnitで記述したテストケースは、
Javaの不得意なレビューアには受け入れられない。

JavaDocにテスト仕様書相当の情報を記述しJavaDocをレビュー対象にするなどやってみたが、テスト情報が二重管理になり、間違う原因となるし、手間もかかる。

【開発者側で努力しますから．．．】

テスト仕様以外の処理を上位クラスなどに隠蔽して、方式に依存する複雑な処理を見えなくし、本来のテスト仕様が目立つようにする

【レビューア側にも努力してほしい】

Javaってそんなに難しくないから。簡単な命令しか使わないようにするから。だから、ちょっとだけJavaを習得してください。

ツールの理想像とは

JUnitは、ツール開発者に「理想のツール像」を示している。
JUnitという極めてシンプルでカスタマイズ可能なツールが
開発手法全般に大きな影響を与えた。

開発支援ツールは、開発手法と一体化し、
現場へ柔軟に適用されてこそ威力を発揮する。

理想的な開発支援ツールを作るためには、開発現場の最前線へ行き、実際のプロジェクトに深く参加する必要がある。そして、「そのプロジェクトを助けるためにはどうすれば良いか」という観点で、現場の開発者たちといっしょに考え、開発手順を改良し、ツールをカスタマイズする必要がある。

まとめ：解決策の効果と今後の課題

開発現場にJUnitを導入して効果的に使用し、適用における課題を解決していくことによって、上記の通り「品質低下要因」のひとつひとつに対して改善することができると考える。

最大のポイント：JUnitを開発現場に導入すると．．．

開発者自らが知恵と工夫と働く意欲で改善する

ソフトウェア開発作業は「人の能力」に大きく依存する。今後、私は「開発者の知恵と工夫と働く意欲」を引き出すことに念頭をおき、開発支援ツールを含めた開発プロセスの改善を実施していきたい。

最後に

鳥取三洋電機で多能工による生産性の
改善が実証された直後、工場改革チームの
メンバの方が語ったコメント

「もっと早くやっておけばよかった。
もったいない。今になって．．．」

早くやらないといけない。

FUJITSU

THE POSSIBILITIES ARE INFINITE

参考文献

1．AntとJUnitを用いた漸進的開発
～単体テストでコードの品質を徐々に向上させる～ (2000年11月)
http://www-6.ibm.com/jp/developerworks/java/010223/j_j-ant.html

2．XPエクストリーム・プログラミング入門
ケント・ベック 著、長瀬 嘉秀 監訳
2000年12月発行 [出版元] ピアソン・エデュケーション
http://www.pearsoned.co.jp/washo/prog/wa_pro37-j.html

3．NHKスペシャル （2001年5月12日(土) 放送）
「常識の壁を打ち破れ ～脱・大量生産の工場改革～」
http://www.nhk.or.jp/special/libraly/01/l0005/l0512.html
→ 番組の内容が本になっています。
常識破りのものづくり NHKスペシャルセレクション
山田 日登志 (著), 片岡 利文 (著)