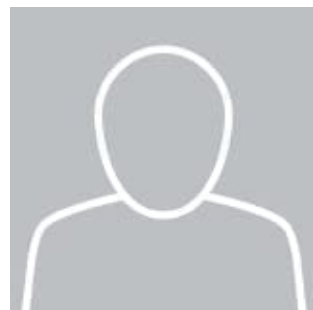




Large-scale study of substitutability in the presence of effects



Jackson Maddox, Yuheng Long, Hridesh Rajan

ESEC/FSE 2018 Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering Pages 528-538

発表者 吉富 楓雅

副作用と継承

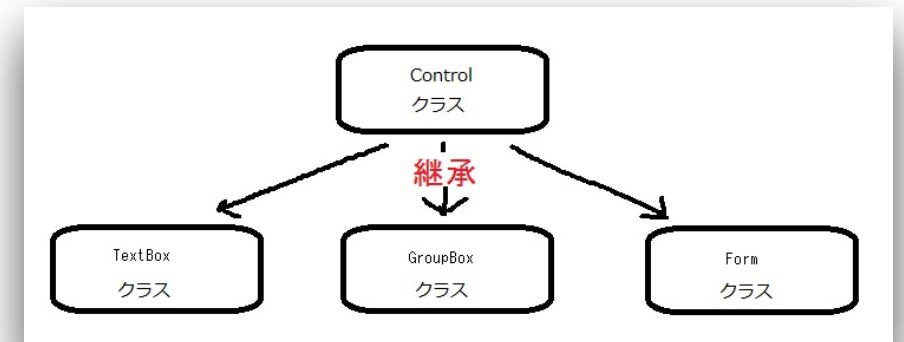
- 現在のプロジェクトは、オブジェクト指向の**副作用 (side-effect)**や**継承**を利用

* 副作用

- ▶ 例外のスロー, ファイルの読み取り, etc...

* 継承

- ▶ クラスのサブタイプ化, メソッドのオーバーライド, etc...



クラスの代用

- あるクラスを別のクラスに置き換える場合”**リスコフの置換原則**”が成立[1][2]
 - * もしリスコフの置換原則が成立しない場合、バグが発生する恐れがある
 - ▶ スーパークラスとそのサブクラス間など
- しかしながら、**クラスの代用による違反(violation)については知られていない**
- **クラスの代用による違反は一般的に発生している？その影響は？**

[1] Gary T. Leavens and William E. Weihl. 1995. Specification and Verification of Object-oriented Programs Using Supertype Abstraction. Acta Informatica 32, 8 (August 1995), 705–778. <https://doi.org/10.1007/BF01178658>

[2] Barbara H. Liskov and Jeannette M. Wing. 1994. A Behavioral Notion of Subtyping. ACM Trans. Program. Lang. Syst. 16, 6 (November 1994), 1811–1841. <https://doi.org/10.1145/197320.197383>

リスコフの置換原則



T型のオブジェクトxに関して真となる属性を $q(x)$ とする。この時SがTの派生型であれば、S型のオブジェクトyについて $q(y)$ が真となる

- オブジェクト指向であれば、スーパークラスとサブクラス間にはリスコフの置換原則が成立する

```
public void Func(ISampleInterface sample)
{
    var sample1 = sample as ConcreteSample1; // やばい！

    if (sample1 != null) // 特定の処理に限定している！よくない！
        sample1.Hogehoge();
}
```

リスコフの置換原則に違反している例

研究の目的

- リスコフの置換原則が破られるときの影響について調査する
 - * どのように置換原則は破られるのか？
 - * 置換原則が破られたことによる調査

リサーチクエスチョン

- RQ1：スーパークラスに基づいているサブクラスで違反が発生することはあるか？
 - RQ2：クラスの代用による違反が発生した場合，スーパークラスとサブクラスで実装はどのように異なるのか？
 - RQ3：クラスの代用による違反が発生した場合，どのような影響があるのか？
- * 代用可能性のバグやCode Smellの原因となる可能性がある

調査対象

- クラスプラットフォームのPython開発用IDE Boa[3][4]

- * Githubで管理

- 2015年9月のデータセットを分析

- * 380,125のJavaプロジェクトを含む



[3]Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N. Nguyen. 2013. Boa: A Language and Infrastructure for Analyzing Ultra-large-scale Software Repositories. In Proceedings of the 2013 International Conference on Software Engineering (ICSE '13). IEEE Press, Piscataway, NJ, USA, 422–431.
[4]<https://github.com/boalang>

クラス代用の調査

- 自動分析と手動分析の2種類で調査
 - * [5]の**型システム**の手法を利用して、スケーラブルにする
 - * コンストラクタやprivateを含まないメソッドのみを抽出、オーバーライドしているメソッドからスーパークラスとサブクラスのペアを形成する

クラス代用の調査

Table 1: Summary of the data studied

Metric	Count
# of projects	380,125
# of source files	20,302,663
# of classes	20,569,922
# of methods	149,294,833
# of method pairs	5,975,136
# of project AST nodes	6,619,264,814

- method pair : スーパーメソッドとサブメソッドを含むペア
- パッケージ名とファイル名が重複している場合は、最後に変更されたファイルを使用

クラス代用の調査

- クラス代用をしているメソッドのペアに違反が見られる場合、サンプルをランダムに収集
- サブメソッドの副作用の効果がスーパーメソッドの本体から推測できるのかを調査

クラス代用の調査

- 各メソッド内でスローされた**例外処理のタイプ**を追跡し、発見できた場合は、メソッドのスローされた例外のタイプのセットに追加
 - * IllegalArgumentException, IOExceptionなど
 - * 実際にはメソッドペアのうち例外があるのは4%
 - ▶ nullオブジェクトや範囲外の配列への参照

RQ2:スーパーメソッドとサブメソッド内の副作用

Table 3: Effect comparison of pairs with effects

Submethod has more effects	2,680,336 (53%)
Supermethod has more effects	1,287,747 (25%)
Both have same effects	395,940 (8%)
All others	710,258 (14%)
Total Pairs	5,074,281

- Submethod has more effects
 - * サブメソッドの方が副作用が多い場合
- Supermethod has more effects
 - * スーパーメソッドの方が副作用が多い場合

メソッドがスローする例外

**Table 4: Number exceptions types thrown per method kind
(NP means non-private, and Init means constructor)**

# Types	Private	NP Static	NP Init	Other
0	14M (91%)	12M (92%)	14M (96%)	99M (94%)
1	1.2M (8%)	961K (7%)	509K (3%)	5.6M (5%)
2	86K (1%)	92K (1%)	39K (0%)	462K (0%)
3+	14K (0%)	12K (0%)	3017 (0%)	98K (0%)
Total	14,936,491	12,938,996	14,883,048	104,776,047

- 94%は例外をスローしていない

* 例外処理：IllegalArgumentException, IOExceptionなど

RQ1:サブメソッドとスーパーメソッドの比較

- 例外処理が適切に”期待される場合”
 - * 例外処理が記述されており，期待通りにスローされる
- 例外処理が期待できない場合
 - * スーパークラスとサブクラスとの間で違反が発生する可能性がある

```
1 class Super {
2     void m(Object x) throws IOException {
3         if (x == null) throw new IllegalArgumentException();
4     } }
5
6 class Sub extends Super {
7     @Override void m(Object x) throws IOException {
8         if (x == null) throw new IllegalArgumentException();
9         else throw new IOException();
10    } }
```

適切な例外処理が行われている例

RQ1:サブメソッドとスーパーメソッドの比較

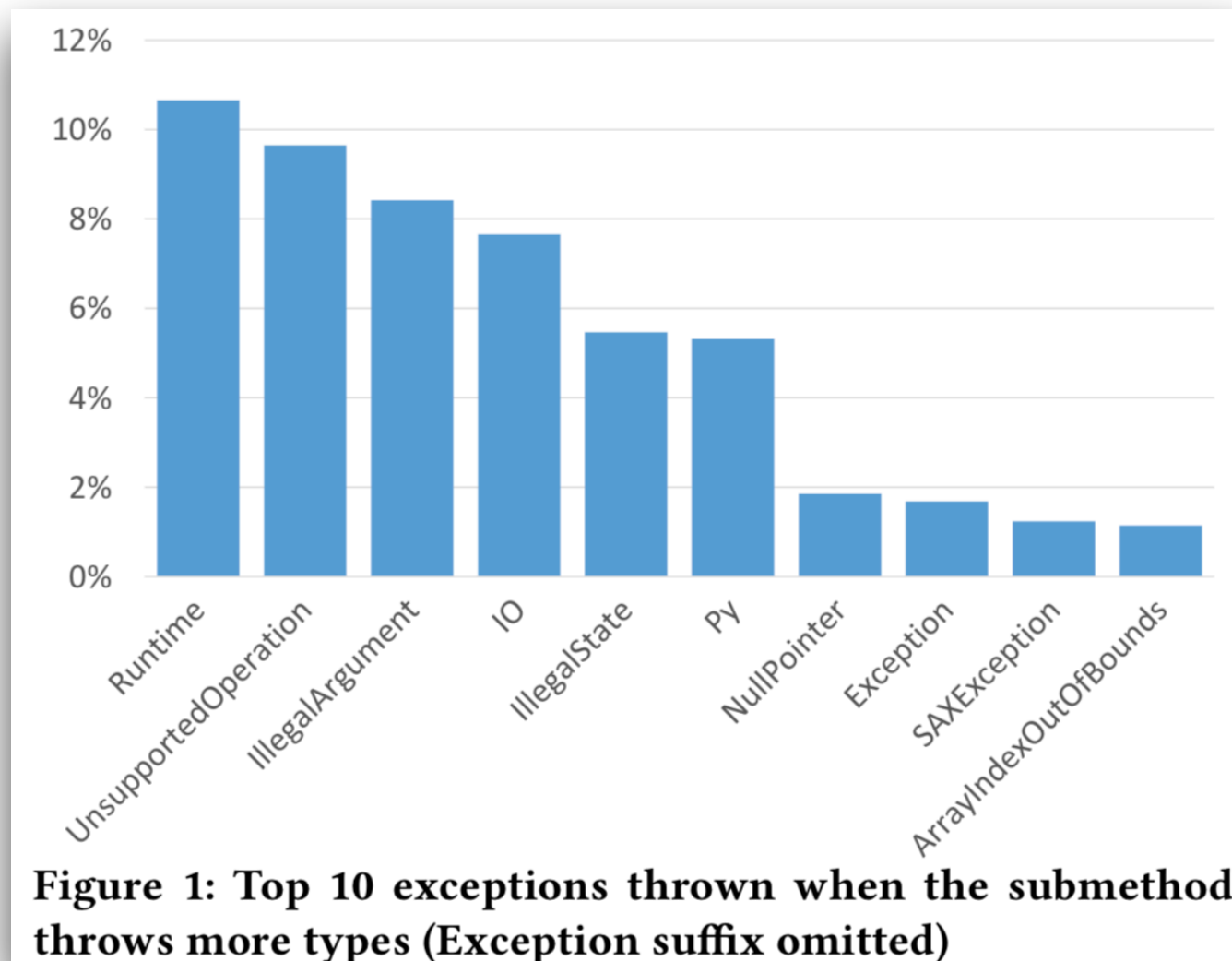
Table 5: Exception pair types (s=sub, p=super)			
Category	Pairs		
	Unexpected	Expected	Total
$s \subset p$	0 (0%)	350K (67%)	350K (48%)
$s \not\subset p, s < p $	5,399 (3%)	643 (0%)	6,042 (1%)
$s \supset p$	163K (79%)	61K (12%)	224K (31%)
$s \not\supset p, s > p $	3,203 (2%)	240 (0%)	3,443 (0%)
$s = p$	0 (0%)	103K (20%)	102K (14%)
$s \neq p, s = p $	34K (17%)	7,222 (1%)	41K (6%)
Total Pairs	205,844	521,880	727,724

- 28%のペアが”期待されない”，つまり違反しているクラス
の代用である
- 開発者はメソッドの効果を分析する時は，サブメソッドの
実装を考慮すべき

RQ3:ペアによってスローされる例外

- 例外の中にはサブメソッドの方がスーパーメソッドよりも強い前提条件を持つものがある

* スーパータイプの抽象化の観点で問題が発生する



まとめ

- クラスの代用による違反について実証研究
 - * リスコフの置換原則に違反している場合
- スーパークラスとサブクラスのペアのパターンについて調査
 - * スーパークラスに与える影響

所感

-  • 自動分析したと書いていたが、その方法がよくわからなかった
-  • オブジェクト指向の言語(継承)における問題を学ぶことができた
-  • 複雑な内容だったので読むのに結構時間がかかった
-  • 型システムについては全く分からなかったなので勉強しようと思う