

Javaプログラムのアクセス修飾子過剰性分析 ツールModi Checkerの機能拡張とその応用例

大阪大学情報工学科研究科: 小堀 一雄 石居 達也 松下 誠
井上 克郎

背景

- ▶ Java言語にはpublic, protected, privateなどのアクセス修飾子があるが、テスト時に不適切なアクセスを行ってしまうことがある。
- ▶ ソースコードの規模が大きいと、すべてのメソッドに対して正確なアクセス修飾子を設定することが困難である。
- ▶ 支援が必要である。

表 1 アクセス修飾子の種類

アクセス修飾子	アクセスを許容する範囲
public	全ての部品
protected	自身と同じパッケージに所属する部品および自身のサブクラス
default (指定なし)	自身と同じパッケージに所属する部品
private	自身と同じクラス

アクセス修飾子によって引き起こされる問題

- ▶ 3つのメソッドを持つクラスXを例に説明する。

```
public class X{  
    private String y = null;
```

```
    private method A( ){  
        y=new String ("hello");  
    }
```

```
    public method B(){  
        y.Length( );  
    }
```

```
    public method C( ){  
        this.method A( );  
        this.method B( );  
    }
```

methodAはyに”hello”を代入する。

methodBはyの長さを返す。

methodCはAとBを実行する

しかしmethod A、B、Cのうちアクセス修飾子が不適切なものがある。

それはどれか？

```
}
```

アクセス修飾子によって引き起こされる問題

- ▶ 3つのメソッドを持つクラスXを例に説明する。

```
public class X{  
    private String y = null;
```

```
    private method A( ){  
        y=new String ("hello");  
    }
```

```
    public method B(){  
        y.Length( );  
    }
```

```
    public method C( ){  
        this.method A( );  
        this.method B( );  
    }
```

```
}
```

method Bのアクセス修飾子に問題がある。

method Bがpublicだと外部からyが無い状態で実行されてしまい。Null Pointer Exceptionが発生する。

AE(Accessibility Excessiveness)

▶ AE

- ▶ 宣言したアクセス修飾子と実際のアクセス修飾子の範囲の差
- ▶ 下の表背景に色がある箇所の6種類ある。
- ▶ 例えば宣言されているものに対して実際にはprotectedの場合のアクセスであればAEはpub1とする。
- ▶ pub0やpro0は適正と判断。

▶ ModiCheker

- ▶ 上記のAEをソースコードから検出するツール。

表 2 AE の種類

† \ ‡	public	protected	default	private
public	pub0	pub1	pub2	pub3
protected	x	pro0	pro1	pro2
default	x	x	def0	def1
private	x	x	x	pri0

†列タイトル: 宣言されているアクセス修飾子

‡行タイトル: 実際にアクセスされている範囲

Modi Checkerの改良

▶ NoAccessの追加

- ▶ 宣言されてはいるが、実際にはアクセスがないものが多数あった。
- ▶ AEの対象にアクセスがなかった場合も追加した。

表 3 拡張した AE の種類

† \ ‡	public	protected	default	private	No Access
public	pub0	pub1	pub2	pub3	pub4
protected	x	pro0	pro1	pro2	pro3
default	x	x	def0	def1	def2
private	x	x	x	pri0	pri1

†列タイトル: 宣言されているアクセス修飾子

‡行タイトル: 実際にアクセスされている範囲

▶ AE修正機能の追加

- ▶ 宣言したアクセス修飾子と、実際のアクセス修飾子が違った場合、選択してソースコードを修正する機能を追加した。

Modi Checkerの画面

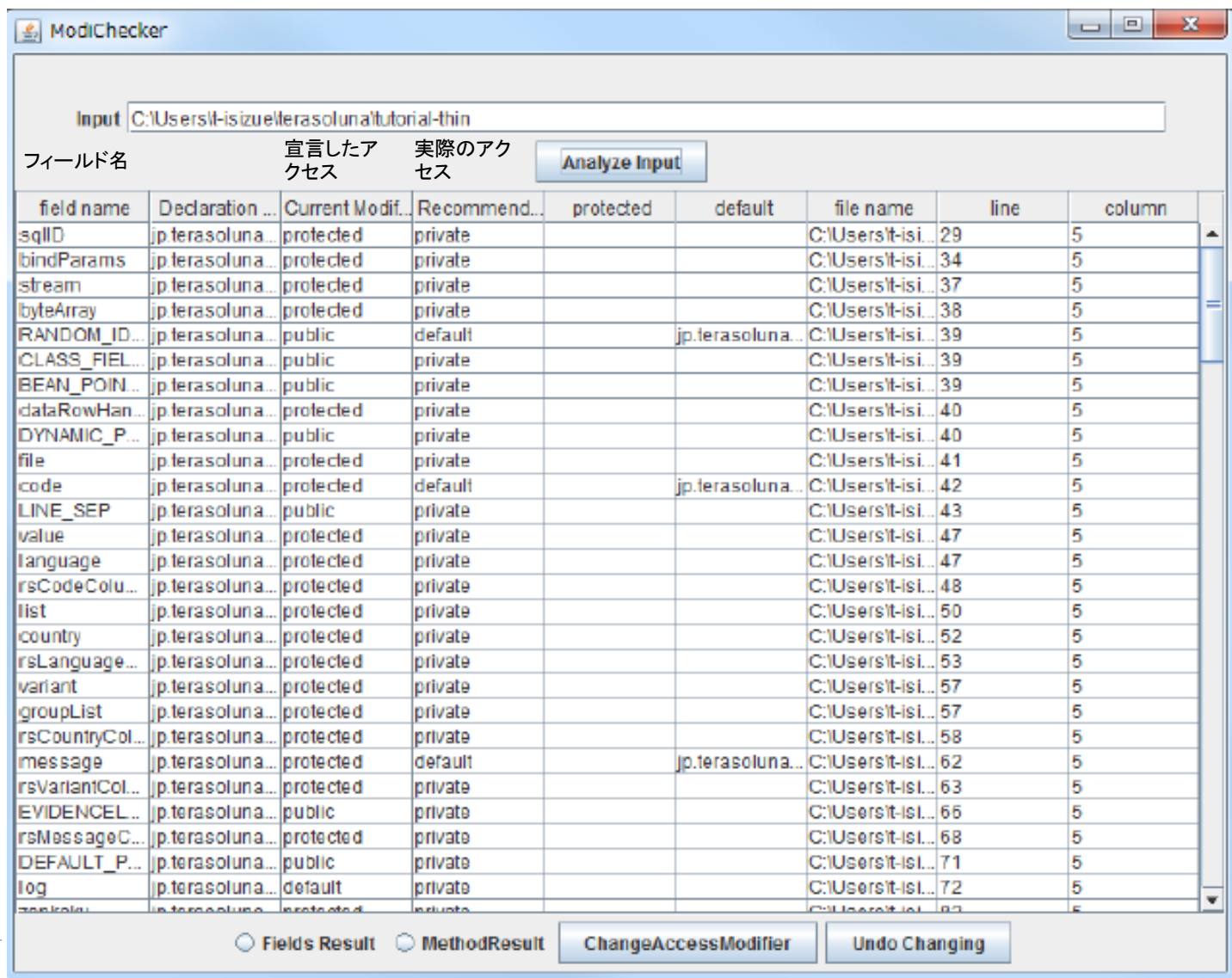


図 1 ModiChecker の AE 表示画面

ModiCheckerの検証

- ▶ 対象プロジェクト: Ant のバージョン1.8.2
 - ▶ v1.8.2はプログラムの動作確認を行うテストスイートを持つ。
- ▶ 手順
 - ▶ 1.Ant-v1.8.2のテストスイートを実行し実行結果を確認する。
 - ▶ 2.ModiCheckerでアクセス修飾子を抽出する。
 - ▶ 3.開発者が意図的に設定したであろうアクセス修飾子を修正対象から外す。
 - ▶ 4.修正した後、ビルドエラーがでなくなるまでビルドする。
 - ▶ 5.テストスイートを行い、実行結果が変わっていないかを確認する。



抽出結果

- ▶ 798 個のフィールドおよび6076個のメソッドがAE であると判定された。
- ▶ 5695個のメソッドがAnt外部からアクセスするため、開発者の意図的なものとみなし、修正対象から外した。
- ▶ それ以外を修正対象とする。

	フィールド数	メソッド数
抽出したAE	798	6076
開発者の意図があるAE	0	5695
修正対象	798	381



修正の検証

▶ 修正の手順

1. 修正し、ソースコードをビルドする。
2. ビルドエラーが出たら問題のあるAEを修正対象から外す。
3. 再度ビルドする。
4. ビルドエラーが出なくなるまで繰り返す。
5. テストスイートを実行し結果が同じであるか確認する。

▶ ビルドにかかった回数と時間

ビルド総回数	ビルド総時間	ModiChecker実行時間
6回	約63分	約34分
	10分30秒(平均)	5分30秒(平均)

ModiCheckerの実行時間は1回あたり約5分30秒であり、実用性があると判断できる。



機能拡張の応用例

▶ バージョン間の分析

- ▶ 複数のバージョン間におけるAE 数の変化量を比較することで、開発者がソフトウェアの機能を成熟したと判断したかどうかを推定することができる。
- ▶ どのようなバージョンが一番過剰なアクセス修飾子が追加されるのか推定することができる。

▶ 予測

- ▶ 多くの変更があるメジャーバージョン
 - ▶ 過剰なアクセス修飾子の変化量がより大きいと予想される。
- ▶ バグ修正など小さな変更があるマイナーバージョン
 - ▶ 過剰なアクセス修飾子の変化量が小さいと予想される。



バージョン間分析予測の検証方法

▶ 検証

- ▶ ソフトウェアのバージョンをメジャー、マイナーの2種類に分け、2つの変化量を調べて有意差検定を行う。

▶ MajorVU: 1.2→1.3 や、1.6.5→1.7.0 など。

▶ MinorVU: 1.4→1.4.1 や、1.6.4→1.6.5 など。

▶ 分析対象

- ▶ Ant のバージョン1.1 からバージョン1.8.4 までの22 個のバージョンを対象とする。
- ▶ 22個の内、メジャーは7個、マイナーは15個であった。



マイナーバージョン間分析結果の比較

表 5 バージョン 1.4 のフィールドの AE 数

† ↕	public	Protected	default	private	No Access
↑					
public	49	3	8	82	6
protected	x	16	51	314	10
default	x	x	2	51	3
private	x	x	X	1214	27

表 6 バージョン 1.4.1 のフィールドの AE 数

† ↕	public	Protected	default	private	No Access
↑					
public	49	3	8	82	6
protected	x	17	51	314	10
default	x	x	2	49	3
private	x	x	X	1217	27

- ▶ このMinorVUでのAEの変化量は-2である。
- ▶ MinorVUのAEの変化量は少ない。

メジャーバージョン間分析結果の比較

表 4 バージョン 1.3 のフィールドの AE 数

† \ ‡	public	Protected	default	private	No Access
public	39	0	20	84	2
protected	x	15	37	181	3
default	x	x	1	43	2
private	x	x	X	952	21

†列タイトル: 宣言されているアクセス修飾子

‡行タイトル: 実際にアクセスされている範囲

表 5 バージョン 1.4 のフィールドの AE 数

† \ ‡	public	Protected	default	private	No Access
public	49	3	8	82	6
protected	x	16	51	314	10
default	x	x	2	51	3
private	x	x	X	1214	27

- ▶ MajorVUではAEの変化が大きいことが分かる。
- ▶ MinorVUより明らかに変化が大きい。
 - ▶ Pro2に関しては133個も増えている。

全バージョンのAE変化量

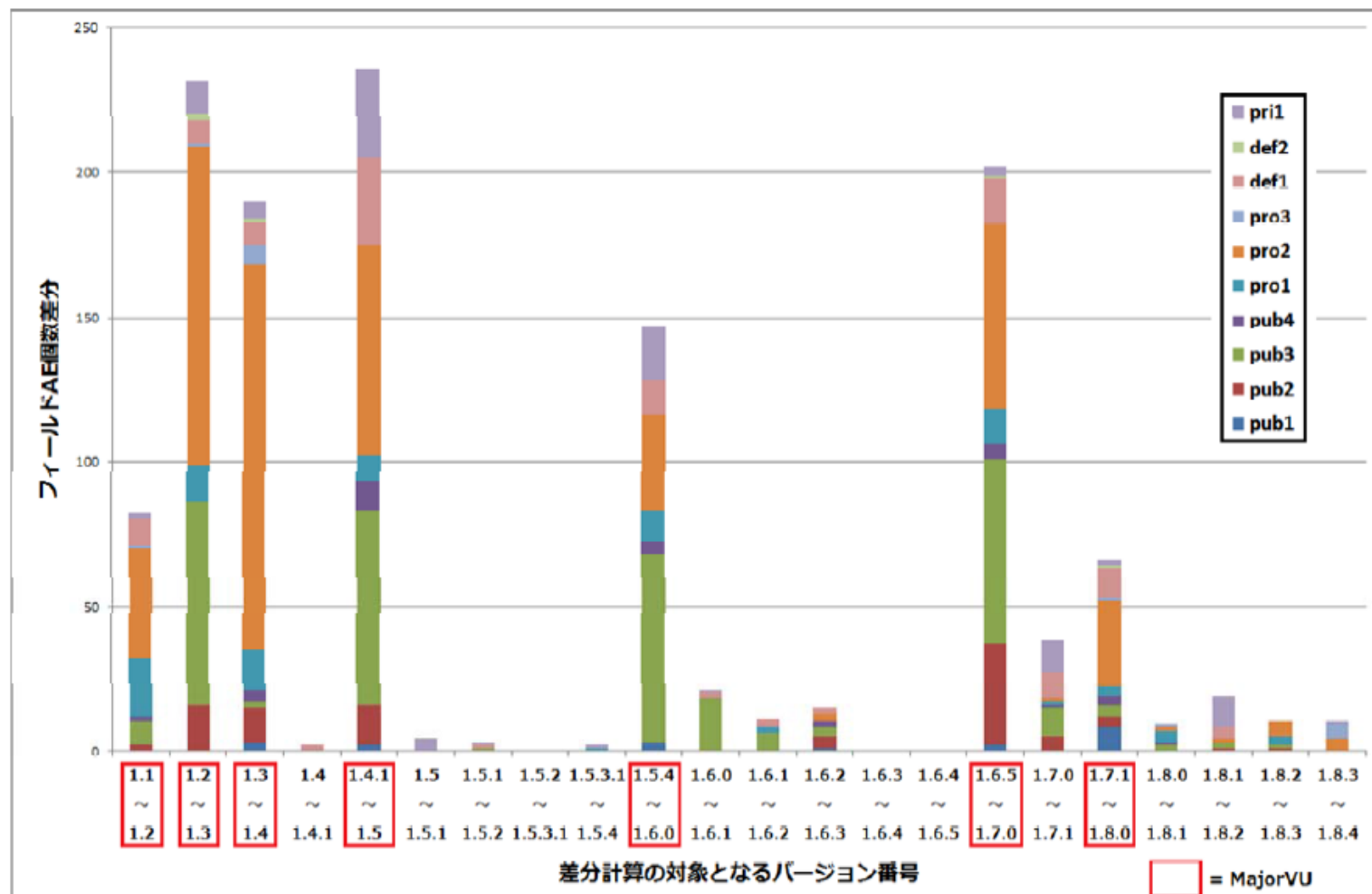


図 2 Ant のバージョン間における、フィールドの AE 数の各要素数の差分

全バージョンのAE変化量からの考察

- ▶ MajorVU 時 (図2 の赤で囲った列) に pro2 と pub3 の変化量が多く検出されている。
 - ▶ pro2とpub3含め、約80%のAEがprivateなので、多くのフィールドでカプセル化が可能となる。
- ▶ NoAccessに関するAEは全体の2~2.5%ほどであり、デッドコードは少ない。
- ▶ 全てのバージョンでMajorVU時のほうがMinorVU時よりAEの変化量が多いことが分かる。
 - ▶ マンホイットニーのU検定にて全てのアクセス修飾子で有意差が認められた。



まとめ

- ▶ ModiCheckerを使うと、多量のソースコードの場合、短い時間でアクセス修飾子の変更ができることを示した。
- ▶ MajorVUの方がMinorVUよりAEの変化量が明らかに大きいことが明らかになった。
- ▶ 本実験の応用として
 - ▶ ある保守対象のソフトウェアに使用し、本実験のMajorVU時のような変化が見られた場合、アクセス修飾子に過剰性を残した機能が追加されたかを発見できる可能性がある。

