

# Collaborative Bug Finding for Android Apps

Shin Hwei Tan , Southern University of Science and Technology Shenzhen,  
Guangdong Province, China

Ziqiang Li , Southern University of Science and Technology Shenzhen,  
Guangdong Province, China

2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)

発表者：富田恭平

# 概要

---

Androidアプリのクラッシュを検出するために多くの自動テスト生成手法が提案されている

他のアプリの似たようなバグから、テスト中のアプリのバグを発見するアプローチを提案する

# 概要

Androidアプリの場合，多くの類似バグレポートが異なるソフトウェアプロジェクト全体に存在する

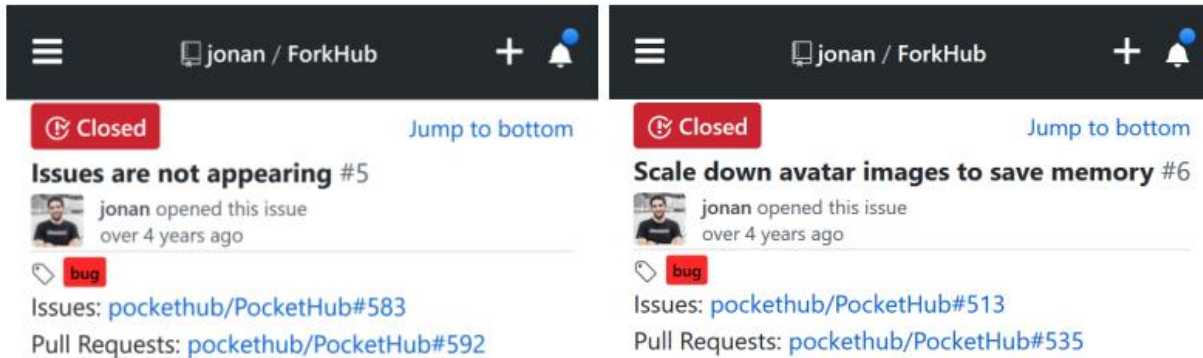


Figure 1: App developers who developed PocketHub and ForkHub found similar bugs across two apps

PocketHubのissueを参照してForkHubで同様のバグを発見

Table 1: Example where there exists one-to-one correspondence for issues in CameraColorPicker (driver) and Gnucash (selected app)

| Titles of issues in CameraColorPicker | Titles of issues in Gnucash                   |
|---------------------------------------|-----------------------------------------------|
| How do I get left top color.          | When an account is edited, its color is lost. |
| publish to F-Droid                    | Publishing on F-Droid                         |
| Make gradlew executable               | Make gradlew executable                       |

CameraColorPickerとGnucashの間でGitHubのissueに1対1の対応が存在

# 概要

---

Androidアプリ間でバグレポートの冗長性を利用して、新しいバグを見つける方法に関する研究はほとんどない

## Collaborative Bug Finding (共同バグ検索)

テスト対象の特定のアプリに特化したテストシナリオを作成するために、Androidアプリ間の類似性を利用する新しいテスト形式

# 概要

---

Collaborative Bug Findingを容易にするために、ペアプログラミングのドライバー・ナビゲーターの役割を呼称として利用する

具体的には、ドライバーとナビゲーター間の相互作用の度合いが異なる3つの設定を設計

# 概要

---

## 1. coder-vs-coders setting

ドライバー（App<sub>A</sub>を開発する開発者A）がバグレポートR<sub>A</sub>を共有し、ナビゲーター（App<sub>B</sub>, ..., App<sub>Z</sub>の開発者）がバグレポートを読んで、R<sub>A</sub>のテストシナリオをApp<sub>B</sub>, ..., App<sub>Z</sub>に適用できるかどうかを検討する

# 概要

---

## 2. coder-vs-manual-issues setting

ドライバーは手動で別のアプリApp<sub>B</sub>（ナビゲーター）からバグレポートを検索し、App<sub>A</sub>に特化したテストシナリオを構築する

# 概要

---

## 3. coder-vs-auto-issues setting

ドライバーはApp<sub>A</sub>をクエリとして**Bugine**に提供

**Bugine**は, App<sub>A</sub>とは別の類似アプリからissueを選択することにより, 関連するGitHubのissueを自動的に識別するソフトウェア

# 概要

## 1. coder-vs-coders setting

Step1: ドライバーがApp<sub>A</sub>のGitHub issue “x”を投稿

Step2: 同組織内のナビゲーターが “x” をApp<sub>A'</sub>に適用できるか考える

Step3: ナビゲーターが “x” を再現した場合、新たなissue “y” を投稿する  
これをペア共有と呼ぶ

x : 最初に共有したissue  
y : 派生したissue

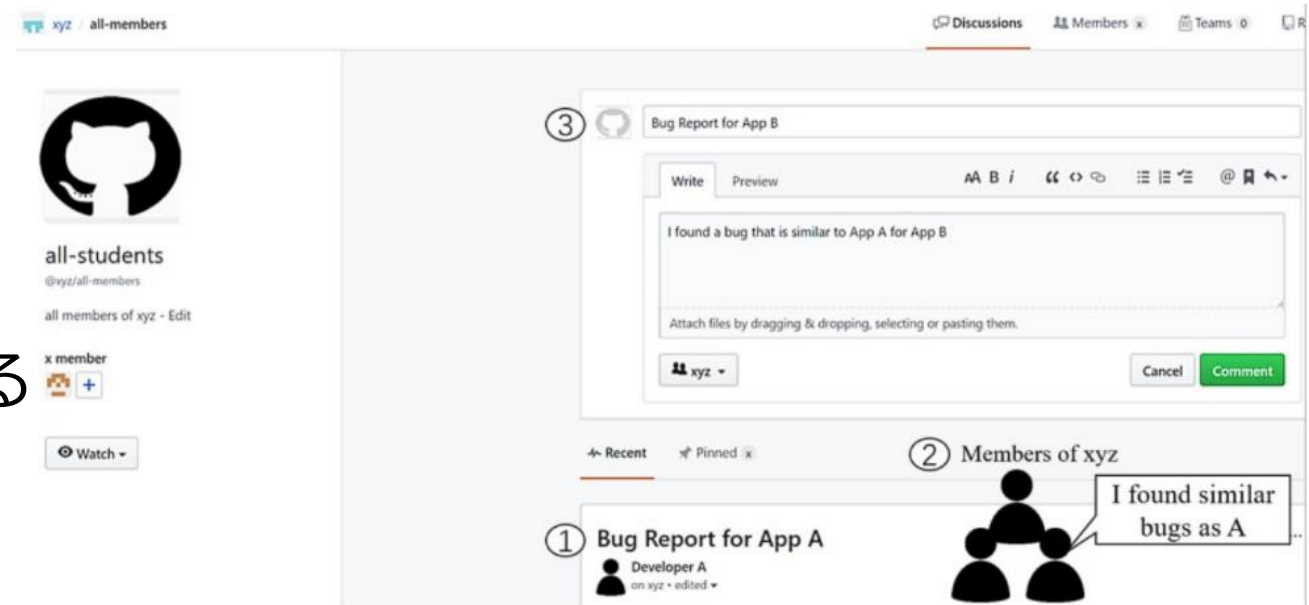


Figure 5: Workflow for the coder-vs-coders setting

# 概要

## 2. coder-vs-manual-issues setting

プログラマーAがApp<sub>A</sub>に影響を与える可能性のあるバグを見つけない

**Step1:** AppAと同じカテゴリのApp<sub>B</sub> or App<sub>A</sub>とは異なるカテゴリのApp<sub>C</sub>を選択

**Step2:** 選択後, AはそれぞれApp<sub>B,C</sub>から5つ以上のissueを選択

**Step3:** 選択したissueのテストシナリオをApp<sub>A</sub>で再現できるかAが検討



Figure 6: Workflow for the coder-vs-manual-issues setting

# RQ

---

Collaborative Bug Findingの適用性を評価する前に, まず  
実現可能性を調査

## **coder-vs-coders settingの実験設定**

Southern University of Science and Technology(SUSTech)に  
29人の学生がいるソフトウェアテストコース(CS409)で,  
各学生に3-4人のチームを作るように依頼  
定めたガイドラインに従い, 20個のアプリが選択され  
た

# RQ1

---

**RQ1a:セメスター全体を通して, githubのディスカッションページにペア共有はいくつあるか?**

**RQ1b:ペア共有で見つかったバグの種類は何か?**

**coder-vs-coders settingの結果**

14人の学生が11のオープンソースAndroidアプリについて29のGitHubのissueを共有  
これらのうち5つのペア共有のみが観察された

RQ1 : ディスカッションページには5つのペア共有がある.  
これらのバグはすべてバージョン更新関係のバグであった.

# RQ2

---

## **coder-vs-manual-issues settingの実験設定**

2つの方法を使用してテストする

(方法1) 最近のコミットのコード変更を網羅する  
(regression-inspired)

(方法2) Collaborative Bug Findingのcoder-vs-manual-issues setting

# RQ2

---

## **coder-vs-manual-issues settingの実験設定**

両方の手法がテストケースの構築時，別の類似アプリに依存しているため，これらの2つの方法を比較

(方法1)

学生は2つのテストケースを作成して，最新のコミットの変更されたコードが正しいことを確認する必要がある

通常の動作と例外的な動作の両方をチェックするテストを設計できるように，2つのテストケースを作成するよう生徒に依頼

# RQ2

---

(方法2)

学生は同じカテゴリ内のアプリから5つのissueを選択し、別のカテゴリアプリから5つのissueを選択する必要がある

# RQ2

---

**RQ2a:**同じカテゴリのアプリから派生したissueと異なるカテゴリのアプリからのissueを参照する場合、関連するGitHubのissueの割合はどのくらいか？

**RQ2b:**ほかのテスト方法と比較して、 Collaborative Bug Findingが発見できるバグはいくつあるか？見つかったバグの種類は何か？

**RQ2c:**学生のフィードバックを考慮し、コードの変更に基づいてテストを構築するのと比較して、 Collaborative Bug Findingの有効性と効率性はどのようなものか？

# RQ2a

---

**RQ2a:**同じカテゴリのアプリから派生したissueと異なるカテゴリのアプリからのissueを参照する場合、関連するGitHubのissueの割合はどのくらいか？

Collaborative Bug Findingのドライバーアプリとして、  
**同じカテゴリのアプリ** と **異なるカテゴリのアプリ** の  
どちらが適しているかを調査

合計で、この課題には27人の学生の提出物があった  
これらの27人の学生は、260のGitHubのissueを選択した  
(同じカテゴリのアプリから131のissue, 異なるカテゴリのアプリから129のissue)

# RQ2a

---

**RQ2a:**同じカテゴリのアプリから派生したissueと異なるカテゴリのアプリからのissueを参照する場合、関連するGitHubのissueの割合はどのくらいか？

各issueについてその種類を分類し、それが次のいずれであるかを測定

## 定義4.1

テスト対象のアプリ $App_{query}$ が与えられた場合、issue “X” に記載されている同様の機能と再現手順が $App_{query}$ に存在し、**予期しない動作を引き起こさない場合**、issue “X”は関連している

## 定義4.2

テスト対象のアプリ $App_{query}$ が与えられた場合、issue “X” に記載されている同様の機能と再現手順が $App_{query}$ に存在し、**予期しない動作が発生した場合**、issue “X”は再現可能である

# RQ2a

**RQ2a:**同じカテゴリのアプリから派生したissueと異なるカテゴリのアプリからのissueを参照する場合、関連するGitHubのissueの割合はどのくらいか？

Type of issues : 学生が選択したissueのタイプ

Relevance : issueが再現可能か関連性があるか

Same Cat : 同じカテゴリのアプリに関連するissueの割合

Diff. Cat : 異なるカテゴリのアプリに関連するissueの割合

**Table 2: The issue relevance comparison between app of same category and app of different category**

| Types of Issues | Relevance    | Same Category | Diff. Category |
|-----------------|--------------|---------------|----------------|
| bug             | reproducible | 21.18%        | 11.96%         |
|                 | relevant     | 32.94%        | 32.61%         |
| feature         | reproducible | 33.33%        | 21.74%         |
|                 | relevant     | 16.67%        | 21.74%         |
| build           | reproducible | 20.00%        | 66.67%         |
|                 | relevant     | 33.33%        | 0.00%          |
| question        | reproducible | 20.00%        | 0.00%          |
|                 | relevant     | 20.00%        | 0.00%          |
| documentation   | reproducible | 50.00%        | 80.00%         |
|                 | relevant     | 0.00%         | 0.00%          |

# RQ2a

---

**RQ2a:**同じカテゴリのアプリから派生したissueと異なるカテゴリのアプリからのissueを参照する場合、関連するGitHubのissueの割合はどのくらいか？

RQ2a:

選択されたバグ関連のissueのうち、  
同じカテゴリのアプリからのissueの21.18%は再現可能で、  
異なるカテゴリのアプリのissueの11.96%は再現可能である

ただし、異なるカテゴリのアプリからバグに関連しないissueを選択すると、他のソフトウェアアーティファクト（ビルドスクリプト、ドキュメントなど）の将来の改善に関する洞察が得られる場合がある。

# RQ2b

**RQ2b:**ほかのテスト方法と比較して, Collaborative Bug Finding で発見できるバグはいくつあるか? 見つかったバグの種類は何か?

全てのテスト手法で発見された  
新しいバグの数

Collaborative Bug Findingは29個の新しいバグのうち, 17個のバグを発見するのに役立つ

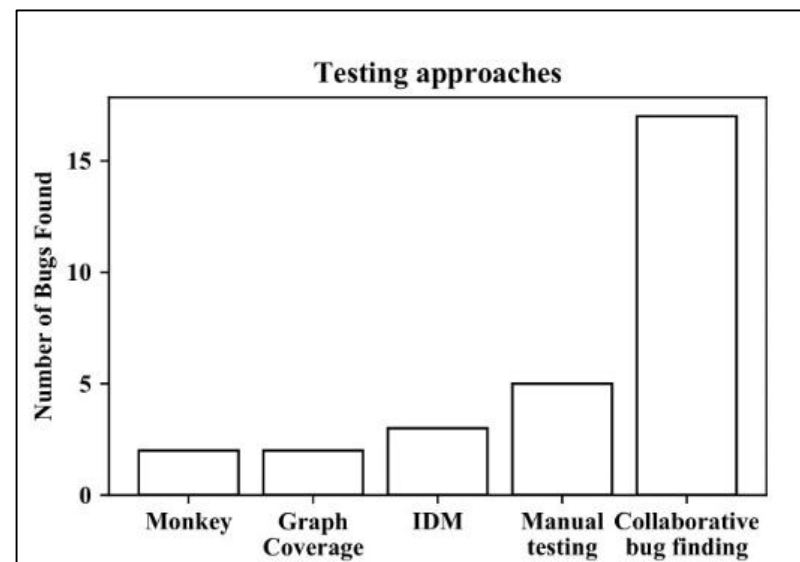


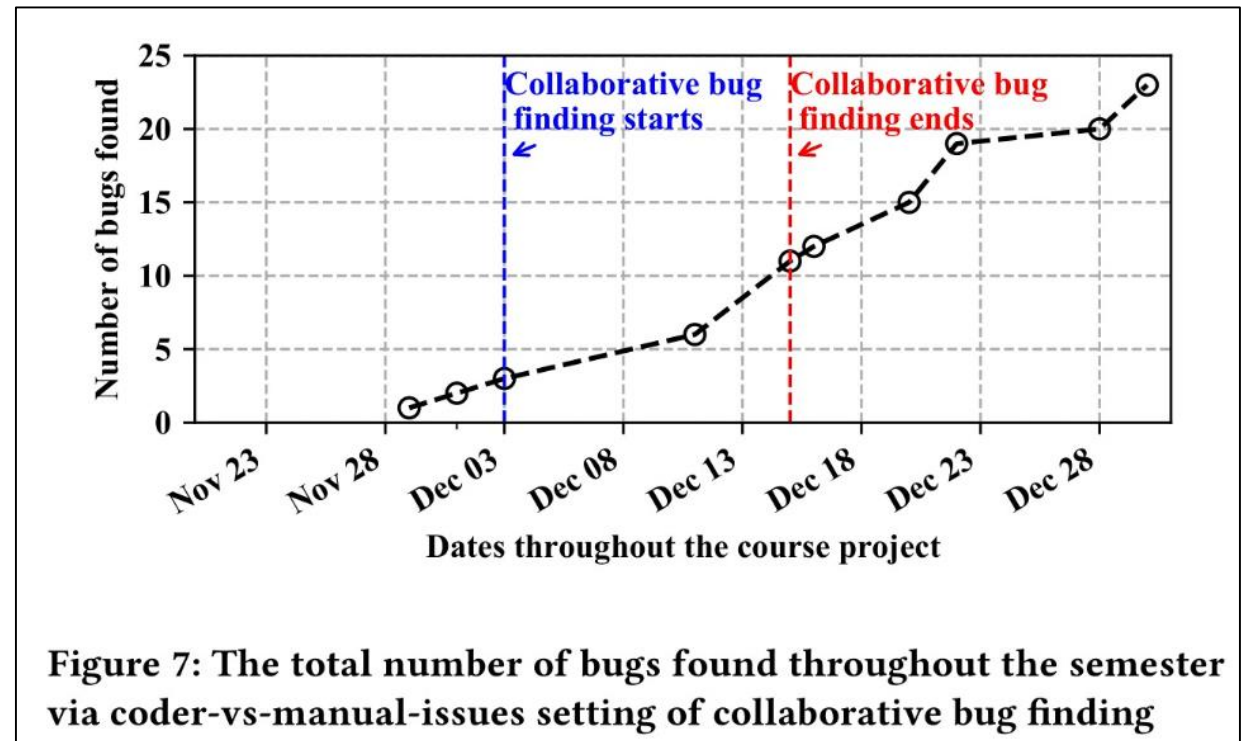
**Figure 8: The number of new bugs discovered through different testing approaches across 11 apps.**

# RQ2b

**RQ2b:**ほかのテスト方法と比較して, Collaborative Bug Finding で発見できるバグはいくつあるか? 見つかったバグの種類は何か?

ディスカッションページで報告されたバグの累積合計数が時間の経過とともにどのように変化するかを分析

マン・ホイットニーのU検定において, Collaborative Bug Findingのアプローチを使用して検出されたバグの数と他のアプローチを使用して検出されたバグの数の差が統計的に有意であった ( $p < 0.05$ )

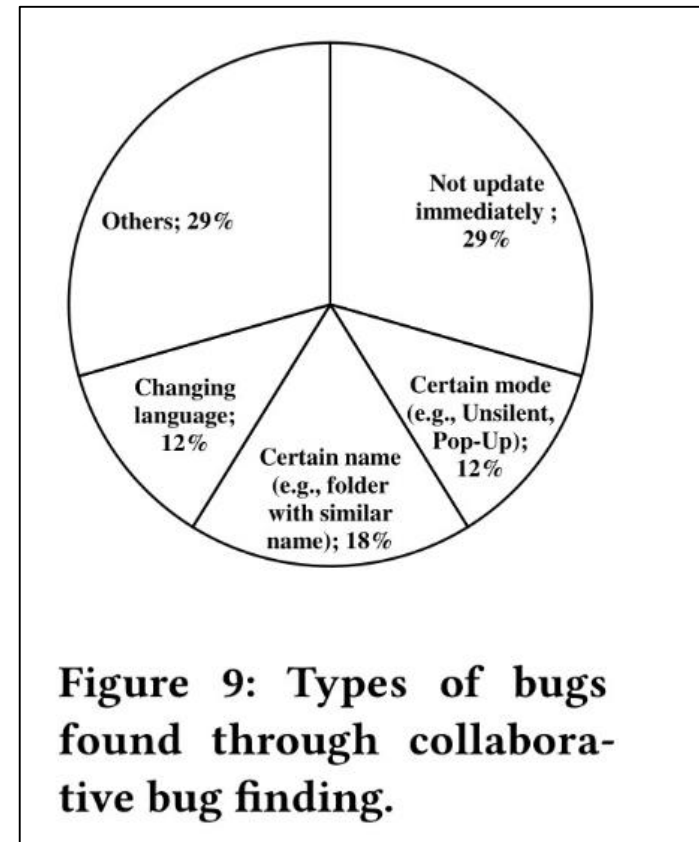


# RQ2b

**RQ2b:**ほかのテスト方法と比較して, Collaborative Bug Finding で発見できるバグはいくつあるか? 見つかったバグの種類は何か?

## 手動分析の結果

- ・バージョン更新関係のバグ(29%)
- ・特定のモードでアプリを開く際に起こるバグ(12%)
- ・ファイルまたはフォルダー作成時, 特定の名前の使用に関連するバグ(18%)
- ・言語の変更によるバグ(12%)



# RQ2b

---

**RQ2b:**ほかのテスト方法と比較して, Collaborative Bug Finding で発見できるバグはいくつあるか? 見つかったバグの種類は何か?

Collaborative Bug Findingのcoders-vs-manual-issue settingによって, 学生が12個の新しいバグを発見した.

Collaborative Bug Findingによって発見されたバグは,  
**多くのアプリに影響する一般的な問題**(バージョン更新関係のバグ(29%) 等)  
と,  
**選択したアプリに影響を与える特定の問題**(others(29%))  
であった

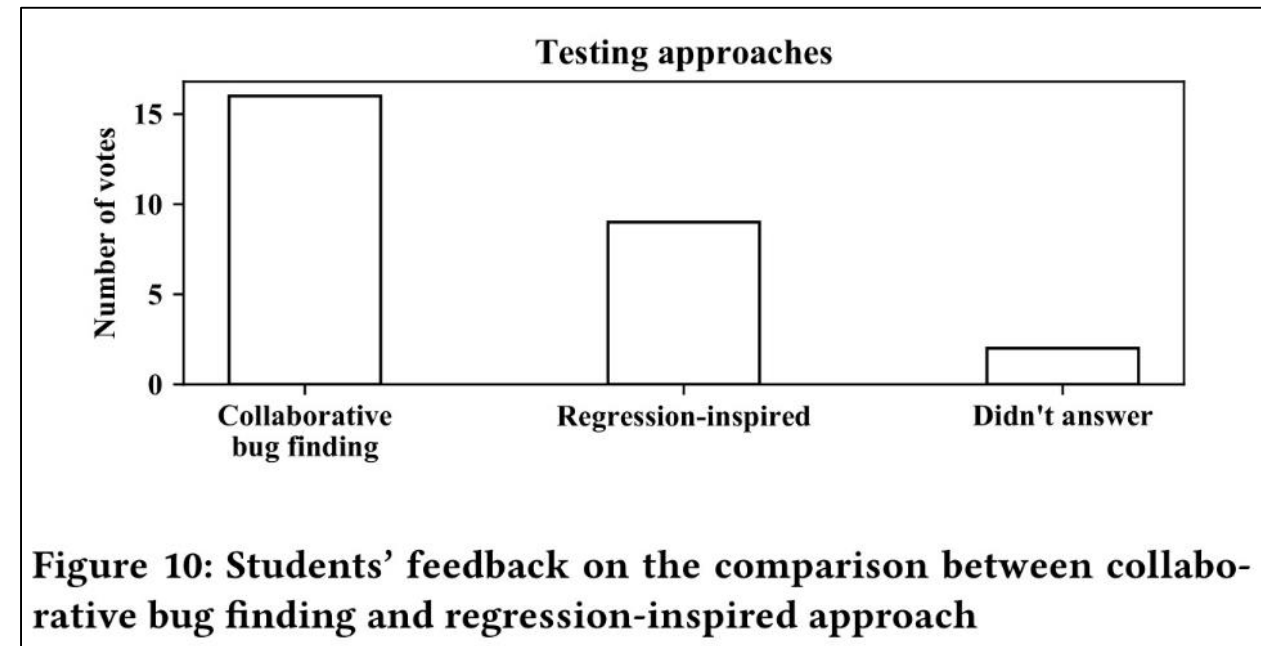
# RQ2c

**RQ2c:**学生のフィードバックを考慮し，コードの変更に基づいてテストを構築するのと比較して，Collaborative Bug Findingの有効性と効率性はどのようなものか？

(方法1)と(方法2)を比較する25の回答を手動で読み，これらのフィードバックを示した

**16人**の学生が新しいバグの発見のために(方法2)を好み，**9人**の学生が(方法1)を好んだ

**16人**のうち，**15人**が**関連するissueの検索に時間がかかる**可能性があることを認めた



# RQ2c

---

**RQ2c:**学生のフィードバックを考慮し、コードの変更に基づいてテストを構築するのと比較して、Collaborative Bug Findingの有効性と効率性はどのようなものか？

25人中16人の学生はregression-inspiredのアプローチよりもCollaborative Bug Findingを好んだ

ただし15人の学生は、関連するissueの検索に時間がかかる可能性があることを認めた

# 問題に対する推奨アルゴリズム

---

Coder-vs-manual-issues settingの弱点に対する学生のフィードバックに基づいて、テスト中のアプリに関連するGitHubのissueを自動的に選択するアプローチであるBugineを提案

Bugineの主な手順

(s1) 自然言語処理を使用して、 $App_{query}$  および  $App_{database}$  の共通のUIコンポーネントをアプリ記述ファイルとして表すことにより、類似のアプリを検索

(s2) (s1) からqueryを自動的に作成して関連するissueを選択

(s3) 品質に基づいてそれらをランク付けする

# Bugineのワークフロー

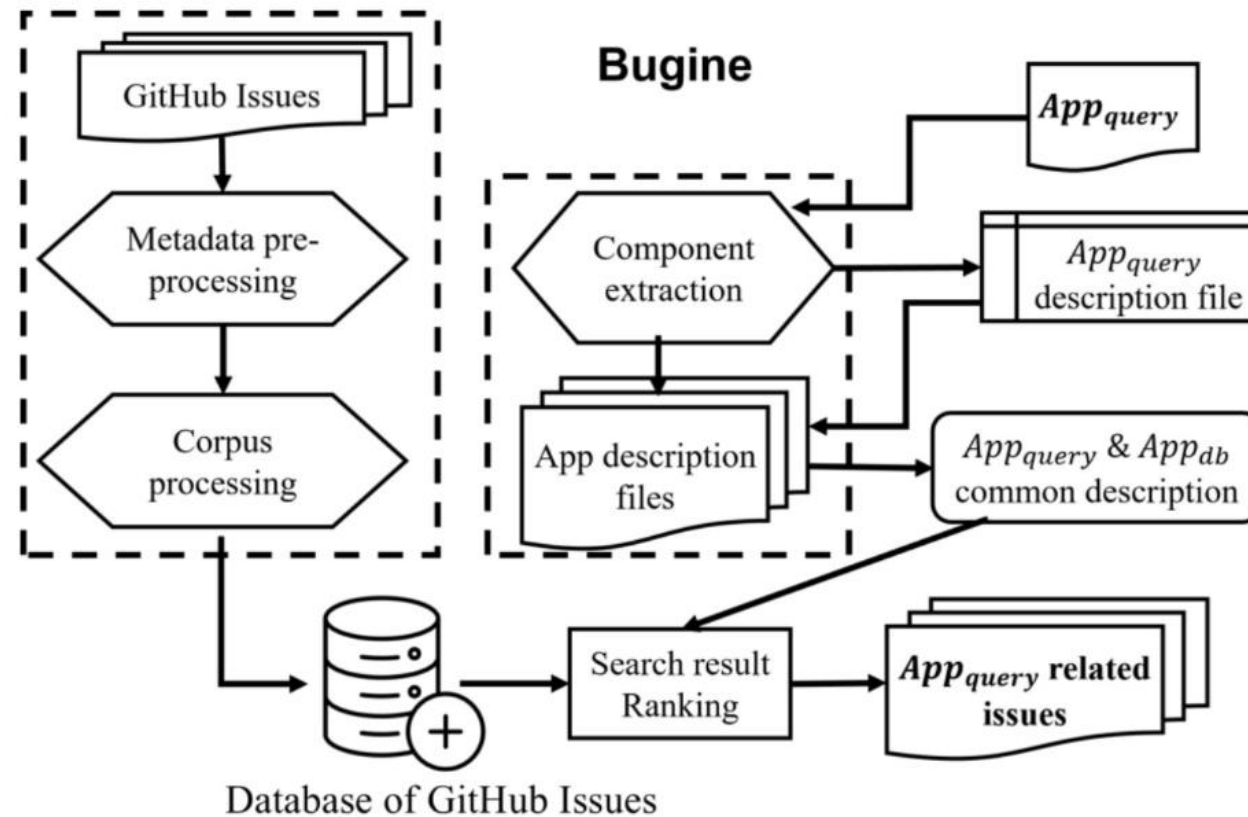


Figure 11: Workflow for our GitHub issues recommendation system

# GitHub Issueのランク付けに用いたメトリック

**Table 4: Factors used in ranking search results**

| <b>Factor</b>   | <b>Description</b>                                                              |
|-----------------|---------------------------------------------------------------------------------|
| Issue length    | Word count of issue body (int)                                                  |
| Issue status    | Closed or opened (binary)                                                       |
| Ref commit SHA  | Commit SHA referenced by issue (binary)                                         |
| Issue reply num | The number of replies that an issue received (int)                              |
| Hit_all         | Find all search keywords in the corpus (binary)                                 |
| Hit_overlap     | Overlap Coefficient between search keywords and corpus (float)                  |
| Hit_hot_words   | Word count of descriptive hot words like <i>reproduce</i> , <i>defect</i> (int) |

# RQ3

---

RQ3a :

関連するGitHubのIssueを推奨する際のBugineの全体的なパフォーマンスはどのくらいか？

RQ3b :

Bugineは実際にはいくつかのバグを見つけることができるか？

# RQ3

## Bugineを5つのオープンソースAndroidアプリで評価

Table 5: Statistics and Evaluation Results of the Android apps used

| App Name            | Category      | KLOC  | #Downloads | Rating | Version No. | #GitHub Stars | #GitHub Issue (closed) | # Bugs Found (new,old) |
|---------------------|---------------|-------|------------|--------|-------------|---------------|------------------------|------------------------|
| Camera-Roll         | Gallery       | 26.00 | 100,000+   | 4.2    | 1.0.6       | 420           | 227(133)               | (11, 0)                |
| PocketHub           | GitHub client | 31.35 | 10,000+    | 3.3    | 0.5.1       | 9429          | 644(526)               | (12, 2)                |
| Simple File Manager | Explorer      | 5.84  | 50,000+    | 4.5    | 6.3.4       | 378           | 189(130)               | (6, 2)                 |
| Zapp                | Broadcast     | 8.41  | N.A.       | N.A.   | 3.2.0       | 60            | 151(137)               | (2, 7)                 |
| Simpletask          | Reminder      | 24.80 | 10,000+    | 4.7    | 10.3.0      | 349           | 821(583)               | (3, 2)                 |

# RQ3

## Bugineの全体的なパフォーマンスの評価

BugineのPrecision @ k  
(Top kアイテムのうち、何割が  
ユーザに嗜好されたか)の結果

結果の範囲は0.1~0.7

Bugineが推奨する上位10のissue  
の中に、少なくとも1つの関連す  
るissueがある

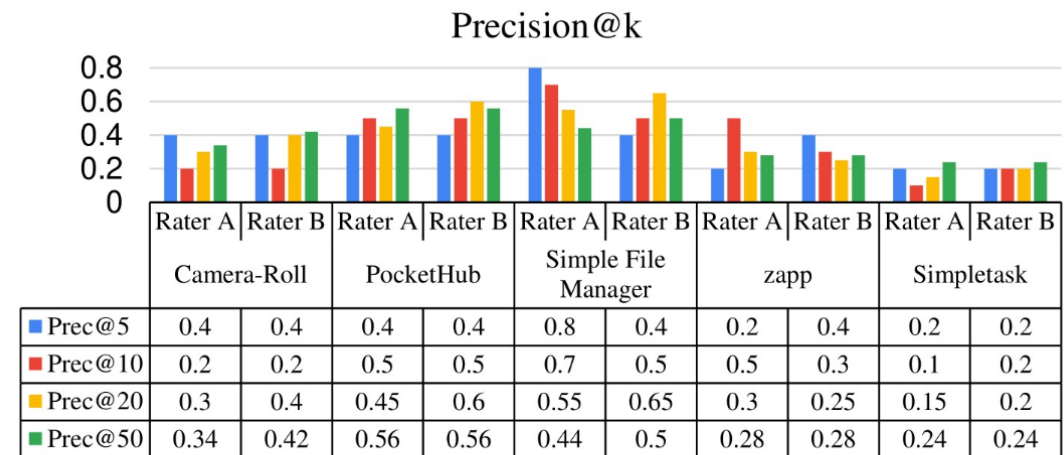


Figure 12: The Prec@k results for BUGINE

# RQ3

## Bugineの全体的なパフォーマンスの評価

BugineのMRR（Mean Reciprocal Rank, 平均相互順位）の結果

BugineのMRR値の範囲は0.34～0.75

バグレポートのソースファイルをランク付けする以前の推奨システムと比較すると、MRR値の範囲は0.2～0.55しかないため、MRR値は比較的高い

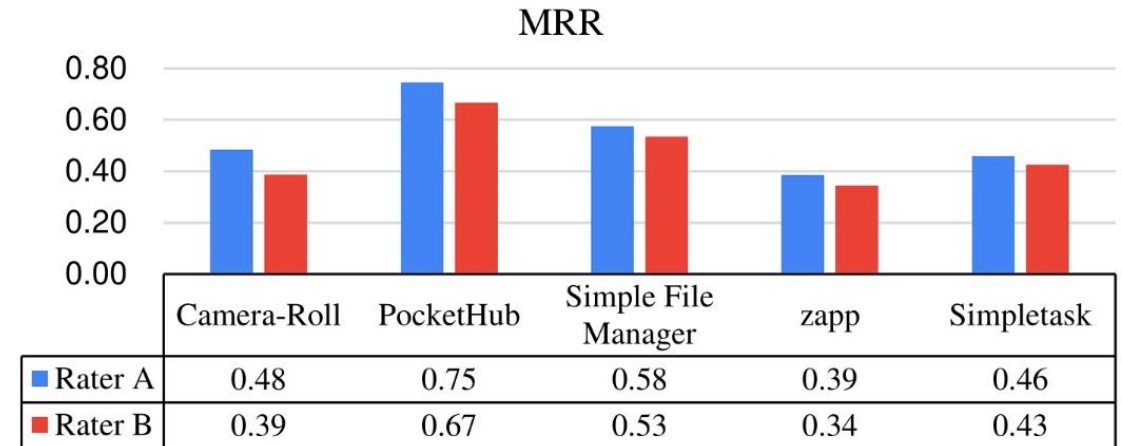


Figure 13: The Mean Reciprocal Rank (MRR) results for BUGINE

# RQ3a

---

RQ3a : 関連するGitHubの問題を推奨する際のBugineの全体的なパフォーマンスはどのくらいか？

RQ3a :

Precision @ kとMRRRの値からわかる通り, Bugineは評価されたすべてのアプリに, 関連するissueを推奨できる

# RQ3a

RQ3b : Bugineは実際にはいくつのバグを見つけることができるか？

Table 5: Statistics and Evaluation Results of the Android apps used

| App Name            | Category      | KLOC  | #Downloads | Rating | Version No. | #GitHub Stars | #GitHub Issue (closed) | # Bugs Found (new,old) |
|---------------------|---------------|-------|------------|--------|-------------|---------------|------------------------|------------------------|
| Camera-Roll         | Gallery       | 26.00 | 100,000+   | 4.2    | 1.0.6       | 420           | 227(133)               | (11, 0)                |
| PocketHub           | GitHub client | 31.35 | 10,000+    | 3.3    | 0.5.1       | 9429          | 644(526)               | (12, 2)                |
| Simple File Manager | Explorer      | 5.84  | 50,000+    | 4.5    | 6.3.4       | 378           | 189(130)               | (6, 2)                 |
| Zapp                | Broadcast     | 8.41  | N.A.       | N.A.   | 3.2.0       | 60            | 151(137)               | (2, 7)                 |
| Simpletask          | Reminder      | 24.80 | 10,000+    | 4.7    | 10.3.0      | 349           | 821(583)               | (3, 2)                 |

評価した5つのアプリすべてで、合計で34個の新しいバグと13個の古いバグを発見

coder-vs-codersおよびcoder-vs-manual-issues setting(20個のアプリを評価したときに29人の学生が17個の新しいバグを発見)と比較して、Bugineは5つのアプリのみの評価で、さらに多くのバグを発見する可能性がある

関連するissueの推奨におけるBugineの有効性を示し、新しいバグの発見につながる

# RQ3a

---

RQ3b : Bugineは実際にはいくつのバグを見つけることができるか？

RQ3b :

Bugineは、評価された5つのアプリ全体で34の新しいバグを発見している

# まとめ

---

ドライバーとナビゲーターの間の様々な程度の相互作用を介して新しいバグの発見を促進するアプローチである、Collaborative Bug Findingを提案

Collaborative Bug Findingを評価したところ、新しいバグの発見に役立つことが分かった

Collaborative Bug Findingで発見されたバグはほとんどクラッシュに関連していないが、既存の自動テストアプローチではクラッシュの発見に重点を置いているため、補完として利用することができる