

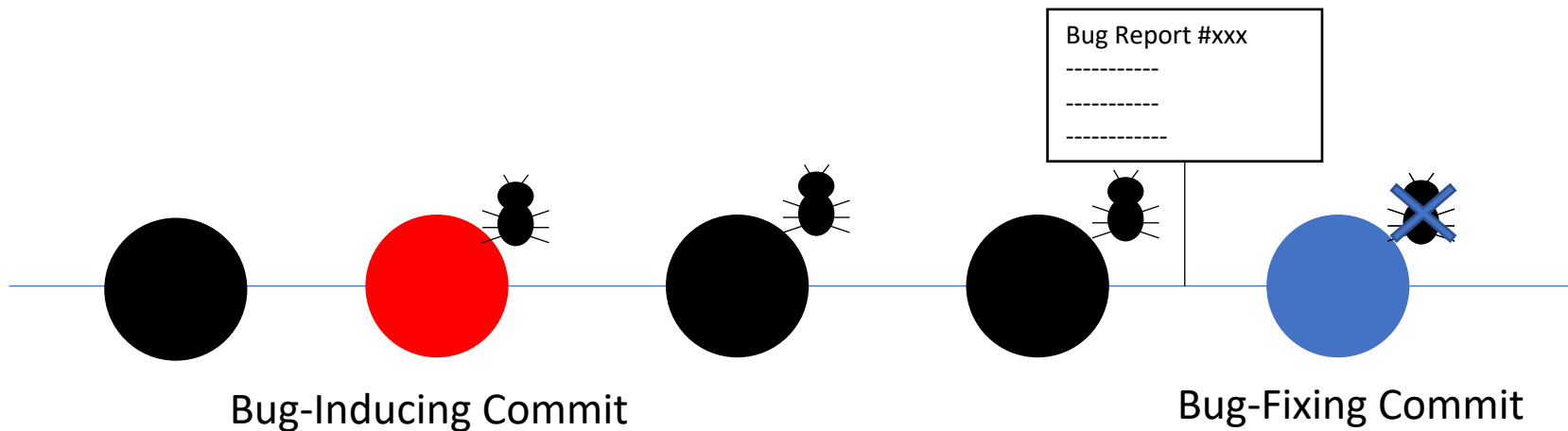
Exploring and Exploiting the Correlations between Bug-Inducing and Bug-Fixing Commits

Ming Wen, Rongxin Wu, Yepang Liu, Yongqiang Tian,
Xuan Xie, Shing-Chi Cheung, Zhendong Su

ESEC/FSE 2019 Proceedings of the 2019 27th
ACM Joint Meeting on European Software
Engineering Conference and Symposium on
the Foundations of Software Engineering
Pages 326-337

Bug-Inducing Commit

- バグが混入したコミット
- バグのデータを利用する研究で必要不可欠



以降からBug-Inducing CommitをBIC, Bug-Fixing CommitをBFCと略す

背景

- **BIC**で混入したバグが**BFC**で正しく修正されているかを明らかにする必要がある
- **BIC**で混入したバグを**BFC**で修正しているかが体系的に評価されていない
- **BIC**はデバッグ技術に有効であるか評価されていない

目的

- **BIC**で混入したバグが**BFC**で修正されているかを評価する
- **BIC**がデバッグ技術で有効かを評価する

リサーチクエスション

1. **BIC**で変更が加えられた部分は**BFC**の同じ箇所で修正されているか
2. **SZZ**アルゴリズムの精度を低下させる原因は何か
 - A) **SZZ**アルゴリズムの不正確さを引き起こす主な原因は何か
 - B) 不正確さによって既存研究の結果に偏りが生じるか
3. **BIC**は自動デバッキング手法の性能を向上させるか

BICとBFCの収集

1. バグレポートから正規表現を用いてBICの候補を収集

- (started with|caused by|introduced by|commit).*?(r*(¥w){6,41}|PROJECT-(¥d){3,7})

2. 目視でBIC候補を絞る

- 以下の例は除かれる
- “The problem is caused by **stack overflow** in SOLR-6213 [5]”

3. BICと関連するBFCを収集する

BICとBFCの収集

Table 1: The Benchmark Dataset Collected in This Study

Project	Description	#Stars	#Files	KLOC	#Changes	Major	Critical	Blocker	Minor	Trivial	#Bugs	#BIC	#BFC
Accumulo	Data Storage and Retrieval	431	2,078	661	9,741	10	4	14	4	1	33	53	36
Ambari	System Administrators for Hadoop	788	3,267	132	22,800	11	8	12	0	0	31	36	38
Lucene	Text Search Engine Library	1,477	7,776	375	29,608	42	6	9	12	0	69	144	71
Hadoop	Distributed System	6,185	10,892	555	18,032	29	11	10	1	2	53	57	53
JackRabbit	Content Repository Management	202	3,158	132	8,571	24	0	2	7	0	33	66	42
Oozie	Web Services for Hadoop	444	1,373	278	2,224	24	6	11	3	0	44	49	45
CoreBench	Projects from GNU Operating System	~	2,317	146	33,433	~	~	~	~	~	70	70	70
Summary		9,527	30,861	2,279	124,409	140	35	58	27	3	333	475	355

#BFC denotes the number of bug-fixing commits; #BIC denotes the number of bug-inducing commits; ~ means the data is not available

RQ1： BICで変更が加えられた部分はBFCの同じ箇所で修正されているか

- 調査内容
 - BICとBFCで同じ部分が修正されているかを調査する
 - ファイル単位で調査
 - ステートメント単位で調査

RQ1： BICで変更が加えられた部分はBFCの同じ箇所で修正されているか

- メトリクスの定義

- $File\ Coverage(b) = \frac{|S_{I_b} \cap S_{F_b}|}{|S_{F_b}|}$

- b :収集されたバグデータ
- S_{I_b} :BICで変更されたファイル
- S_{F_b} :BFCで変更されたファイル

- $Statement\ Coverage(b) = \frac{|L_{FI_b} \cap L_{F_b}|}{|L_{F_b}|}$

- L_{FI_b} :BICとBFCで修正されたステートメント
- L_{F_b} : BFCで修正されたステートメント

File Coverageの調査結果



Figure 1: File Coverage between Bug-Fixing and Bug-Inducing Commits

- 平均は73.2%
- 異なる部分を変更した26.8%の主な理由
 - 設定ファイルやログ関連(17.9%)
 - 不完全な変更(14.2%)

Statement Coverageの調査結果

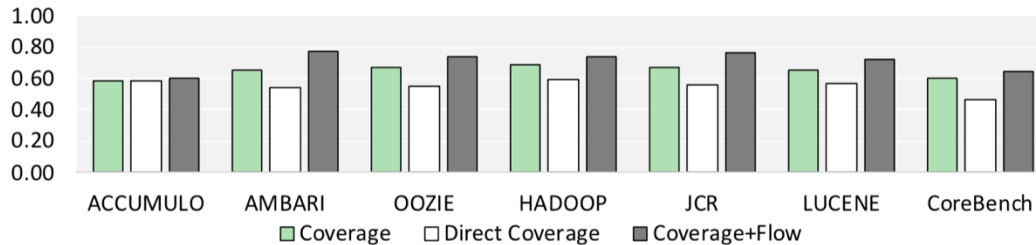


Figure 4: Statement Coverage (with Flow Analysis) and Direct Coverage between Bug-Inducing and Fixing Commits

Coverage : 先述の数式をそのまま使用
Direct Coverage : BICとBFCの間に変更を含まない場合
Coverage+Flow : Coverageに加えてデータフローとコントロールフローを考慮した場合

- Coverageの平均は64.6%
- Direct Coverageの平均は55.1%
- Coverage+Flowの平均は71.1%
- データフローやコントロールフローを考慮するとCoverageが向上することがわかった

RQ2#A：SZZアルゴリズムの不正確さを引き起こす主な原因は何か

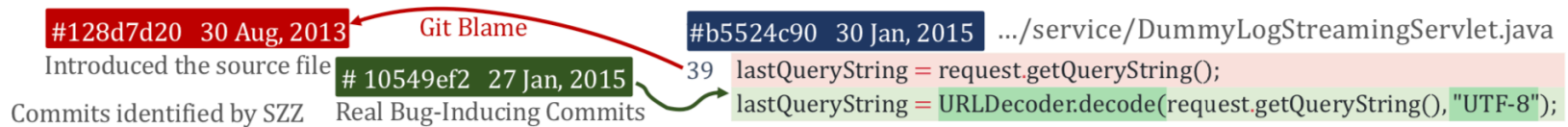


Figure 7: Identifying the Bug-Inducing Commits of OOOIE-2320 by SZZ

- 収集したバグデータに対してSZZアルゴリズムを適用
 - 再現率： 63.7%
 - 適合率： 68.7%
- SZZアルゴリズムがBICと判定したコミットとBFCのfile coverageを計測すると26.8%でありcoverageが低いことが原因と考えられる

RQ2#B：不正確さによって既存研究の結果に偏りが見られるか

Table 2: Bias and Effect Sizes of the Findings Based on Bug-Inducing Commits Revealed by Earlier Studies

ID	Findings	ACCUMULO		AMBARI		LUCENE		HADOOP		JCR		OOZIE		CoreBench	
		p-value	effect	p-value	effect	p-value	effect	p-value	effect	p-value	effect	p-value	effect	p-value	effect
1 ^L	The number of bug-inducing commits per bug [18, 19]	0.000	0.554	0.000	1.117	0.000	0.901	0.000	1.138	0.000	0.796	0.000	1.236	0.000	0.654
2 ^L	The number of inducing source files per bug [28, 31, 67, 68]	0.000	1.087	0.000	0.711	0.000	0.779	0.000	0.615	0.000	0.762	0.000	1.097	0.000	0.755
3 ^U	The time of the bug inducing changes (in terms of the 24 hours) [21]	0.288	0.105	0.252	0.259	0.618	0.042	0.592	0.158	0.860	0.045	0.383	0.014	0.000	0.092
4 ^U	The time of the bug inducing changes (in terms of the 7 weekdays) [54]	0.658	0.098	0.434	0.124	0.727	0.048	0.696	0.062	0.746	0.060	0.025	0.350	0.243	0.010
5 ^G	The experiences of the developers who submitted the buggy commit [21, 28, 67, 68]	0.000	0.661	0.140	0.170	0.000	0.369	0.037	0.292	0.104	0.176	0.002	0.470	0.032	0.069
6 ^L	The life span of software bugs [17, 30]	0.000	1.512	0.000	1.240	0.000	1.434	0.000	1.544	0.000	1.249	0.000	1.509	0.000	1.067
7 ^L	The time scatter among bug-inducing commits for a bug [18]	0.000	1.594	0.000	1.596	0.000	1.493	0.000	1.503	0.000	1.426	0.000	1.571	0.000	1.212
8 ^L	The code churn of added lines of bug-inducing commits [28, 31, 67, 68]	0.000	0.368	0.270	0.238	0.000	0.251	0.000	0.237	0.000	0.350	0.000	0.470	0.266	0.190
9 ^L	The code churn of deleted lines of bug-inducing commits [28, 31, 67, 68]	0.006	0.343	0.252	0.243	0.000	0.272	0.002	0.391	0.000	0.276	0.001	0.301	0.011	0.244
10 ^L	The change entropy of the bug-inducing commits [28, 67, 68]	0.000	0.416	0.145	0.286	0.000	0.458	0.002	0.523	0.000	0.599	0.000	0.693	0.022	0.365

Value **0.000** indicates the *p-value* is less than 0.0001. Emphasis on Cohen's *d* indicates effect size: **large** ($|d| \geq 0.8$), **medium** ($0.5 \leq |d| < 0.8$), **small** ($0.2 \leq |d| < 0.5$) and **negligible** ($|d| < 0.2$). **Bold** values indicate the difference is significant (*p-value* < 0.05). Superscript **L** denotes the findings measured based on I_b , **b** are **L**ess than S_b , **G** denotes **G**reater, and **U** denotes unequal distribution

- SZZアルゴリズムで取得したBICと本研究で収集したBICを用いて既存研究の結果に違いが出るかを調査
- 10個の既存研究のうち8個で違いが見られた

RQ3 : BICは自動デバッグで活用できるか

- Fault Localization

- Spectrum-Based Fault Localization(SBFL)で有効利用できるかを以下のメトリクスを用いて調査

- $$BOOST(s) = \begin{cases} SBFL(s) + 1.0 & s \in \mathcal{A} \wedge s \in L_{IF_b} \\ SBFL(s) + 0.5 & s \in \mathcal{A} \wedge s \notin L_{IF_b} \wedge AL_{IF_b} \\ SBFL(s) & otherwise \end{cases}$$

- L_{IF_b} :BICとBFCで変更が加えられたステートメント
- AL_{IF_b} : L_{IF_b} とデータフローとコントロールフローの影響が及ぶ部分
- $\mathcal{A}$:SBFLで検出されたバグが混入している可能性があるステートメント
- $SBFL(s)$:SBFLで検出できたバグの割合 (0~1までの値になる)

RQ3：BICは自動デバッグで活用できるか

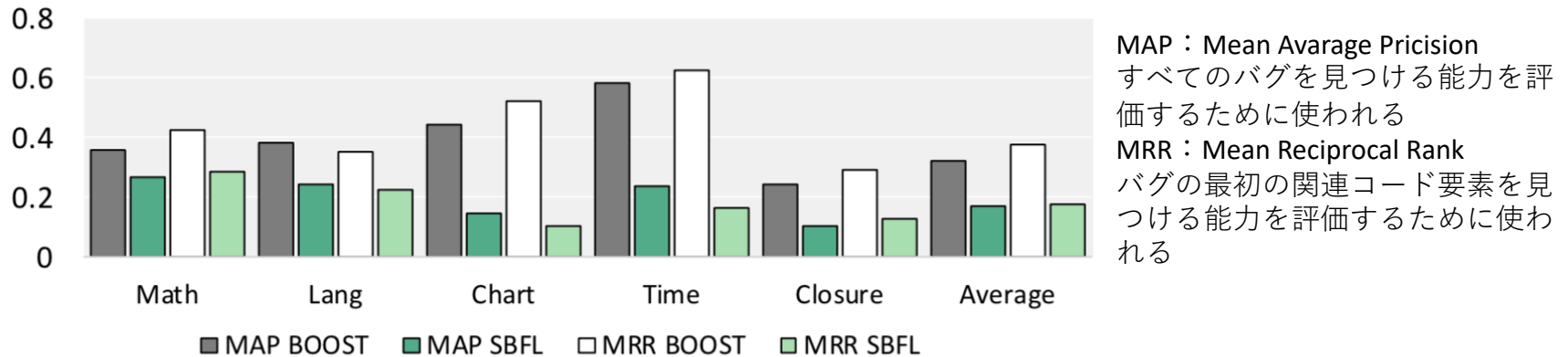


Figure 9: Comparing our Boost Model and SBFL

- MAP, MRR共に既存手法よりも向上していることがわかる
- BICはFLの性能を向上させる可能性が高いことがわかった

RQ3 : BICは自動デバッグで活用できるか

- Automated Program Repair(APR)
 - BICとBFCで加えられた変更操作内容を見て, 自動修正に適用できそうな変更パターンがあるかを調査
- GumTree[1]を用いてBICとBFCで加えられた変更操作を収集
 - GumTree : リビジョン間の差分をASTを用いて得るツール
 - <Operation,Node>といった形で加えられた変更操作を取得することができる

RQ3 : BICは自動デバッグで活用できるか

- Automated Program Repair(APR)

- 以下のメトリクスを測定し変更操作の重複度を測定

- $Action\ Coverage(b) = \frac{|A_{I_b} \cap A_{F_b}|}{|A_{F_b}|}$

- A_{I_b} :BICで加えられた変更操作
- A_{F_b} :BFCで加えられた変更操作

- $Action\ Inverse\ Coverage(b) = \frac{|IA_{I_b} \cap A_{F_b}|}{|A_{F_b}|}$

- IA_{I_b} :BICで加えられた変更がBFCで加えられた変更と逆の操作
- 逆の操作 : <Insert,Node>の逆の操作は<Delete,Node>

RQ3 : BICは自動デバッグで活用できるか

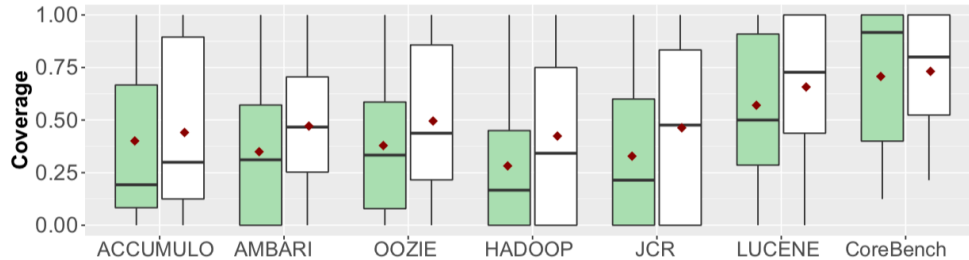


Figure 5: Change Action Coverage Between Bug-Fixing and Bug-Inducing Commits. The Green Bar Denotes Coverage and the White Bar Denotes Inverse Coverage

```
a4c526da .../ jscomp/Compiler.java // Bug Inducing Commit
- 482 CompilerInput previous = inputsById.put(id, input);
+ 482 CompilerInput previous = putCompilerInput(id, input);

- 1272 if (options.dependencyOptions.needsManagement()) {
+ 1272 if (options.dependencyOptions.needsManagement() &&
+ 1273 !options.skipAllPasses &&
+ 1274 options.closurePass) {

0a670cb5 .../ jscomp/Compiler.java // Bug Fixing Commit
1284 if (options.dependencyOptions.needsManagement() &&
- 1285 !options.skipAllPasses &&
1286 options.closurePass) {
```

Figure 10: Bug-Inducing and Fixing Commit of Closure 31

- 重複した操作の平均は43.1%
- 逆の操作が行われた平均は52.6%
- 自動修正に適用する操作として逆の操作が有効となる可能性があることがわかった

まとめ

- **BIC**で混入したバグの6～7割は**BFC**で修正されている
- **SZZ**アルゴリズムの不正確さは**BIC**と**BFC**の変更箇所の重複度が小さいことに起因する
- **BIC**は自動デバッグに有効である可能性があることがわかった

所感

- より精度が高い**BIC**を自動的に収集する手法が必要だと思った
- **BIC**を活用した自動デバッグ手法がこの先開発されるかもしれない