How to Design a Program Repair Bot?
Insights from the Repairnator Project

Simon Urli, Zhongxing Yu, Lionel Seinturier, Martin Monperrus
simon.urli@inria.fr
February, 26\textsuperscript{th}, 2018

Inria & University of Lille
Proceedings of ICSE, SEIP track, 2018
After **one year** of operating a repair bot: what **pitfall** should you avoid?
What is Repairnator?

Repairnator
If the main objective of Terminator was “Seek and Destroy”, the main goal of Repairnator is “Scan and Repair”.
→ Fix a maximum of failing builds from TravisCI.
Overview & Design choices
Overview

GitHub Projects
Commits
Builds with failing tests
CI Build Analysis
Bug Reproduction
Patch Synthesis
Nopol
Astor
NPEFix
collected repair data
Research community

Developers
Repairnator patch analyst
Repairnator Bot
List of projects
Patches
Nopol
Astor
NPEFix
collected repair data
Research community
Repairnator targets:

- **Java projects using Maven**
  - Expertise in program repair for Java
  - Standard build tool

- Build-based repairing bot

- GitHub projects using TravisCI
Design choices

Repairnator targets:

- Java projects using Maven
- **Build-based repairing bot**
  - Easy oracle: failing builds → project to repair
  - Long-term view: Repairnator as part of the CI
- GitHub projects using TravisCI
Design choices

Repairnator targets:

- Java projects using Maven
- Build-based repairing bot
- **GitHub projects using TravisCI**
  - GitHub: largest open-source code hosting service
  - TravisCI: standard CI for open-source on GitHub & open API
Step 1 : CI Build Analysis
Considered Projects

Different ways to produce the list:

- TravisTorrent
- GHTorrent
- GitHub API & Trends

Criteria to be selected:

1. Open-source and available on Github
2. Use Java and Maven
3. With a test suite
4. Popular and active: the most starred first and activity in previous months
List of projects to consider from:

- TravisTorrent: not so many data
- GHTorrent: needs to be filtered
- GitHub Trends: no API

The usage of tools over 14,188 Java projects hosted on GitHub.

Results: **1609 projects** selected.
Build analysis

Process: builds are pulled from Travis, then status and language are checked and finally logs are analyzed for test failure.
Problem: Current build analysis is tedious and time-consuming.

What can we do?

- trigger bot from the test-failing build if possible
  - it might depend on the considered CI
- avoid as much as possible log analysis
  - get test results from CI
  - launch reproduction even when not sure
Step 2: Local bug reproduction
Steps for local bug reproduction

1. Clone the repository
2. Checkout the right commit
3. Compile the build (i.e. `mvn install -DskipTest`)
4. Run test (i.e. `mvn test`)
5. Parse test information (i.e. read xml files)

All steps are done inside a docker container and if a bug is successfully reproduced all data are pushed to a repository.
Local bug reproduction: obtained results 1/2

Build statuses (all times - 14385 builds)

- Error when compiling: 5215 (36.3%)
- Successful Bug Reproduction: 4510 (31.4%)
- Test without failure: 2874 (20.0%)
- Error when testing: 1415 (9.8%)
- Error when checking out: 337 (2.3%)
- Error when cloning: 34 (0.2%)
<table>
<thead>
<tr>
<th>Rank</th>
<th>Project</th>
<th>Builds with test failure</th>
<th>Rank</th>
<th>Reproduced bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>druid-io/druid</td>
<td>579</td>
<td>2</td>
<td>359 (62.00%)</td>
</tr>
<tr>
<td>2</td>
<td>apache/flink</td>
<td>477</td>
<td>3</td>
<td>326 (68.34%)</td>
</tr>
<tr>
<td>3</td>
<td>prestodb/presto</td>
<td>1000</td>
<td>1</td>
<td>194 (19.40%)</td>
</tr>
<tr>
<td>4</td>
<td>hubspot/singularity</td>
<td>437</td>
<td>5</td>
<td>182 (41.65%)</td>
</tr>
<tr>
<td>5</td>
<td>corfudb/corfudb</td>
<td>313</td>
<td>7</td>
<td>126 (40.26%)</td>
</tr>
<tr>
<td>6</td>
<td>apache/storm</td>
<td>349</td>
<td>6</td>
<td>111 (31.81%)</td>
</tr>
<tr>
<td>7</td>
<td>geoserver/geoserver</td>
<td>118</td>
<td>18</td>
<td>109 (92.37%)</td>
</tr>
<tr>
<td>8</td>
<td>spotify/docker-client</td>
<td>111</td>
<td>21</td>
<td>99 (89.19%)</td>
</tr>
<tr>
<td>9</td>
<td>xetorthio/jedis</td>
<td>100</td>
<td>25</td>
<td>94 (94.00%)</td>
</tr>
<tr>
<td>10</td>
<td>4pr0n/ripme</td>
<td>94</td>
<td>28</td>
<td>87 (92.55%)</td>
</tr>
</tbody>
</table>
Local bug reproduction

Bug reproduction is **HARD**.

Build failure reproduction errors can come from:

- build environment (OS, JDK, ...)
- build setup (bash script to start a server, ...)
- flaky tests or custom failing goals (checkstyle, coverage threshold...)
- right source code version not found
- timeout (after 24 hours we kill build)
Local bug reproduction

Bug reproduction is **HARD**.

What can we do?

- reproduce in sandboxed environment (docker)
- use the same setup as in the CI
- don’t try to get back missing commits
Step 3 : Patch Synthesis
**Repair tools**

**Nopol:**
dedicated to repair conditionnal bugs by modifying existing conditions or inserting preconditions.

**Astor:**
a generate-and-validate repair tool derived from Genprog.

**NPEFix:**
dedicated to repair only NullPointerException by inserting preconditions.
Patch synthesis steps

1. Analyze test information from bug reproduction step
2. if a NullPointerException is detected: run NPEFix
3. Run Astor & Nopol (budget based)

At each point, send an email if a Patch is found.
Patch synthesis is even **HARDER**

Successful Reproduction Builds (all times - 14307 builds)

- Bug reproduction and patch created: 0.4% (17)
- Bug reproduction without patch: 99.6% (4464)
## Obtained patches

<table>
<thead>
<tr>
<th>Project</th>
<th>Builds w/ patches</th>
<th>Nopol patches</th>
<th>NPEFix patches</th>
<th>Rank (rep. build)</th>
</tr>
</thead>
<tbody>
<tr>
<td>jamesagnew/hapi-fhir</td>
<td>1</td>
<td>35</td>
<td>0</td>
<td>88</td>
</tr>
<tr>
<td>spotify/cassandra-reaper</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>121</td>
</tr>
<tr>
<td>xmlunit/xmlunit</td>
<td>1</td>
<td>145</td>
<td>0</td>
<td>203</td>
</tr>
<tr>
<td>apache/pdfbox</td>
<td>1</td>
<td>120</td>
<td>0</td>
<td>95</td>
</tr>
<tr>
<td>LiveRamp/hank</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>225</td>
</tr>
<tr>
<td>spring-cloud/spring-cloud-dataflow</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>56</td>
</tr>
<tr>
<td>IQSS/dataverse</td>
<td>2</td>
<td>0</td>
<td>16</td>
<td>40</td>
</tr>
<tr>
<td>bonigarcia/webdrivermanager</td>
<td>3</td>
<td>30</td>
<td>0</td>
<td>27</td>
</tr>
<tr>
<td>GeoWebCache/geowebcache</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>107</td>
</tr>
<tr>
<td>timmolter/XChange</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>58</td>
</tr>
<tr>
<td>phax/jcodemodel</td>
<td>1</td>
<td>624</td>
<td>0</td>
<td>193</td>
</tr>
<tr>
<td>phoenixnap/springmvc-raml-plugin</td>
<td>1</td>
<td>348</td>
<td>0</td>
<td>66</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>15</strong></td>
<td><strong>1307</strong></td>
<td><strong>23</strong></td>
<td></td>
</tr>
</tbody>
</table>
### Valid patches

<table>
<thead>
<tr>
<th>Total</th>
<th>15</th>
<th>1307</th>
<th>23</th>
</tr>
</thead>
</table>

Number of **valid** patch obtained and accepted: 1.

**Fix NPE with queryParams #1**

- **Merged**
- aaime merged 1 commit into `aaime:post_form` from `lucesape:aaime-post-form` on 12 Jan
- **Conversation** 1
- **Commits** 1
- **Files changed** 1

lucesape commented on 12 Jan

This should fix your failing travis build on GeoWebCache#582

aaime commented on 12 Jan

Weird, I thought I already fixed this... maybe I did in some other place. Thanks for the patch!

aaime merged commit e48f17e into `aaime:post_form` on 12 Jan
Valid patches

| Total | 15 | 1307 | 23 |

Number of **valid** patch obtained and accepted: 1.

Fix NPE with queryParams #1

```
Fix NPE with queryParams #1

Merged aaine merged 1 commit into aaine:post_form from lucesage:aaine-post-form on 12 Jan

Changes from all commits ~ Jump to... +10 -6  

File: geowebcache/core/src/main/java/org/geowebcache/layer/wms/WMSHttpHelper.java

@@ -306,11 +306,15 @@
    @ public HttpMethodBase executeRequest(Total URL url, final Map<String, String> qu

    // prepare the request

-   NameValuePair[] params = new NameValuePair(queryParams.size());
-   int i = 0;
-   for (Map.Entry<String, String> e : queryParams.entrySet()) {
-     params[i] = new NameValuePair(e.getKey(), e.getValue());
-     i++;
+   NameValuePair[] params;
+   if (queryParams != null) {
+     params = new NameValuePair(queryParams.size());
+     int i = 0;
+     for (Map.Entry<String, String> e : queryParams.entrySet()) {
+       params[i] = new NameValuePair(e.getKey(), e.getValue());
+       i++;

```
## Top 10 error types

<table>
<thead>
<tr>
<th>Rank</th>
<th>Exception</th>
<th>Occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>java.lang.AssertionError</td>
<td>2,162</td>
</tr>
<tr>
<td>2</td>
<td>java.lang.NullPointerException</td>
<td>641</td>
</tr>
<tr>
<td>3</td>
<td>org.junit.ComparisonFailure</td>
<td>419</td>
</tr>
<tr>
<td>4</td>
<td>java.lang.Exception</td>
<td>250</td>
</tr>
<tr>
<td>5</td>
<td>java.lang.IllegalStateException</td>
<td>202</td>
</tr>
<tr>
<td>6</td>
<td>java.lang.NoClassDefFoundError</td>
<td>197</td>
</tr>
<tr>
<td>7</td>
<td>java.lang.RuntimeException</td>
<td>191</td>
</tr>
<tr>
<td>8</td>
<td>junit.framework.AssertionFailedError</td>
<td>163</td>
</tr>
<tr>
<td>9</td>
<td>java.lang.ExceptionInInitializerError</td>
<td>117</td>
</tr>
<tr>
<td>10</td>
<td>java.io.IOException</td>
<td>110</td>
</tr>
</tbody>
</table>
- Current generic repair tools (Astor & Nopol) are really time and resources consuming
- Repairing assertion errors = guessing a behaviour which is pretty hard
- Repairing explicit errors (NPE, NumberFormatException, ...) seems easier to achieve
- For production-readiness, repair tools should use sophisticated setups (multimodule, external resources, ...)
1. Bigger scope & faster response time: use directly last finished builds on TravisCI instead of relying on a list of projects. ✔️

2. Avoid false positive: Use directly TravisCI to reproduce failures AND to produce patches.

3. Integrate Repairnator into the CI.
Play with it

- Repairnator sourcecode:
  https://github.com/Spirals-Team/repairnator
- Repository of bugs:
  https://github.com/Spirals-Team/seip-2018
  (consolidated data from February 2017 to January 2018)
- Live data: http://repairnator.lille.inria.fr (almost 15,000 builds this morning. 14,385 two weeks ago)
- Want to integrate your own program repair tool? contact us!