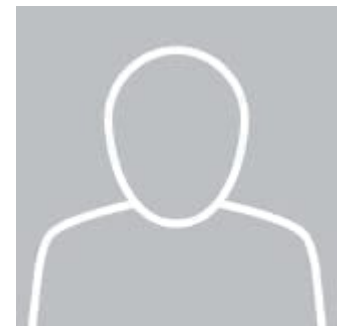


An industrial study on the differences between pre-release and post-release bugs



Renaud Rwemalika, Marinos Kintis, Mike Papadakis, Yves Le Traon, Pierre Lorrach

2019 IEEE International Conference on Software Maintenance and Evolution
(ICSME), Pages 92-102

発表者 吉富 楓雅

研究を商業利用へ

- ソフトウェア開発で発生する問題を最小限に抑えるために多くの研究が行われてきた
 - * コードレビュー, ソフトウェアテストなど
 - ▶ これらの手法はデータセットに含まれる**バグの特性に関連する**
- 商用プロジェクトではリリース後に**バグ**が発生すると大問題

ソフトウェアのバグとはそもそも何なのか??

バグの種類

欠陥タイプ (Type)	説明 (Description)
代入/初期化 (Assignment/Initialization)	(変数・定数への)値の代入の誤り、または初期化の誤り(Incorrect)、または欠如(Missing)
チェック/条件 (Checking)	If文などの条件判定文での、引数、データに対する判定の誤り、または欠如(ただし、引数やデータの取得に関数が用いられる場合はアルゴリズムとする。)
アルゴリズム/メソッド (Algorithm/Method)	アルゴリズムの誤り、または欠如
タイミング/順序 (Timing/Serialization)	データアクセスの順番制御やリソースアクセスに対する排他制御、スレッドの同期処理の誤り、または欠如
インターフェイス/オブジェクト間通信 (Interface/O-O Messages)	機能ブロック間、コンポーネント間、プロダクト間、又はハードウェアとソフトウェアの間のコミュニケーションに関する誤り、または欠如
関連付け (Relationship)	プロシージャ、データ構造やオブジェクトの継承()に関連する誤り、または欠如
機能/クラス/オブジェクト (Function/Class/Object)	要求仕様との不整合、機能性、ユーザインターフェイス又はグローバルデータ構造の誤り、または欠如

<おまけ>ODC分析

ODCはバグ分析の一手法である。1992年IBMワトソン研究所のRam Chillarege氏により発表された。

信頼度成長曲線

SGRM

(Software Reliability Growth Model)

- 収束を見る
- 具体的対策ない

直交欠陥分類

ODC

(Orthogonal Defect Classification)

- 即状況把握できる
- 習得・導入が大変

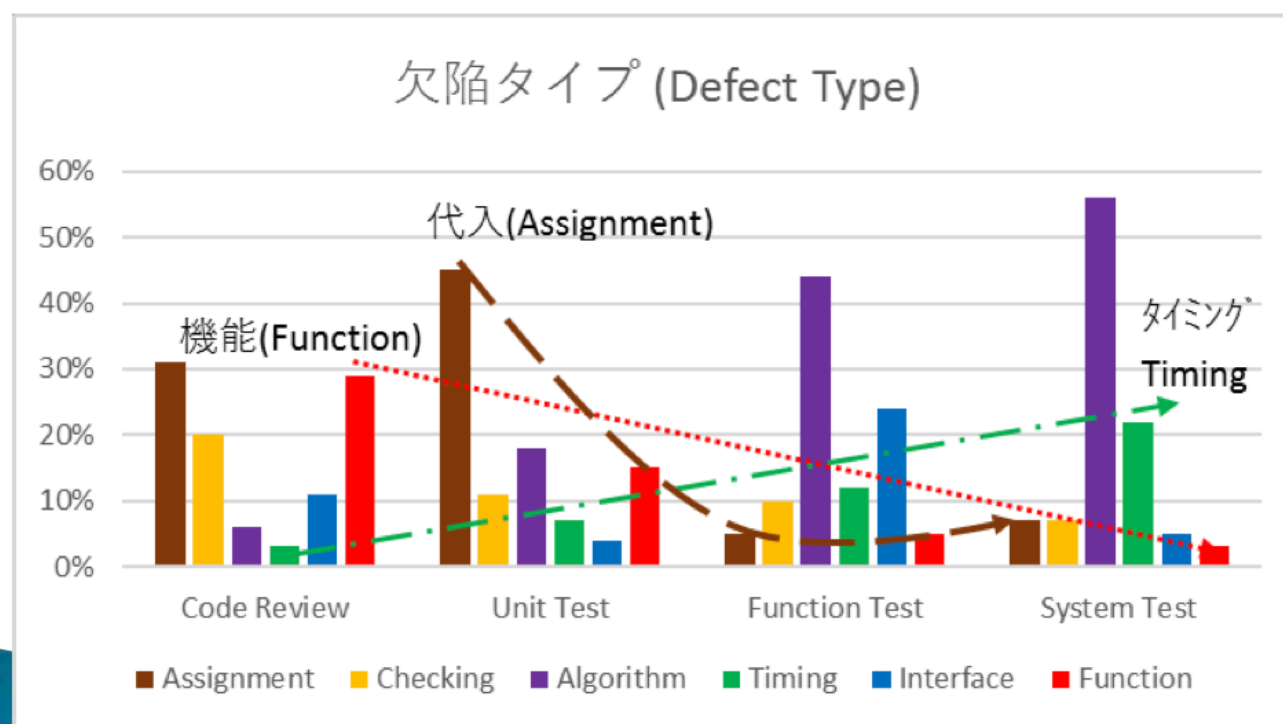
根本原因解析

RCA

(Root Cause Analysis)

- 真因を特定する
- 時間がかかる

Signature



ODCは開発プロセスに着目し、あるべき状態と現在の状態を比較し、どのプロセスに問題があったのか目星をつける。タイムリーに対策を打つことができ、プロセス改善に役立つ。

研究の目的とアプローチ

- 目的

- * リリース前のバグとリリース後のバグの特性を**商用プロジェクトを用いて**調査すること

- アプローチ



- * BGL BNPパリパ(銀行会社)のシステムのそれぞれのバグを抽出, 分析

リサーチクエスチョン

- RQ1：リリース前のバグはどの程度複雑か？
- RQ2：リリース後のバグはどの程度複雑か？
- RQ3：リリース前・リリース後のバグの類似点，相違点は何か？
- RQ4：静的解析ツール(SpotBugs)はリリース前・リリース後のバグの検出にどの程度有効であるか？

バグ修正の分析方法

バグ修正の識別方法

- リリース前のバグ修正
 - * "fix", "defect"などの特定のキーワードを含むコミットメッセージから抽出
- リリース後のバグ修正
 - * 顧客によって報告されたバグに紐づくコミットIDから抽出する

<case1>変更ファイル

- 商用プロジェクトは複数のプログラミング言語によって構成されていることが多い
 - * BGL BNPパリパ(銀行会社)のシステムの場合
 - ▶ **Javaファイル：拡張子がjavaのファイル，今回の分析対象**
 - ▶ その他ソースファイル：JavaScriptなど
 - ▶ 構成ファイル：pom.xmlファイル， Gradleファイルなど
 - ▶ その他ファイル：git.ignoreファイル， mdファイルなど

<case2>パッチサイズ

- 追加, 削除, 変更されたコードの行数を計算する方法
 - * ソースコードの複雑さを評価する最も簡単な方法の1つ

Listing 1. Patch fixing bug CLOSURE-19 from Defects4J

```
1 Node parent = n.getParent();
2 -if (parent.isVar()) {
3 - if (n.hasChildren() && n.getFirstChild().isQualifiedName()) {
4 +if (parent.isVar() &&
5 + n.hasChildren() && n.getFirstChild().isQualifiedName()) {
6 String name = n.getString();
```

2行「変更」とみなす

<case3>チャンク

- 行の追加・削除・変更の組み合わせで構成される箇所

* ファイル内のチャンクの数进行計算(Change Distiller[2])

```
1 + #: assets/js/public/search.js:259 website/views/frontend.py:213
2 + #, python-format
3 + msgid "search.header_title.restaurant %(count)s"
4 + msgstr ""
5 +
1 6 #: assets/js/public/search.js:258 website/views/frontend.py:212
2 7 #, python-format
3 8 msgid "search.header_title.bar %(count)s"
4 9 msgstr ""
5 10
6 - #: assets/js/public/search.js:259 website/views/frontend.py:213
7 - #, python-format
8 - msgid "search.header_title.restaurant %(count)s"
11 + #: assets/js/public/vue/direct-debit-mandate-form.vue:7
12 + msgid "direct_debit_mandate.form.title"
9 13 msgstr ""
10 14
```

<case4>変更パターン

変更パターン	説明
Condition	プログラムの実行フローに影響を与える変更
Statement	呼び出しとステートメントの順序に影響する変更
Comment	コメントまたはJavaDocの変更
Interface	クラスのインタフェースあるいは メソッドのシグネチャの変更
Parameter	メソッドのパラメータと属性の変更
Return type	戻り値の型の追加・削除・変更

<case5>SpotBugs

- FindBugsの後継にあたる静的バグ解析ツール

* 以下のようなViolationを検出可能

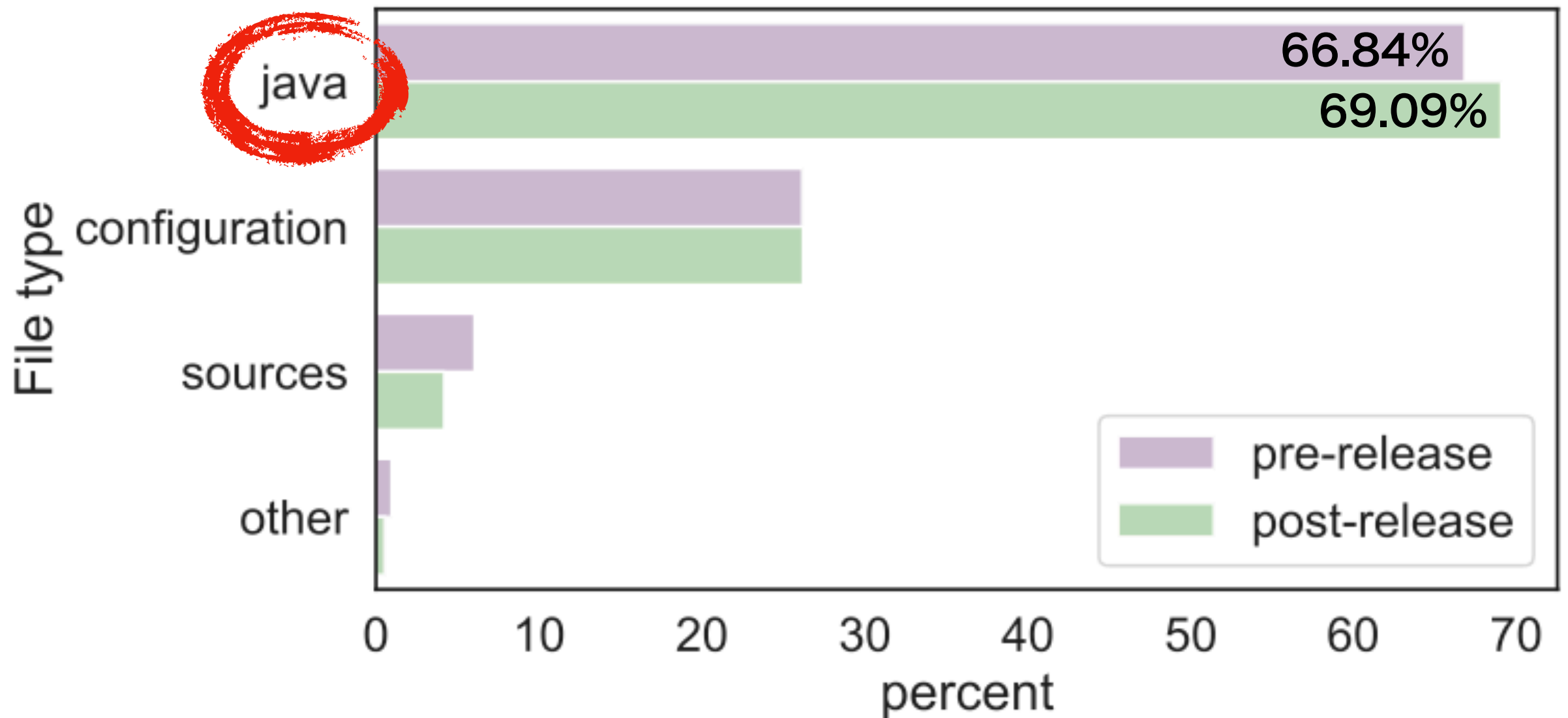
変更パターン	説明
Bad Practice	バグに繋がる恐れのあるコード(Code Smell)
Correctness	予期しない動作に繋がる見かけ上のコーディングの誤り
Malicious code vulnerability	文字エンコードによるもの. 不適切な文字列変換をもたらす
Multithreaded correctness	マルチスレッドの正当性に関するバグ
Performance	非効率な実行につながるようなコード
Security	信頼できない第三者からの攻撃を受ける可能性がある
Style	外部の関係者によるセキュリティのバグ

分析結果

データセット

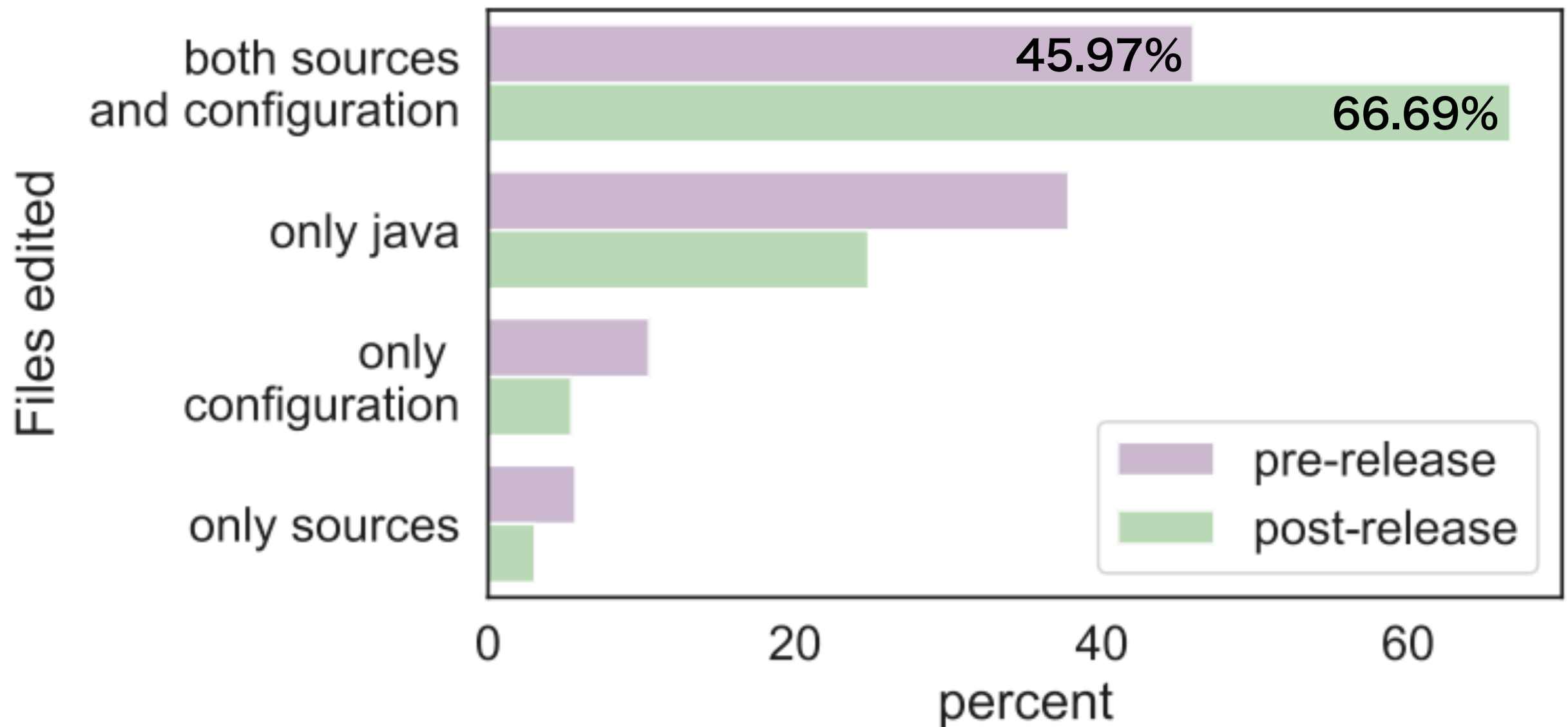
- BGL BNPパリパが開発したJavaプロジェクト37件
- Mockusらの手法[1]によりリリース前とリリース後のバグ修正を分類
 - * リリース前のバグ修正：3623件(99.94%)
 - ▶ "fix", "defect"などの特定のキーワードを含むコミットメッセージから抽出
 - * リリース後のバグ修正：250件(0.06%)
 - ▶ 顧客によって報告されたバグに紐づくコミットIDから抽出する

<case1>ファイル変更



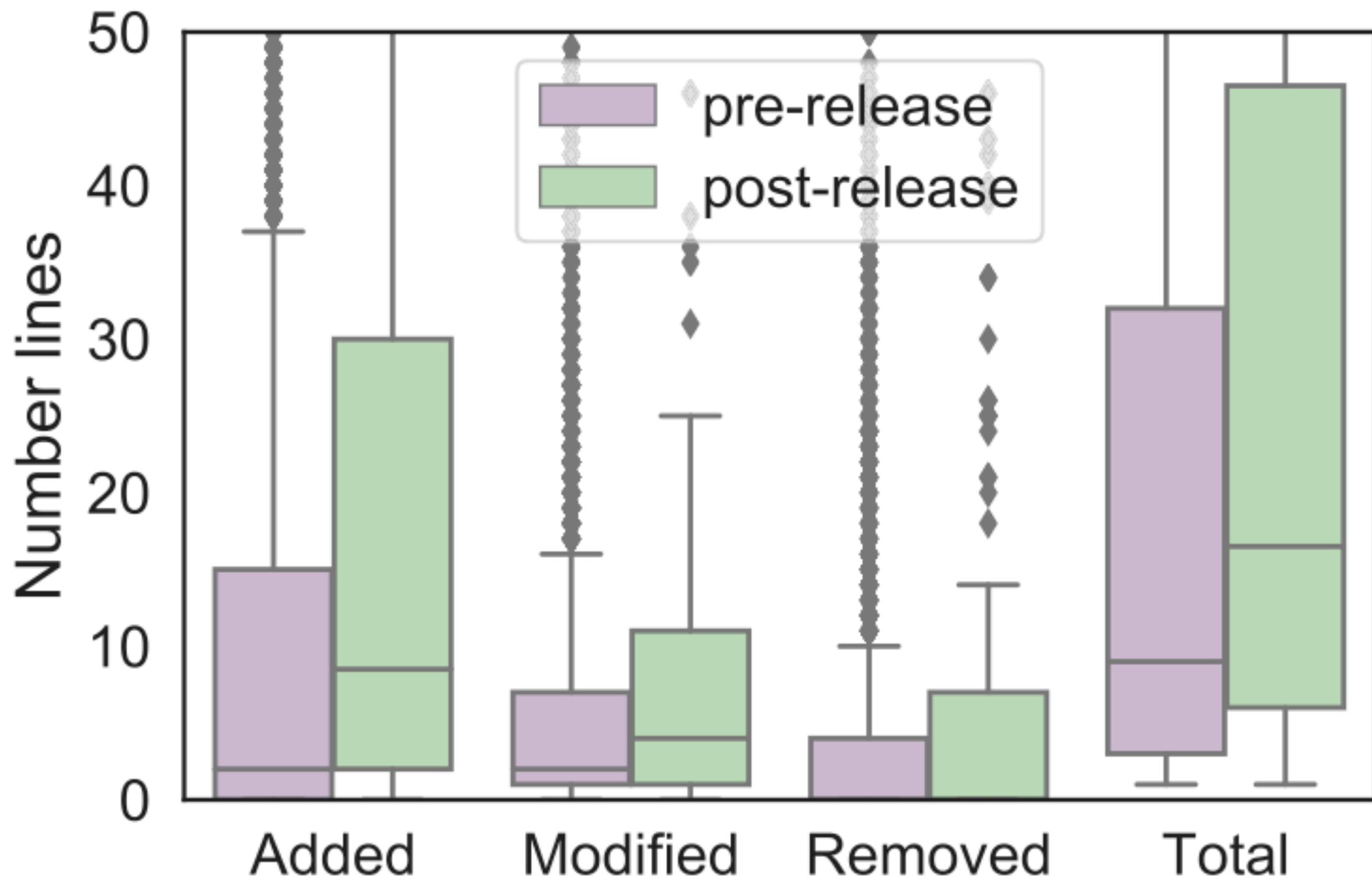
<全てのパッチにおける編集されたファイルの割合>

<case1>ファイル変更

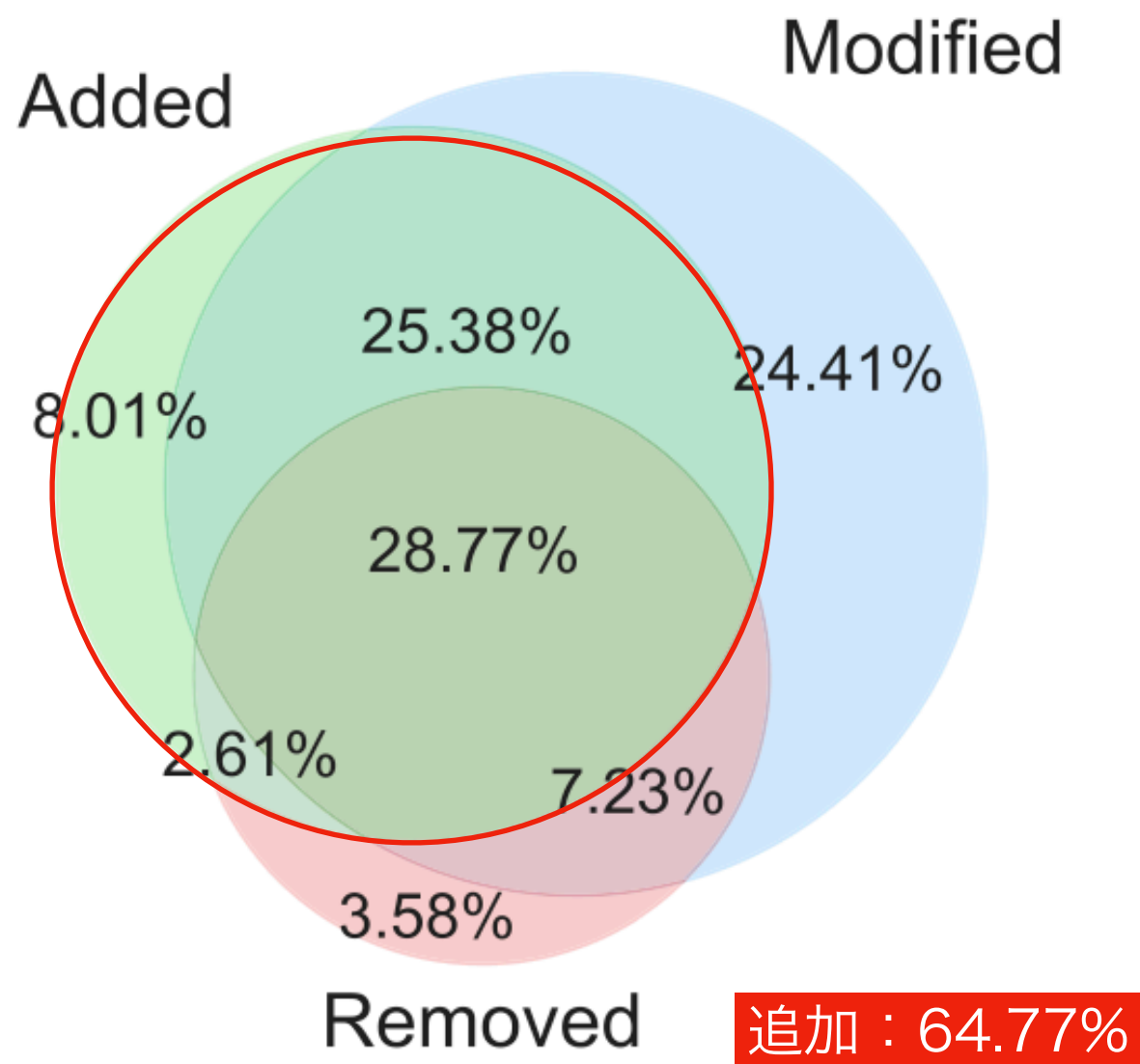


<各パッチにおけるファイル修正の編集箇所>

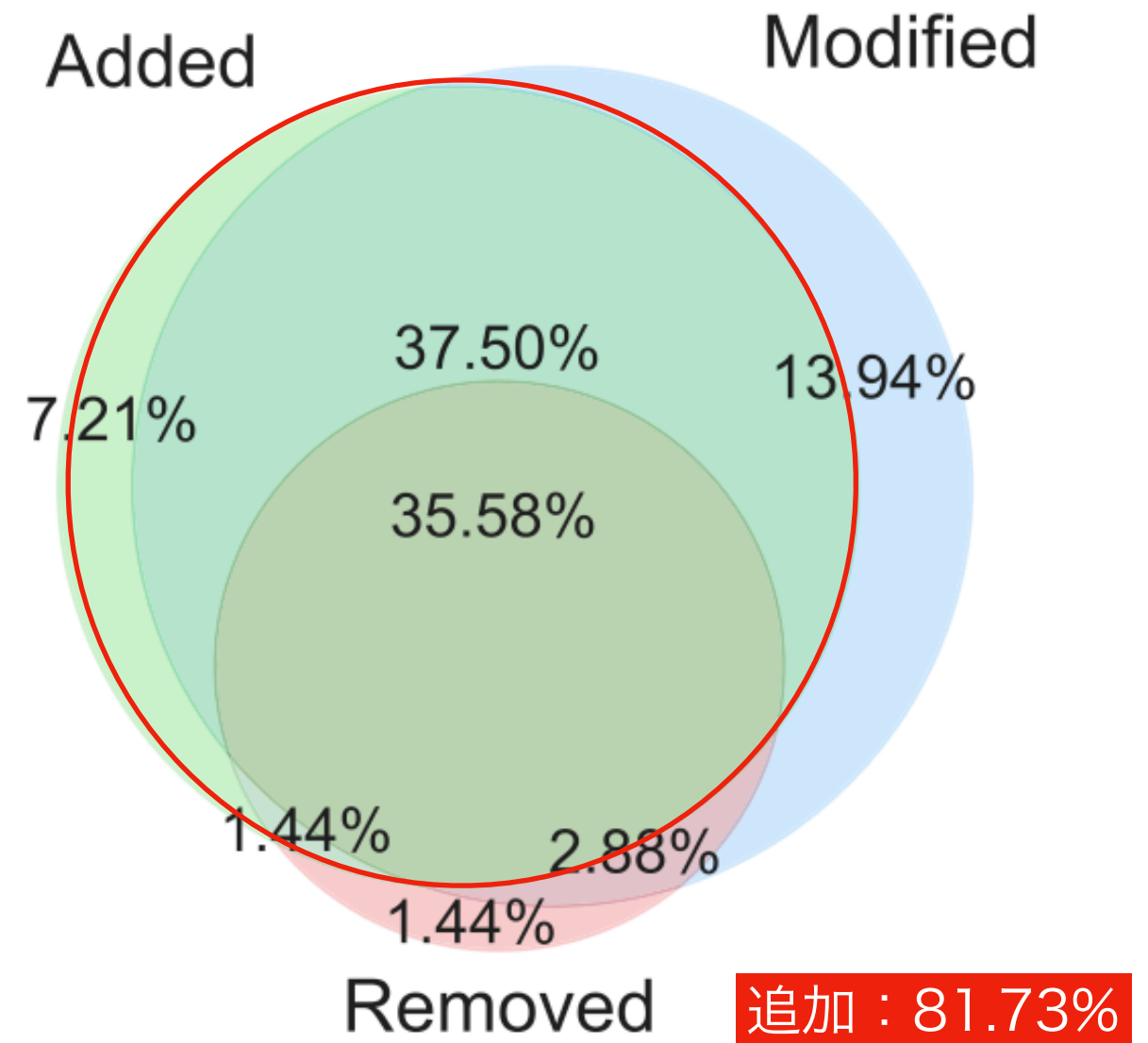
<case2>パッチサイズ



<case2>パッチサイズ

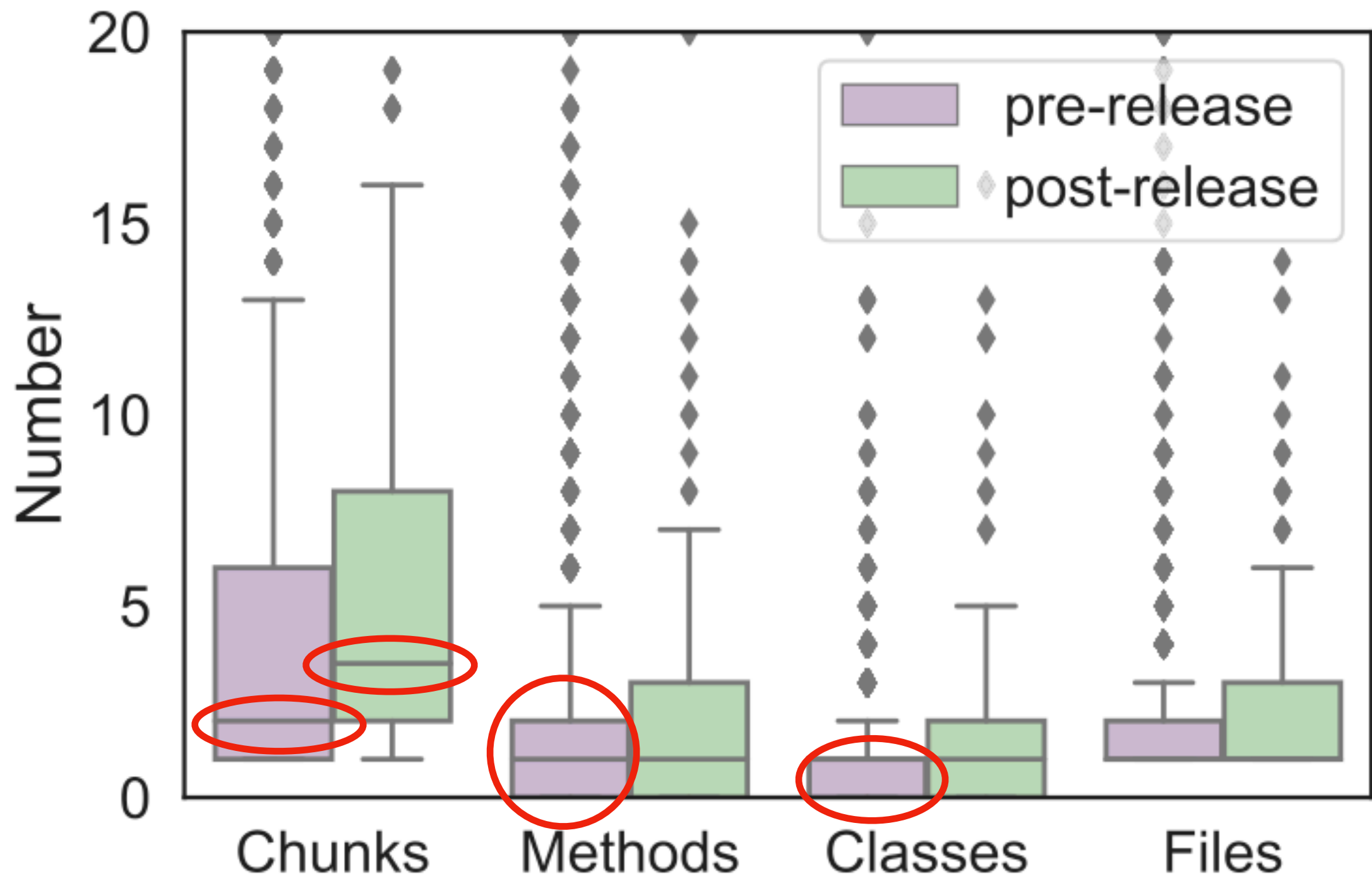


(a) Pre-release patches



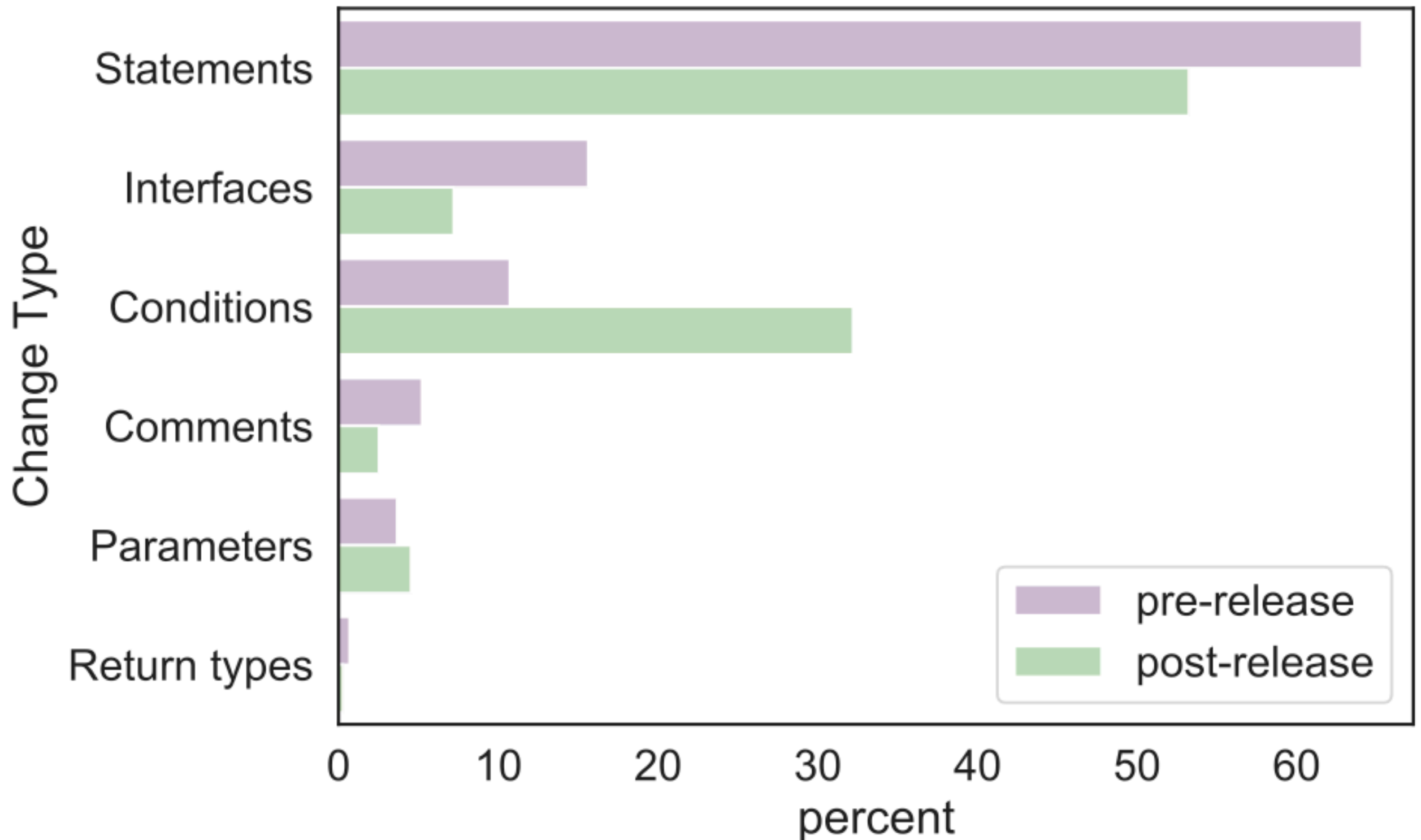
(b) Post-release patches

<case3>チャンク



リリース前のバグ修正の少なくとも25%はクラス・メソッドの変更に関係していない

<case4>変更パターン



<case5>SpotBugs

Violation	Pre-release	Post-release
<i>Bad Practices</i>	3.68%	0.00%
<i>Correctness</i>	2.63%	0.00%
<i>Malicious code vulnerability</i>	1.05%	0.00%
<i>Multithreaded correctness</i>	0.52%	0.00%
<i>Performance</i>	2.37%	0.00%
<i>Security</i>	0.00%	0.00%
<i>Style</i>	3.42%	0.00%
<i>No violation</i>	94.21%	100.00%

まとめ・所感

まとめ

- ソフトウェアのバグを商用的な観点から分析した
 - * リリース前とリリース後のバグの特性を調査
- リリース前とリリース後のバグも修正は主にコードの追加であった
 - * リリース前とリリース後ではチャンクの数が違う
 - * 修正箇所は共通しているが、修正量が大きく異なる
- 静的解析ツールで検出されたリリース前のバグの約5%は修正されないまま放置されていた
 - * 現地点では静的解析ツールがバグ修正を捉えるのは困難な場合もある

所感

- コードの追加が大半の修正であることには驚き
 - * リリース後にバグ修正を特定するのがかなり辛そう
- 日本企業で似たような調査したらどうなるのだろうか？
- プロジェクト内の他の言語でも調査してみたい