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, 26th, 2018

Inria & University of Lille
Proceedings of ICSE, SEIP track, 2018

After **one year** of operating a repair bot: what **pitfall** should you avoid?
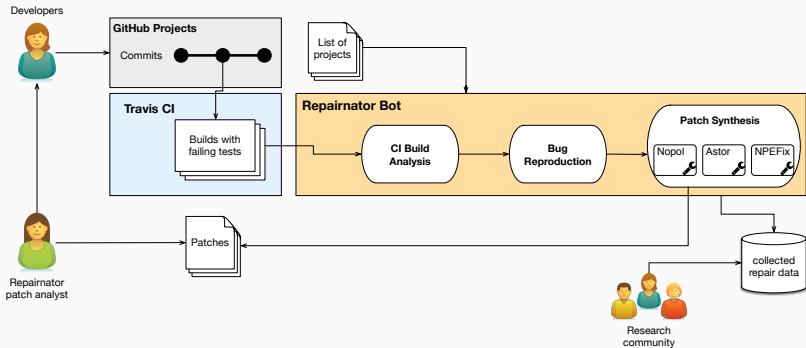
Repairnator

If the main objective of Terminator was "Seek and Destroy", the main goal of Repairnator is "**Scan and Repair**".

$\rightarrow$ Fix a maximum of failing builds from TravisCI.

Overview & Design choices

Design choices

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 $\rightarrow$ 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:

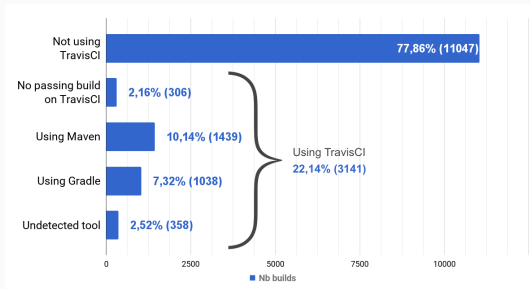- TravisTorrent
- GHTorrent
- GitHub API & Trends

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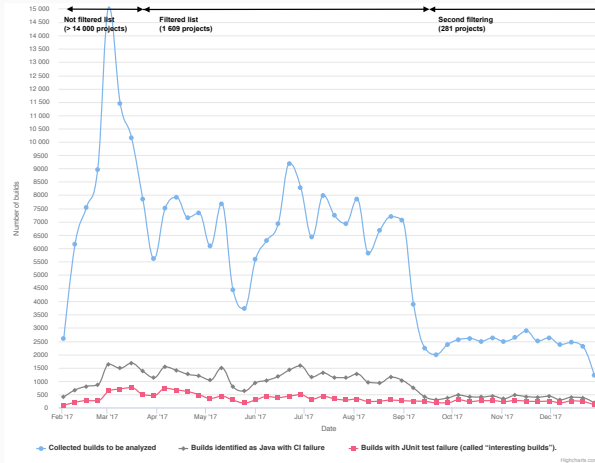 **not so many data**
- GHTorrent:
  needs to be filtered
- GitHub Trends:
  no API

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

Results: **1609 projects** selected.

Process: builds are pulled from Travis, then status and language are checked and finally logs are analyzed for test failure.

Build analysis

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
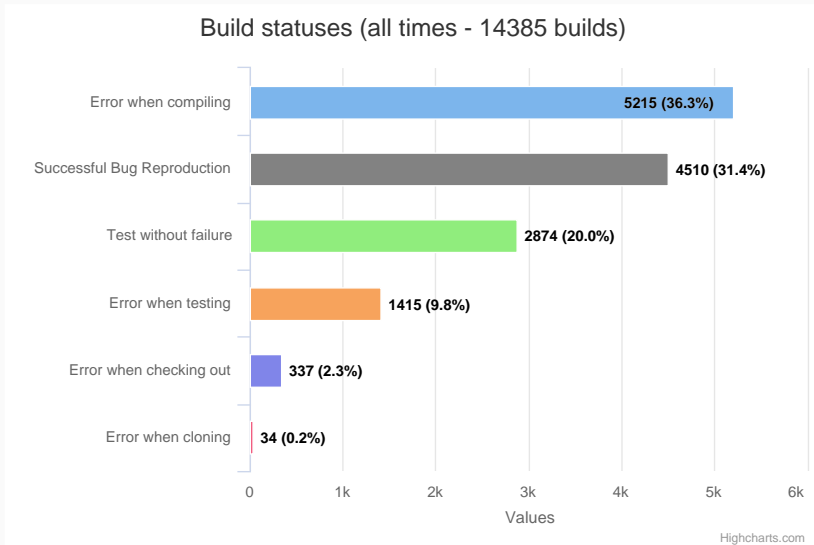 - launch reproduction even when not sure

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.

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%)

Values

Highcharts.com

Local bug reproduction: obtained results 2/2

Rank	Project	Builds with test failure	Rank (test failure)	Reproduced bugs
1	druid-io/druid	579	**2**	359 (62.00%)
2	apache/flink	477	**3**	326 (68.34%)
3	prestodb/presto	1000	**1**	194 (19.40%)
4	hubspot/singularity	437	**5**	182 (41.65%)
5	corfudb/corfudb	313	**7**	126 (40.26%)
6	apache/storm	349	**6**	111 (31.81%)
7	geoserver/geoserver	118	**18**	109 (92.37%)
8	spotify/docker-client	111	**21**	99 (89.19%)
9	xetorthio/jedis	100	**25**	94 (94.00%)
10	4pr0n/ripme	94	**28**	87 (92.55%)

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 exisiting 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

Patch synthesis is even **HARDER**

Successful Reproduction Builds (all times - 14307 builds)

Bug reproduction and patch created: 0.4% (17)
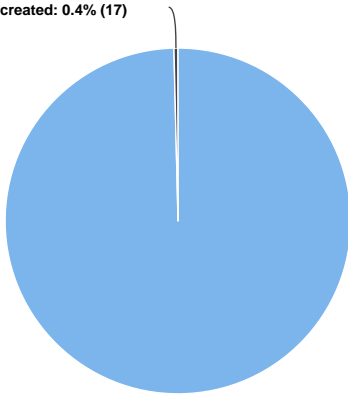
Bug reproduction without patch: 99.6% (4464)

Highcharts.com

Obtained patches

Project	Builds w/ patches	Nopol patches	NPEFix patches	Rank (rep. build)
jamesagnew/hapi-fhir	1	35	0	**88**
spotify/cassandra-reaper	1	1	0	**121**
xmlunit/xmlunit	1	145	0	**203**
apache/pdfbox	1	120	0	**95**
LiveRamp/hank	1	4	0	**225**
spring-cloud/spring-cloud-dataflow	1	0	1	**56**
IQSS/dataverse	2	0	16	**40**
bonigarcia/webdrivermanager	3	30	0	**27**
GeoWebCache/geowebcache	1	0	2	**107**
timmolter/XChange	1	0	4	**58**
phax/jcodemodel	1	624	0	**193**
phoenixnap/springmvc-raml-plugin	1	348	0	**66**
Total	15	1 307	23	

Valid patches

Total		15	1 307	23

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

Valid patches

Total	15	1 307	**23**

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

Top 10 error types

Rank	Exception	Occurrences
1	java.lang.AssertionError	2 162
2	java.lang.NullPointerException	641
3	org.junit.ComparisonFailure	419
4	java.lang.Exception	250
5	java.lang.IllegalStateException	202
6	java.lang.NoClassDefFoundError	197
7	java.lang.RuntimeException	191
8	junit.framework.AssertionFailedError	163
9	java.lang.ExceptionInInitializerError	117
10	java.io.IOException	110

Patch synthesis: discussion

- 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, ...)

Future of Repairnator

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!