クラスサイズメトリクスを用いたソフトウェア変更量予測に関する考察

阿萬　裕久† 　望月　尚美†† 　山田　宏之† 　野田松太郎†

† 愛媛大学工学部情報工学科　〒 790–8577 松山市文京町 3
†† 愛媛大学大学院理工学研究科情報工学専攻　〒 790–8577 松山市文京町 3
E-mail: †{aman,yamada,noda}@cs.ehime-u.ac.jp, ††mochi@hpc.cs.ehime-u.ac.jp

あらまし　オブジェクト指向ソフトウェア開発において，一般にクラスサイズが大きくなれば変更・発展のための作業も複雑になり，その作業量やテスト工数も多くなると考えられる．そのためクラスサイズはある程度以下に押さえておく方が望ましい．そのような適切なクラスサイズについて検討するため，本論文ではソフトウェアのバージョンアップに伴う変更量に着目し，クラスサイズを用いて “後に多大な変更が施されることになるクラス” と “そうでないクラス” とを予測判別する方法を提案している．適用例として約 15000 個の Java プログラムを用いた測定実験とその結果分析を行っている．

キーワード　オブジェクト指向，メトリクス，クラスサイズ，ソフトウェア保守，コスト予測

An Estimation of Software Modification Cost Using Class Size Metric

Hirohisa AMAN†, Naomi MOCHIDUKI††, Hiroyuki YAMADA†, and Matu-Tarow NODA†

† Department of Computer Science, Faculty of Engineering, Ehime University　Bunkyo-cho 3, Matsuyama-shi, Ehime, 790–8577 Japan
†† Advanced Course of Computer Science, Graduate School of Science and Engineering, Ehime University　Bunkyo-cho 3, Matsuyama-shi, Ehime, 790–8577 Japan
E-mail: †{aman,yamada,noda}@cs.ehime-u.ac.jp, ††mochi@hpc.cs.ehime-u.ac.jp

Abstract　In general, the larger object-class will need the more cost for the maintenance. So class would have to be construct in a middle or small size. In order to discuss such desirable class size, this paper proposes a method for detecting costly classes using class size metric, where a costly class would need many modifications in the source code through the version-up. Then this paper applies the proposed method into about 15000 Java programs, and discusses the result.

Key words　object-orientation, metrics, class size, software maintenance, cost estimation

1. まえがき

近年，ソフトウェアの大規模化・高機能化に伴い，ソフトウェアの開発及び保守に費やされるコストは増大してきている．それゆえ，開発性・保守性・再利用性等に優れた高品質なソフトウェアの設計・開発が望まれており，そのための技術としてオブジェクト指向技術が広く用いられている．しかしながら，実際のソフトウェア品質は技術者の力量や経験に大きく依存しており，必ずしもオブジェクト指向の利点が活かされているとはいえない．そのため，ソフトウェア品質を客観的かつ定量的に評価・予測することが重要である．ソフトウェア品質を評価・予測するための定量的尺度をソフトウェアメトリクスあるいは単にメトリクスという [1]．これまで，メトリクスを用いたソフトウェア品質予測に関する研究がいくつか報告されている．例えば，プログラム行数 Line Of Code (*LOC*) [2], [3] を用いた信頼性の予測 [4] や Halstead のソフトウェア科学理論 (programming effort 式) [5] による障害件数の予測 [6] 等が報告されている．オブジェクト指向ソフトウェアに対しても，障害件数が少ないような最適なクラスサイズについて検討されている [7]．クラスサイズは，オブジェクト指向ソフトウェア開発において最も基本的かつ直感的なメトリクスであり，利用も比較的容易である．したがって，これを用いて有用な知見が得られれば，ソフトウェア開発者に対して平易かつ有用な開発基準を提供できる．本論文では，オープンソースソフトウェアの開発にも適用可能な開発基準の提供を目指し，バージョンアップに伴うソースコードの変更量に着目する．ソフトウェアの変更量はバージョンアップテストの工数にも関係する重要な要素と考えられる．そこで，“後に多大な変更が施されることになる

```
*** HelloWorld.java     2003-02-20 15:12:38.000000000 +0900
--- HelloWorld.java     2003-02-20 15:15:39.000000000 +0900
***************
*** 1,8 ****
 public class HelloWorld {
     public static void main(String[] args){
!       for ( int i = 0 ; i < 10 ; i++ ){
            System.out.println("Hello World!!");
-           System.out.println("Hello");
        }
     }
 }
--- 1,9 ----
 public class HelloWorld {
     public static void main(String[] args){
!       for ( int i = 0 ; i < 5 ; i++ ){
!           System.out.print("[" + i + "] ");
            System.out.println("Hello World!!");
        }
+       System.out.println("END");
     }
}
```

図 1　diff コマンドを用いて得られた出力ファイルの例

クラス” と “そうでないクラス” とをクラスサイズメトリクスによって（予測）判別するための閾値とその決定法を提案する．

以下，**2** 節では，ソフトウェアの変更量及びクラスサイズについて説明し，クラスサイズを用いた予測判別のための閾値とその決定法を提案する．**3** 節では，提案した手法の適用例として，多数の Java プログラムに対する測定実験とその結果分析について述べる．**4** 節では，本論文のまとめと今後の課題を示す．

2. ソフトウェア変更量とクラスサイズ

2.1 ソフトウェア変更量

本論文では，ソフトウェアのバージョンアップに伴うソースコードの変更行数を “ソフトウェア変更量” とする．ただし，コメント文及び空文についてはこの限りでない．ここでいう変更には “修正，” “追加，” “削除” の 3 種類があるが，それぞれ次のように定義し，変更量に計上する：

便宜上，バージョンアップ前のソースコードを S_1，バージョンアップ後のソースコードを S_2 とする．S_1 に対して以下の 3 種類の操作のいずれかを繰り返し適用すれば S_2 と同一のコードが得られる．

（1） 連続した n ($\geqq 1$) 行のコードをそれとは内容の異なる連続した m ($\geqq 1$) 行のコードに置き換える．ただし，いずれの行も内容は異なるものとする．

（2） 連続した m ($\geqq 1$) 行のコードを挿入する．

（3） 連続した n ($\geqq 1$) 行のコードを削除する．

このとき，(1) の操作に該当する部分をソフトウェアの “修正” 部分と定義し，変更量の計上には n と m の大きい方を用いる．(2) の操作に該当する部分を “追加” 部分と定義し，変更量の計上には m を用いる．(3) の操作に該当する部分を “削除” 部分と定義し，変更量の計上には n を用いる．　□

本来，ソフトウェアの変更作業にはフォールトの除去のための作業と機能変更・向上のための作業が混在しているが，ソースコードのみからこれらを分類するのは困難であるため本論文では区別しない．

上述の変更行数の計上には， UNIX コマンド “diff” が利用できる．変更前及び変更後の Java プログラムに対して diff コマンド(注1)を用いた例を図 1 に示す．ただし，オプションには “`-cbwB`” を用いている．各オプションの意味は以下の通りである．

- “`-c`” オプション：出力形式を context 形式とする．
- “`-b`” オプション：空白の数だけが違う場合には違いを無視する．
- “`-w`” オプション：行を比較する際に空白を無視する．
- “`-B`” オプション：空行を挿入・削除するだけの変更を無視する．

(本研究では，視覚による確認と測定ツール開発の両方の利便

(注1)：GNU diffutils バージョン 2.8.1 を使用．

性を考慮して context 出力形式を採用している．あわせて，空白や空行の存在による違いはソースコードの変更とは見なさないことにしている.）

図中の “!” で始まる行は修正箇所を，“+” で始まる行は追加箇所を，“-” で始まる行は削除箇所をそれぞれ表す．diff コマンドによって得られた出力ファイルから変更（修正，追加，削除）のあった行の合計数が計算できる．上述したように，修正については変更前と変更後のプログラムに対して “!” と表示された行を比較し，行数の多い方を計算に用いる．例えば図 1 の場合，“修正 2 行，追加 1 行，削除 1 行” となり，あわせて 4 行の変更として数えられる．

2.2 クラスサイズ

クラスサイズを測定するメトリクスとして，PIM（パブリックなインスタンスメソッドの数）[9] や LOC 等があるが，本論文では ソースコードの特性が強く反映されるメトリクスとして $Stmts$（Statements）[10] を用いる．$Stmts$ は “クラスにおけるメソッドの宣言文と実行文の数” として定義されている．単にプログラム行数，つまり LOC を用いるよりも，このメトリクスを用いた方がコーディングスタイルの影響を受けることなくクラスサイズを測定できる.（前節で説明したソフトウェアの変更量測定についても同様のことがいえるが，ソフトウェアの変更量という観点からいえば，変更のあった “文” の数よりも変更のあった “行” の数の方が実際の変更作業量に近いと考え，変更量測定には “文数” ではなく “行数” を採用した.）

測定対象を Java プログラムとした場合，宣言文及び実行文の定義は Java 言語仕様 [11] に従う．このときのメソッドの宣言文及実行文の一覧を表 1 に示す.

表 1　メソッドの宣言文及び実行文の一覧

While 文	If 文
Switch 文	Do 文
Break 文	Throw 文
Return 文	Continue 文
For 文	Synchronized 文
Try 文	式文
ローカル変数宣言文	ラベル付き文

$Stmts$ 値はソースコードから算出できるが，そのためには Java プログラムの字句・構文解析が必要である．そのため，ツール開発の容易さを考慮し，いったん Java プログラムを JavaML ファイル（XML ファイル）[12]〜[14] に変換し，それに対して構文解析を行うという方法が考えられる．Java プログラムから JavaML ファイルへの変換は IBM の Java コンパイラ “jikes” [15] を用いることで可能である.

2.3 ソフトウェア変更量とクラスサイズとの関係

ソフトウェア開発において，一般的にクラスサイズが大きくなれば変更・発展のための作業も複雑になり，その作業量も多くなると考えられる．そこで我々は，クラスサイズメトリクスを利用し，特に多くの変更が行われるようなクラスの検出を目指している．その一環として，本論文ではバージョンアップに伴うソースコードの変更量（変更行数）が特に多くなる場合とそうでない場合とをクラスサイズメトリクスを用いて（予測）判別するための一つの閾値について考える．このような閾値はソフトウェア開発において一つの品質管理基準になると考えられる．以下ではこれを順を追って形式的に定義する.

（1） あるクラス（Java プログラム）のソースコードを異なるバージョンごとに用意する．つまり，あるクラス C に対し，その複数の異なるバージョン $v_1, v_2, \ldots, v_n$ を用意する．ただし，$v_i < v_j$ $(i < j)$ とする．以下では，バージョン v_i のクラス C を “$C(v_i)$” と書く.
例えば，“Foo” というクラスのソースコード（Foo.java）について，バージョン “1.0,” “1.1,” “1.11,” “1.2” を用意した場合，これらをそれぞれ “Foo(1.0),” “Foo(1.1),” “Foo(1.11),” “Foo(1.2)” と書く.

（2） （1）で用意したバージョンの異なる複数のソースコードに対し，それぞれ直後のバージョンとの変更行数（差分量）を算出する．つまり，$(C(v_i), C(v_{i+1}))$ という組合わせ $(i = 1, \ldots, n-1)$ について，両者の間での変更行数を算出する．これを “$d(C, v_1, v_{i+1})$” と書く.
上述の例の場合，d(“Foo,”1.1, 1.11) 等を算出する.

（3） サイズメトリクス $Stmts$ を用い，各バージョンでのクラスサイズ（$Stmts$ 値）を測定する．以下では，“$C(v_i)$” の $Stmts$ 値を “$Stmts(C, v_i)$” と書く.
上述の例の場合，$Stmts$(“Foo,”1.0) 等を測定する.

（4） 他のクラスについても（1）,（2）,（3）を繰り返し，変更行数データ $d(C, v_i, v_{i+1})$ 及びサイズデータ $Stmts(C, v_i)$ を多数収集する $(i = 1, \ldots, n-1)$.

（5） 収集した多数の変更行数データ $d(C, v_i, v_{i+1})$ を降順に整列させ，最初の α % 以内，すなわち，変更行数の上位 α % 以内にあるようなバージョンアップを “特に変更の多かった” サンプルとする．α は品質管理者が経験的に定めるものとする．例えば，$\alpha = 5$ として上位 5% を特に変更の多かったサンプルと考える.

（6） ある自然数 τ を考え，

$$\begin{cases} \text{“}Stmts(C, v_i) \geqq \tau\text{” ならば} \\ \qquad \text{“}d(C, v_i, v_{i+1}) \text{ は上位 } \alpha\% \text{ に含まれる”}, \\ \text{さもなくば} \\ \qquad \text{“}d(C, v_i, v_{i+1}) \text{ は上位 } \alpha\% \text{ に含まれない”}, \end{cases}$$

と予測する．すなわち，サイズデータ “$Stmts(C, v_i)$” から変更行数 “$d(C, v_i, v_{i+1})$” が特に大きくなる場合を予測する.（図 2）

図 2 に示したように，閾値 τ をある値に決めると，それに対応した予測誤りのサンプルがいくつか現れる．したがって，予測誤りが最小となるような τ を求めればよい.

いま，実際に変更のあったサンプルのうち，変更行数が上位 α% 以内にあるサンプル数を N_1，それ以外のサンプル数を N_2 とする．そして，それぞれ

- $Stmts$ 値が τ 未満（$Stmts(C, v_i) < \tau$）であって変更行数が上位 α% 以内に含まれるサンプル数を E_1，
- $Stmts$ 値が τ 未満（$Stmts(C, v_i) < \tau$）であって変更

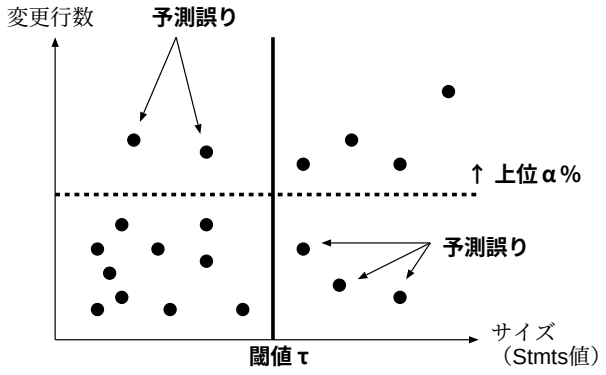

図 2　閾値 τ を用いた予測の概念図

行数が上位 α% 以内に含まれないサンプル数を S_1,

- $Stmts$ 値が τ 以上（$Stmts(C, v_i) \geqq \tau$）であって変更行数が上位 α% 以内に含まれるサンプル数を S_2,
- $Stmts$ 値が τ 以上（$Stmts(C, v_i) \geqq \tau$）であって変更行数が上位 α% 以内に含まれないサンプル数を E_2,

とする.（表 2）

表 2　場合分けされたサンプル数

		$Stmts$ 値		
		τ 未満	τ 以上	計
変更	上位 α%	E_1	S_2	N_1
行数	その他	S_1	E_2	N_2

このとき，E_1 及び E_2 が予測誤りに対応する．そこで，閾値 τ に対する誤り率を次式で定義する．

$$\varepsilon(\tau) = \frac{E_1}{N_1} + \frac{E_2}{N_2} \ . \qquad (1)$$

そして，$\varepsilon(\tau)$ が最小となるような閾値 τ を見つける．なお，$\varepsilon(\tau)$ を最小となるような自然数 τ が複数見つかるような場合は，その中で最小のものを目的の閾値とする．

□

上述の手順で得られた閾値 τ は，ソフトウェア開発に際して一つの開発基準として利用でき，保守コストの削減に対する一助になると考えられる．

3. 測定実験及び分析

本節では，前節で説明した手法の適用例として，Java プログラムに対する測定実験と閾値の算出を行う．

3.1 測 定 実 験

本実験では，Java 言語で記述されたオープンソースソフトウェア Relaxer [16] (注2)，JBoss (注3)，JEdit [17] (注4) を測定対象とした．その総クラス数は（バージョン違いを含めて）15405 個であった．

実験は以下の手順（1）〜（3）を繰り返した：

（1） クラス C（バージョン $v_1, \ldots, v_n$）に対し，各 $C(v_i)$ $(i = 1, \ldots, n)$ からコメント文を除去し，変更行数 $d(C, v_i, v_{i+1})$ $(i = 1, \ldots, n-1)$ を算出する．なお，変更箇所の抽出には **2.1** 節で説明した diff コマンドを利用した．

（2） **2.2** 節で説明したコンパイラ “jikes” を用いて Java プログラム（$C(v_i)$，$i = 1, \ldots, n$）からそれぞれに対応した JavaML ファイルを生成する．

（3） 各 JavaML ファイルを解析し，クラスサイズ $Stmts(C, v_i)$ を測定する（$i = 1, \ldots, n$）．具体的には，文の種類（表 1 参照）に対応したタグ “`<if>`” 等の数を計上する．

3.2 実験結果とその分析

変更行数の測定を行ったクラスのうち，3514 個のクラスのバージョンアップについて変更が見られた．この中で特に変更が大きかったサンプルに着目して分析する．そのようなサンプルの例として，今回は変更行数の上位 5 %（$\alpha = 5$）を用いた．本実験で得られたサンプルでは，上位 5 % に位置する変更行数は 154（昇順で 3339 番目のサンプル）であった．ここで変更行数が特に多くなるケースとそうでないケースとを $Stmts$ 値でもって予測・判別することを考え，そのための閾値 τ を考える．つまり，(1) 式で定義された誤り率 $\varepsilon(\tau)$ の値が最小となるような τ を算出する．

図 3 に $\varepsilon(\tau)$ のグラフを示す．

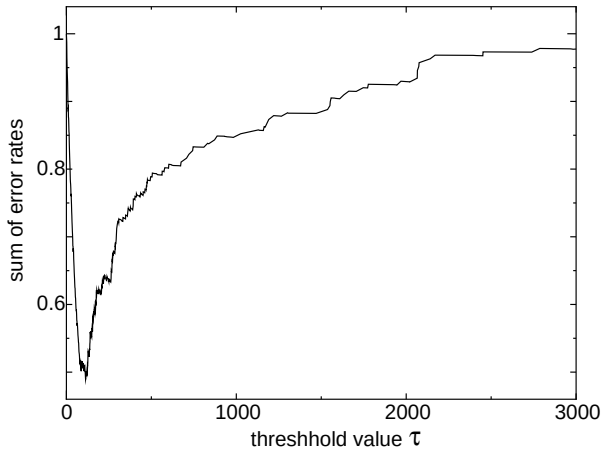

図 3　閾値 τ に対する誤り率の変化

結果として，$\tau = 113$ において $\varepsilon(\tau)$ が最小になっていた（図 3，最小値 0.489.）(注5)．この結果から，$Stmts$ 値に対する一つの閾値として 113 が考えられる.（参考：$Stmts$ 値が 113 である Java プログラムの平均行数は 246 （コメント文を含まず）であった.）

3.3 有効性に関する評価実験

上述の測定実験によって算出された $Stmts$ 値に対する閾値 113 の有効性について検討するため，上述のものとは別に（Java 言語で記述されたクラスの）バージョンアップのサンプルを 556 件用意した(注6)．この中で 15 個のクラスについて特に大きな変更が見られた（変更行数が 154 以上：上述の実験

(注2)：XML 処理プログラム（Java プログラム）を自動生成するツール.
(注3)：100%Pure Java で開発された J2EE アプリケーションサーバ.
(注4)：Java で記述されたプログラマ用のテキストエディタ.
(注5)：逐次比較により算出.
(注6)：SourceForge.net [17] より入手.

データ参照）．この 15 個のクラスの中で $Stmts$ 値が閾値 113 以上のものは 10 個であった．すなわち，特に変更が大きいクラスの約 67% については予測に成功していた．逆に，特に大きな変更は見られなかった 541(= 556 − 15) 個のクラスの中で $Stmts$ 値が閾値以上であったもの，すなわち，予測に失敗していたクラスは 143 個（約 26%）であった．決して高い予測精度ではないが，ある程度の有効性は確認できたと考えられる．

ここでの実験は一例に過ぎないが，上述の閾値は，変更量の大きいクラスの予測にある程度有効であることが期待できる．

4. むすび

本論文では，バージョンアップに伴うソースコードの変更量に着目し，変更量が特に大きいクラスとそうでないクラスとをクラスサイズから予測判別するための閾値とその決定法を提案した．適用例として多数の Java プログラムに対して測定実験を行い，クラスサイズメトリクス $Stmts$ の値に対して一つの閾値を見い出した．測定実験から算出した閾値をソフトウェア開発の際の品質基準として利用することで，クラスサイズを適切に抑制でき，バージョンアップに伴う変更コストやテスト工数の増大を未然に防ぐことができると期待される．今後の課題として，他のメトリクス（例えば，複雑度，凝集度等）を用いた場合の実験・検討，ならびに，複数のメトリクスを複合的に用いた判別分析 [18] による予測精度の向上が挙げられる．

謝辞

この研究は栢森情報科学振興財団の助成を受けて遂行された．ここに感謝の意を表する．

文　献

[1] 片山卓也，土居範久，鳥居宏次 監訳，“ソフトウェア工学大事典，” 朝倉書店，東京，1998.
[2] V.R. Basili, D.H. Hutchens, “An empirical study of a syntactic complexity family,” IEEE Trans. Software Eng., vol.SE-9, no.6, pp.652–663, 1983.
[3] S.D. Conte, V.Y. Shenm, H.E. Dunsmore, Software engineering metrics and modles, Benjamin Cummings Publishing Inc., Calif., 1986.
[4] J.L. Elshoff, “An investigation into the effects of the counting method used on software science measurements,” SIGPLAN Notices, vol.13, no.2, pp.30–45, 1978.
[5] M.H. Halstead, Elements of Software Science, Elservier North-Holland, 1997.
[6] 高橋良英，“C 言語ソフトウェア保守行程における Halstead のソフトウェアサイエンス計測と障害密度との関係の分析，” 信学論（D-I），vol.J82-D-I, no.8, pp.1017–1034, Aug. 1999.
[7] K.E. Emam, S. Benlarbi, N. Goel, W. Melo, H. Lounis, S.N. Rai, “The optimal class size for object-oriented software,” IEEE Trans. Software Eng., vol.28, no.5, pp.494–507, May 2002.
[8] 望月尚美，阿萬裕久，山田宏之，野田松太郎，“ソフトウェアの変更量とクラスサイズとの関係解析，” ソフトウェア工学の基礎 X，pp.109–112, 近代科学社，2003.
[9] M. Lorenz, J. Kidd, Object-Oriented Software Metrics, PTR Prentice Hall, New Jersey, 1994 （宇治邦明 監訳，オブジェクト指向ソフトウェアメトリクス，プレンティスホール出版，東京，1995）.
[10] L.C. Briand, J. Wust, J.W. Daly, D.V. Porter, “Exploring thd relationships between design measures and software quality in object-oriented systems,” J. Systems and Software, vol.51, pp.245–273, 2000.
[11] http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html
[12] 山中祐介，大畑文明，井上克郎，“プログラム解析情報の XML データベース化——提案と実現，” コンピュータソフトウェア，vol.19, no.1, pp.39–43, Jan. 2002.
[13] 阿萬裕久, 坂井一憲, 山田宏之, 野田松太郎，“JavaML を用いたクラス設計メトリクス測定ツールの開発とその利用，” 情処学論，vol.43, no.12, pp.4005–4008, Dec.2002.
[14] G.J. Badros, “JavaML : A markup language for Java source code,” Proc. 9th International World Wide Web Conference, 2000.
[15] http://www-124.ibm.com/developerworks/opensource/jikes/
[16] http://www.asahi-net.or.jp/˜dp8t-asm/java/tools/Relaxer/
[17] http://sourceforge.net/
[18] 竹村彰通，統計，共立出版，東京，1997.