VI Workshop on Software Visualization, Evolution and Maintenance (VEM)

Towards an automated approach for bug fix pattern detection



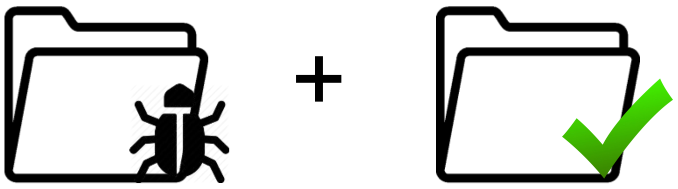

Fernanda Madeiral[1] **Thomas Durieux[2]** **Victor Sobreira[1]** **Marcelo Maia[1]**

[1]Federal University of Uberlândia, Brazil

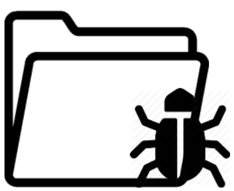
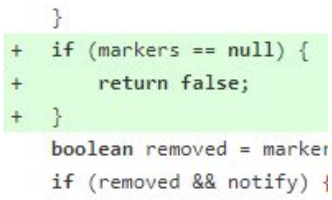
[2]INRIA & University of Lille, France

Datasets of bugs

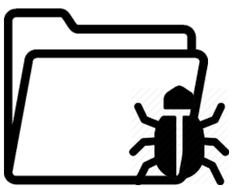
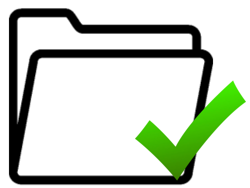
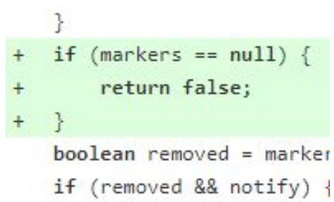
Datasets of bugs



Datasets of bugs

 +  diff = 

Datasets of bugs

 +  diff = 

<u>No knowledge on the composition of them</u>

25th International Conference on Software Analysis, Evolution and Reengineering (SANER '18)

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

25th International Conference on Software Analysis, Evolution and Reengineering (SANER '18)

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

25th International Conference on Software Analysis, Evolution and Reengineering (SANER '18)

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

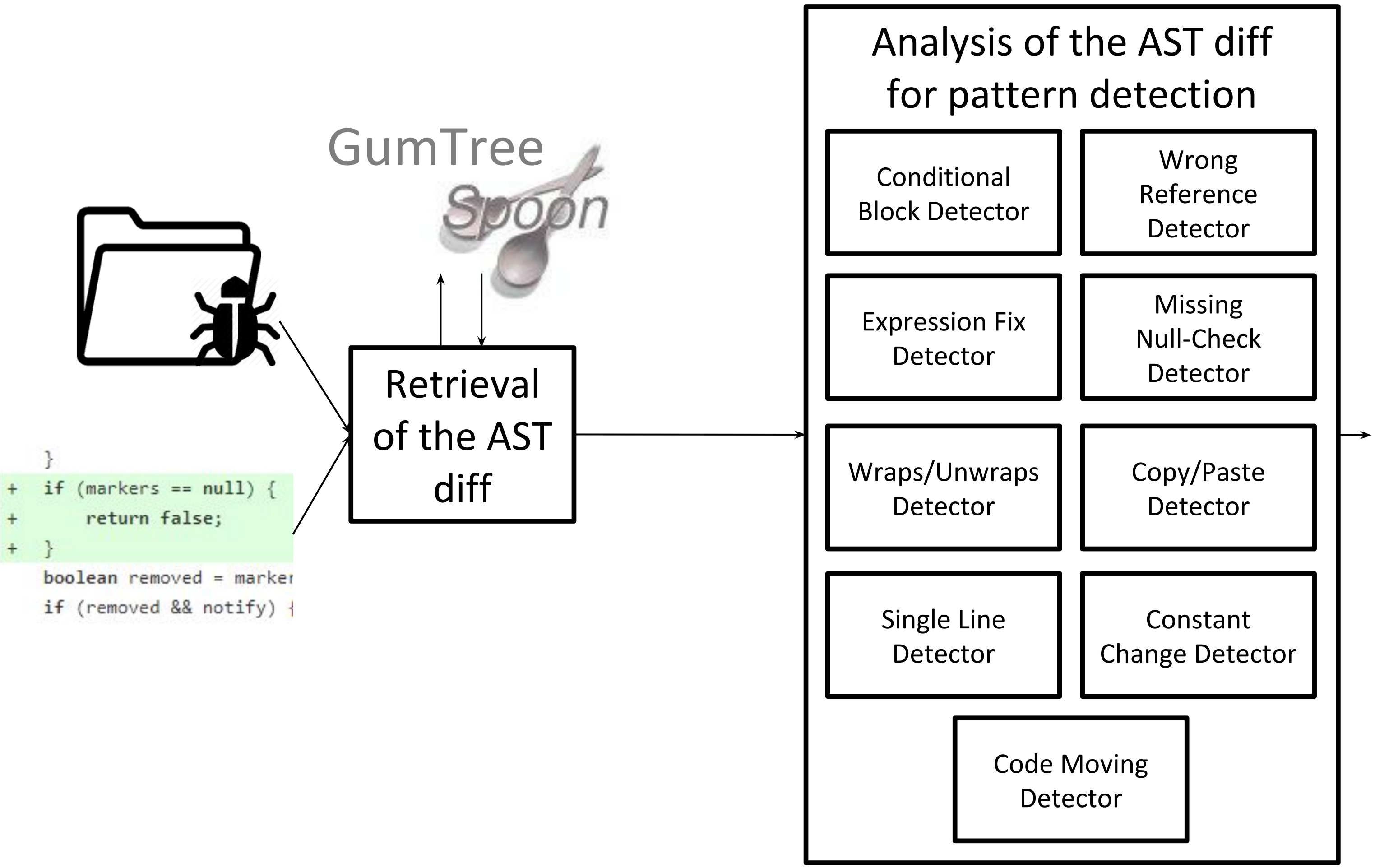
PPD
(Patch Pattern Detector)

From previous work (SANER'18):

9 pattern groups

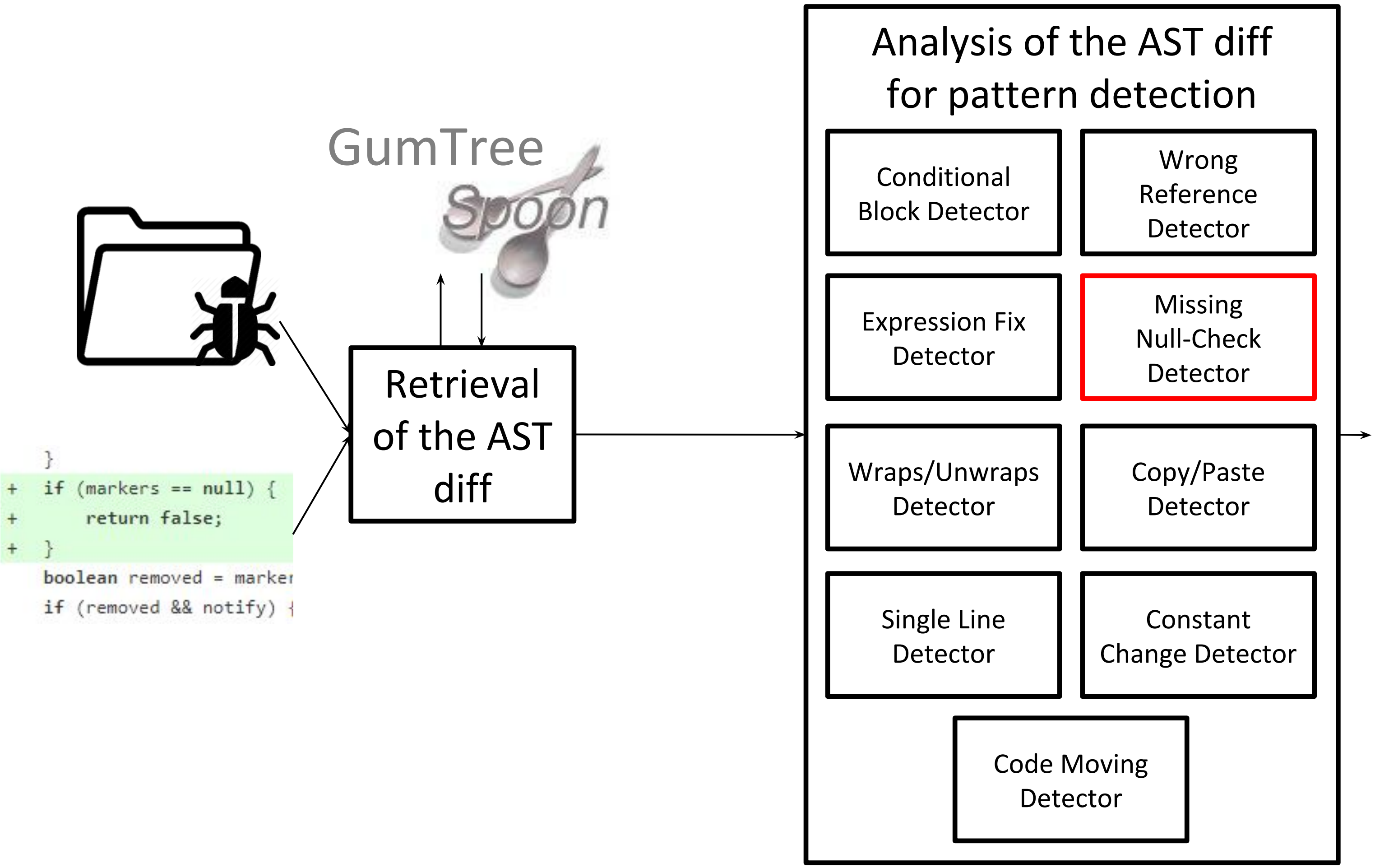
25 patterns in total

PPD overview



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

PPD overview



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

Missing Null-Check Detector

```
            }
+           if (markers == null) {
+               return false;
+           }
            boolean removed = markers.remove(marker);
```

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

Missing Null-Check Detector

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

```
        }
+       if (markers == null) {
+           return false;
+       }
        boolean removed = markers.remove(marker);
```

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

Missing Null-Check Detector

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

```
        }
+       if (markers == null) {
+           return false;
+       }
        boolean removed = markers.remove(marker);
```

Variable: markers

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

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

```diff
             }
+            if (markers == null) {
+                return false;
+            }
             boolean removed = markers.remove(marker);
```

markers is not new (rule a)

```diff
-                double valueDelta = dataset.getStdDevValue(row, column).doubleValue();
+            Number n = dataset.getStdDevValue(row, column);
+            if (n != null) {
+                double valueDelta = n.doubleValue();
                 double highVal = rangeAxis.valueToJava2D(meanValue.doubleValue()
                         + valueDelta, dataArea, yAxisLocation);
                 double lowVal = rangeAxis.valueToJava2D(meanValue.doubleValue()
@@ -341,6 +346,7 @@ else if (lclip <= 0.0) { // cases 5, 6, 7 and 8
                 line = new Line2D.Double(lowVal, rectY + rectHeight * 0.25,
                                          lowVal, rectY + rectHeight * 0.75);
                 g2.draw(line);
+            }
```

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
  - <u>PPD: 30 false positives</u>

Evaluation: Discussion on the reasons why the manual and automatic detections differ

- Reason #1: Global human vision versus AST-based analysis

```
+ } else if (type == Iterable.class) {
+     return new ArrayList<Object>(0);
  } else if (type == Collection.class)
        {
  [...]
```

Listing 4. Human vision.

```
+ } else {
+     if (type == Iterable.class) {
+         return new ArrayList<Object>(0);
      } else {
          if (type == Collection.class) {
  [...]
```

Listing 5. 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-dissection

VI Workshop on Software Visualization, Evolution and Maintenance (VEM)

Towards an automated approach for bug fix pattern detection

Fernanda Madeiral[1]

Thomas Durieux[2]

Victor Sobreira[1]

Marcelo Maia[1]

[1]Federal University of Uberlândia, Brazil

[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