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

和田　憲明（富士通株式会社）wadan@jp.fujitsu.com

あらまし　現在の開発現場には様々な「品質低下要因」が存在しており、収益悪化の原因となっている。私が JUnit を現場で実際に使用したところ、このツールは「品質低下要因」に対し驚くべき８つの改善効果をもたらした。

JUnit は極めてシンプルなテストツールだが、このツールの影響は、テスト手法のみにとどまらず、設計方法、開発管理方法、開発ツールのあり方といった開発技術の分野全般に波及している。

私はこの論文で、実際の開発現場で得た様々な改善効果を具体的に紹介する。開発現場にはたくさんの課題が存在する。「JUnit」のようなシンプルでカスタマイズ可能なツール群は、現場の開発者から「品質低下要因」を改善するための知恵と工夫を引き出す。そして、開発者の知恵と工夫こそがＳＩビジネスの収益を劇的に改善すると確信する。本当である。

１．　はじめに

１．１　開発における課題

私は２年間、開発リーダとして、ソフトウェア開発の最前線にいた。そして、現在の開発現場が抱えている多くの「品質低下要因」に直面した。主な要因を以下に列挙する。

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
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

単体テスト時の品質低下は後の工程で猛烈に工数を食いつぶすため、上記の課題を解決していかないと、ＳＩビジネスは決して好転しない。

２．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。以下に、各改善効果について具体例とともに説明する。

２．１　JUnit とは

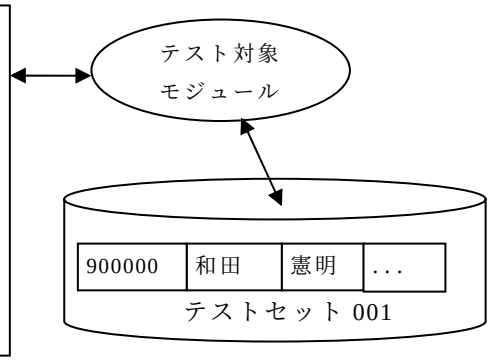
「JUnit」は、テストケースの記述と実行を支援する極めて軽量かつシンプルな Java フレームワークである。テストケースを Java で記述し、一括して自動実行することができる（概要は【参考文献１】を参照）。JUnit 自体も Java で作られているため、継承の機構などを利用して容易にカスタマイズし機能追加することができる。以下は JUnit で記述した単純なテストケースの例である。assertEquals メソッドを使用して、結果の予想値と実際の動作結果を比較する。

```
/**
*【観点】１＋２は「３」になるはずである。
*/
public void test001 {
        assertEquals( 3, calc.add(1,2));
}
```

予想値
実際の実行結果
比較ﾒｿｯﾄﾞ

次は業務的な例を示す。テストケース実行前に、従業員「和田」の情報を含む「テストセット 001」は事前にＤＢへロードしておく。

```
/**
* [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 {
  EmployeeMsg resultMsg = employee.findByKey("900000");
  assertEquals("900000#和田#憲明#ＬＭＣ技術部", format(resultMsg));
}
```



上記テストケースを実行し、業務メソッドから実際に返却される値が予想値と等しければテスト成功、異なれば失敗である。また JUnit は、まとめて実行したテストケースの数と成功数を集計し、ユーザに教えてくれる。シンプルである。この JUnit というシンプルなツールが以下の８つの改善効果を開発技術の世界にもたらしたという事実は、10 年以上も開発支援ツールの開発・適用を本業としてきた私にとって、本当に驚くべきことであった。

２．２　各課題に対する改善効果　（以下の ①～⑧ は上記１．１の課題 ①～⑧ にそれぞれ対応）

①．Improve Quality　品質向上

JUnit で書くテストケースは実行可能であるため、手抜きできず、正確に記述する必要がある。そのため、テストケース・テストデータの品質が向上し、そのテスト対象であるプログラムも品質が向上する。

私は、過去のプロジェクトで実際に作成されたテスト仕様書（Excel 文書）を検証し、以下の２つの問題点を発見した。

問題点１．あいまいな記述

テストケースの入力値について、２箇所の部分で記述があいまいだった。

	入力内容
テストケースＡ	存在しない日付
テストケースＢ	Null or 空文字

テストケースＡについて、実際に入力される日付データはテスト担当者によって様々である。例えば、2/30 と入力する人、13/32 と入力する人、0A/0B と入力する人。この場合、2/30 と 0A/0B という入力値では、テストの観点が明らかに異なるが、実際にどんな値でテストしたかは記録に残らないし、再テスト時には、値をもう一度考えないといけない。

テストケースＢについて、１行に入力値が２つ書かれており、テスト実施時にどちらの値（null、空文字）でテストされたのかわからない。私が JUnit で再テストしたところ、空文字を入力した場合は仕様通り ABCRuntimeException（ABCRuntimeException とは、java.lang.RuntimeException をプロジェクト固有クラスでラップしたもの）が発生したが、null の場合は仕様書と異なる NullPointerException が発生し、バグを発見した。（実際の仕様書の結果欄は ○ となっていた。）

問題点２．結果の目視判定が間違っていた例

異常系テストケースの多くは、確認結果欄に「ABCRuntimeException が出力される」となっていた。JUnit でテストケースを記述し実行したところ、ほとんどのテストケースでは ABCRuntimeException が発生したが、ひとつのテストケースだけ java.lang.RuntimeException が発生した。これはバグである（ちなみに仕様書の結果欄は ○ となっていた）。担当者にテスト実施方法を確認したところ、DOS 上のテストドライバから、手で値を入力し、表示された結果を目視で確認する方法だった。目視での確認作業で、ABCRuntimeException ではなく RuntimeException が表示されたことを見落としてしまったのではないかと考えられる。Excel で記述したテスト仕様書はそのまま実行されるわけではないため、あいまいさが混入しても排除できない。さらに、テスト実施時に直接手で入力し、目視で確認することをベースにしたテストは、間違いを見落としてしまう可能性がある。

JUnit で記述したテストケースは入力値、結果予想値を具体的にテストケース上に記述しておくため、障害発生時には入力値と予測値を後から確認することができ、追跡性に優れている。また、ツールで機械的に実行結果と比較することができるため、正確である。

②．Keep Quality　品質維持

JUnit で書かれたテストケースは簡単に実行できるため、コストをかけずに何回でも（毎日でも）全てを

再テストすることができる。そのため、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。

私は、ある受託開発プロジェクトにおいて、ＥＪＢ処理に仕様変更が発生するたび、常に全テストケースを実行する運用を行なったところ、数件のデグレードを即座に検出することができた。一般的に、デグレードの発生はお客様に不信感を与えてしまう大きな要素のひとつであるため、開発現場では絶対に発生させてはいけないものである。しかし、仕様変更によるモジュール修正時には、影響を受けると思われるモジュールに対する全テストは、コストの関係で部分的にしか実施されず、影響する範囲の中の重要な部分のピックアップテストで代用してしまうことが多い。JUnit では、低コストで全てのテストケースを実行できるため、影響する全ての範囲を含む十分なリグレッションテストを容易に実施することができる。

また、Excel 文書で記述されたテスト仕様書は、仕様変更時に修正が漏れても気がつかず、プログラムと乖離してしまうことがある。テスト仕様書が一部でも乖離してしまうと、どのテスト仕様書が現実と合っているのかがわからなくなり、テスト仕様書全体の信頼性が低下し、使われなくなってしまう。大きな工数をかけて作成したテスト仕様書なのだから、これは非常にもったいない。

常に全てのテストケースを実行させる JUnit では、仕様の乖離が発生したらすぐにエラーとなるため、常に修正される。そのため、テストケースの信頼性が保持できる。

③．Ease Of Management　テスト完了基準の明確化

JUnit は、テスト対象機能に対するテスト進捗を、テストケース実行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。また、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。

私は、プロジェクトでの進捗管理で、毎日必ず全テストケースを実行し、テストケース件数、テスト成功件数を簡単に把握することができた。また、集計が容易であるため、開発者・テスト担当者に負担をかけず、開発およびテストに集中してもらうことができた。

④．Education Of Test　テストの教育

JUnit は開発者に「テストの技術」を実践を通じて身につけさせることができる。

私は、あるプロジェクトで「テスト要員」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。これまでテストに対して漠然としか理解していなかった２名が「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。

テストの教育は難しい。特に、実際の業務システムに対するプログラム経験がない人がテスターになった場合、当事者にとっては、どういった観点でテストケースを作成すればいいか、かいもく見当がつかない状態である。「テスト実施要員」としてのテスト作業は、結局テスト全体を把握できないため、テストに対する理解がなかなか深まらず、スキルアップできない。

JUnit では、テストは自動実行され、結果確認工数も不要なため、従来の「テスト実施要員」という役割は必要ない。そのため、テスト担当者はテストに関する作業を全て行なうので、テストに関する全てのこと（テストケース作成、テストデータ作成、テスト実行、結果の分析など）を実際のテスト作業から学ぶことができ、テスト全体のスキルが向上する。

⑤．Motivation　働く意欲

JUnit は、開発者・テスト担当者の「働く意欲」を高めることができる。

私は、開発現場で JUnit を適用することによって、開発者・テスト担当者の働く意欲を向上することができた。まず周知の事実として、Excel でのテスト仕様書作成作業、テスト実行時の手入力作業・目視での確認作業は疲れる。また、プログラムを作成しないため、プログラマとしてのストレスがたまる。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。

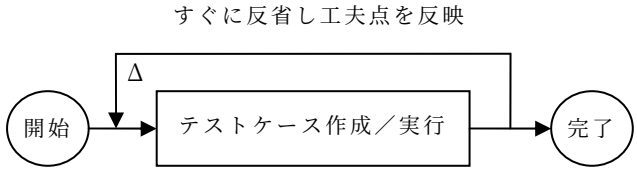
人間のやる気によって品質が大きく左右されるソフトウェア開発作業では、楽しさは大事である。目標が見えず、達成感が感じられないまま淡々とプログラムしている開発者と、目標がはっきりしていて、達成感のある仕事を任せられ、個人ベースでの工夫を生か

せる開発者の生産性を比較すると、おおげさでなく数倍もの開きがある。

⑥．Versatility　知恵と工夫

JUnit は、開発者・テスト担当者の知恵と工夫を引き出す効果がある。

私は、実際に JUnit を導入することにより、個人が知恵と工夫によりテストの生産性を向上する場を提供することができた。JUnit を利用したテスト作業では、「テストケースを作成／実行」を次々に繰り返す。テスト担当者は、その繰り返しの中で、反省や工夫を次の作業にすぐに反映することができる。また、工夫の対象はテストケースだけに留まらず、テストの道具である JUnit そのものを現場の開発者が即座に改造して作業効率を上げた。



テスト作業は、「テスト仕様書作成・テストデータ作成」と「テスト実施」に分業されている場合が多い。ここで問題となるのが「分業による生産性の低下」である（生産性の“向上”ではないことに注意）。

非常に興味深い番組がＮＨＫで放映された（【参考文献３】より）。鳥取三洋電機で携帯電話生産ラインを改革する話である。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。」

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。

ソフトウェア業界では、個々のソフトウェアモジュール仕様はそれぞれ異なるため、ひとつひとつ「手作り」する。そのため、生産性は「人」に大きく依存する。開発作業を分業してしまうと、それぞれの作業に対するゴールが不明確となり、達成感が薄れる。テスト仕様書を作成する作業は、その仕様書に基づいてテストを実施してこそ、貴重なフィードバックを得られるのだが、分業してしまうと達成感もフィードバックも得られない。

JUnit による開発・テストでは、業務ロジック設計、業務ロジック作成、テストケース作成、テスト実行をひとりで行なうため、各作業での工夫や、各作業にまたがった工夫を作業改善に直結することができ、生産性を大きく向上できる。

また、JUnit は容易にカスタマイズできるため、作業の無駄を現場の開発者が発見した時点で即座に JUnit を機能強化し、直後のテスト作業から効率を向上することができた。例えば、複数個のテストケースを作成したところ、同じようなファイル制御処理をそれぞれのテストケースで数十行に渡って記述していたが、その制御処理を共通処理として上位クラスに隠蔽することで、テストケースを作成する時間が短くなり、テストケース自体の可読性も向上した。

⑦．Leverage　基礎

JUnit は、プログラマに必要な基礎力を効率良く学習できる。

私は、低スキルの開発者に対しては、JUnit によるテスト作業を担当させるのが良いと考え、実際にテスト作業を担当してもらった。その結果、テストケース作成当初は、HashMap や配列などといった基本的な文法に対する理解が不足して苦戦したが、テストケースの作成と実行を通して基本的な文法が身につき、開発者としてのスキルをアップさせることができた。JUnit でのテストケース作成では、業務ロジックメソッドに渡す引数の組み立てと、戻り値の解析処理を中心にプログラミングする。使用する文法は、データクラス、配列、HashMap に関連するものが中心である。私は、過去のプロジェクト経験にて、事務処理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。

⑧．Embrace Change　開発手法の変革

JUnit は、テスト工程だけでなく、設計そのものにまで変革を促した。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 をベースとした新しい開発手法「エクストリーム・プログラミング」には、テスト実行が自動化され、リグレッションテストを容易に行なえることから可能となる、様々な新しい開発手法が盛り込まれている。（【参考文献２】）

２．３　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。

１．品質管理指標

「バグ検出率による品質管理」について。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。そのため、プログラムのコーディングが完了してから単体テストを行なうことが前提となっている「バグ検出率」は、測定できない。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。

２．Java でテスト仕様書を書くということ

JUnit で記述したテスト仕様書は Java で記述されているため、Java の不得意なレビューアには受け入れられにくい。開発者側は、テスト仕様以外の処理を上位クラスなどに隠蔽してテスト仕様が目立つようにする工夫、JavaDoc 部分に日本語でテスト観点を記述する工夫、テストソースから表形式のテスト仕様書を自動生成する工夫、などを考える必要がある。

３．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。

３．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。この改善における最大のポイントは「開発者自らが知恵と工夫と働く意欲で改善する」という点である。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。

４．最後に

ハード、ＯＳは、この十数年間で劇的な変化を遂げた。実は開発手法でも劇的な変化が起こりつつある。JUnit によるテストファースト手法や、アジャイル開発手法は、最も大きな変化と感じている。これらを大規模プロジェクトに適用するうえで解決しないといけない課題は多いが、積極的に検討して良い部分を素早く導入し、開発者の知恵と工夫を最大限に活用すれば、ＳＩビジネスは飛躍的な収益改善をなし遂げることができると確信している。最後に以下の言葉で締めくくりたい。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が語ったコメントである。

「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。

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「常識の壁を打ち破れ　～脱・大量生産の工場改革～」　2001 年 5 月 12 日(土) 放送
http://www.nhk.or.jp/special/libraly/01/l0005/l0512.html