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

望月尚美
愛媛大学大学院 理工学研究科 情報工学専攻）

阿萬裕久 山田宏之 野田松太郎
愛媛大学 工学部 情報工学科）

本研究の概要

目的

オブジェクト指向ソフトウェアの変更・発展に伴う変更量とクラスサイズとの関係解析を行い，ソフトウェア技術者に対して開発基準を提供．

方法

オープンソースソフトウェアに対し，バージョンアップに伴う変更量の測定及びクラスサイズの測定．

結果

測定結果を分析し，ソフトウェア変更量をクラスサイズによって予測・判別するための閾値を算出．

研究背景

- ハードウェアの進歩により，ソフトウェアは大規模かつ高機能なものが求められている．

ソフトウェアの開発・保守コストの増大

- 一般的にオブジェクト指向設計は開発性・保守性に優れているといわれている．

現実には技術者の力量・経験に依存

研究背景

ソフトウェア品質の評価・予測やコストの見積もりが必要

- ライフサイクルコストにおいて最も大きいといわれている保守コストに注目．
- 保守において施された変更の量とクラスサイズとの関係について考察．

テスト工数にも強く影響

ソフトウェア変更・発展に伴う変更量

ソフトウェアの変更量

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．

- 連続したn行のコードを連続したm行のコードと置換
- 連続したm行のコードを挿入
- 連続したn行のコードを削除

※コメント文・空文は除く

変更行数の測定方法

(1) ソースコードからコメント文を除去．

(2) UNIX コマンド「diff」を用いて，あるバージョンとその直後のバージョンとを比較．

(3) diff コマンドの出力結果から変更行数を測定．

diffを用いた出力ファイルの例

```
public class HelloWorld {
    public static void main(String[] args){
        for ( int i = 0; i < 10; i++ ){
            System.out.println("HelloWorld!!");
            System.out.println("H
        }
    }
}
```

version 1.0: HelloWorl

```
public class HelloWorld {
    public static void main(String[] args){
        for ( int i = 0; i < 5; i++ ){
            System.out.println("HelloWorld!!");
        }
        System.out.println("END");
    }
}
```

version 1.1: HelloWorld.java

diff の出力

```
*** 1, 8 ****
public class HelloWorld {
    public static void main(String[] args){
!           for ( int i = 0; i < 10; i++ ){
                System.out.println("HelloWorld!!");
-           System.out.println("Hello");
        }
```

削除

修正 ・削除 ・追加の 合計数 ＝ 変更行数
※但し, 修正は変更前と変更後の行数を比較し大きい方を用いる)

```
    public static void main(String[] args){
!           for ( int i = 0; i < 5; i++ ){
!               System.out.println("[" + i + "]");
                System.out.println("HelloWorld!!");
+       version 1.0 からversion 1.1 へ
        バージョンアップしたときの変更行数 :3
    }
}
```

追加

diff の出力

```
*** 1, 8 ****
public class HelloWorld {
    public static void main(String[] args){
!           for ( int i = 0; i < 10; i++ ){
                System.out.println("HelloWorld!!");
-           System.out.println("Hello");
        }
```

削除

修正 ・削除 ・追加の 合計数 ＝ 変更行数
※但し, 修正は変更前と変更後の行数を比較し大きい方を用いる)

```
    public static void main(String[] args){
!           for ( int i = 0; i < 5; i++ ){
!               System.out.println("[" + i + "]");
                System.out.println("HelloWorld!!");
+       version 1.0 からversion 1.1 へ
        バージョンアップしたときの変更行数 :4
    }
}
```

追加

クラスサイズ

各バージョンにおけるクラスのサイズ を
メトリクス Stmts（Statements）で測定.

メトリクス Stmts

クラスにおけるメソッドの宣言文と実行文の数.
※ ≠ LOC :プログラム行数）
コーディングスタイルに依存しない測定.

Java 言語における メソッドの宣言文と実行文の一覧

- ローカル変数宣言文
- 式文
- 代入文
- インクリメント文
- デクリメント文
- Break 文
- Synchronized 文
- Continue 文
- Return 文
- Throw 文
- ラベル付き文
- メソッド呼び出し文
- クラスインスタンス生成文
- If 文
- While 文
- For 文
- Switch 文
- Do 文
- Try 文

Stmtsの測定方法

Stmts はソースコードから測定可能.
⇒ Javaプログラムの構文解析が必要.

JavaソースコードをXML（**JavaML**）へ変換.
⇒ Stmts の測定が容易になる.

(1) コメント文を除去したソースコードを JavaML に変換.
⇒ IBM が提供しているコンパイラ "jikes "を用いる.

(2) JavaMLファイルからメソッドの宣言文と実行文の数を測定. 対応する**タグの数を数える**）

Stmtsの測定方法

変換前のJavaプログラム

```
public class HelloWorld {
    public static void main(String[] args){
        int i = 0 ;
        while( i < 5 ){
            System.out.println("Hello World!!");
            i++;
        }
    }
}
```

```
<method name="main" visibility="public" static="true">
<type name="void" primitive="true"/>
<formal-arguments><formal-argument name="args" id="HelloWorld:frm-9">
<type name="String" dimensions="1"/>
</formal-argument></formal-arguments>
<block>
    <local-variable name="i" id="HelloWorld:var-26">
    <type name="int" primitive="true"/>
    <literal-number kind="integer" value="0"/></local-variable>
    <loop kind="while"><test>
    <binary-expr op="&lt;"><var-ref name="i"/>
    <literal-number kind="integer" value="5"/></binary-expr></test>
        <block>
        <send message="println">
        <target><field-access field="out"><var-ref name="System"/>
        </field-access></target><arguments>
        <literal-string value="Hello World!!"/></arguments></send>
       <unary-expr op="++" post="true"><var-ref name="i"/></unary-expr>
        </block></loop>
</block></method>
```

JavaML の一部

public static void main(String[] args){ ・・・・・} に対応

```
<method name="main" visibility="public"  static="true">
  <type name="void" primitive="true"/>
  <formal-arguments>
    <formal-argument name="args" id="HelloWorld:frm-9">
       <type name="String" dimensions="1"/>
    </formal-argument>
  </formal-arguments>
            ・・・・・・・
</method>
```

JavaML の一部

int i = 0 に対応

```
<local-variable name="i" id="HelloWorld:var26">
<type name="int" primitive="true"/>
<literal-number kind="integer"value="0"/>
</local-variable>
```

i++ に対応

```
<unary-expr op="++" post="true">
  <var-ref name="i"/>
</unary-expr>
```

```
<method name="main" visibility="public" static="true">
<type name="void" primitive="true"/>
<formal-arguments><formal-argument name="args" id="HelloWorld:frm-9">
<type name="String" dimensions="1"/>
</formal-argument></formal-arguments>
<block>
  <local-variable name="i" id="HelloWorld:var-26">
    <type name="int" primitive="true"/>
    <literal-number kind="integer" value="0"/></
  <loop kind="while"><test>
  <binary-expr op="&lt;"><var-ref na
  <literal-number kind="integer" value="5"/></
    <send message="println">
      <target><field-access field="out"><var-ref name="System"/>
      </field-access></target><arguments>
      <literal-string value="Hello World!!"/></arguments></send>
    <unary-expr op="++" post="true"><var-ref name="i"/>
      </unary-expr></block></loop>
</block></method>
```

メソッドにおける宣言文と実行文に対応

変更・発展のための変更量とクラスサイズとの関係

- 一般的にクラスサイズが大きくなれば，変更・発展のための変更量も多くなると考えられる．

バージョンアップに伴うソースコードの変更量が特に多くなる場合とそうでない場合を予測・判別するための一つの閾値について考える．

ソフトウェア開発において品質管理の目安にできる

分析方法

1. あるクラス C に対して, 複数のバージョン(v_1, v_2, … v_n)を用意
2. あるバージョンのクラス C に対して, 直後のバージョンと比較し変更行数を算出

   $C(v_i)$ と $C(v_{i+1})$ を比較
3. クラスサイズメトリクス Stmts を用い, 各バージョンでのクラスサイズを測定

   $C(v_i)$ の Stmts 値を測定

分析方法

測定を行ったクラスのうち, 特に変更が大きかったサンプル（変更行数の上位α%）に着目.

※αは品質管理者が経験的に決定.

閾値を導入し, 変更行数が特に多くなる場合とそうでない場合とを Stmts 値でもって予測・判別することを考える.

測定実験と結果分析

測定対象

オープンソースソフトウェア Relaxer, JBoss, JEdit に含まれていた Java プログラム． 複数の異なるバージョンのものを含め, 総クラス数 **15405個**）

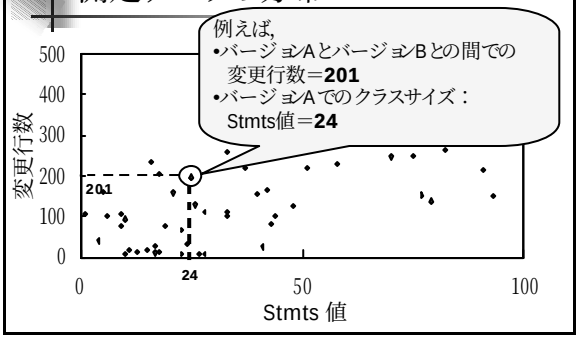
特に変更が大きかったクラス（変更行数の上位α%）に着目.

※αは品質管理者が経験的に決定.

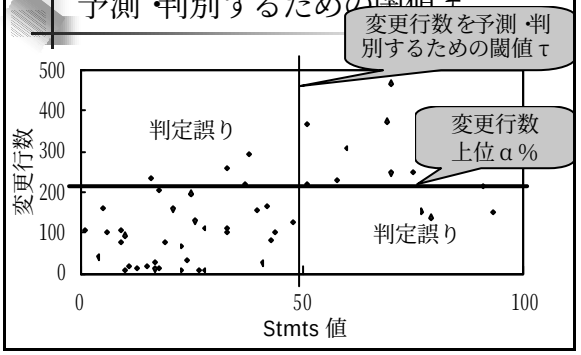
今回は変更行数の上位5%に着目し, 分析を実施.

閾値を導入し, 変更行数が特に多くなる場合とそうでない場合とを Stmts 値でもって予測・判別することを考える.

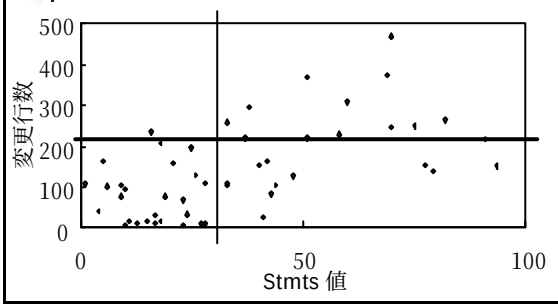
測定データの分布



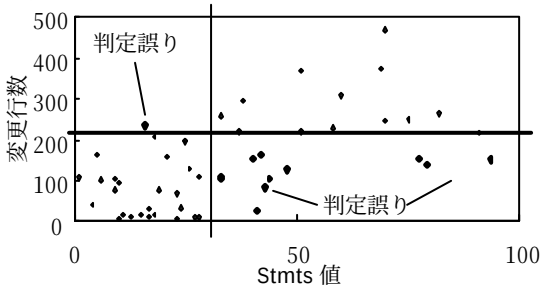
予測・判別するための閾値τ



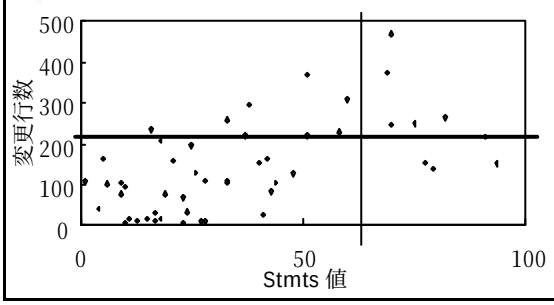
予測・判別するための閾値τ



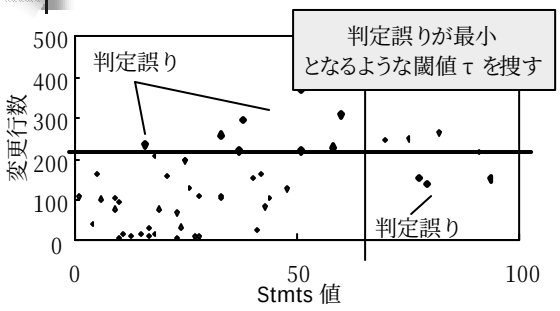
予測 判別するための閾値 τ



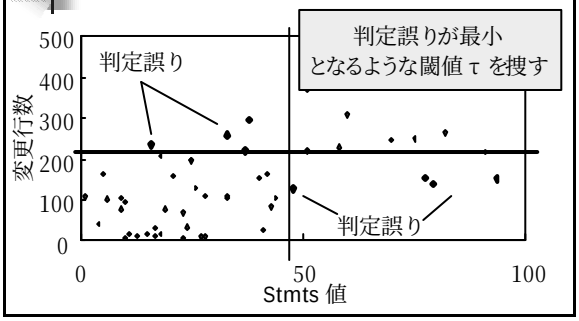
予測 判別するための閾値 τ



予測 判別するための閾値 τ



予測 判別するための閾値 τ



予測 判別するための閾値 τ

測定データの分割表

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

予測 判別誤差の大きさ ε (τ):

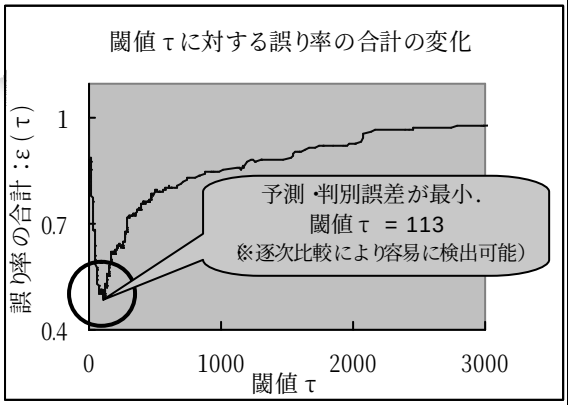
$$\varepsilon(\tau) = E1/N1 + E2/N2$$

測定実験と結果分析

測定対象

オープンソースソフトウェア Relaxer, JBoss, JEdit に含まれていた Java プログラム． 複数の異なるバージョンのものを含め, 総クラス数 **15405個**)

測定を行ったクラスのうち, 特に変更が大きかったサンプル (今回は変更行数の上位5%)に着目し, 分析.



ε（τ）が最小となるτ=113

この結果の持つ意味

- Stmts 値≧113のクラスではバージョンアップに伴う変更量が特に多い傾向にあった
  ⇨113未満となるよう開発するとよい．

参考）

文の数 113 ⇔ プログラム行数 250行程度 ※コメント文を除く

おおよそ対応

※実験データによる

τ = 113 における有効性の検討

- 新たに 556個 のサンプルを取得し，実験を実施

		Stmts 値		
		τ未満	τ以上	計
変更行数	上位5%	5	**10**	15
	その他	**398**	143	541

トータルで**約73%**の予測判別に成功
今後の課題：予測精度の向上

まとめ

- 変更・発展に伴う変更量が多かったクラスに着目し，その変更量とクラスサイズとの関係解析を実施．
- 閾値を導入し，変更行数が特に多くなる場合とそうでない場合とをStmts 値でもって予測・判別する方法について検討．

今後の課題

- 他のメトリクスを用いた場合の実験・検討．
- 複数のメトリクスを用いた判別分析による予測精度の向上．