OUKA
Osaka University Knowledge Archive

Title	クラス間関係を利用した単体テストおよび静的検査の網羅率可視化手法
Author(s)	武藤, 祐子; 岡野, 浩三; 楠本, 真二
Citation	
Version Type	VoR
URL	https://hdl.handle.net/11094/50598
rights	ここに掲載した著作物の利用に関する注意 本著作物の著作権は情報処理学会に帰属します。本著作物は著作権者である情報処理学会の許可のもとに掲載するものです。ご利用に当たっては「著作権法」ならびに「情報処理学会倫理綱領」に従うことをお願いいたします。
Note	

Osaka University Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

Osaka University

クラス間関係を利用した 単体テストおよび静的検査の網羅率可視化手法

武 藤 祐 子† 岡 野 浩 三† 楠 本 真 二†

近年ソフトウェアの可視化の技術が重要視されてきている．本研究ではソフトウェア品質のうち機能性を評価するためにプログラムの仕様と実装の一致性を2つの観点から可視化する方法を提案する．具体的には単体テストおよび静的検査の結果を重み付き有向グラフで表現する．提案手法によって得られる知見を評価するため，実際にプロトタイプツールを作成し評価例題として在庫管理プログラムに適用した．これらから，提案手法からソフトウェア品質の推定および仕様記述の改善戦略を得ることができることを確認した．

A Visualization Technique for the Coverage of Unit Testing and Static Cheking using Caller-Callee Relationship between Classes

YUKO MUTO,† KOZO OKANO† and SHINJI KUSUMOTO†

Software visualization has attracted lots of attention. We propose a new hybrid method based on both of the two categories, especially from point of view of development process. The proposed method visualizes coincidence between specification and implementation from two aspects; unit testing and static checking. In the method, the ratio of the coincidence are drawn by pie charts which represent classes of the target software. Whole software is represented in a weighted digraph structure. We have prototyped a tool implemented the proposed method. We have evaluated the availability of the proposed method by applying the tool to the warehouse management program. As a result, we conclude that the proposed method provides the estimation of software quality and the strategy to improve specifications of two checking methods.

1. はじめに

近年，ソフトウェア規模の増大に伴い，ソフトウェアの理解を助けるためにソフトウェアの可視化の技術が重要視されてきている．可視化対象は以下の2つに大別できる．プログラムやソフトウェアコンポーネントの構造に着目し，プログラムのフローをPDG(Program Dependency Graph)として図示する手法[1)]や，ソフトウェアプロセスに着目し，各種ソフトウェアメトリクスを時系列で可視化する手法[2)]がある．

また可視化対象の粒度もコードレベルから，オブジェクト，クラス，ファイル，ライブラリなど複数に分かれる．オブジェクト指向プログラムではクラスに着目しクラス間の関係を図示する試みがなされている[3)]．

プログラムやソフトウェアコンポーネントの構造を可視化する手法はいくつか提案されている[4)5)]が，ソフトウェアプロセスに着目したもの，特にソフトウェアの品質の可視化に関する研究はまだ少ない．

ソフトウェアの品質は国際標準化機構において定義されており[6)]，機能性を満たすためには少なくとも仕様に記述されている内容が実装されている必要がある．

ソフトウェア開発における仕様と実装の一致性を測る手段として単体テストや静的検査などがある．

単体テストとはモジュール単位で設計された仕様を満たすかどうかを確認するテストである．これにはテスト仕様書で想定するケースに漏れがある場合はそのケースのテストが行われないため，テストの品質がテストケースに依存する問題点がある．

一方，プログラムを実行せずにソースコードを検査することを一般に静的検査という．静的検査を行うツールの一つとしてESC/Java2[7)]がある．これはメソッドが満たすべき性質を仕様としてJML(Java Modeling Language)[8),9)]を用いて記述されたソースコードに対

† 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University

して，ソースコードが仕様と矛盾しないか否かを判定するツールである．このツールの問題点として，検査の品質が記述した仕様の質に依存するということや，結果がクラス単位であり，かつテキストで出力されるため，クラス間の呼び出し関係が読み取りにくく，結果が分かりづらいということが挙げられる．

ソフトウェアの品質向上のために静的検査と単体テストを組み合わせることが考えられ，文献10)の手法が提案されている．欠陥を自動で発見することを目的とし，ESC/Java2を利用した静的検査により出力された反例情報を用いて単体テストのテストケースを生成する．この手法の問題点として，静的検査の結果が妥当でなかった部分のみを用いており，妥当であった部分に関しては使用していないことが挙げられる．

本研究では単体テストおよび静的検査の結果とクラスにおけるメソッド呼び出し情報を重み付き有向グラフとして可視化する手法を提案する．このグラフのノードはクラスに対応し単体テストおよび静的検査の結果を円グラフで表現し，エッジはクラスの呼び出し関係に対応する．これにより二つの検査手法を同一のフォーマット上で比較することができ，ソフトウェア品質の推定を補助し，またそれが分からない場合は改善のための戦略を提供する．

提案手法を実装したツールを統合開発環境 Eclipseのプラグインとして試作し，観測できる情報を評価するため実験を行った．単体テストおよび静的検査のカバレッジや単体テストのテストケース品質，静的検査の仕様記述品質を用いてソフトウェア品質を推測する仮説を立て，試作したツールを JML による仕様が記述された在庫管理プログラム[11]に対して適用した．仮説に対する評価を行い，結果として，推定したソフトウェアの品質およびテストケース記述の状況は現状と一致した．

以降，2章で本研究で用いる用語と既存のツールについて述べ，3章で提案手法について説明する．4章において実装したツールについて述べ，5章で仮説および実験の結果とその考察を論じる．最後に6章で主な結果と今後の課題についてまとめる．

2. 準　　備

本章では単体テストおよび静的検査に関する一般的な定義や使用されるツールについて述べる．

2.1 単体テスト

単体テスト とはモジュールを単位として行うテストである．Java において用いられるツールとして JUnit[12]，テストの網羅率を算出する JCoverage[13]，

```
 1: class Main {
 2:   public static void main(String[] args) {
 3:     Person p = new Person();
 4:     p.setFullName("John Smith");
 5:     System.out.println(p.getFamilyName());
 6:   }
 7: }
 8: class Person {
 9:   private String fullName = "";
10:   public Person() {}
11:   /*@ public behavior
12:     requires nm != null && !nm.equals("");
13:     ensures fullName.equals(nm); @*/
14:   public void setFullName(String nm) {
15:     fullName = nm;
16:   }
17:   public String getFamilyName() {
18:     return fullName.split(" ")[1];
19:   }
20: }
```

図 1 Main クラスおよび JML の記述された Person クラス
Fig. 1 Main class and Person class with JML

djUnit などがある．djUnit[14]は JUnit と JCoverage を手軽に使えるようにしたツールである．Eclipse プラグインとして提供されており，カバレッジ・レポートをエクスポートする機能を持つ[*]．

2.2 静的検査

2.2.1 JML

JML[8),9)]とはソースコードに対して満たすべき仕様を表現するための言語であり，Java ソースコード中にアノテーションとして記述する．DbC(Design by Contract)[15]に基づいた事前条件や事後条件などの制約を記述することができる．

JML の記述例として**図 1** の Person クラスを取り上げる．Person クラスは名前を表す *fullName* フィールド，コンストラクタ，*fullName* に対する setter である setFullName メソッド，および姓を得る getFamilyName メソッドを持つ．setFullName メソッドにおいては，引数に与えられた格納したい名前 *nm* が Null や空文字列であってはならないため 12 行目のように事前条件を記述し，このメソッドの処理が完了した時点において *fullName* フィールドと引数 *nm* が等しい必要があるため 13 行目のように事後条件を記述する．

2.2.2 ESC/Java2

ESC/Java2[16]は JML で仕様が記述された Java ソースコードに対して仕様とソースコードが妥当であるかを検査するツールである．完全性も健全性も保障されていないが効率的にバグ発見ができるため，軽量形式手法のための有用なツールとみなされている．現

* djUnit の内部で使用している JCoverage に変更が加えられている部分があり，分岐網羅率に関してはオリジナルの JCoverage とは異なる値となる．

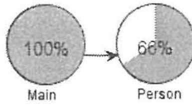

図 2 Main クラスおよび Person クラスの静的検査結果
Fig. 2 Visualization of static analysis for Main class and Person class

バージョンでの対象 Java のバージョンは Java1.4 であり，基本的にはソースコードに JML を記述しておく必要があるが，Collection，regex，io など主要なライブラリについてはあらかじめ JML 仕様が与えられており，実用性は高い．このツールの入力は 1 つの Java ソースコードファイルである．出力は各メソッド毎の妥当であるまたは妥当でないという結果であり，妥当でない場合は反例情報が同時に出力される．図 1 の Person クラスを ESC/Java2 で検査すると，コンストラクタおよび setFullName メソッドは妥当であり，getFamilyName メソッドは配列の添え字が大きすぎる可能性があるため妥当でないと出力される．

3. 提案手法

本章では提案手法および用いる定義について述べる．

3.1 概　　要

図 1 に対して静的検査を行った結果を可視化したものを**図 2** に示す．単体テストおよび静的検査のカバレッジの値を重み付き有向グラフとして表現する．その際クラスの呼び出し関係を用いる．重み付き有向グラフのノードはクラスに対応し，単体テストまたは静的検査のカバレッジを円グラフで表現する．エッジは呼び出し関係に対応する．呼び出し回数をエッジの重みとし，エッジの太さで表現する．

3.2 手法によって期待できる効果

クラスの品質が可視化できるだけではなく，クラスの呼び出し関係をたどってもう少し大きな単位での品質を評価することができるようになる．複数の異なった尺度で可視化することにより，より精緻な解析が可能になると期待できる．

3.3 定　　義

用いる単体テストおよび静的検査のカバレッジについて述べる．なおいずれもクラスを単位としている．

3.3.1 単体テストのカバレッジ

単体テストのカバレッジは命令網羅率を用いる．その理由は命令網羅率は最も単純でありわかりやすいこと，分岐網羅率は djUnit で再計算が行われており JCoverage の結果と異なること，条件網羅率は JCoverage や djUnit が対応していないことである．

3.3.2 静的検査のカバレッジ

クラス A の持つコンストラクタおよびメソッド数を $M(A)$ とし，そのうち妥当である数を $M_{passed}(A)$ とする．このときクラス A の静的検査のカバレッジ $C_s(A)$ を

$$C_s(A) = M_{passed}(A)/M(A) \qquad (1)$$

と定義する．

図 1 の Person クラスを ESC/Java2 で検査すると，コンストラクタおよび setFullName メソッドは妥当であり，getFamilyName メソッドは妥当でないと判断される．従って $M_{passed}(Person) = 2$，$M(Person) = 3$ であり，式 (1) より Person クラスの静的検査のカバレッジ $C_s(Person)$ は 66%となる．

3.3.3 呼び出し関係

クラス A がクラス B の持つコンストラクタおよびメソッドを n 回呼び出す場合，クラス A はクラス B を n 回呼び出すと定義する．呼び出し回数 n はソースコード中に登場する回数である．

図 1 の Main クラスの main メソッドでは，3 行目で Person クラスのコンストラクタを呼び出し，4 行目で Person クラスの setFullName メソッド，5 行目で getFamiliName メソッドを呼び出している．従って Main クラスは Person クラスを 3 回呼び出していることになる．

4. 実　　装

本章では作成したツールについて述べる．

4.1 概　　要

Eclipse のプラグインとして実装を行った．規模は空行とコメント行を除いておよそ 2000 行であり，パッケージ数は 14，クラス数は 33 である．

単体テストや静的検査結果のカバレッジデータを記録する XML 文書を中間ファイルとして用い，それを得るために Perl スクリプトを実装した．単体テストに用いる Perl スクリプトは djUnit のカバレッジレポートを読み込み XML 文書を生成し，静的検査に用いる Perl スクリプトは ESC/Java2 を実行し出力結果から XML 文書を生成する．

4.2 処理の流れ

処理の流れを**図 3** に示す．単体テストおよび静的検査結果それぞれから生成される XML 文書からカバレッジ等のクラス情報を取得し，対象ソースコードから呼び出し関係を求め，重み付き有向グラフを生成して表示する．

4.3 開発環境

開発言語は Java 1.6 であり，Eclipse Galileo 上

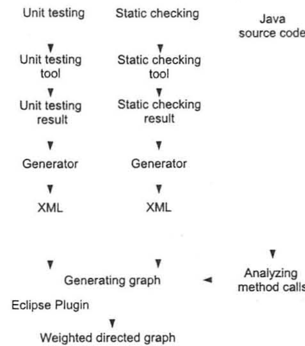

図 3 処理の流れ
Fig. 3 Flow of the process

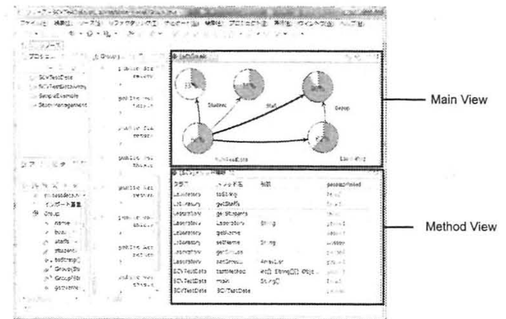

図 4 スクリーンショット
Fig. 4 Screenshot

で開発を行った．外部ライブラリとして MASU[1] と JUNG[17] を用いた．

4.4 仕 様

4.4.1 入 力

入力は対象ソースコードディレクトリ，XML 文書生成コマンド，XML 文書設置ディレクトリである．解析対象言語は Java であり，ESC/Java2 の制約によりバージョン 1.4 までに対応する．対象ソースコードディレクトリ以下の.java ファイルが解析対象となる．XML 文書生成コマンドは，静的検査や単体テストを実行し XML 文書を生成するスクリプトを実行するものである．XML 文書設置ディレクトリは XML 文書生成コマンドにより生成される XML 文書を設置するディレクトリであり，ここに設置された XML 文書が Eclipse プラグインに読み込まれる．

4.4.2 ビュー

図 4 にツール全体のスクリーンショットを示す．このツールは重み付き有向グラフを表示するメインビューとそこで選択されるクラスが持つメソッドを表示するメソッドビューの 2 つのビューを持つ．

5. 評 価 実 験

提案手法から得られる情報を評価するため実験を行った．推測できる事柄に対する 1 つの適用例およびツールの有用性について考察した．

5.1 提案手法より推測できる事柄

静的検査の仕様記述および単体テストのテストケース記述が妥当である場合とそうでない場合それぞれについて考える．記述が妥当である場合は，結果を信頼することができ，それを用いて他方の検査およびソフトウェアの品質を推定することができる．記述が妥当でない場合は，他方の検査結果を用いて記述の問題点を推定することができる．

なお記述が妥当であるかそうでないかを判別することはできておらず，今後の課題である．

5.1.1 静的検査のカバレッジが高く，単体テストのカバレッジが低い場合

この場合推測できることを**表 1** に示す．

静的検査の仕様記述が妥当である場合，静的検査結果を信頼することができ，その結果の低さからソフトウェア品質は低いと言える．にもかかわらず単体テストのカバレッジは高くなっているため，単体テストのテストケース記述においてテストを行うパターンが不足していると考えられる．

静的検査の仕様記述が妥当でない場合，それを修正する必要がある．その際単体テストの結果を信頼すると，静的検査の仕様記述が一般的すぎると推測できる．

静的検査と単体テストを入れ替えても同様である．

5.1.2 静的検査のカバレッジが低く，単体テストのカバレッジが高い場合

この場合推測できることを**表 2** に示す．説明については同様なので省略する．

5.2 方 法

図 5 に示すクラス図を元に実装された在庫管理プログラム[11] を評価対象として，ツールに適用した結果を分析する． このプログラムにはソースコード中に JML が厳密に記述されており，制約条件という意味での仕様と実装の一致率が高く，ソフトウェア品質が高いと言ってよい．

単体テストを行うために各クラスに対応するテストケースを独自に記述した．コンストラクタと setter/getter メソッドのみを実行することを方針として記述したため，不充分なテストケースとなっている．

表 1 静的検査結果のカバレッジが高く，単体テスト結果のカバレッジが低い場合
Table 1 The coverage of unit testing is high, the coverage of static analysis is low

		静的検査の仕様記述	単体テストのテストケース記述	ソフトウェア品質
静的検査の仕様記述	妥当	OK	パターン不足	高
	妥当でない	制約の範囲が小さすぎる	-	-
単体テストのテストケース記述	妥当	制約の範囲が小さすぎる	OK	低
	妥当でない	-	パターン不足	-

表 2 静的検査結果のカバレッジが低く，単体テスト結果のカバレッジが高い場合
Table 2 The coverage of unit testing is low, the coverage of static analysis is high

		静的検査の仕様記述	単体テストのテストケース記述	ソフトウェア品質
静的検査の仕様記述	妥当	OK	パターン不足	低
	妥当でない	制約の範囲が大きすぎる	-	-
単体テストのテストケース記述	妥当	制約の範囲が大きすぎる	OK	高
	妥当でない	-	パターン不足	-

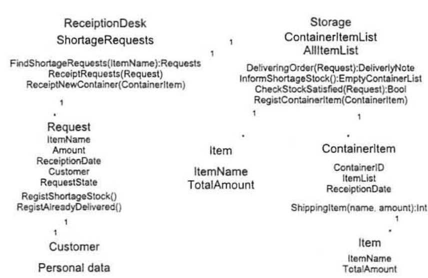

図 5 在庫管理プログラムのクラス構成[11)]
Fig. 5 Class constitution of warehouse management program

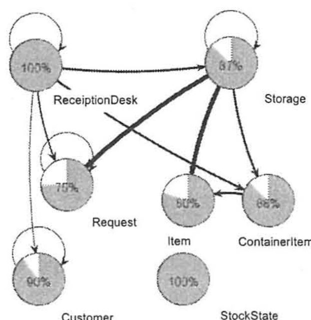

図 7 静的検査の実行結果
Fig. 7 Result of static checking

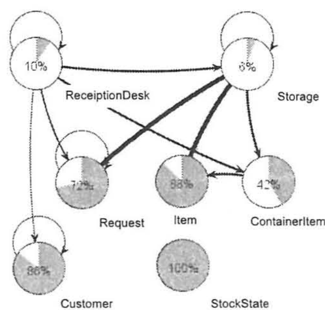

図 6 単体テストの実行結果
Fig. 6 Result of unit testing

5.3 結　　果

単体テストおよび静的検査に関してメインビューに出力されたグラフをそれぞれ図 6, 図 7 に示す[*].

5.4 考　　察

5.4.1 単体テスト

図 6 において，呼び出し関係のエッジが多いクラスほどより多くのクラスを呼び出している．従ってエッジの本数から必要なスタブの量を把握できる．

5.4.2 静 的 検 査

文献 11) と同様に，図 7 より全てのクラスが高いカバレッジを持っていることが確認できる．

静的検査結果と呼び出し関係について Receiption-Desk クラスを例として考える．このクラスはカバレッジが 100%であり問題がないと考えられるが，Request クラス (75%) を呼び出しており，もし Request クラスに問題があれば ReceiptionDesk クラスにも影響が出ることが考えられる．従って呼び出し元クラスの品質は呼び出し先クラスに影響を受けると考えられ，呼び出し関係を考慮したカバレッジの算出が課題となる．

5.4.3 単体テストと静的検査の比較

図 6, 図 7 から 5.1 節に基づいて考察する．

単体テストのカバレッジと静的検査のカバレッジが

[*] 見やすいようにクラス名を張り替えている．

共に高いクラス* に関しては仕様と記述が一致しており，ソフトウェア品質が高いと言える．

単体テストのカバレッジは低く，静的検査のカバレッジは高いクラス**を表 1 に適用すると，文献 11) より静的検査の仕様記述は妥当であることからテストケースのパターンが不足していると考えられ，実際にテストケースは不足している．対象ソフトウェアの品質は高いことが推定でき，実情と一致する．

6. おわりに

本研究ではソフトウェアの品質を 2 つの観点から可視化する手法を提案した．提案手法を実装したツールを試作し評価実験を行った．結果として，仕様と記述が一致しておりソフトウェアの品質は高いものの，テストケース記述が不足していることを推測し，実情と一致した．このことより，本手法によってより精緻な解析が可能であることが分かった．

今後の課題として，仕様記述品質の妥当性を評価する手法の考案，呼び出し関係を考慮したカバレッジの算出，単体テストや静的検査だけでなく他のメトリクスへ対応することが挙げられる．

謝辞 本研究の一部は科学研究費補助金基盤 C(21500036) と文部科学省「次世代 IT 基盤構築のための研究開発」(研究開発領域名：ソフトウェア構築状況の可視化技術の普及) の助成による．

参 考 文 献

1) 三宅達也, 肥後芳樹, 楠本真二, 井上克郎: 多言語対応メトリックス計測プラグイン開発基盤 MASU の開発, 電子情報通信学会論文誌 D, Vol.92, No.9, pp. 1518–1531 (2009).

2) Morisaki, S. and Matsumoto, K.: Toward Optimized Collection and Visualization of Software Metrics for Progress Sharing in Offshore Software Development Project, *In Proc. of the 2nd Workshop on Accountability and Traceability in Global Software Engineering (ATGSE2008)*, pp. 3–4 (2008).

3) Kleyn, M.F. and Gingrich, P.C.: GraphTrace—understanding object-oriented systems using concurrently animated views, *OOPSLA '88: Conference proceedings on Object-oriented programming systems, languages and applications*, New York, NY, USA, ACM, pp. 191–205 (1988).

4) Gonzalez, A., Theron, R., Telea, A. and Garcia, F. J.: Combined visualization of structural and metric information for software evolution analysis, *IWPSE-Evol '09: Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*, New York, NY, USA, ACM, pp. 25–30 (2009).

5) Lowe, W., Ericsson, M., Lundberg, J. and Panas, T.: Software comprehension - integrating program analysis and software visualization (2002).

6) ISO: Software Engineering-Product Quality-Part 1 : Quality Model, *ISO/IEC : 9126-1:2001* (2001).

7) Chalin, P.: Early detection of JML specification errors using ESC/Java2, *SAVCBS '06: Proceedings of the 2006 conference on Specification and verification of component-based systems*, pp. 25–32 (2006).

8) Yoonsik, C. and Ashaveena, P.: Specifying and checking method call sequences of Java programs, *Software Quality Journal*, Vol. 15, No. 1, pp. 7–25 (2007).

9) Burdy, L., Huisman, M. and Pavlova, M.: Preliminary Design of BML: A Behavioral Interface Specification Language for Java bytecode, *In Fundamental Approaches to Software Engineering (FASE 2007)*, pp. 215–229 (2007).

10) Csallner, C. and Smaragdakis, Y.: Check 'n' crash: combining static checking and testing, *ICSE '05: Proceedings of the 27th international conference on Software engineering*, New York, NY, USA, ACM, pp. 422–431 (2005).

11) 尾鷲方志, 岡野浩三, 楠本真二: 在庫管理プログラムの設計に対する JML 記述と ESC/Java2 を用いた検証の事例報告, 電子情報通信学会論文誌 D, Vol. 91, No. 11, pp. 2719–2720 (2008).

12) Gamma, E. and Beck, K.: JUnit: A Regression Testing Framework, *http://www.junit.org/*.

13) jcoverage ltd.: JCoverage, http://www.jcoverage.com/

14) 株式会社デジック : djUnit, http://works.dgic.co.jp/djwiki/.

15) Meyer, B.: *Object-oriented software construction (2nd ed.)*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1997).

16) Flabagan, C., Rustan, K., Leino, M., Lillibridge, M., Nelson, G., Saxe, J. B. and Stata, R.: Extended static checking for Java, *Proc. of the ACM SIGPLAN 2002*, pp. 234–245 (2002).

17) JO'Madadhain, Fisher, D., White, S. and Boey, Y.: The jung (java universal network/graph) framework, Technical report, UC Irvine (2003).

---

* Customer クラス，Request クラス，Item クラス，StockState クラスが該当する．

** ReceiptionDesk クラス，Storage クラス，ContainerItem クラスが該当する．