Empirical Review of Java Program Repair Tools

A Large-Scale Experiment on 2,141 Bugs and 23,551 Repair Attempts

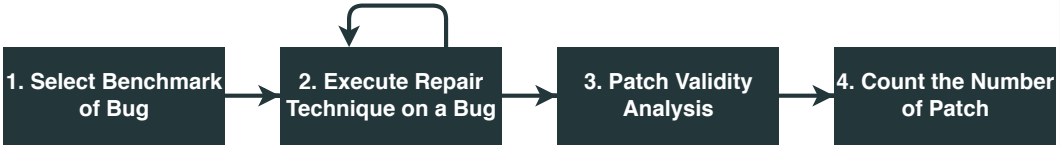
Thomas Durieux, Fernanda Madeiral, Matias Martinez, Rui Abreu

FSE'19

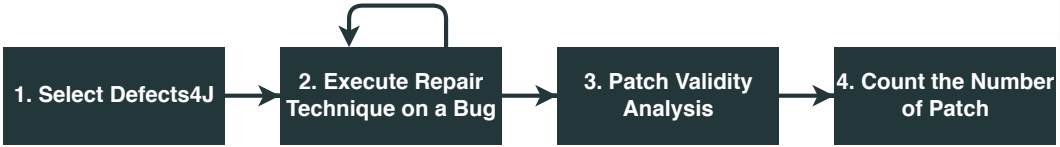
INESC-ID, Lisbon, Portugal, Federal University of Uberlândia, Brazil, Université Polytechnique Hauts-de-France

Empirical Evaluation for Automatic Repair Techniques

Empirical Evaluation for Program Repair Techniques



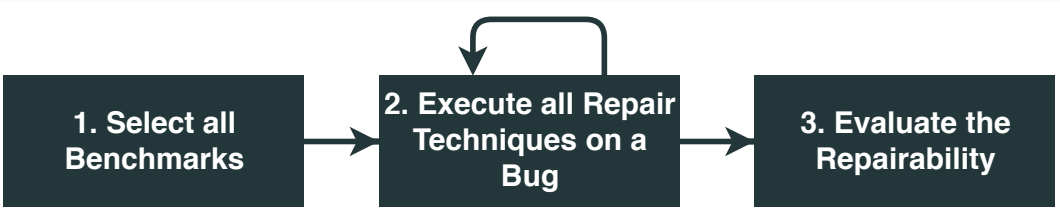
Empirical Evaluation for Program Repair Techniques



22/24 Java repair tools were evaluated on (a subset of) bugs from **Defects4J**.

How automatic repair results generalize on other benchmarks?

Study Design of RepairThemAll



1. Benchmark Selection

Inclusion criteria: All benchmarks of Java bugs with a test-suite.

Benchmark	# Projects	# Bugs	LOC (Java)
Bears	72	251	62,597
Bugs.jar	8	1158	212,889
Defects4J	6	395	129,592
IntroClassJava	6	297	230
QuixBugs	40	40	190
Total	130	2,141	146,428

2. Java Repair Tool Selection

24 considered automatic repair tools.

1. ACS	9. Jaid	17. SimFix
2. ARJA	10. jGenProg	18. SketchFix
3. CapGen	11. jKali	19. SOFix
4. Cardumen	12. jMutRepair	20. ssFix
5. DeepRepair	13. Kali-A	21. xPAR
6. Elixir	14. LSRepair	22. DynaMoth
7. GenProg-A	15. PAR	23. Nopol
8. HDRepair	16. RSRepair-A	24. NPEFix

2. Repair Tools Selection

Four inclusion criteria.

C1. Publicly available

C2. Still Executable

C3. Runs on bugs from different benchmarks beyond the one used in its original evaluation

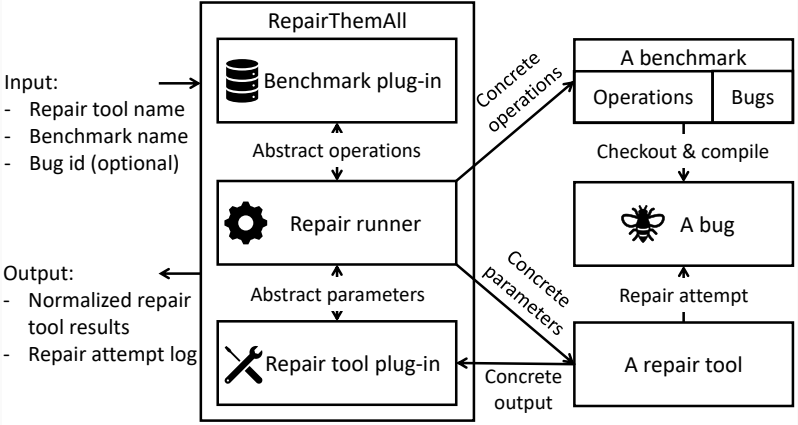
C4. Requires only the source code and the test suite of the program under repair.

2. Repair Tools Selection

11 Selected automatic repair tools.

	Criteria	Repair Tools
Excluded (13)	Not public	Elixir, PAR, SOFix, xPAR
	Not working	ACS, CapGen, DeepRepair
	Multi-bench support	LSRepair, SimFix
	Only source required	HDRepair, Jaid, SketchFix
Included (11)	ARJA, Cardumen, DynaMoth, jGenProg, GenProg-A, jKali, Kali-A, jMutRepair, Nopol, NPEFix, RSRepair-A	

3. Build Repair Framework



Available on GitHub and on Dockerhub.

3. Build Repair Framework

Command line interface

```
python repair.py Nopol --benchmark Defects4J --id Chart-5
```

or

```
docker run tdurieux/repairthemall Nopol --benchmark Defects4J
--id Chart-5
```

Available on GitHub and on Dockerhub.

4. Execution & Analysis

2,141 bugs x **11** tools = **23,551** repair attempts.

Executed on Grid5000 with a total execution time of 314 days.

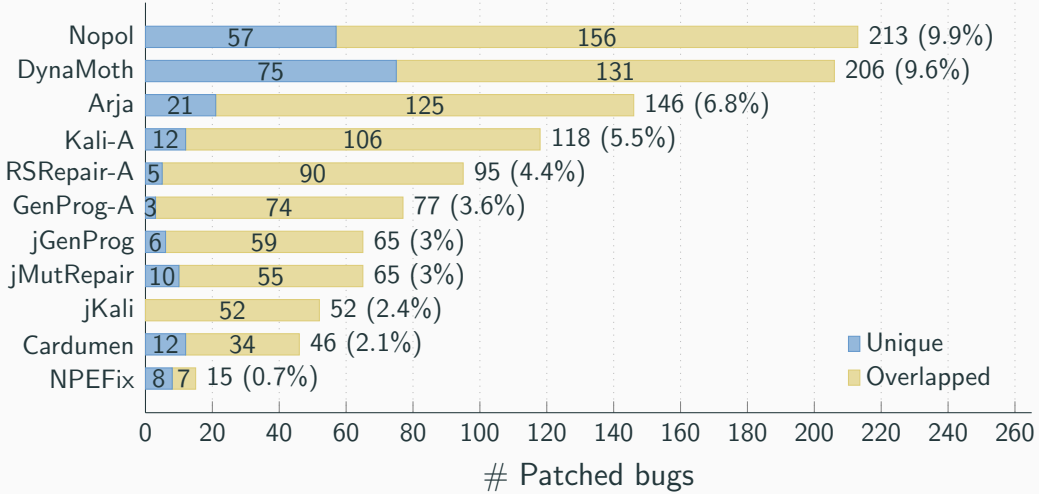
Results are available at

Website:
`program-repair.org/RepairThemAll_experiment/`

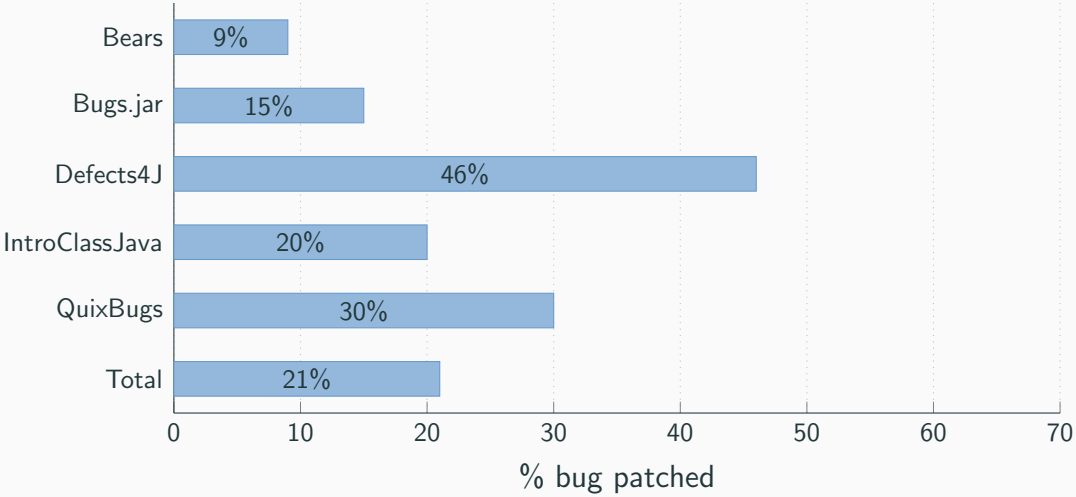
Repository:
`github.com/program-repair/RepairThemAll_experiment`

The tools generated 66,596 patches for 459 bugs!

Repairability of the tools



% of Bugs Patched per Benchmark



Defects4J has a significant more patched bugs than the other benchmark.

Proportion of Patched Bugs

	Defects4J	Other benchmarks	Total	p-value
ARJA	21%	3%	7%	< 0.00001
GenProg-A	11%	2%	3%	< 0.00001
Kali-A	18%	3%	5%	< 0.00001
RSRepair-A	15%	2%	4%	< 0.00001
Cardumen	4%	2%	2%	0.00107
jGenProg	7%	2%	3%	< 0.00001
jKali	6%	1%	2%	< 0.00001
jMutRepair	5%	2%	3%	0.009309
Nopol	27%	10%	11%	< 0.00001
DynaMoth	18%	7%	10%	< 0.00001
NPEFix	2%	<1%	<1%	0.000031

The repairability of the tools does not seem to generalize on the other benchmarks.

Discussion on repairability

What could impact the repairability of tools on the different benchmarks?

1. Controlled environment: the tools do not handle the complexity of the diversity of the environments.
2. The type of bugs and projects.
3. The bug fix isolation of `Defects4J`.

Discussion on repairability

What can be done to reduce benchmark overfitting?

1. Use several benchmarks during the creation of new techniques
2. Don't hard code one specific benchmark in your artifacts.

Reasons of non-patch generation

1. The repair tool cannot repair the bug
2. Incorrect fault localization
3. Multiple fault locations
4. Small time budget
5. Incorrect configuration
6. Other technical issues

Summary

Take away

Use a diversity of benchmarks to evaluate your automatic repair tools

Contributions

- An Investigation on the **benchmark overfitting**: we shown that this is a problem
- The RepairThemAll framework
- **66,596** test-suite adequate patches

Open-science

All the artifacts are open-science and available on GitHub:

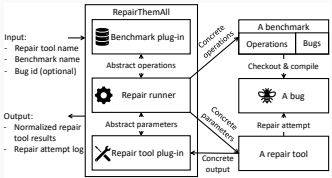
`https://github.com/program-repair/RepairThemAll`

`https://github.com/program-repair/RepairThemAll_experiment`

Available

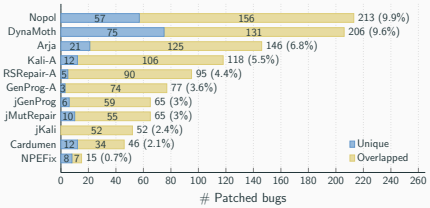
Reusable

3. Build Repair Framework



Available on GitHub and on Dockerhub.

Patched Bugs



Proportion of Patched Bugs

	Defects4J	Other benchmarks	Total	p-value
ARJA	21%	3%	7%	< 0.00001
GenProg-A	11%	2%	3%	< 0.00001
Kali-A	18%	3%	5%	< 0.00001
RSRepair-A	15%	2%	4%	< 0.00001
Cardumen	4%	2%	2%	0.00107
jGenProg	7%	2%	3%	< 0.00001
jKali	6%	1%	2%	< 0.00001
jMutRepair	5%	2%	3%	0.009309
Nopol	27%	10%	11%	< 0.00001
DynaMoth	18%	7%	10%	< 0.00001

Reasons of non-patch generation

1. The repair tool cannot repair the bug
2. Incorrect fault localization
3. Multiple fault locations
4. Small time budget
5. Incorrect configuration
6. Other technical issues