契約を用いたソフトウェア開発の修正傾向調査



岡野　浩三[†1]　　吉岡　一樹[†2]　　楠本　真二[†3]

ソフトウェアに対して Design by Contract に基づく契約記述を付加することによって，ソフトウェアの品質の向上が期待されている．しかし，契約記述がソフトウェアの品質に影響を実際に与えるか否かについて実証的な観点からの調査を行った研究は報告されていない．そこで本研究では，契約記述の存在する複数のオープンソフトウェアに対してリポジトリマイニングを行ない，ソフトウェアの品質への影響を調査した．具体的には，契約記述のあるファイルと無いファイルで，そのファイル中の不具合の修正傾向 (修正の頻度，タイミング) に違いがあるかどうかを調査した．その結果，正しい契約記述が開発の効率向上に結びつき，また，早期に契約記述をすることで，不具合の早期・短期修正の可能性があることが確認された．今後，開発過程で正しい契約記述がなされたかどうかを担保する方法を検討しなければならない．

Empirical Research on Design by Contract based Software Development

Kozo OKANO[†1], Kazuki YOSHIOKA[†2], and Shinji KUSUMOTO[†3]

A program based on Design by Contract is said to have high software quality. Any past study, however, does not report that such contract actually affects the quality of the software from the view point of empirical approach. In this research, we mine software repository for open software, and confirm whether contract affects software quality. In particular, we research whether there is differences of a tendency to fix defects between programs with contracts and without them. In the results, we find that programs with correct contracts improve development process and early contracts promote early and rapid bug-fixing. Consequently studies on verification methods for DbC written in real software in software development are needed.

1. はじめに

ソフトウェア開発が大規模化すれば，それだけ仕様の漏れやテストケースの不足などが発生しやすくなり，細かい不具合の検出が困難になる．Design by Contract (以下 DbC) [Meyer1992] の概念では，ある処理の呼び出し側と呼び出される側の責任を，契約として明確に分けることにより不具合が発生したときの修正箇所を明確にする．このような契約に対して，形式手法などを用いることで実装が契約を満たしているか否かを調べることができる．DbC に基づく仕様記述言語の例として，Java Modeling Language (以下 JML) [Leavens1999] や Spec# [Barnet2005] などが挙げられる．JML を用いて Java プログラムに対して仕様記述する

【脚注】
† 1　信州大学 工学部情報工学科
† 2　株式会社日立製作所 インフラシステム社モノづくり統括設計部
† 3　大阪大学 大学院情報科学研究科

ことにより，ESC/Java2[Flanagan2002] などを用いて仕様に対するプログラムの正しさを検証できる．ESC/Java2 の結果は正当性，完全性は保証されていないが，不具合検出などに活用できる．このような軽量的手法 (light-weighted approach)，すなわち，形式手法が使える場面で活用する方法を用いることにより，契約記述が付加されたプログラムの正しさを保証し，ソフトウェアの品質，特に，保守性，再利用性の向上を図ることができる．しかしながら，契約記述がソフトウェアの品質に実際に影響を与えるか否かについて実証的に調査をした研究は報告されていない．そこで，本研究では契約記述の存在するソフトウェアに対してリポジトリマイニングを行ない，ソフトウェアの品質に影響を与えているかについて実証的に調査する．この論文は [吉岡 2010] で報告した内容を整理し，記述追加を行ったものである．

リポジトリマイニングとはソフトウェアの特徴を調べる手法の一つであり，開発履歴情報の集合であるリポジトリに対してデータマイニングを行う．リポジトリマイニングを行う際のデータの収集対象としては，ソフトウェアの変更履歴や開発者情報などのプロセス情報，開発期間などの時間情報 [Kim2008] などがある．本研究では，契約記述の付加されたファイルとそうでないファイルで不具合の修正率や修正までにかかる時間で差があるかを調査した．その結果，契約を正しく書くことが契約記述による効率向上につながるであろうことが分かった．

以降，2，3，4 章で，それぞれ，本研究の背景，調査内容および実装，得られた結果について述べる．5 章で考察し，6 章では関連研究について言及する．最後に 7 章でまとめる．

2. 背景

2.1. Design by Contract

DbC は Bertland Meyer が提案した概念 [Meyer1992] であり，クラスとそのクラスを利用する側との間での仕様の取り決めを契約とみなすことにより，ソフトウェアの品質を向上させることを目指している．契約とは，クラスの利用側がそのクラスのメソッドを利用する際に，ある指定された事前条件を保証すれば，そのメソッド実行後は指定された事後条件を満たすことを保証するというものである．

いくつかのプログラム言語では，標準で DbC を提供している．一方で，オリジナルの言語定義とは独立に DbC をサポートする言語や処理系も存在する．例えば，Eiffel [Meyer1991] は標準で DbC を提供している．C# と Java は標準ではその機能はないが，それぞれ仕様記述用の言語が存在する．Spec# [Barnet2005] は C# に契約を記述する言語である．Spec# に対する静的解析のために制約記述の中間言語としての Boogie [1] や Boogie のための種々の処理系が挙げられる．Java においては，仕様記述用の言語としては JML がよく知られている．また Daikon[Ernst2007] は複数のプログラム言語を対象に契約の構成要素である表明を動的解析によって導出するツールであり，すでに作成されたプログラムコードの仕様保守などに有用視されている．

2.2. Java Modeling Language

JML[Cok2011]，[Leavens1999] は DbC に基づいて Java に契約を付加するための仕様記述言語である．契約はメソッド実行前に満たすべき事前条件，事前条件を満たした上でメソッド実行後に満たされるべき事後条件，オブジェクトが生存中に，各フィールド変数が常に満たすべき不変条件などがあり，JML ではこれらを Java の文法を拡張した言語で記述することができる．また，この記述に対して種々の解析を行えるツールがある．ここでは，契約の基本的な概念である事前条件，事後条件，不変条件について説明する．図 1 に説明例として用いる BankAccount クラスを示す．

@requires 句は JML において事前条件を示すために用いられる．図 1 において，withdraw メソッドと deposit メソッドはそれぞれ，9 行目，10 行目と 16 行目に事前条件が記述されている．@ensures 句は事後条件を示すために用いられる．コンストラクタと withdraw メソッド, deposit メソッド，getBalance メソッドは事後条件を持つ．図 1 の 4 行目はインスタンスを生成した直後はフィールド balance の値は 0 であることを事後条件として記している．

図 1 の 11 行目は事前条件を満たしている状況で withdraw メソッドが実行された後には事後条件として balance == \old (balance) – amount が成り立つことを記している．ここで \old() はメソッド実行前の状況でのフィールド変数の値を参照するのに用いられる．

JML において，不変条件を示すためには @invariant 句が

```
1 public class BankAccount{
2    private int balance ;
3    // @invariant balance >= 0;
4    // @ensures balance == 0;
5    // @assignable balance ;
6    public BankAccount ( ) {
7    this.balance = 0 ;
8    }
9    // @requires amount >= 0;
10   // @requires balance >= amount ;
11   // @ensures balance == \old (balance) - amount ;
12   // @assignable balance;
13   public void withdraw ( int amount ) {
14   this.balance  -= amount ;
15   }
16   // @requires amount >= 0;
17   // @ensures balance == \old (balance) + amount ;
18   // @assignable balance ;
19   public void deposit ( int amount ) {
20   this.balance += amount ;
21   }
22 }
```

図 1　JML が記述された BankAccount Class

用いられる. 図1の4行目は, フィールドbalanceはオブジェクトが生存する間は必ず0以上ということを示している.

その他，任意のフィールド変数aの宣言時に@non_nullが修飾された場合は，@invariant a != null という表現と同等の意味を持つ．これらの事前条件，事後条件，不変条件やプログラムの着目する1点で成立すべき条件などを併せて表明 (assertion) と呼ぶ.

以上で述べた各種の表明はESC/Java2[Cok2004] や，JML4c [Sarcar2010] などのツールを用いることで解析できる.

静的検査器の1つであるESC/Java2 [KindSoftware2013] はESC/Java[Flanagan2002] の後継ツールであり，対象プログラムとJML表明を述語論理式に変換し，充足可能性の判定を行う.

ESC/Java2 がプログラムの実行を伴わない静的な検査ツールなのに対し，JML4c は動的にプログラムを実際に動かして表明とプログラム動作との適合検査を行う.

2.3. リポジトリマイニング

ソフトウェアの開発履歴が蓄積されているソフトウェアリポジトリに対するデータマイニング (リポジトリマイニング) は，実際のソフトウェア開発における有用な知見の発見が期待され，近年の実証的ソフトウェア工学で多くの研究が行われている [小林 2010]．近年のソフトウェア工学の動向としても，実証的証拠を発見，収集することが重視されている [MacDonell2010].

3. 調査手法

3.1. 動機

契約記述はソフトウェアの品質を高めていると一般的に考えられているが，実際の開発に与える影響に関して調査した研究は存在しない．我々の研究グループの既存研究 [武藤 2011] において，実装中に出現した変数のうち，契約記述に出現した変数の割合を計測している．この手法では契約記述の網羅性を測ることはできるが開発においてどれだけ必要な記述であるかまでは判定することはできない．そこで，本研究では実際にソフトウェアリポジトリを用いて，開発履歴情報がソフトウェアリポジトリに蓄積されているソフトウェアを対象に調査を行う．ソフトウェアリポジトリには開発の変更履歴の情報が残っているために，開発プロセスにおいて契約記述がどのように変更されているかを調査することができる．JML が付加されたソフトウェアにおいてどのような変更が行われているかを調査し，実際の開発における JML の影響に関する知見を得る.

3.2. 調査内容

本研究では，3つの調査を行った．1つ目は契約記述が付加されたソフトウェアの修正傾向の調査，2つ目はJML記述を付加されたプログラムと付加されていないプログラムにおける変更率の比較調査である．3つ目はJML記述を付加されたプログラムが付加されていないプログラムと比較して不具合が早期修正される傾向にあるかという調査である．これらにより，JML記述の付加されたプログラム自体の修正傾向 (修正の頻度，タイミング) と，JML記述の付加されていないプログラムとの比較による，JML記述が付加されることによる開発の違いの調査を行う.

3.3. リサーチクェッション

本研究における各調査の求めるリサーチクェッション（RQ）は以下の通りである.

調査1のRQ: 契約記述が付加されたソフトウェアの修正内容は特徴付けできるか？

調査2のRQ: 契約記述が付加されることにより，不具合の修正回数が減るか？

調査3のRQ: 契約記述が付加されることにより実際に不具合が早期に修正される傾向にあるか？

3.3.1. 調査1：契約記述が付加されたソフトウェアの修正内容の調査

調査対象のリポジトリからdiff情報を取得し，変更前，変更後，それぞれのソースコードを目視で検査した.

3.3.2. 調査2：契約記述を含むコードとそうでないコードの変更率の調査

契約記述が付加されたソースコードのあるソフトウェアにおいても，契約記述を含まないソースコードは大量に存在する．そこで，契約記述の付加されたソースコードと，契約記述の付加されていないソースコードにおける修正率及び修正影響量の比較を行う．具体的には，それぞれの修正された

- メソッド数
- メソッド行数

を調査する．これらの修正された値を，最新バージョンにおける

- 総メソッド数
- 総メソッド行数

によって正規化を行う．このようにして，不具合に関わるコミットに関しての修正率及び修正影響量の比較による調査を行う.

本研究における不具合修正の定義について述べる．ソースコードに対して加えた修正が，不具合修正であるのか機能追加であるのかなどは，厳密には開発者以外判断できない．不具合修正に関する既存研究においては，通常ソフトウェアリポジトリと関連付けられた，バグ管理システムの情報を用いて不具合の混入したリビジョンを特定している [Sliweski2005] [Wu2011].

これらは，ソフトウェアリポジトリのコミットログや編

集履歴において，バグ管理システムのバグ管理チケットと関連のあるものを取得することで，バグの混入した時期を特定している．しかしながら，我々の研究グループが調査した限りにおいて，契約記述の付加されたソフトウェアリポジトリと関連付けられ，かつ公開されているバグ管理システムは存在しなかった．従って，上記のツールやアルゴリズムを用いることはできない．

そこで，各リビジョンにおいて不具合に関する単語 (bug, fix, problem, issue, error) を含むコミットログを持ち，明らかに不具合修正ではないものを手作業で取り除いた結果を不具合修正の行われたコミットとする．一般的なソフトェアにおいては，不具合修正を行った場合，上記の単語を含むコミットログを残す．この方法は，不具合修正が行われたかどうかを確かめる際に不具合修正が行われたコミットをおおよそに特定するために用いられる [Li2012].

今回調査する対象のソフトウェアには公開されたバグ管理システムが存在しないため，この方法を用いて不具合に関するコミットを特定する．

3.3.3. 調査 3：契約記述を含むコードはそうでないコードと比較して不具合が早期修正される傾向にあるかの調査

一般的に，契約記述が付加されたソフトウェアにおいては不具合が早期に修正される傾向にあると考えられている．本調査では，契約記述の付加されたファイルがそうでないファイルと比較して不具合が早期に修正されているかを調査する．具体的には，不具合が修正されたメソッドを特定し，そのファイルがいつ生成されたのかを特定する．また，不具合が修正されたメソッドの存在するファイルにおいて，最後に修正された日時を取得する．これらの日時情報から，不具合を修正するのにかかった時間を特定する．通常の調査においてはバグ管理情報から不具合 (バグ) の混入した日時を特定するが，今回の調査対象ではバグ管理システムとの連携を行なっていなかったため，不具合が修正されたメソッドが存在するファイルにおいて最後に修正された日時を不具合混入日時と仮定している．このためこの調査は「早期の修正」と「短期の修正」の 2 つの意味を併せて行ってしまっているが，本稿ではこの調査を「早期の修正」の調査と見なす．そして，契約記述が付加されている場合とそうでない場合で，どちらの方が早期に修正される傾向があるのか，ウィルコックス検定を行う．

3.4. 調査手法の詳細

前述した調査手順の実装の詳細について述べる．本調査では，svn の解析のために，svnkit を用いた．svnkit は Java から svn を操作するためのオープンソースソフトウェアである．また，Java ソースコード及び JML 記述を解析するために Java Development Tool (以下 JDT) を用いた．JDT は Eclipse プロジェクトのサブプロジェクトとして開発された Java ソースコードの静的解析ツールである．

調査 2，調査 3 共通の前処理は以下の通りである．

ステップ 1 svnkit を用いてリポジトリにアクセスし，全 diff 情報をファイルに出力する．

ステップ 2 取得した unified diff 情報から，変更のあったリビジョンおよびその前のリビジョンにおけるファイルをローカルマシンにエクスポートする．必要なファイルをすべて先にローカルに取得しておくことで，後の工程の作業を効率化する．

ステップ 3 取得したソースコード中のメソッドが JML 記述を含むかどうかを判定する．まず，ソースコード中のコメントノードを取得し，そのコメントが JML 記述を含むかどうかを判定する．

そのコメントがラインコメントであれば，//@ で始まり，直後に Override などの Java 言語自体のアノテーションが続かなければ JML であると判定する．また，ブロックコメントも同様に /*@ で始まり，直後に Override などの Java 言語自体のアノテーションが続かなければ JML であると判定する．その後，コメントがどのメソッドに対するものなのかを判定する．そのコメントがラインコメントであれば，その行を含むメソッドのコメントとする．そのコメントがブロックコメントであれば，コメントの開始行と終了行を計算し，終了行の直後から始まる実装を確認する．もしも直後にある実装がフィールド変数であればクラスのコメントとし，メソッドやコンストラクタであった場合にはそのメソッドやコンストラクタのコメントとする．

ステップ 4 unifeied diff 情報から適切に差分のあるメソッドを取得するために，diff の開始行と終了行に含まれているメソッドを計算する．このために，まず JDT を用いてソースコード中の全メソッドのノードを取得する．取得したノードからメソッドの開始行と終了行を計算し, diff の開始行と終了行と照らしあわせて, そのメソッドが diff に含まれているかを確認する．これをコメントにおいても同様に確認し，変更されたのが JML 記述なのか，実装なのか，あるいは両方なのかを確認する．

ステップ 5 **ステップ 3** において差分があると判定されたメソッドにおいて，diff となっているのが JML 記述なのか実装なのかを判定する．

調査 2 固有の手順は以下の通りである．

ステップ 1 svnkit を用い，コミットログを取得する．

ステップ 2 不具合に関すると思われるコミットを取得する．**ステップ 1** において取得したコミットログを単語に分割して次の単語群に含まれる単語があるかを判定する．

- bug
- fix
- error

・problem
・issue

なお，このとき先頭の大文字や複数形，活用等の語形の変化したものも検出している．

ステップ 3 ローカルにファイルをエクスポートして，そのファイルが JML 記述を含むかどうかを判定する．JDT を用いて構文解析を行う．このステップにおける比較は，ファイル単位，メソッド単位，行単位で行う．

調査 3 固有の手順は以下の通りである．

ステップ 1 全コミットにおいて，変更パスのうち svn のバージョン管理下に加えられたものを取得し，それぞれのコミットされた日時との対応を記録する．

ステップ 2 同様に不具合修正に関するコミットを取得し，修正されたファイル，メソッドも同様に取得する．このとき修正されたファイル，メソッド毎に不具合修正のコミットがなされた日時の対応を記録する．

ステップ 3 **ステップ 2** において修正されたファイルにおいて，そのファイルが作成された時間を**ステップ 1** において取得したファイルと日時との対応情報から取得する．

ステップ 4 **ステップ 3** までの操作によって，ファイル毎，メソッド毎の実装から不具合修正までの時間が取得できるので，これを JML の付加されたファイル，付加されていないファイル，JML の付加されたメソッド，付加されていないメソッド，の 4 つに分類し，修正にかかる時間の情報を作成する．

ステップ 5 ファイル単位，メソッド単位それぞれにおいて JML の付加されたものと付加されていないものでどちらの方が早期に修正される傾向があるのかについて検定を行う．ファイル単位の修正に関しては，まず全リビジョン毎にリビジョンと不具合情報を対応つける．次に，不具合修正のあるリビジョンにおいて，log コマンドで変更パス情報を取得する．取得した不具合修正の行われたリビジョンの変更パスを用いて，対応情報から最新更新リビジョンを取得する変更されたファイルパスを取得し，ファイルパスと最新更新リビジョンを結びつける．

検定にはウィルコックス検定を用いて R で行った．

4. 結果

4.1. 調査対象

バージョン管理システムには CVS，Subversion，git などがあるが，今回の調査では Subversion で管理されたソフトウェアを対象とした．対象としたのは以下のソフトウェアである．

・Community Z Tools（以下 CZT）
・Weka3
・JavaFE

CZT は，Z 言語のための，編集，タイプチェック，アニメーションを行う開発ツールである．

Weka3 は，Java コードに対する機械学習アルゴリズムを用いたデータマイニングツールである．JavaFE は Java1.4 および，Java1.5 のパーサであり，ESC/Java2 や RCC のフロントエンドコンパイラとして用いられている．

JML の付加されたファイル数，JML の付加されたファイルに関するコミット数，ソフトウェアサイズなどのデータを表 1 に示す．

表 1　調査対象

	# files with JML	# files w/o JML	# revisions	LOC*
Weka3	10	1259	7342	282285
CZT	40	116	8248	157653
JavaFE	106	223	188	69500

*LOC of the latest revision

4.2. 調査 1 の結果

結果を表 2 にまとめる．add, delete, modify はそれぞれ追加，削除，修正を意味する．

Weka3 において特徴的なのは，実装が修正されておらず，かつ JML 記述が修正されているメソッドが他のソフトウェアの結果と比較すると非常に多いという点である．また，61 メソッドにおいて追記がなされているが，これらは全て後から追記したものであり同様の修正であった．これらの追記は 16 リビジョンという短い期間でおこなわれており，実際のコミットした日時を見ても 11 日間で行われている．Weka3 において発見された特徴的な修正の例をあげる．

変更前のプログラムにおいてあるメソッドの事前条件は m_Dataset != null; となっていた．しかしながら，変更後のプログラムにおいては事前条件が m_Dataset == null; に変更されていた．これは，プログラム中において m_Dataset != null; であるときには例外をなげているため変更前の事前条件では矛盾が発生するためである．従って，事前条件の式を m_Dataset == null; に変更することで問題を解決していた．

表 2　記述修正量

		実装，JML 記述共に修正	実装無修正 JML 記述修正	JML 記述無修正実装修正
Weka3	add	176	61	13
	delete	1	5	2
	modify	27	4	29
	total	204	70	44
CZT	add	6	0	22
	delete	33	0	47
	modify	11	0	16
	total	50	0	85
JavaFE	add	134	32	11
	delete	9	5	15
	modify	74	8	91
	total	217	45	117

また，このプログラムでは例外を発生させているために //@signals (RuntimeException)　m_Dataset != null という契約を記述することが可能である．これにより，m_Dataset != null であるときに RuntimeException が発生することを明示的に示すことができる．

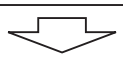
誤契約の修正の別の例を挙げる．

変更前のプログラムにおいてはある変数 index に対して代入可能であることを示す @assignable 節が記述されていた．これにより変数 index への代入が可能であるということが契約として指定されている．しかしながら，このメソッドの実行中に index という変数に代入を行われることはない．すなわち，厳密には誤契約というわけではないが意味のない契約となってしまっている．よって変更後のプログラムにおいては，@assignable 節を削除している．このソフトウェアにおいては，類似した変更が存在した． 変更後のメソッドにおいてはこのような事前条件の削除が同一コミットで３つのメソッドにおいて同様に行われていた．ある時期において，開発者が不要な契約であると発見，認識し，同時修正したものであると思われる．

次に，JavaFE において発見された特徴的な修正の例を示す．

JavaFE においては，JML 記述のリファクタリングが行われている．その例を図２に示す．図２の変更前のプログラムにおいては，事前条件および事後条件がそれぞれ @requires 節，@ensures 節で記述されている．これは JML 記述における最も基本的な契約の記述例である．しかしながら，変更後のプログラムにおいては変更前のプログラムに記述された契約と同等の契約が /*@non_null*/ 節によって記述されている．変更前および変更後のプログラムにおいて記述されている契約の意味的差異は存在しない．しかし，変更後のプログラムにおける契約記述の方がより契約に関連する実装部分との距離が近く，厳密には直前に記述されるようになっている．

```
//@ requires signature != null;
//@ ensures \nonnullelements(\result);
private FormalParaDecl[] makeFormals(MethodSignature
signature) {
    int length = signature.countParameters();
    FormalParaDecl[] formals = new FormalParaDecl[length];
            :
            :
return formals;
```



```
private/*@non_null*/FormalParaDecl[]
makeFormals(/*@non_null*/MethodSignature signature) {
    int length = signature.countParameters();
    FormalParaDecl[] formals = new FormalParaDecl[length];
            :
            :
    return names;
}
```

図２　リファクタリングによる契約記述変更例

この変更は，引数が満たすべき事前条件や，戻り値が満たすべき事後条件が，コードを見る開発者にとってよりわかりやすくなることを意図してリファクタリングを行ったものであると考えられる．

最後に CZT について簡単に述べる．

CZT においては表２から分かるようには実装が修正されずに JML 記述が修正されたメソッドの数が０である．すなわち，JML 記述が修正される場合には常に実装も修正されるということあり，仕様を決めてから実装を行っていること，修正の際も仕様と実装の一貫性を維持していることを伺わせる．すなわち契約を記述する際に，正しさに十分に注意を払って記述していると考えられる．

4.3. 調査２の結果

JML 記述の付加されたソフトウェアにおける，実際に JML 記述が付加されたファイルとそうでないファイルにおける変更率の比較結果を示す．本調査における変更単位にはメソッドを用いている．この結果を表３に示す．

不具合修正の発生率，影響量を以下で定義する．

不具合修正の発生率＝
　不具合修正に関わるメソッド数 /
　　最新バージョンのメソッドの総数

不具合修正の影響量＝
　不具合修正に関わるメソッドの総行数 /
　　最新バージョンのメソッドの総行数

Weka3 においては JML 記述を含むほうが，修正頻度，正影響量ともに値が小さい．すなわち JML 記述が付加された効果があるといえる．一方，CZT と JavaFE において逆の結果が出ている．すなわち，JML 記述による優位性はこの２つのソフトウェアでは見られなかった．

4.4. 調査３の結果

表４の中で示される平均値および中央値の値はファイルが作成されてから不具合修正が行われるまでの時間で，単位は時間（hour）となっている．また，それぞれの平均値，中央値，ウィルコックスンの順位和検定によって導かれる p 値も表４に附す．順位和検定を用いたのはウィルコックスン検定の値だけでは有意水準しか示されず，比較した際の値の大小がわからないためである．p 値が 0.05 より小さい場合，分布に差があると判断する．

ファイル単位で見たときには，僅かながら JML に有意な傾向があるが，より細かい粒度であるメソッド単位で見たときには，２つのソフトウェアにおいて，統計的に JML 記述が付加されていない方が不具合の発生から修正までの期間が短いといえる．

表 3　最後に修正されてから不具合修正にかかる時間の調査結果

調査対象		修正全体における		不具合修正に関わる			
		メソッド数	総行数	メソッド数	総行数	不具合発生率	不具合影響量
Weka3	with JML	312	4316	20	336	0.115	0.088
	w/o JML	20415	701110	3466	122142	0.175	0.316
CZT	with JML	461	5870	78	1036	5.423	0.918
	w/o JML	1723	29806	407	6827	1.702	0.539
JavaFE	with JML	2598	81841	1070	29165	0.996	1.393
	w/o JML	822	11846	175	3289	0.135	0.194

表 4　不具合の発生率と影響量

調査対象			平均値（時間）	中央値（時間）	p 値
Weka3	file	with JML	4892.36	3378	0.2476
		w/o JML	5510.42	1675	
	method	with JML	13196.88	15598	$2.542e^{-6}$
		w/o JML	13106.33	6373	
CZT	file	with JML	2574.14	433	0.3867
		w/o JML	2766.07	652	
	method	with JML	5647.06	2991	0.8764
		w/o JML	5616.15	3182	
JavaFE	file	with JML	3338.15	2138	0.0111
		w/o JML	5778.33	3647	
	method	with JML	7089.99	3647	$7.516e^{-2}$
		w/o JML	3887.10	3647	

残り 1 つのソフトウェアである CZT に関しては統計的に有意差が出ないだけでなく，p 値が 0.87 と非常に近しい傾向にあるという結果になった．

5.　考察

本章において考察を述べる．

5.1. 調査 1, 2 の考察

調査 2 の 3 つの対象のうち，Weka3 は JML 記述の付加されたメソッドの方が不具合の発生率，影響量が小さかった．すなわち，Weka3 は JML による効果があったといえる．実際にこのソフトウェアにおいて JML の付加されたファイルを調べたところ，その全てが /core 以下にあるファイルであった．これらのファイル群はソフトウェアの根幹部分を成す機能を担っており，他のファイルと比較すると重要度が高く，変更も容易には行われなかったのではないかと考えられる．また，重要度の高いファイルだからこそ，JML 記述を付加することで信頼度を高めようとしたと思われる．調査 1 の結果から併せて考察すると契約をきちんと早期にチェックした Weka3 においては契約記述の効果があり，そうでない他の 2 つのプロジェクトは開発時間を多くとったと考えることができる．

5.2. 調査 3 の考察

今回の調査では，不具合の修正にかかる時間を，最後に修正を行った時間から不具合修正を行った時間と定義している．この方法では，開発自体が特定の期間に偏っていると，その期間における不具合修正にかかる時間が短くなる．Weka3 においては，開発期間が早い時期に偏っていたために不具合修正にかかる時間に影響を与えたと考えられる．

なお，メソッド単位よりはファイル単位でみたときに DbC についてやや肯定的な結果がでていることについては，1 つの仮説としてファイル，すなわち，クラスの単位で DbC を考えるのが適切である，ということが考えられる．この考えはそもそも DbC がオブジェクト指向プログラミングを対象に考案されたことに付合する．ただしこの仮説の妥当性についての調査は今後の課題である．

5.3. 妥当性の脅威

今回の調査における妥当性への脅威として，実験対象が小規模かつ少数であることや，実際の不具合修正情報を用いていないことが挙げられる．特に，実験対象の一つである Weka3 などは JML 記述の付加されたファイルが 10 しかないのに対して JML が付加されていない通常の Java ファイルが 1259 あるなど，結果へのゆらぎが大きい．実験対象を今回の調査よりも大規模かつ大量にすることで今回の調査よりも信頼性の高い結果が得られるものと考えられる．また，今回の調査では，実際の不具合情報を用いずにキーワードによる判定をし，不具合の発生した時期も最後に修正をした時期と見なしている．一般的なリポジトリマイニングの研究においては，バグチケットが fix された時間を不具合の修正された時間とし，不具合の発生した時期に関しては推定用のアルゴリズムを利用して精度を上げている．もしも，契約記述の付加されたソフトウェアリポジトリにおいて，対応するバグ管理システムが存在すればそれを利用することにより，異なる調査結果が得られる可能性がある．その他，各種表明の効

果の比較をするためのデータがまだまだ不十分だと考えられる．今後多くのデータの蓄積が望まれる．

6. 関連研究

武藤らは文献 [武藤 2011] において，形式仕様記述のメトリクスである変数カバレッジを提案した．このメトリクスは実装中に出現した変数のうち，形式仕様記述にも出現する変数の割合を表したものである．変数カバレッジはメトリクスを利用することで形式仕様が十分に記述されていないメソッドやフィールド変数を明示し，開発者に情報を仕様として記述すべきものを提示することを目的としている．

また，不具合の修正に関して本論文では最終修正日時を不具合の混入時期と見なしているが，既存研究においては，SZZ アルゴリズム [Sliweski2005] や混入時期特定ツール [Wu2011] などが提案されている．これらのツールは，ソフトウェアリポジトリと連携しているバグ管理システムの情報を利用することで，不具合の混入時期を推定している．契約を用いた開発手法が，文献 [Leitner2007] で提案されている．この手法では，実行時に契約を満たさない実行であった場合，その実行履歴をトレースしテストケースを生成する．すなわち，契約を記述することでテストケースを記述するコストが軽減される．文献 [Knauth2009] では，mutation の概念を利用することで契約記述の有効性を検証している．テストケースとテストケースの mutant を実行し，双方の振舞いが異なるとき，detectable であるとする．この detectable な mutant のうち契約となる表明の発生数を調べる．表明の発生数が多いほど契約が網羅されている（完全性）に近いと見なしている．

7. おわりに

本研究では，契約記述の付加されたソフトウェアに対して，リポジトリマイニングを利用した 3 種類の調査を行った．調査 1 では契約記述が付加されたソフトウェアの修正内容の調査を行った．調査 2 では，契約記述の付加されたソフトウェアにおいて契約の付加されたファイル，メソッドが付加されていないものと比較して，修正率，修正に関わる量の差があるかを調査した．調査 1，2 より，正しい契約記述が開発の効率向上に結びつくということが分かった．

調査 3 では，契約記述の付加されたソフトウェアにおいて契約の付加されたファイル，メソッドの不具合が早期に修正される傾向にあるのかを調査した．結果として，Weka3 には早期あるいは短期の修正の傾向があり，契約を早期に記述する効果があったことが分かった．

また，いくつかのプロジェクトでは DbC の改変に関する特徴的な変更が観測できた．

今後の課題として，修正が行われた部分と契約との関連の調査などが挙げられる．

謝辞

本研究に関して，実験対象のツールの調査にご協力頂いた肥後芳樹，井垣宏両氏に深く感謝致します．

本研究の一部は科学研究費補助金基盤 C（21500036）の助成による．

【参考文献】

[Barnet2005] M. Barnett, K. Rustan M. Leino, W. Schulte, G. Barthe, L. Burdy, M. Huisman, Jean-Louis Lanet, and T. Muntean: The Spec# Programming System: An Overview, Vol. 3362, LNCS, pp.49-69, 2005.

[Ernst2007] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao: The Daikon system for dynamic detection of likely invariants, Science of Computer Programming, Vol.69, No. 1-3, pp.35-45, 2007.

[Cok2011] D. R. Cok: OpenJML: JML for Java 7 by extending Open JDK, In Proceeding NFM'11 Proceedings of the Third international conference on NASA Formal methods, pp.472-479, 2011.

[Cok2004] D. R. Cok and J. R. Kiniry: ESC/Java2: Uniting ESC/Java and JML, In International Workshop on Construction and Analysis of Safe Secure and Interoperable Smart Devices CASSIS 2004, Vol.3362, LNCS, pp.108-128, 2004.

[Flanagan2002] C. Flanagan, K. Rustan M. Leino, M. Lillibridge, G. Nelson, James B. Saxe, and Raymie. Stata: Extended Static Checking for Java, ACM SIGPLAN Notices, Vol.37, No.5, p.234, 2002.

[Kim2008] S. Kim, E.J. Whitehead, and Y. Zhang: Classifying software changes: Clean or buggy? IEEE Transactions on Software Engineering, Vol.34, No.2, pp.181-196, 2008.

[KindSoftware2013] KindSoftware: ESC/Java2, http://kindsoftware.com/products/opensource/ESCJava2/.

[Knauth2009] T. Knauth, C. Fetzer, and P. Felber: Assertion-driven development: Assessing the quality of contracts using meta-mutations, In Software Testing, Verification and Validation Workshops 2009, pp.182-191, 2009.

[Leavens1999] G. T. Leavens, A. L. Baker, and C. Ruby: JML: A Notation for Detailed Design, Behavioral Specifications of Businesses and Systems, pp.175-188, 1999.

[Leitner2007] A. Leitner, I. Ciupa, M. Oriol, B. Meyer, and A. Fiva: Contract driven development = test driven development - writing test cases, In Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The Foundations of Software Engineering, ESEC-FSE '07, pp.425-434, 2007.

[Li2012] J. Li and M. D. Ernst: Cbcd: cloned buggy code detector, In Proceedings of the 2012 International Conference on Software Engineering, pp.310-320, 2012.

[MacDonell2010] S. MacDonell, M. Shepperd, B. Kitchenham, and E. Mendes: How reliable are systematic reviews in empirical software engineering? IEEE Transactions on Software Engineering, Vol.36, No.5, pp.676-687, 2010.

[Meyer1991] B. Meyer: Eiffel: The Language, (Prentice Hall Object-Oriented Series). Prentice Hall, 1991.

[Meyer1992] B. Meyer: Applying 'Design by Contract,' IEEE Computer, Vol.25, No.10, pp.40-51, 1992.

[Sarcar2010] A. Sarcar: A New Eclipse-Based JML Compiler Built Using AST Merging, In 2010 Second World Congress on Software Engineering, pp.287-292, 2010.

[Sliweski2005] J. Sliwerski, T. Zimmermann, and A. Zeller: When do changes induce fixes? In Proceedings of the 2005 international workshop on Mining software repositories, pp.1-5, 2005.

[Wu2011] R. Wu, H. Zhang, S. Kim, and S. Cheung: Relink: recovering links between bugs and changes, In Proceedings of the 19th ACM SIGSOFT symposium and the 13th European Conference on Foundations of Software Engineering, ESEC/FSE '11, pp.15-25, 2011.

[武藤 2011] 武藤祐子 , 岡野浩三 , 楠本真二 : 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 , 情報処理学会関西支部大会 , B-03, 2011.

[小林 2010] 小林隆志 , 林晋平 : データマイニング技術を応用したソフトウェア構築・保守支援の研究動向 , コンピュータソフトウェア , Vol. 27, No.3, pp.13-23, 2010.

[吉岡 2010] 吉岡一樹 , 岡野浩三 , 楠本真二 : 契約記述の変更傾向の開発履歴情報を用いた調査 , 電子情報通信学会技術研究報告 , Vol.112, No.457, pp.121-126, 2013.