

自動生成されたパッチが与える コーディング規約違反の分析

発表者：野口 優輝

背景

| ソフトウェア開発におけるデバッグ

- デバッグ：バグを特定し、適切に修正を行なっていく
- 開発者がかける労力では半数以上を占める^[1]



自動化

自動プログラム修正手法を提案

| 自動プログラム修正(Automatic Program Repair)

- バグの特定と適用を自動化するアプローチ
- パッチを適用することで、プログラムを自動的に修復

[1]肥後 芳樹, 堀田 圭佑, 楠木 真二, 自動プログラム修正の修正可能バグ数に関する考察

パッチ

プログラムを修正，改善するための更新ファイル

```
--- a/src/main/java/org/apache/commons/lang3/math/NumberUtils.java  
+++ b/src/main/java/org/apache/commons/lang3/math/NumberUtils.java  
@@ -464,11 +464,20 @@ public class NumberUtils {
```

パッチを適用前後のパス

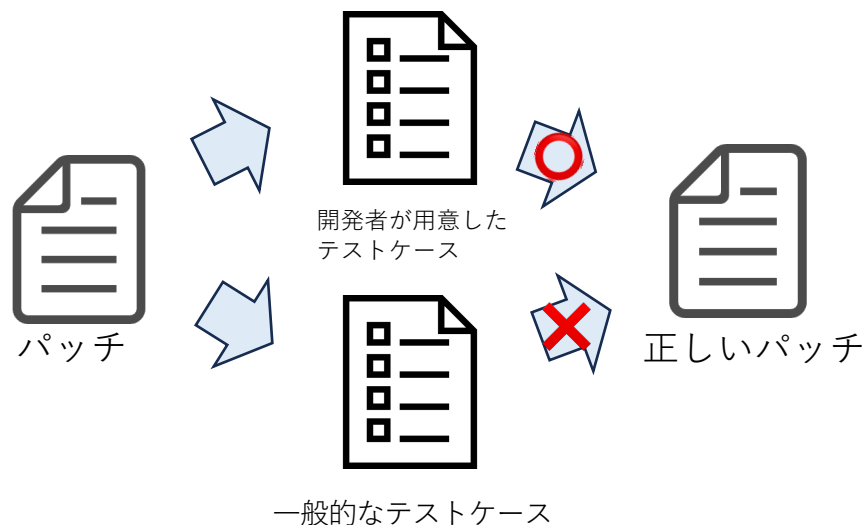
```
    }  
    if (pfxLen > 0) { // we have a hex number  
+        char firstSigDigit = 0; // strip leading zeroes  
+        for(int i = pfxLen; i < str.length(); i++) {  
+            firstSigDigit = str.charAt(i);  
+            if (firstSigDigit == '0') { // count leading zeroes  
+                pfxLen++;  
+            } else {  
+                break;  
+            }  
+        }  
        final int hexDigits = str.length() - pfxLen;  
-        if (hexDigits > 16) { // too many for Long  
+        if (hexDigits > 16 || (hexDigits == 16 && firstSigDigit > '7')) { // too many for Long  
            return createBigInteger(str);  
        }  
-        if (hexDigits > 8) { // too many for an int  
+        if (hexDigits > 8 || (hexDigits == 8 && firstSigDigit > '7')) { // too many for an int  
            return createLong(str);  
        }  
        return createInteger(str);
```

パッチファイルの例

パッチテキスト

自動プログラム修正での問題点

パッチのオーバーフィット



オーバーフィット

開発者が用意したテストケースは通過するが、一般的なテストケースでは脆弱、不十分で問題が発生すること [2]

パッチを適用した際のコーディング規約

- 開発者の期待している、コードになっているか
→ コーディング規約に従っていない場合、開発に不具合が生じる可能性がある

[2] Shangwen Wang*, Ming Wen*†, Bo Lin, Hongjun Wu, Yihao Qin, Deqing Zou†‡, Xiaoguang Mao (2020)
Automated Patch Correctness Assessment: How Far are We?

コーディング規約

| ソフトウェア開発におけるコードの書き方の定義

- 開発者のコードへの理解
- 静的プログラム解析により検査
→本研究ではcheckstyleを使用

| 違反での不具合^[3]

- ソフトウェアの可読性や保守性が低下
→開発者間でのコードの理解が低下



ソフトウェアの品質が下がる

^[3]末次 健太郎, 名倉 正剛, 高田 眞吾, 尾原 秀登, 角 幸一郎, 浅原 明広, ossプロジェクトを対象にしたコーディング規約違反発生状況の分析

目的とアプローチ

| 目的

- 自動生成されたパッチを適用することで、プロジェクトのコーディング規約に沿ったコードになっているかを分析する

| アプローチ

- パッチを適用する前のコミットから、ファイルを取得
- 自動生成されたパッチを適用
- checkstyleを使って、コーディング規約を調べる

check style

| Javaのソースコードがコーディング規約に即しているかを判定するための静的なツール

- インデントサイズ, 変数の付け方, 空白のスペースなどを判定

| xmlファイルを用いて, ルールをプロジェクトごとに決める

- 適用したいモジュールを<module>タグで記述
- <module>にパラメータを渡す場合は, <property>タグを使用

```
<module name="Checker">
  <module name="FileLength">
    <property name="max" value="5"/>
  </module>
</module>
```

check style 例

データセット

| 32のAPRツールによって生成されたパッチ^[4]

- checkstyle.xmlを含むLang, Math, Timeプロジェクト
- オーバーフィットしない正しいパッチ
→ Lang : 123個, Math : 231個, Time : 21個

| APRツール^[4]

3sFix [9]
ACS [81]
ARJA [91]
CapGen [77]
Cardumen [55]
CoCoNut [53]
ConFix [34]
DeepRepair [78]

DynaMoth [13]
ELIXIR [65]
FixMiner [36]
GenPat [29]
GenProgA [91]
HDRRepair [40]
Hercules [66]
JAID [7]

jGenProg2015 [12]
jKali [54]
jMutRepair [54]
KaliA [91]
kPAR [45]
LSRepair [48]
Nopol2015 [12]
PraPR [21]

RSRepairA [91]
SequenceR [8]
SimFix [30]
SketchFix [26]
SOFix [51]
ssFix [79]
TBar [47]
VFix [83]

[4] HAOYE TIAN, YINGHUA LI, WEIGUO PIAN, and ABDOUL KADER KABORÉ, University of Luxembourg, Luxembourg, KUI LIU, Software Engineering Application Technology Lab, Huawei, China ANDREW HABIB, JACQUES KLEIN, and TEGAWENDÉ F. BISSYANDÉ, University of Luxembourg, Luxembourg (2022), Predicting Patch Correctness Based on the Similarity of Failing Test Cases

進捗

| 現在までの進捗

- パッチを適用するためのベースとなるjavaファイルを取得
- 実際にパッチ適用
 - パッチ適用部分にcheckstyleでエラーが発見された.
(Langプロジェクトで58/123個にエラー)

| 今後

- checkstyleでのエラー分析
 - それぞれのAPRがどのような生成をしているのか
- Math, Timeプロジェクトでの検証