Towards an automated approach for bug fix pattern detection

Fernanda Madeiral\textsuperscript{1}  Thomas Durieux\textsuperscript{2}  Victor Sobreira\textsuperscript{1}  Marcelo Maia\textsuperscript{1}

\textsuperscript{1}Federal University of Uberlândia, Brazil
\textsuperscript{2}INRIA & University of Lille, France
Datasets of bugs
Datasets of bugs
Datasets of bugs

+  

diff =

```java
+ if (markers == null) {
+   return false;
+ }

boolean removed = markers;
if (removed && notify) {
```


Datasets of bugs

No knowledge on the composition of them
Dissection of a Bug Dataset: Anatomy of 395 Patches from Defects4J

Victor Sobreira*, Thomas Durieux†, Fernanda Madeiral*, Martin Monperrus‡, and Marcelo de Almeida Maia*
*Federal University of Uberlândia, Brazil, {victor, fernanda.madeiral, marcelo.maia}@ufu.br
†INRIA & University of Lille, France, thomas.durieux@inria.fr
‡KTH Royal Institute of Technology, Sweden, martin.monperrus@csc.kth.se

Abstract—Well-designed and publicly available datasets of bugs are an invaluable asset to advance research fields such as fault localization and program repair as they allow directly and fairly comparison between competing techniques and also the replication of experiments. These datasets need to be deeply understood by researchers: the answer for questions like “which bugs can my technique handle?” and “for which bugs is my technique effective?” depends on the comprehension of properties related to bugs and their patches. However, such properties are usually not included in the datasets, and there is still no widely adopted methodology for characterizing bugs and patches. In this work, we deeply study 395 patches of the Defects4J dataset. Quantitative properties (patch size and spreading) were automatically extracted, whereas qualitative ones (repair actions

We focus on the analysis of Defects4J [14], a dataset containing 395 real bugs collected from six open-source Java projects. Although extensively used in recent research on fault localization [17], [18], [19] and program repair [20], [21], [8], Defects4J does not come with fine-grained information about bugs and their patches. We contribute to Defects4J with the extraction and study of both quantitative (e.g. metrics) and qualitative properties (e.g. patterns) regarding patches. This new data is very valuable to 1) interpret past published results based on Defects4J under the light of the extracted properties; 2) provide and guide future research using Defects4J with fine-grained information; 3) understand the representativeness of
Dissection of a Bug Dataset: Anatomy of 395 Patches from Defects4J

Victor Sobreira*, Thomas Durieux†, Fernanda Madeiral*, Martin Monperrus‡, and Marcelo de Almeida Maia*
*Federal University of Uberlândia, Brazil, {victor, fernanda.madeiral, marcelo.maia}@ufu.br
†INRIA & University of Lille, France, thomas.durieux@inria.fr
‡KTH Royal Institute of Technology, Sweden, martin.monperrus@csc.kth.se

Abstract—Well-designed and publicly available datasets of bugs are an invaluable asset to advance research fields such as fault localization and program repair as they allow directly and fairly comparison between competing techniques and also the replication of experiments. These datasets need to be deeply understood by researchers: the answer for questions like “which bugs can my technique handle?” and “for which bugs is my technique effective?” depends on the comprehension of properties related to bugs and their patches. However, such properties are usually not included in the datasets, and there is still no widely adopted methodology for characterizing bugs and patches. In this work, we deeply study 395 patches of the Defects4J dataset. Quantitative properties (patch size and spreading) were automatically extracted, whereas qualitative ones (repair actions

We focus on the analysis of Defects4J [14], a dataset containing 395 real bugs collected from six open-source Java projects. Although extensively used in recent research on fault localization [17], [18], [19] and program repair [20], [21], [8], Defects4J does not come with fine-grained information about bugs and their patches. We contribute to Defects4J with the extraction and study of both quantitative (e.g. metrics) and qualitative properties (e.g. patterns) regarding patches. This new data is very valuable to 1) interpret past published results based on Defects4J under the light of the extracted properties; 2) provide and guide future research using Defects4J with fine-grained information; 3) understand the representativeness of
Dissection of a Bug Dataset: Anatomy of 395 Patches from Defects4J

Victor Sobreira*, Thomas Durieux†, Fernanda Madeiral*, Martin Monperrus‡, and Marcelo de Almeida Maia*
*Federal University of Uberlândia, Brazil, {victor, fernanda.madeiral, marcelo.maia}@ufu.br
†INRIA & University of Lille, France, thomas.durieux@inria.fr
‡KTH Royal Institute of Technology, Sweden, martin.monperrus@csc.kth.se

Abstract—Well-designed and publicly available datasets of bugs are an invaluable asset to advance research fields such as fault localization and program repair as they allow directly and fairly comparison between competing techniques and also the replication of experiments. These datasets need to be deeply understood by researchers: the answer for questions like “which bugs can my technique handle?” and “for which bugs is my technique effective?” depends on the comprehension of properties related to bugs and their patches. However, such properties are usually not included in the datasets, and there is still no widely adopted methodology for characterizing bugs and patches. In this work, we deeply study 395 patches of the Defects4J dataset. Quantitative properties (patch size and spreading) were automatically extracted, whereas qualitative ones (repair actions

We focus on the analysis of Defects4J [14], a dataset containing 395 real bugs collected from six open-source Java projects. Although extensively used in recent research on fault localization [17], [18], [19] and program repair [20], [21], [8], Defects4J does not come with fine-grained information about bugs and their patches. We contribute to Defects4J with the extraction and study of both quantitative (e.g. metrics) and qualitative properties (e.g. patterns) regarding patches. This new data is very valuable to 1) interpret past published results based on Defects4J under the light of the extracted properties; 2) provide and guide future research using Defects4J with fine-grained information; 3) understand the representativeness of

1) Patch size
2) Patch spreading
3) Repair actions
4) Repair patterns
PPD
(Patch Pattern Detector)

From previous work (SANER’18):
- 9 pattern groups
- 25 patterns in total
PPD overview

GumTree

Retrieval of the AST diff

Analysis of the AST diff for pattern detection

Patterns:
1) Conditional block addition with return statement
2) Missing null check addition
3) Copy/Paste

- Conditional Block Detector
- Wrong Reference Detector
- Expression Fix Detector
- Missing Null-Check Detector
- Wraps/Unwraps Detector
- Copy/Paste Detector
- Single Line Detector
- Constant Change Detector
- Code Moving Detector
PPD overview

Patterns:
1) Conditional block addition with return statement
2) Missing null check addition
3) Copy/Paste
Missing Null-Check Detector

```java
2165 2165 }
2166 + if (markers == null) {
2167 +     return false;
2168 + }
2169 + boolean removed = markers.remove(marker);
```

```java
315    - double valueDelta = dataset.getStdDevValue(row, column).doubleValue();
318    + Number n = dataset.getStdDevValue(row, column);
319    + if (n != null) {
320    +     double valueDelta = n.doubleValue();
316 321    double highVal = rangeAxis.valueToJava2D(meanValue.doubleValue()
317 322        + valueDelta, dataArea, yAxisLocation);
318 323    double lowVal = rangeAxis.valueToJava2D(meanValue.doubleValue()
346 348    @-341,6 +346,7 else if (lclip <= 0.0) { // cases 5, 6, 7 and 8
341 343    line = new Line2D.Double(lowVal, rectY + rectHeight * 0.25,
342 344    lowVal, rectY + rectHeight * 0.75);
343 348    g2.draw(line);
349 + }
```
Missing Null-Check Detector

1. Searches for the addition of a binary operator where one of the two elements is null;

```java
2165 2165 }    
2166 +   if markers == null { 
2167 +     return false; 
2168 +   }
2166 2169   boolean removed = markers.remove(marker);
```
Missing Null-Check Detector

2. Extracts from the null-check the variable being checked
(variable <operator> null);

```java
2165 2165 } 
2166 if markers == null { 
2167 return false; 
2168 
}
2169 boolean removed = markers.remove(marker);
```

```java
315 315 - double valueDelta = dataset.getStdDevValue(row, column).doubleValue(); 
318 + Number n = dataset.getStdDevValue(row, column); 
319 + if n != null {
320 + double valueDelta = n.doubleValue();
321 +
322 +
323 +
324 +
325 +
326 +
327 +
328 +
329 +
330 +
331 +
332 +
333 +
334 +
335 +
336 +
337 +
338 +
339 +
340 +
341 +
342 +
343 +
344 +
345 +
346 +
347 +
348 +
349 +
```

Variable: markers

Variable: n
Missing Null-Check Detector

3. Verifies if the variable is new (added in the patch):
   a) if not new, a missing null-check was found
   b) if new, it verifies if the new null-check wraps existing code: if it does, a missing null-check was found

```java
2165  2165  }
2166 +  if markers == null {
2167 +      return false;
2168 +  }
2166  2169  boolean removed = markers.remove(marker);
```

Markers is not new (rule a)

```java
315  315  double valueDelta = dataset.getStdDevValue(row, column).doubleValue();
318 + Number n = dataset.getStdDevValue(row, column);
319 +  if n != null {
320 +      double valueDelta = n.doubleValue();
321 +      double highVal = rangeAxis.valueToJava2D(meanValue.doubleValue() + valueDelta, dataArea, yAxisLocation);
322 +      double lowVal = rangeAxis.valueToJava2D(meanValue.doubleValue() - 341,6 +346,7 else if (lclip <= 0.0) { // cases 5, 6, 7 and 8
341  346  line = new Line2D.Double(lowVal, rectY + rectHeight * 0.25,
342  347  lowVal, rectY + rectHeight * 0.75);
343  348  g2.draw(line);
349 +  }
```

n is new (rule b)
Evaluation: Method

• Running PPD

• Subject Dataset: 395 patches from Defects4J

• Result analysis:
  – Step 1: direct comparison with manual pattern detection (previous work)
  – Step 2: disagreement analysis
Evaluation: Results
Overall precision and recall

• Step 1: direct comparison with manual pattern detection (previous work)
  
  precision 78.26%; recall 86.95%

• Step 2: after disagreement analysis
  
  precision 91.53%; recall 92.39%
Evaluation: Results

Highlights

• Conditional Block (precision 98%; recall 96%)
  – Agreed: 194 instances
  – PPD: 39 new instances
Evaluation: Results

Highlights

• Single Line (precision 100%; recall 100%)
  – Agreed: 96 instances

• Missing Null-Check (precision 100%; recall 98%)
  – Agreed: 50 instances
Evaluation: Results

Highlights

- Wraps/Unwraps (precision 79%; recall 89%)
  - Agreed: 95 instances
  - PPD: 7 new instances
  - PPD: 30 false positives
Evaluation: Discussion on the reasons why the manual and automatic detections differ

• Reason #1: Global human vision versus AST-based analysis
Evaluation: Discussion on the reasons why the manual and automatic detections differ

• Reason #2: The automatic detection relies on rules defined by humans, and it is difficult to identify all cases where an instance of a pattern may exist
Conclusion

PPD is able to detect repair patterns in patches, which can be helpful to characterize datasets of bugs.
Future works

• To conduct experiments over other bug datasets
  – to evaluate the scalability of PDD
  – to compare bug datasets

• To create a visualization for patches where the repair patterns are highlighted
PPD is open-source

https://github.com/lascam-UFU/automatic-diff-diff-dissection
Towards an automated approach for bug fix pattern detection

Fernanda Madeiral\textsuperscript{1} Thomas Durieux\textsuperscript{2} Victor Sobreira\textsuperscript{1} Marcelo Maia\textsuperscript{1}

\textsuperscript{1}Federal University of Uberlândia, Brazil
\textsuperscript{2}INRIA & University of Lille, France
Threats to validity

• Internal validity: manual disagreement analysis
  – To mitigate this: each pattern group was analyzed by two authors of this paper

• External validity: evaluation only on Defects4J