The University of Osaka
Institutional Knowledge Archive

Title	JMLによって記述された契約に対する品質評価手法の提案
Author(s)	武藤, 祐子; 岡野, 浩三; 楠本, 真二
Citation	平成23年度情報処理学会関西支部支部大会講演論文集. 2011, 2011
Version Type	VoR
URL	https://hdl.handle.net/11094/50252
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	

The University of Osaka Institutional Knowledge Archive : OUKA

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

The University of Osaka

B-03

JML によって記述された契約に対する品質評価手法の提案

A Proposal of Metrics for Quality of Contracts Described by JML

武藤　祐子 †　岡野　浩三 †　楠本　真二 †
Yuko Muto　Kozo Okano　Shinji Kusumoto

あらまし ソフトウェアの設計開発プロセスにおいて形式手法が注目されている．Design by Contract の考え方に基づき，Java に対して契約を付加する JML という言語がある．本研究では JML によって記述された契約に対する品質カバレッジを提案する．これにより，契約に漏れがないかどうかを判定でき，より正確にソフトウェア品質を測定できる．

Abstract. Formal approach has been attracted in software development. We focus on JML which is a behavioral interface specification language for Java based on Design by Contract. In this paper, we propose Variable Coverage, a set of metrics for quality of contracts described by JML. Software quality will be able to be measured more correctly using Variable Coverage because the metrics judge whether the contracts are enough or not.

１．　はじめに

Design by Contract（以下 DbC）[1] はソースコードや設計書に対して満たすべき契約を記述する手法である．Java に対しては JML (Java Modeling Language) [2] が，UML に対しては OCL (Object Constraint Language) [3] という契約記述のための言語が存在する．

Java において，JML によって書かれる契約の量や品質は，開発者によって差異がある [4].

契約とソースコードが一致しているとき，ソフトウェア品質が高いとする．表 1 に JML によって記述された契約の品質および検査結果とソフトウェア品質の関係を示す．契約の品質が高くなければ，検査の結果を信頼できないことが表 1 からわかる．

以下に DbC に基づいてソフトウェア品質の評価を行う場合の評価の観点を挙げる．

観点 1 契約はソースコードと矛盾しないか？

観点 2 チェックすべき項目に対して契約が存在するか？

観点 3 意味のある契約か（開発者の意図が反映されているか）？

観点 1 については，JMLrac[5] による実行時の検査や，ESC/Java2[6] による静的検査などが提案されている．契約の品質評価を行っている研究では，この観点のみを対象としていることが多い．

観点 2 について言及している研究は少ない．本研究では観点 2 について評価メトリクスを提案し，評価方法について述べる．これにより契約の書き忘れを検出することができる．

観点 3 については，ソースコードや設計書のみを用いる場合，開発者の意図を汲み取ることは難しいため，今回は対象としない．

表 1: 契約の品質および検査結果とソフトウェア品質

契約の品質	検査の結果	ソフトウェア品質
高	高	高
高	低	低
低	高	不明
低	低	低
不明	高	不明
不明	低	不明

２．　関連研究

2.1　契約の品質

文献 [4] では，Mutation Coverage という JML によって記述された契約の品質を評価するためのメトリクスを提案し，評価を行っている．結果から，契約の質は開発者間に差があることがわかり，契約の質を向上させるためのツールサポートが必要であると述べられている．この論文では，変異テストを契約の質の評価に用いている．ランダムにメソッドを呼ぶようなテストケースを生成し，テスト対象プログラムに対してテストを実行する．テスト対象プログラムから一部を変更した変異とよばれるプログラムを生成し，再度ランダムテストを実行する．するとテスト対象プログラムと変異によって検出されるアサーション違反に差が出るため，それを用いて契約の完全性について上界と下界を求め，Mutation Coverage を

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

計算する．

2.2 ソフトウェア品質の可視化

我々の研究チームでは，単体テストおよび静的検査に関する可視化手法 [7] を提案している．この研究では，ソフトウェアの品質を (1) テストケースの品質，(2) 静的検査に用いる契約記述の品質，(3) 単体テスト結果の品質，(4) 静的検査結果の品質の 4 つの観点から評価する．この手法では，単体テストの結果および静的検査の結果を同一の形式で比較するために，両者に対して通過率というメトリクスを適用し，クラスの呼び出し関係を用いて結果をグラフ構造で表現する．

３． JML

Java Modeling Language(JML)[2] とは，Java で記述されたソースコードに対して契約を付加することのできる言語である．アノテーションとしてコメントの形で書かれるため，ソースコード本体へは影響を及ぼさないという利点がある．

JML を用いて付加することのできる契約として事前条件，事後条件，代入可否，不変条件などがある．例として銀行口座クラスに対して JML を記述したソースコードを図 1 に示す．*balance* フィールド変数は口座の残高を表す．deposit(int) メソッドは口座への預け入れを，withdraw(int) メソッドは口座からの引き出しを行うメソッドである．

事前条件とは，メソッドが実行される前に満たされるべき条件であり，@requires 節に記述する．deposit(int) メソッドでは，預け入れる金額が 0 以上であるという事前条件が記述されている．この条件を満たすのはこのメソッドを呼び出す側の責任であるため，deposit(int) メソッド内では引数のチェックを行っていない．

事後条件とは，事前条件が満たされた状態で，メソッドが実行された後に満たされるべき条件であり，@ensures 節に記述する．戻り値は *\result* という語句で表される．deposit(int) メソッドでは，メソッド実行前の残高 *\old*(*balance*) と預け入れ金額 *amount* の和がメソッド実行後の残高 *balance* と等しいという事後条件が記述されている．

代入可否とは，変数に対して代入を行ってよいことを明示する契約であり，@assignable 節に記述する．deposit(int) メソッドでは，口座への入金後に *balance* フィールド変数の値を更新するため，@assignable 節に *balance* が記述されている．

不変条件とは，常に成立すべき条件であり，@invariant 節に記述する．*balance* フィールド変数に対して，必ず 0 以上であるという不変条件が記述されている．

```
public class BankAccount{

  //@invariant balance >= 0;
  private int balance;

  //@ensures balance == 0;
  //@assignable balance;
  public BankAccount() {
      balance = 0;
  }

  //@requires amount >= 0;
  //@ensures balance == amount;
  //@assignable balance;
  public BankAccount(int amount) {
    balance = amount;
  }

  //@requires amount >= 0;
  //@ensures balance == \old(balance) + amount;
  //@assignable balance;
  public void deposit(int amount) {
    balance = balance + amount;
  }

  //no contracts
  public void withdraw(int amount)
              throws InsufficientFundsException {
    if (balance < amount) {
      throw new InsufficientFundsException();
      return;
    }
    balance = balance - amount;
  }

  //@ensures \result == balance;
  public int getBalance() {
    return balance;
  }

}
```

図 1: 銀行口座クラス

４． 提案手法

契約の品質を評価することを目的として，変数カバレッジを提案する．この変数カバレッジは必要最低限の契約について評価を行い，契約の漏れ（書き忘れ）を確かめることができる．一般的に，契約が記述される対象は変数である．事前条件は，メソッド実行前に引数やフィールド変数が満たすべき条件を契約する．但しフィールド変数に関しては，メソッド内で使用されるものに限定される．事後条件は，メソッド実行後に戻り値やフィールド変数が満たすべき条件を契約する．但しフィールド変数は代入可能 (assignable) なものに限定される．クラス不変条件は，フィールド変数が常に条件を満たしているかどうかを検査する．従って，制約条件を持つべき変数のうち，実際に制約条件が記述されている割合をカバレッジと考えることができる．

4.1 変数カバレッジ

引数，戻り値，フィールド変数に関して，JML で記述された契約の品質を評価するカバレッジメトリクスを提案する．これらのカバレッジを複合的に扱うことで，契約の質を表現できる．戻り値カバレッジとフィールド変数カバレッジを用いることで事後条件の品質評価を行うことができる．

4.1.1 戻り値カバレッジ

戻り値カバレッジは，戻り値が存在するメソッドのうち，事後条件に登場する戻り値の割合である．なお，メソッド単位で計測した場合は必ず 0%または 100%になる．

4.1.2 引数カバレッジ

引数カバレッジは，あるメソッドにおいて，メソッドの引数のうち，事前条件に登場する割合である．

4.1.3 フィールド変数カバレッジ

フィールド変数カバレッジについては，事後条件のみを対象とする．その理由は，事前条件においてフィールド変数に制約を設けるべきかどうかは開発者の意図に依存するため，判定する方法がないためである．同様にクラス不変条件についても，不変条件を設けるべきかどうかはソースコードからは判定できないため除外する．フィールド変数カバレッジを，メソッドにおいて代入可否 (@assignable) が設定されているフィールド変数のうち，事後条件に出現する割合と定義する．

4.2 測定例

図 1 の銀行口座の例について，BankAccount クラスを単位とする変数カバレッジを計測する．

戻り値が void でないのは，getBalance メソッド 1 つである．このメソッドの事後条件には $\backslash result$ が登場している．従って BankAccount クラスの戻り値カバレッジは 1/1 となる．

引数が存在するメソッドは，BankAccount(int) コンストラクタ，deposit(int) メソッド，withdraw(int) メソッドの 3 つである．これらのうち，withdraw メソッドには契約が記述されておらず，事前条件に引数が登場しない．その他のメソッドについては事前条件に引数が登場している．従って BankAccount クラスの引数カバレッジは 2/3 となる．

代入可否が設定されているメソッドは，BankAccount() コンストラクタ，BankAcount(int) コンストラクタ，deposit(int) メソッド，withdraw(int) メソッドの 4 つである．そのうち前者 3 つには事後条件にフィールド変数 balance が記述されているが，withdraw(int) メソッドには事後条件が存在しない．従って BankAccount クラスのフィールド変数カバレッジは 3/4 となる．

５． 予備実験

変数カバレッジを測定した．計測対象は JML 公式サイト [2] の例題の一部（4 クラス）である．これらは Java ソースコードと JML によって書かれた契約を含み，契約の品質が高いと言える．JML を含む形で Javadoc を生成する JMLdoc の出力結果を目視で計測した．

5.1 変数カバレッジの測定結果

結果を表 2 に示す．-は計測対象メソッドが存在しなかったことを示す．左下の 5/6 は，IntegerSetAsTree クラスに対して，すべての可視性のメソッドを対象に戻り値カバレッジを測定した結果を表す．戻り値が void でないメソッドが 6 つ存在するうち，5 つに関して事後条件に戻り値が登場していたことを示す．

5.2 考察

戻り値カバレッジについては，public メソッドに限定すれば 100%であり，JML の品質メトリクスとして採用できると考えられる．

引数カバレッジに関しては，0%〜100%まで幅広い結果となった．事前条件が存在しない引数は int や boolean 等の primitive 型であった．例として，SLList は int 型であれば特に制限なしに格納することができる単方向リストであり，リストへの追加を行うメソッドでは引数に対して契約が不必要である．契約が不必要なのか，それとも必要であるが書き忘れているのかを判定するのは観点 3 の範囲であるため，対象としない．このような場合も考慮して，primitive 型の変数を除外して計測することが考えられる．

フィールド変数カバレッジに関しては，代入可否が記述されているメソッドが少なく測定対象不足であったため，計測対象を増やして評価を行うこととする．

６． 研究計画

6.1 公式サイトの例題について計測

JML 公式サイトの例題について変数カバレッジを計測する．測定条件を以下のように変化させ，カバレッジの計測結果を観測する．これらのうちカバレッジが高いものを変数カバレッジとして採用する．

表 2: 変数カバレッジの測定結果

	Return Value		Parameter		Field Variable	
Visibility	All	Public	All	Public	All	Public
BoundedStack	6/6	6/6	1/2	1/2	0/1	–
DLList	4/4	4/4	3/4	3/3	–	–
SLList	5/5	5/5	5/5	3/3	–	–
IntegerSetAsTree	5/6	5/5	0/6	0/3	0/1	–

1. 計測対象メソッドの可視性による違い
2. setter/getter を含める・除外する
3. primitive 形引数を含める・除外する

6.2 一般的な JML について計測

在庫管理プログラム [8]，住所録プログラムについて計測を行う．例題と比較することで，一般的に JML 記述時に起こしやすい問題やバグを発見する．

6.3 可視化手法への統合

我々の研究チームで提案している，単体テストおよび静的検査に関する可視化手法 [7] との統合を行う．この研究で対象としている 4 つの品質のうち，(2) 静的検査に用いる契約記述の品質 に対するメトリクスとして変数カバレッジを採用し，可視化手法に組み込み評価を行う．

7. おわりに

本研究では，DbC に基づき JML を用いて Java ソースコードに対して記述された契約に対する品質評価メトリクスを提案した．これにより，JML を用いてソフトウェア品質を評価する際に，より正確な結果の評価が実現する．

今後は研究計画に従って研究を進めていく予定である．

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

[2] JML, http://www.eecs.ucf.edu/~leavens/JML/.

[3] OCL, http://www.omg.org/spec/OCL/.

[4] Knauth, T., Fetzer, C. and Felber, P.: Assertion-Driven Development: Assessing the Quality of Contracts Using Meta-Mutations, *Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops*, Washington, DC, USA, IEEE Computer Society, pp. 182–191 (2009).

[5] Cheon, Y. and Leavens: A Runtime Assertion Checker for the Java Modeling Language, International Conference on Software Engineering Research and Practice (SERP ’02), pp. 322–328 (2002).

[6] Flabagan, C., Leino, K. R. 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).

[7] Muto, Y., Okano, K. and Kusumoto, S.: A Visualization Technique for the Passage Rates of Unit Testing and Static Checking with Caller-Callee Relationships, *In proc. of Parallel and Distributed Processing with Applications Workshops (ISPAW), 2011 Ninth IEEE International Symposium*, pp. 336–341 (online), DOI: 10.1109/ISPAW.2011.54 (2011).

[8] 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).