外皮・躯体と設備・機器の総合エネルギーシミュレーションツール「BEST」の開発（その 7）

ソフトウェアテスト手法

Development of an Integrated Energy Simulation Tool
for Buildings and MEP Systems, the BEST (Part 7)

Software Testing Methods

正会員　○丹羽　勝巳（日建設計）　　特別会員　村上　周三（慶応義塾大学）
正会員　石野　久彌（首都大学東京）　　正会員　郡　公子（宇都宮大学）
正会員　柳井　崇（日本設計）　　正会員　小池　正浩（竹中工務店）
正会員　瀧澤　博

Katsumi NIWA*[1]　Shuzo MURAKAMI*[2]　Hisaya ISHINO *[3]　Kimiko KORI*[4]
Takashi YANAI*[5]　Masahiro KOIKE *[6]　Hiroshi TAKIZAWA

*[1] Nikken Sekkei　*[2] Keio University　*[3] Tokyo Metropolitan University
*[4] Utsunomiya University　*[5] Nihon Sekkei Inc.　*[6] TAKENAKA Corporation,

Total Energy Saving Technology, To Unify Building Envelopes with MEP Facilities, BEST(Building Energy Simulation Tool)、Software Testing
Abstract　This paper presents software testing methods for BEST program. Software testing is the process used to help identify the correctness, completeness, and quality of developed computer software. Software testing is very important and it is a trade-off between budget, time and quality.

はじめに

最近のプログラム開発においては、システムの複雑化や大規模化、開発期間の短縮化とともにテストの重要性が増しており、ソフトウェア開発の費用の 30～50%がテストのためのコストといわれている（図１）※1), 2)。

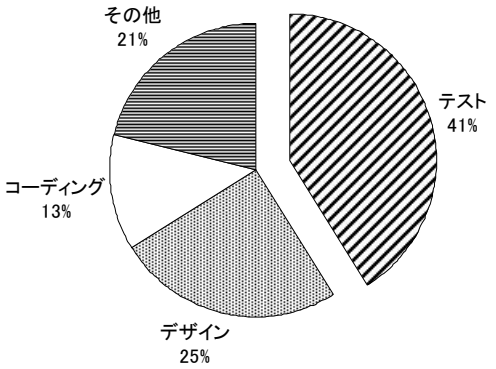

図１　ソフトウェア開発に必要な費用の内訳

BEST プログラムのソフト開発においても、従来のエネルギー消費量算出ツールの開発（例えば、HASP/ACLD/ACSS）に比較して、ソフトウェアテストに重点を置いて開発を進めている。多機能化や開発途中における機能追加の要請に応えつつ、ソフトウェアの品質を保つためには、開発の初期段階からテストを計画・実行することが必要である。

ＢＥＳＴプログラム開発におけるテストの方針は次の通りであり、本報ではその詳細について今後の課題とともに紹介する。

1)　プログラム作成者によるテスト
テスト自動化の有力なツールである JUnit を用いた単体テストを行う

2)　テストチームによるテスト
開発者とは独立したテストチームを立ち上げ、システムテスト、受け入れテストなどによって、プログラムが公表可能なレベルか検証を行う

3)　IEEE 829 に準ずるテスト
ソフトウェアテストの世界標準である IEEE 829（Standard for Software Test Documentation）に則ってテストプラン立案し、開発と無縁のクリーンなマシンでテストを実行する。

１．ＢＥＳＴプログラムの規模

テスト戦略を検討する上で第一に重要となるのがソフトウェア規模を知ることである。時間や人・費用に限りがある中で、ソフトウェア規模を念頭にテスト範囲・テスト環境・テスト方法を決定する必要がある。

表 1 に市販の解析ツールを用いて BEST0703(パイロット版)を解析した結果を示す。この解析は、ソフトウェア・メトリクス（品質測定）と呼ばれており、ソフトウェア開発をさまざまな視点から定量的に評価したものである。代表的な例としてはコード行数、設計仕様ドキュメント量、開発工数、期間、人数などが挙げられる。

表１　BEST0703 のソフトウェア規模

メトリクス（品質測定）	値
ファイル数	262
パッケージ数	32
全クラス数	294
公開クラス数	230
インタフェース数	32
全メソッド数	1488
公開メソッド数	1301
行数	27,290
命令行数	17,812
コメント行数	7,332
ステートメント数	11,233
１パッケージ当たり平均クラス数	9.2
１クラス当たり平均メソッド数	5.1
１メソッド当たり平均ステートメント数	7.5
１ファイル当たり平均行数	104.2
１ファイル当たり平均命令行数	68.0
平均コメント比率	26.9%

表２　BEST0703 と他のソフトウェア規模の比較

ソフトウェア	命令行数	
BEST0703	17,812 行	（約２万行）
薄型テレビ[3),4)]	600,000 行	（ 60 万行）
HDD 内蔵 DVD レコーダ[3),4)]	1,000,000 行	（100 万行）
カーナビ[3),4)]	3,000,000 行	（300 万行）

表１より、全部でファイル数が 262 存在し、コメント行や空白行を除いた有効ソースコード行は約 20,000 行であったことが分かる。

表２に BEST0703 のソフトウェア規模を他の代表的なソフトと比較した結果を示す。有効ソースコード行が 100,000 行以下で、かつファイル数が 1,000 以下であることから、現段階の Best は小規模 Java アプリケーションに分類される。小規模アプリケーションにおいては単体テストの比重が相対的に高くなり、JUnit によるテストが重要となる。JUnit による実際のテストコードと、コードカバレッジについては後述する。

２．プログラムの開発プロセスとテストの対応

システムの開発とテストのプロセスはＶ字モデルと呼ばれる以下の図で表現されることがある（図２）。

開発プロセスは、要求分析（システムに対する要求を分析する）、分析（システムが「何をするのか」を把握する）、設計（システムを「どう作るのか」）を決める）、詳細設計（プログラミングの詳細を決める）、実装（詳細設計を基にコーディングする）の各フェーズに分かれる。この図に示されるように、各開発フェーズに対応するテストフェーズがあり、例えば分析作業の成果はシステムテスト（総合テスト）によって確認される。いうなれば PDCA サイクルにも通じる考え方である。以下にテストの各フェーズについて詳述する。

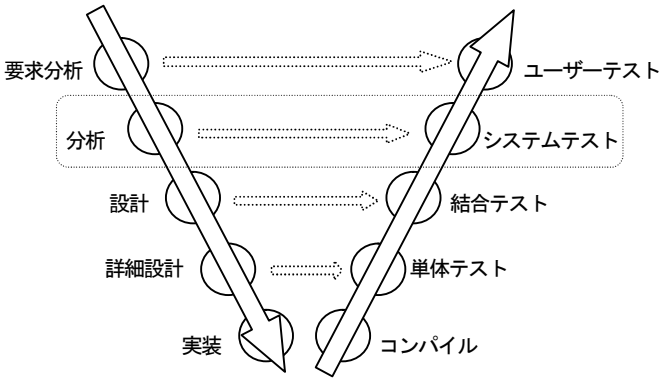

図２　ソフトウェア開発プロセスとテストの対応関係[5)]

３．単体テスト

単体テストは、モジュールを単位としたテストである。BEST で使用している Java 言語においては、クラス、メソッドが単体テスト対象となる。この単体テストはテスト工程には含めず、プログラミングの一環として位置づけられることもある。

単体テストは、JUnit を用いてプログラム担当者自身が行う。JUnit は単体テストを効率よく行うためのツールであり、Java のプログラミング単位であるクラス毎に単体テストを実行する。作成したテストケースとプログラムの実行結果が合致しているかチェックする機能を持っている。チェック結果は、テストケースをクリアすれば緑で、クリアしなければ赤のバーグラフで表示され、一目瞭然の結果として示される。

JUnit の利用により、次のメリットが得られる。
①プログラム仕様の明確化、②テストの自動化、③テスト結果のドキュメント化、④プログラム完成の明確化などが、その代表的なものである。

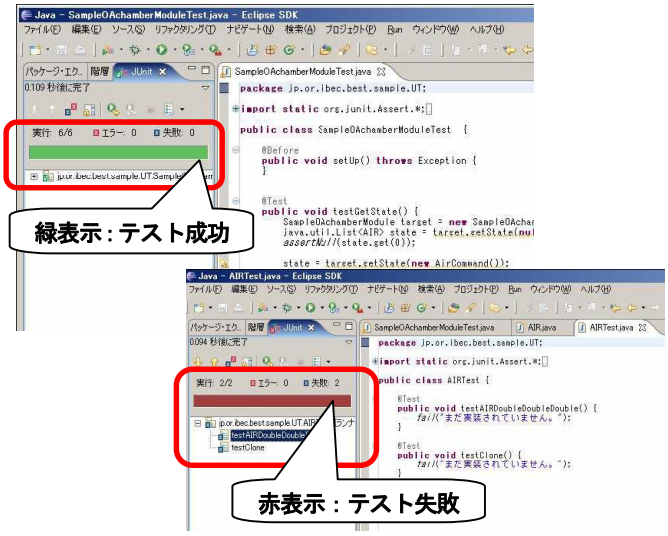

図３　JUnit によるテストの実行例

表３　Junitによる実際のテストコード(例)

```
package building_test_edu;
import static org.junit.Assert.*;
import java.lang.reflect.Field;
import java.util.ArrayList;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import multiSpace.MultiSpaceData;
public class MultiSpaceDataTest {
        @Before
        public void setUp() throws Exception {
        }
        @After
        public void tearDown() throws Exception {
        }
        @Test
        public void testGetMultiSpaceData() {

                String[] args = new String[]{"room","a"};

                MultiSpaceData multi_space_data = new MultiSpaceData(args);

                String[] rcode = multi_space_data.getMultiSpaceData();

                assertEquals("GetMultiSpaceData 戻り値のテスト","room",rcode[0]);
                assertEquals("GetMultiSpaceData 戻り値のテスト","a",rcode[1]);
        }
```

テストに利用する Assert クラスのインポート

JUnit で定義されている After,Before,Test をインポート

テスト対象の初期化：例外発生時の記述

テスト対象の後処理：例外発生時の記述

テストメソッドの記述

assert Equals()：2 つのオブジェクトが等しいことを表明。等しければテスト成功。
第１引数：メッセージ
第２引数：期待値
第３引数：処理結果

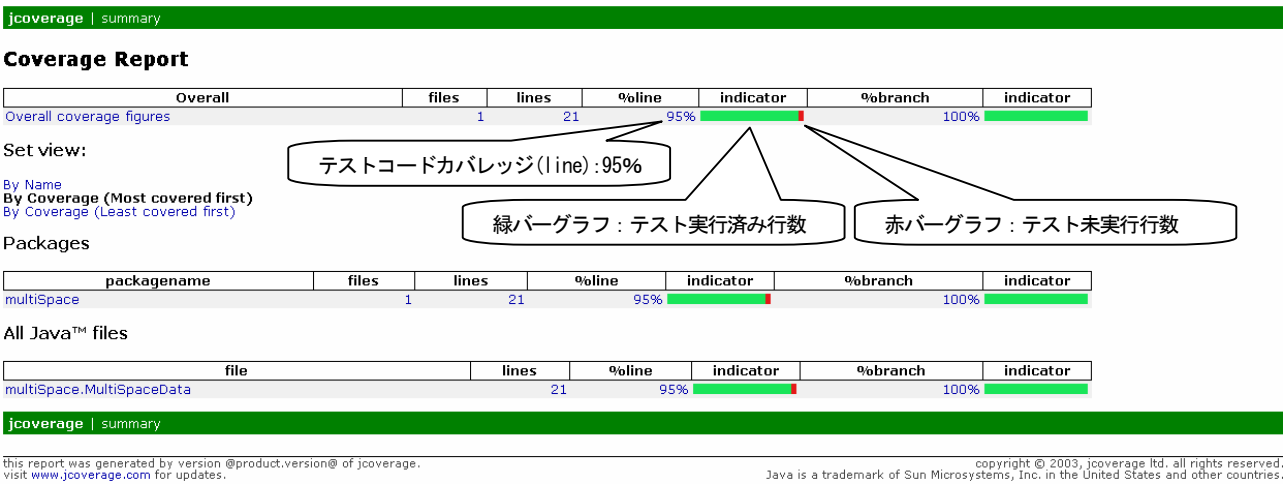

図４　jcoverageによるコードカバレッジレポートの例

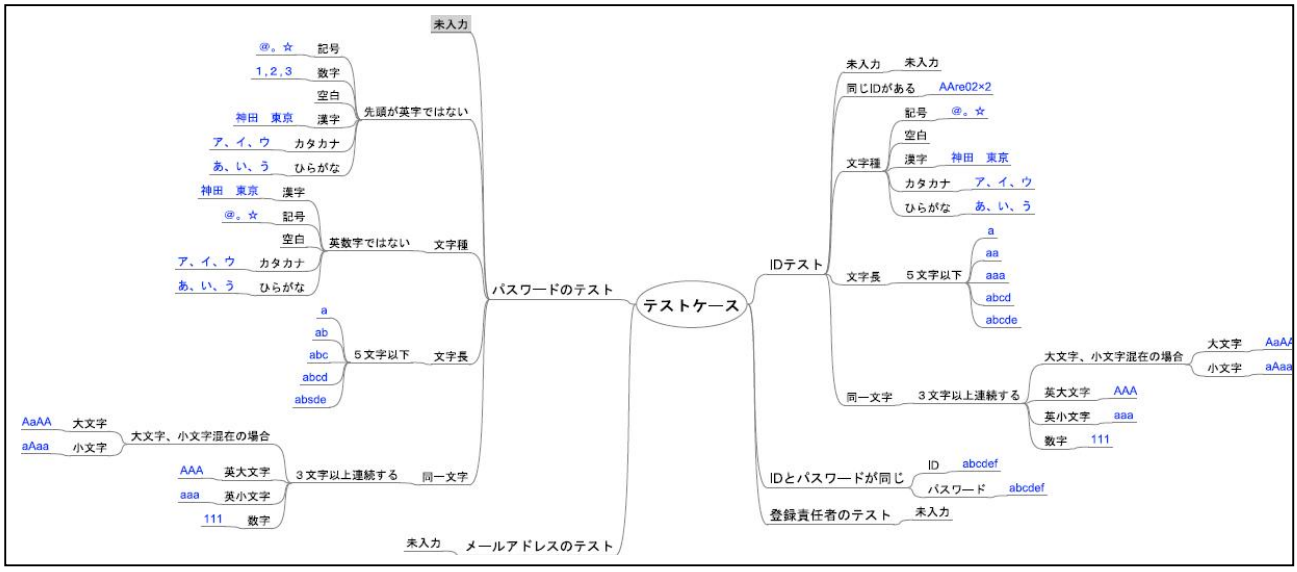

図５　テストケース例(マインドマップによる表現)

今回のJUnitを用いた単体テストの実際のテストコードの一例としてMulti Space Dataクラスの戻り値テストの例を表３に示す。この例では、assertEqualsを利用して引数や戻り値のチェックを行っている。

単体テストとしては、他にも例外テスト（null 値テスト、メソッド順序入れ替え）、同値クラステスト（代表値を利用したテスト）,境界値テストなどを実行している。

さらに、テストコードのカバレージ（網羅率）の検証例を図４に示す。テストケースの範囲や数が十分なものかどうかについての検討ツールとして、カバレージモニタ(jcovarage)が用意されている。jcoverrage が作成するレポートはグラフで表示され、どのモジュールがどの程度のカバレージであるかを示している。レポートでは全体のカバレージ、パッケージとファイル毎にカバレージ度が緑（実行)、赤（未実行）で表示される。

カバレージモニタを利用して カバレージを測定し、たとえばカバレージ 95%以上を単体テストの完了基準とすることで、テストケースの漏れを最小限に抑えることができる。

４．結合テスト（統合テスト）

単体テストは開発マシンで行うことが多いが、本来のテスト工程に移行した結合テスト以降は、実際にユーザーが使用するマシンを使ってテストを実施することとなる。すなわち、ソフト開発とは無縁のテストチームを立ち上げ、クリーンな環境のパソコンに、実際にソフトのインストールから行い、テスト環境を構築することが必要である。

テストプランは文書化を行い具体的な成果品とすることで、テストチームで共有化することが重要である。BEST のソフトウェアテストにおいては、プログラミングテストの世界標準として規定されている IEEE 829 -1998 （Standard for Software Test Documentation、表-４）を文書化のテンプレートとして利用することとしている。

IEEE829 によるテスト計画文書は、小規模アプリケーションでは結合テスト以降を対象として作成される。テスト計画書においては、テストケースの作成がポイントとなり、マインドマップによるテストケース作成が有効である。図５に今回のテストケースの一例を示す。

５．まとめ

ＢＥＳＴプログラム開発におけるテストの方針について紹介した。ＢＥＳＴプログラムは、現在も新バージョンの開発中であり、その規模が拡大中である。そのプログラム内容も、レガシーコードの利用や衛生・電気分野への拡張など多種なクラスを使用することになり、そのソフトウェアテストは複雑なものとなる。プログラムの開発プロセスに併せてテスト工程も進行中である。限られた時間や人・費用のもとでバグのより少ない、品質の高いプログラムを目指すために更にソフトウェアテスト手法を充実させていく予定である。

表-４　IEEE 829-1998 (Standard for Software Test Documentation) によるテストプランのテンプレート

- Test plan identifier （テスト計画書文書番号）
- References（レファレンス）
- Introduction（はじめに）
- Test items（テストアイテム）
- Features to be tested（テストする機能）
- Features not to be tested（テストする必要のない機能）
- Approach（アプローチ）
- Item pass/fail criteria（テストアイテムの合否判定基準）
- Suspension criteria and resumption requirements（中止基準と再開要件）
- Test deliverables（テスト成果物）
- Testing tasks（テスティングタスク）
- Environmental needs（実行環境）
- Responsibilities（責任範囲）
- Staffing and training needs（人員計画、トレーニングプラン）
- Schedule（スケジュール）
- Risks and contingencies（リスクとその対策）
- Approvals（承認）

【引用文献】
1)現場の仕事がバリバリ進むソフトウェアテスト手法､高橋寿一･湯本剛著､技術評論社 2006 年 6 月
2) Sharon Waligora and Richard Coon,“Improving the Software Testing Process in the SEL”
http://sel.gsfc.nasa.gov/website/documents/techstudy/impr-stp.htm
3)日経エレクトロニクス　2000 年 9 月 11 日号(No.778)
4)ずっと受けたかったソフトウェアエンジニアリングの授業２、鶴保征城・駒谷昇一著、翔泳社、2006 年 10 月
5)JUnit と単体テスト技法、福島竜著、ソフト・リサーチ・センター2006 年 8 月
【謝辞】本報は、(財) 建築環境・省エネルギー機構内に設置された産官学連携による環境負荷削減のための建築物の総合的なエネルギー消費量算出ツール開発に関する「BEST 開発普及事業研究会 (村上周三委員長)」ならびにアーキテクチュア検討部会（坂本雄三部会長)、テスト WG（丹羽勝巳主査）の活動成果の一部であり、関係各位に謝意を表するものである。テスト WG 名簿（順不同）主査:丹羽勝巳（日建設計)、委員：柳井崇（日本設計)、瀧澤博（自営)、小池正浩（竹中工務店)、協力委員：石野久彌（首都大学東京)、郡公子（宇都宮大学)、浅野良吾、神田千恵美 (以上、シグマ・システム・エンジニアリング)、事務局：生稲清久（建築環境・省エネルギー機構)