

テストヘルパー関数をコンテキストとして利用したLIBROの性能向上

辻川大稀

研究テーマ

バグ再現

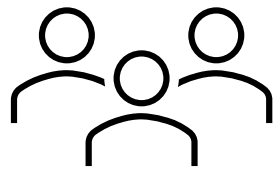
- ソフトウェアの利用者から提供されるバグレポートを用いて、開発者の環境でそのバグを再現する。

自動化するメリット

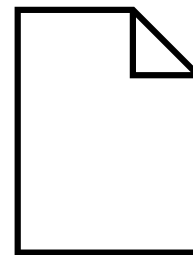
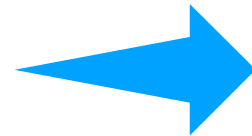
- バグの修正は手元でバグを再現することから始まるが、これには多くの手間と時間がかかる。
- バグの修正が遅れたり、複数回同じバグが発生したりすると、ユーザが離れる原因になる。

研究テーマ

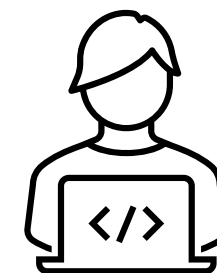
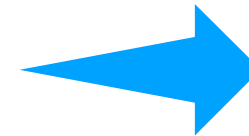
バグ修正の主な手順



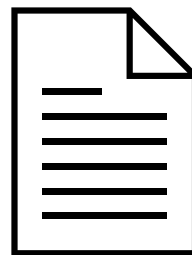
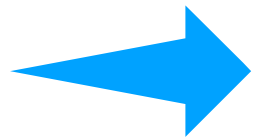
ユーザがバグを発見



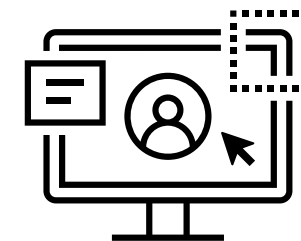
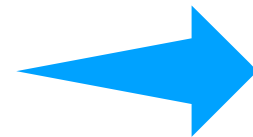
バグレポートとして報告



報告されたバグを再現



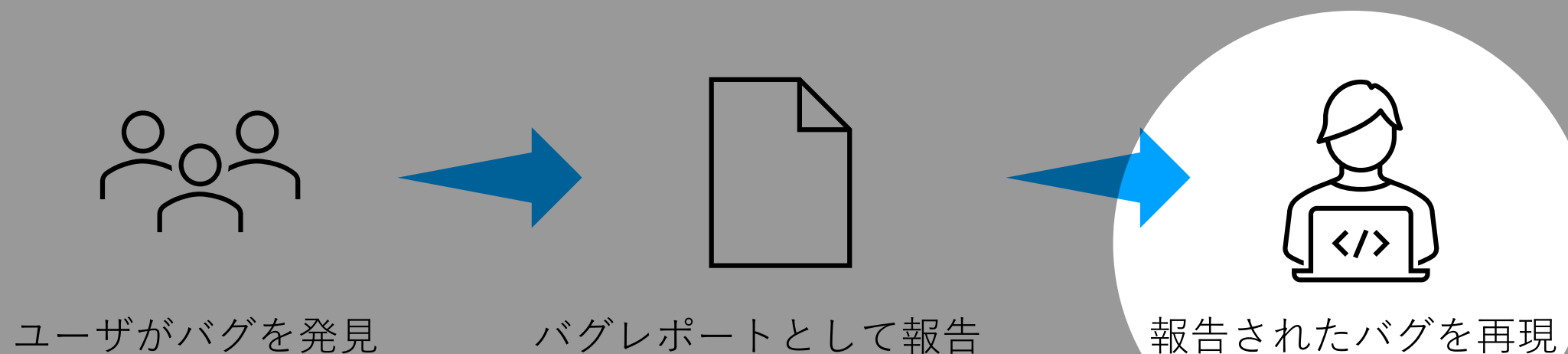
バグの原因を特定して修正



修正完了

研究テーマ

バグ修正の主な手順



手間がかかるが時間はかけたくない

バグの原因を特定して修正

修正完了

先行研究①

アクションリストを出力

- アプリとバグレポートを解析することによって、レポートに記述されているバグを再現する動作手順を出力する。

長所

- かなり高い精度を発揮している。
- バグレポートで欠落した手順を補完することも可能になっている。

課題

- 解析で使用するUTGなどが不十分だと精度が下がる。

先行研究②

テストを生成（本研究の対象）

- LLMを用いてバグレポートからバグ再現テスト（**報告されたバグによってエラーが出るテスト**）を自動生成する。

長所

- バグレポートから直接テストを生成する。
- 学習にそこまで力を注がず一定の精度を発揮した。

課題

- ユーザー定義関数への対応が不十分。
- 外部ファイルや設定ファイルが必要なものは再現できない。
- 実用化するにはまだ精度が低い。

LIBRO

LIBROとは . . .

LLM Induced Bug ReprOductionの略

バグレポートから，バグを再現するテストを生成する.

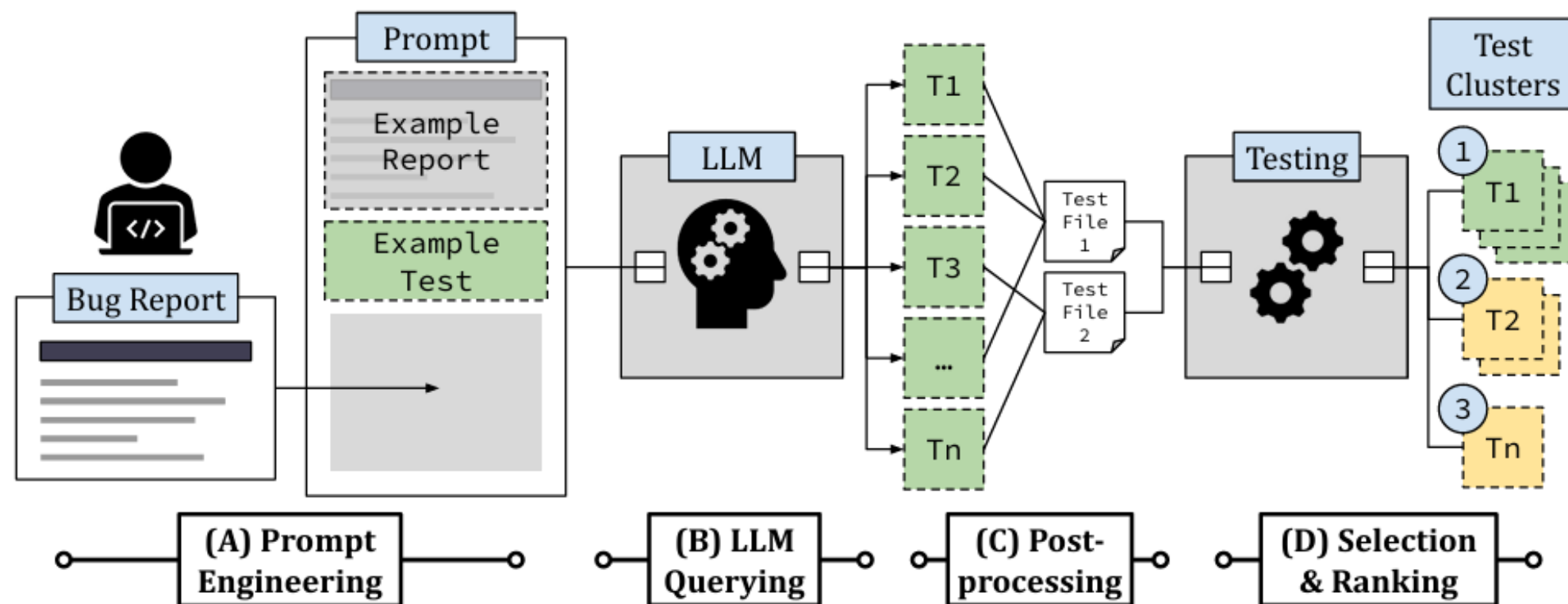


Fig. 1: Overview of LIBRO

LIBRO (再現実験)

LIBROとは . . .

LLM Induced Bug ReprOductionの略

バグレポートから、バグを再現するテストを生成する.

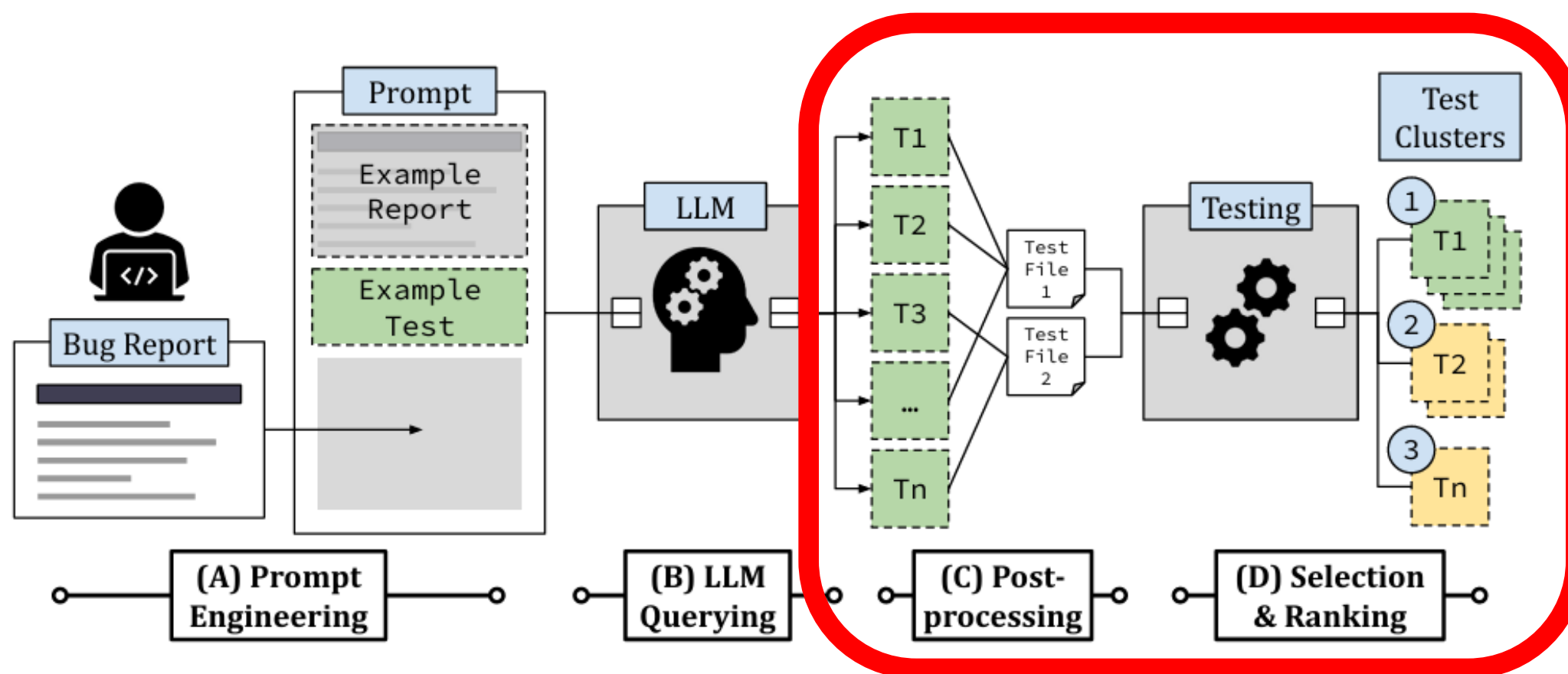


Fig. 1: Overview of LIBRO

テスト生成後のみ再現

再現実験

実行時間

- テストの実行フェーズで184時間（論文の説明では100時間程度）かかった.

成功率

- 再現実験では31.3%(254/812)
- 論文の中では33.5%(251/750)

評価指標	論文の結果 (Table VI)	今回の再現結果	差
acc@1 (精度)	149件 (0.43)	141件 (0.42)	-8件
acc@3 (精度)	184件 (0.53)	178件 (0.53)	-6件
acc@5 (精度)	199件 (0.57)	194件 (0.57)	-5件
提案対象となったバグ数	350件	338件	-12件
うち、成功したバグ数 (精度)	219件 (0.63)	212件 (0.63)	-7件

再現実験

プロジェクトごと

- 全体的に成功率は下がったものの、誤差の範囲内であった。
- Closureは極端に低かった。
(独特なテスト構造を持つという記述)

プロジェクト名	成功率 (論文)	成功率 (今回)	差
Gson	63.6% (7/11)	100.0% (11/11)	▲ 36.4%
Chart	71.4% (5/7)	71.4% (5/7)	± 0.0%
Time	68.4% (13/19)	68.4% (13/19)	± 0.0%
Jsoup	60.9% (56/92)	59.8% (55/92)	▼ 1.1%
Codec	55.6% (10/18)	55.6% (10/18)	± 0.0%
Cli	48.3% (14/29)	55.2% (16/29)	▲ 6.9%
Mockito	7.7% (1/13)	53.8% (7/13)	▲ 46.1%
Lang	73.0% (46/63)	49.2% (31/63)	▼ 23.8%
Math	41.3% (43/104)	43.3% (45/104)	▲ 2.0%
Csv	37.5% (6/16)	37.5% (6/16)	± 0.0%
JacksonCore	33.3% (8/24)	33.3% (8/24)	± 0.0%
JacksonXml	33.3% (2/6)	33.3% (2/6)	± 0.0%
JacksonDatabind	28.0% (30/107)	30.8% (33/107)	▲ 2.8%
Collections	25.0% (1/4)	25.0% (1/4)	± 0.0%
JXPath	15.8% (3/19)	15.8% (3/19)	± 0.0%
Compress	8.7% (4/46)	8.7% (4/46)	± 0.0%
Closure	1.2% (2/172)	2.3% (4/172)	▲ 1.1%
総成功数	251件	242件	-9件

再現実験

失敗した原因

- ヘルパー関数が要因のものが最多

失敗原因	手動分析での件数 (論文, 40件中)	推定件数 (論文, 499 件中)	推定件数 (今回, 558 件中)
ヘルパー関数が必要だった	13件	約162件	約181件
バグ報告書の品質が低い	11件	約137件	約153件
LLMが期待される動作を誤認した	8件	約100件	約112件
外部ファイルへの依存があった	6件	約75件	約84件
LLMの生成長が不十分だった	3件	約37件	約42件

研究趣旨

ヘルパー関数がわかれば本当に性能が向上するのか？

- 著者によると最も大きな原因がヘルパー関数によるもの.
- LIBROはバグ報告といくつかのサンプルのみを与えられるため、プロジェクト固有の知識をほとんど持たない.
- ヘルパー関数はテストをスムーズに行うためだけに作成・使用される関数であるため、これがわかると妥当なテストを生成できる可能性が上がる. (予想)

研究設計

データ抽出

- 再現実験により，失敗例が抽出できたのでそのうち何件かを抽出する。（当面の目標は50件）

ヘルパー関数を手作業で抽出

- 設けた基準に基づいて，ヘルパー関数を特定する．

プロンプト設計

- 抽出したヘルパー関数を利用するためのプロンプトを設計．

実行

- 設計したプロンプトを用いてテストを生成，その先も実行し，性能を検証する．

抽出する条件

実際に使用されたバグ再現テストで呼び出されている.

- 実際に使用されたという客観的に最も有力な情報.

バグ報告よりも前に作成されている.

- バグを修正する際に開発者が作成した関数は、LIBROに与えることができない.

深さ（レイヤー）が2以内

- 理想的な関数を求めるとキリがないため、現実的なボーダーラインを定める.

除外する条件

具体的な処理を含んでいない

- 関数を呼び出すことが主目的の関数（ラッパー関数）を与えても、テスト生成に役立てることができない。（後述）

```
private void fold(String js, String expected) {  
    test(js, expected);  
}
```

レイヤーが深すぎる

- ラッパー関数が呼び出している関数がラッパー関数であることや、複数回オーバーロードされているパターンが存在。

```
public void test(String js, String expected) {  
    test(js, expected, (DiagnosticType) null);  
}
```

```
public void test(String js, String expected,  
    DiagnosticType error, DiagnosticType warning) {  
    test(externsInputs, js, expected, error, warning, description: null);  
}
```

除外する条件

バグレポートで報告されていた再現テストがそのまま実装されている。

- 明らかにヘルパー関数の知識不足が原因でないと言える。

抽出・除外の方針

- この関数を与えていれば成功していたかもしれない、かつ、失敗した原因がヘルパー関数の知識不足かもしれない、かつ、将来自動的に抽出することができるかもしれない。

実行

プロンプト作成の際の注意点

- JAVAの仕組み上、ヘルパー関数を与えたとしても挿入されたクラスから呼び出せない可能性がある。

→与えたヘルパー関数を呼び出して使用させるのではなく、必要に応じてそこに含まれる処理を再び記述するように指示する。

補足

```
public void testFoldComparison3() {
    fold(js: "x = !1 == !0", expected: "x = false");

    fold(js: "x = !0 == !0", expected: "x = true");
    fold(js: "x = !1 == !1", expected: "x = true");
    fold(js: "x = !1 == null", expected: "x = false");
    fold(js: "x = !1 == !0", expected: "x = false");
    fold(js: "x = !0 == null", expected: "x = false");

    fold(js: "!0 == !0", expected: "true");
    fold(js: "!1 == null", expected: "false");
    fold(js: "!1 == !0", expected: "false");
    fold(js: "!0 == null", expected: "false");

    fold(js: "x = !0 === !0", expected: "x = true");
    fold(js: "x = !1 === !1", expected: "x = true");
    fold(js: "x = !1 === null", expected: "x = false");
    fold(js: "x = !1 === !0", expected: "x = false");
    fold(js: "x = !0 === null", expected: "x = false");

    fold(js: "!0 === !0", expected: "true");
    fold(js: "!1 === null", expected: "false");
    fold(js: "!1 === !0", expected: "false");
    fold(js: "!0 === null", expected: "false");
}
```









```
if (getName().endsWith("/")) {  
    if (!isPaxHeader() && !isGlobalPaxHeader() && getName().endsWith("/")) {
```

```
s276150@DESKTOP-19GHAVJ:~$ cd ~/d4j_analysis/Compress_38_buggy  
s276150@DESKTOP-19GHAVJ:~/d4j_analysis/Compress_38_buggy$ defects4j test  
Running ant (compile.tests)..... OK  
Running ant (run.dev.tests)..... OK  
Failing tests: 1  
- org.apache.commons.compress.archivers.tar.TarArchiveInputStreamTest::survivesPaxHeaderWithNameEndingInSlash  
s276150@DESKTOP-19GHAVJ:~/d4j_analysis/Compress_38_buggy$ cd ~/d4j_analysis/Compress_38_fixed  
s276150@DESKTOP-19GHAVJ:~/d4j_analysis/Compress_38_fixed$ defects4j test  
Running ant (compile.tests)..... OK  
Running ant (run.dev.tests)..... OK  
Failing tests: 0
```

```
@Test  
public void survivesPaxHeaderWithNameEndingInSlash() throws Exception {  
    final TarArchiveInputStream is = getTestStream("/COMPRESS-356.tar");  
    try {  
        final TarArchiveEntry entry = is.getNextTarEntry();  
        assertEquals("package/package.json", entry.getName());  
        assertNull(is.getNextTarEntry());  
    } finally {  
        is.close();  
    }  
}  
  
private TarArchiveInputStream getTestStream(final String name) {  
    return new TarArchiveInputStream(  
        TarArchiveInputStreamTest.class.getResourceAsStream(name));  
}
```

改善案

| <https://arxiv.org/abs/2502.01821#:~:text=This%20paper%20investigates%20automated%20BRT%20generation%20within%20an,industry%20bugs%20extracted%20from%20Google%27s%20internal%20issue%20tracker.>