BugsPHP: A dataset for Automated Program Repair in PHP

K.D. Pramod
University of Moratuwa
Moratuwa, Sri Lanka
dushan.18@cse.mrt.ac.lk

W.T.N. De Silva
University of Moratuwa
Moratuwa, Sri Lanka
thushann.18@cse.mrt.ac.lk

W.U.K. Thabrew
University of Moratuwa
Moratuwa, Sri Lanka
udithk.18@cse.mrt.ac.lk

Ridwan Shariffdeen*
National University of Singapore
Singapore, Singapore
ridwan@nus.edu.sg

Sandareka Wickramanayake
University of Moratuwa
Moratuwa, Sri Lanka
sandarekaw@cse.mrt.ac.lk

ABSTRACT

Automated Program Repair (APR) improves developer productivity by saving debugging and bug-fixing time. While APR has been extensively explored for C/C++ and Java programs, there is little research on bugs in PHP programs due to the lack of a benchmark PHP bug dataset. This is surprising given that PHP has been one of the most widely used server-side languages for over two decades, being used in a variety of contexts such as e-commerce, social networking, and content management. This paper presents a benchmark dataset of PHP bugs on real-world applications called BugsPHP, which can enable research on analysis, testing, and repair for PHP programs. The dataset consists of training and test datasets, separately curated from GitHub and processed locally. The training dataset includes more than 600,000 bug-fixing commits. The test dataset contains 513 manually validated bug-fixing commits equipped with developer-provided test cases to assess patch correctness.

KEYWORDS

Automated Program Repair, PHP Application Errors

ACM Reference Format:
K.D. Pramod, W.T.N. De Silva, W.U.K. Thabrew, Ridwan Shariffdeen, and Sandareka Wickramanayake. 2024. BugsPHP: A dataset for Automated Program Repair in PHP. In *Proceedings of 21st International Conference on Mining Software Repositories (MSR 2024).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Manually fixing software bugs is a tedious and time-consuming task. Software engineers must invest significant time and effort in finding and fixing bugs in software programs. For example, O'Dell et al. in [13] have mentioned that developers spend 50% of programming time finding and fixing bugs. As a solution, Automated Program Repair (APR) [5] has been explored to reduce the bug-fixing effort and increase developer productivity. Given the buggy program, APR tools automatically generate potential patches for software bugs. APR tools analyze the buggy program to determine the root cause and create a patch that fixes the bug while preserving the original program functionality [11]. Most of the existing APR tools, particularly the learning-based ones, have been implemented for bug repair in C, Java, Python, and JavaScript languages owing to the availability of large datasets for those languages. Surprisingly, despite being the most popular server-side scripting language, there is no benchmark bug dataset for PHP.

PHP is an open-source and general-purpose language that powers many large-scale web applications like Facebook, Wikipedia, and WordPress. Owing to the easiness of programming, PHP is still among the top choices for web development, and PHP is used by 77.5% of websites as the server-side scripting language [1]. Weak typing, poor performance, and lack of debugging tools cause errors in PHP web applications. Despite its popularity, the research community has paid little attention to developing techniques to improve PHP programs. A well-organized labeled PHP bugs dataset must be introduced to facilitate further studies of the software evolution, maintenance, and repair of PHP applications.

In this work, we introduce BugsPHP, a data-set of PHP programs that can be used to train deep learning models and evaluate any APR technique including traditional techniques that are not learning models. BugsPHP contains 653606 bug fixing commits and a separate set of 513 bug fixing commits equipped with at least one failing test case, by crawling 5000 GitHub repositories with PHP code. We collect commits from 01 Jan 2020 to Mar 2023. For the test data-set construction, we carry out a manual test case validation by running developer provided test cases on the fixed version of the program. We select the commits with at least one failing test case from its parent commit, resulting in 513 bugs from 15 PHP applications that constitute the buggy version, fixed version, and related test cases. Excluding the project repositories used for the test data-set, the remaining projects are used to construct the training data-set. This paper makes three main contributions:

- A PHP bug dataset called BugsPHP consisting of 513 PHP bugs for testing and 653606 for training deep learning models from popular open-source projects.
- Analysis of the types of errors in PHP applications captured in BugsPHP
- Preliminary results of the effectiveness of existing APR Models to fix errors in PHP applications

*corresponding author

MSR 2024, April 2024, Lisbon, Portugal
© 2024 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 MOTIVATION

PHP is the most popular server-side language used by major web applications, including Facebook, Wikipedia, Tumblr, Slack, MailChimp, Etsy, and WordPress. Facebook alone boasts 2.9 billion users globally, while Wikipedia has received around 5 billion visits in the first 5 months of 2022. Slack is expected to have 32.3 million monthly users by 2023, and WordPress has been used to create 810 million websites, accounting for 43% of all websites.

However, like any other software, PHP programs can have bugs ranging from minor issues to major security flaws. These bugs can lead to severe problems such as data loss or unauthorized access by attackers who exploit vulnerabilities in web applications. For instance, the 2017 Equifax data breach [10] cost the company $700 million in expenses and lost revenue due to a flaw in the company's web application and affected nearly 150 million clients, about 56% of Americans. 87 million records of US resident profiles were obtained through Facebook as the most significant data breaches on online social networks as of 2020, and the data was used to create software that could predict and influence electors [2].

Nevertheless, manually fixing program bugs can be costly and time-consuming, causing delays in releasing new features or products. To save resources and time, developers can use efficient tools and methods to identify and resolve these issues, enabling them to focus on creating high-quality software. To better understand the nature of bugs in PHP applications, there is a need for a standard dataset to study the different types of bugs and their required fixes. Additionally, a comprehensive dataset can facilitate the analysis of patterns and traits of PHP bugs over time, investigating their effects on web application security and developing innovative software testing and verification methods.

This work aims to fill this gap by curating a list of PHP bugs from the most popular open-source PHP applications. Researchers can use our work to evaluate program repair tools for PHP applications. Furthermore, this dataset can be applied to identify insecure code, detect code smells and create new methods and tools for debugging PHP bugs. Overall, this research aims to benefit the community by offering a valuable dataset of PHP bugs.

3 RELATED WORK

The existing work in analyzing program bugs has curated datasets focused on Java, C, Python, and JavaScript bugs. For example, Defects4J[8] is a framework and database that offers genuine Java program bugs for reproducible research in software testing. It has 357 bugs from five open-source programs, each with a complete test suite that exposes the bug. It's expandable, and new bugs can be easily added to the database once a program is set up. Bugs.jar [14] is an extensive data collection useful for researching automated Java program debugging, testing, and patching. It contains 1,158 bugs and patches from 8 open-source Java projects representing eight important application categories. BEARS [12] is a project that creates a flexible benchmark for automatic repair studies in Java. It collects and stores bugs by identifying potential pairs of faulty and fixed program versions from open-source projects on GitHub. BEARS is publicly accessible and includes 251 reproducible bugs from 72 projects that use the Travis CI and Maven build environment.

The ManyBugs and IntroClass datasets [9] comprise 1,183 defects found in 15 C/C++ programs. ManyBugs contains bugs from well-known open-source projects, while IntroClass contains bugs from programming assignments done by a small group of students. The datasets BugsJS [6] and FixJS [4] are collections of JavaScript bugs. BugsJS features 453 actual bugs that have been verified manually. These bugs are taken from 10 widely used server-side JavaScript programs, which collectively have 444,000 lines of code. FixJS, on the other hand, gathers bugs from GitHub and provides information on the faulty and fixed versions of the same program. This dataset comprises over 300,000 samples and includes details on the commit, before/after states, and three source code representations. BugsInPy [16] is a dataset that collects real world bugs from Python programs, where each is accompanied by a failing test case that passes once the bug is fixed. Vul4J [3] is a more recent work collecting vulnerability fixing commits for Java programs.

In contrast, BugsPHP is generic and contains information on PHP bugs. Unlike previous datasets, we provide support for learning-based and traditional APR techniques by providing a training dataset and test cases for patch validation.

4 BUGSPHP DATASET

In this section, we present the methodology we followed to curate a dataset of PHP bugs and provide a statistical overview. Our new dataset, BugsPHP, includes a training dataset of 653,606 PHP bug-fixing commits and a test dataset of 513 bugs from the most popular open-source applications collected from GitHub between 01 Jan 2021 and March 2023.

4.1 Methodology

The process of curating the BugsPHP dataset is outlined in Figure 1. Based on their stargazers count, we use the GitHub REST API to retrieve the top 5000 PHP repositories with at least one commit after 01 Jan 2021 (indicating recent development activities). We avoid repositories that are not maintained by filtering those which do not have recent development activities. Bugs are then collected from these repositories and filtered based on the number of file and line changes they have. After filtering, the top 75 repositories' bugs are used for the testing dataset, while the remaining repositories are used for the training dataset. For the bugs in the training dataset, we extract the buggy and fixed versions files for the BugsPHP training dataset. Meanwhile, we conduct a dynamic validation for the bugs in the testing dataset to obtain each bug's corresponding buggy version, fixed version, and test cases. Below, we elaborate on each step of curating our dataset.

Repository Selection: To collect commits, we retrieve PHP repositories using the GitHub REST API [1]. Then, we sort them by popularity, measured using the stargazers count (e.g., the number of stars) and the latest commit date. We select only repositories with at least one commit date after 01 Jan 2021 to ensure that we collect bugs from recently updated repositories. Finally, we collect commits from the top 5,000 repositories.

Bug Collection: To collect bugs, we search for PHP projects on GitHub and use the git version control system's history to locate issues and the corresponding solutions developers provided. We

[1] https://docs.github.com/en/rest?apiVersion=2022-11-28

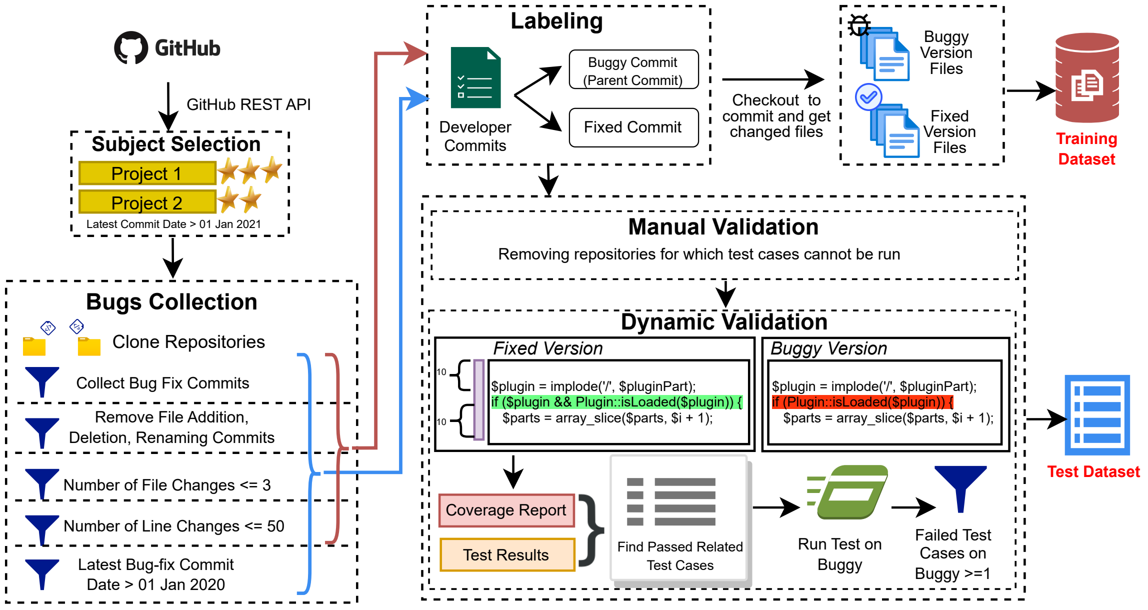
GitHub
GitHub REST API
Subject Selection
Project 1
Project 2
Latest Commit Date > 01 Jan 2021
Bugs Collection
Clone Repositories
Collect Bug Fix Commits
Remove File Addition, Deletion, Renaming Commits
Number of File Changes <= 3
Number of Line Changes <= 50
Latest Bug-fix Commit Date > 01 Jan 2020
Labeling
Developer Commits
Buggy Commit (Parent Commit)
Fixed Commit
Checkout to commit and get changed files
Buggy Version Files
Fixed Version Files
Training Dataset
Manual Validation
Removing repositories for which test cases cannot be run
Dynamic Validation
Fixed Version
Buggy Version
$plugin = implode('/', $pluginPart);
if ($plugin && Plugin::isLoaded($plugin)) {
$parts = array_slice($parts, $i + 1);
$plugin = implode('/', $pluginPart);
if (Plugin::isLoaded($plugin)) {
$parts = array_slice($parts, $i + 1);
Coverage Report
Test Results
Find Passed Related Test Cases
Run Test on Buggy
Failed Test Cases on Buggy >=1
Test Dataset

Figure 1: The overview of the process of curating BugsPHP dataset. Blue lines —— and red lines —— represents the filtering criteria for the test data-set and training data-set respectively.

follow previous studies (e.g., [15]) and look for commits with messages containing words like “fix", “solve", “bug", “issue", “problem" or “error". Finally, we apply the following criteria similar to those in [6] to filter out unusable data. a) developer fix is applied only to PHP files b) developer patch should not contain file addition, deletion, or renaming, as such modifications to the source files are usually refactoring effects and can obfuscate the relevant fix c) number of file changes should be 3 or fewer and d) number of line changes should be equal to or less than 50. We then sort through repositories containing the bugs that meet the criteria and have bug fixes committed after 01 Jan 2020. Then, we choose the top 75 most popular repositories from this list to create the *BugsPHP testing dataset*. The remaining repositories are used to select bugs for the *BugsPHP training dataset*.

Dataset Labeling: We obtain the developer commits from the set of repositories selected and designate the commit before the bug-fixing commit as the *buggy version* of the program and the current version as the *fixed version* of the program. Based on the labeled developer commits, we extract the corresponding buggy version and fixed version files from the repositories designated to obtain the training commits to create the *BugsPHP training dataset*. The labeled developer commits obtained from the test dataset repositories are fed to the validation process. In creating the *BugsPHP testing dataset*, we perform manual and dynamic validations on the bugs selected for the testing dataset.

Manual Validation: In this step, we examine the latest bug to validate a repository and see if its fixed version has accompanying test cases that can verify the fix. We use the latest stable PHP version, 8.1, to validate fixed versions with these test cases. We selected PHP version 8.1 since it is the latest stable PHP version at the time of writing. If the latest bug passes this step, we move that repository to the dynamic validation. If not, we repeat the process with the following bug with the next lower PHP version until we find a bug that passes. This validation is done for each repository until we find at least one bug that successfully passes the validation. Thus, we remove repositories for which we do not find at least one bug that can successfully run the validation.

Dynamic Validation: We run all the test cases on the fixed version of the program and generate the test coverage report. Then, using the test coverage report, we identify the relevant test cases that reach the fixed version’s modified location(s). Next, we select the appropriate passing test cases covering the modified fix locations within a neighborhood of 10 lines. Next, we run selected relevant test cases on the buggy version of the program to identify at least one test case that fails for that buggy version. Finally, we choose the bugs with at least one failing test case. These validation steps result in a testing dataset of 513 bugs, each consisting of the buggy version, the fixed version, and the corresponding test cases. The details of the *BugsPHP testing dataset* are shown in Table 1.

Training Data: We also curated a training dataset following similar criteria for subject selection and bug collection in bug dataset construction. In Subject selection, we used GitHub REST API[2] to retrieve the GitHub repositories. Following steps similar to those previously taken, we selected the first 5k repositories and removed the repositories included in our bug dataset to curate a training dataset. In the bugs collection step, we collected the bug fix commits by checking `"fix"`, `"solve"`, `"bug"`, `"issue"`, `"problem"` or `"error"` keywords included in the commit message. Then, we filtered the bug fixes with the number of file changes less than or equal to 3, and the number of line changes less than or equal to 50. Then, we extracted the changed files and labeled them buggy

[2]https://docs.github.com/en/rest?apiVersion=2022-11-28

Table 1: Overview of BugsPHP test dataset of PHP applications. BugsPHP tests were taken from the fix commit of the subject, which corresponds to a passing test-case with the fix-commit. The test coverage columns summarize the coverage information typically having a high coverage in terms of lines and functions.

Subject		Repository Information			Test Coverage			LOC	# Bugs	# Tests
Name	Description	Stars	Forks	Commits	Lines	Functions	Classes			
cakephp	web application framework	8.6k	35k	44.6k	84.69%	73.59%	36.92%	358k	33	8322
carbon	API extension for DateTime	16.1k	12k	3.4k	99.98%	99.75%	98.25%	316k	11	5898
composer	dependency manager	27.6k	65k	12.0k	63.34%	49.35%	14.53%	107k	18	2240
dbal	database abstraction layer	9k	12k	10.9k	65.52%	55.72%	29.28%	81k	9	4122
easywechat	WeChat API for PHP applications	10k	24k	2.1k	56.02%	50.00%	20.25%	14k	9	178
framework	web application framework	29.3k	99k	35.4k	74.91%	69.09%	32.28%	355k	94	8425
google-api-php-client	Google API for PHP applications	8.5k	35k	1.8k	66.60%	55.56%	11.76%	11k	3	228
laravel-permission	permission manager	11.2k	17k	1.3k	93.72%	80.29%	53.85%	8k	6	432
magento2	e-commerce platform	10.6k	92k	135.3k	46.15%	37.66%	29.52%	2854k	23	16224
monolog	library for logging	20.3k	19k	2.6k	63.98%	52.97%	23.85%	27k	7	1139
orm	object relation mapper	9.6k	25k	13.1k	84.27%	68.01%	45.09%	187k	15	3706
PHP-CS-Fixer	linter for coding standards	11.9k	15k	8.8k	93.97%	87.45%	64.85%	225k	82	31774
PHP-Parser	static analyzer for PHP	16.1k	0.9k	1.4k	92.69%	91.29%	85.90%	30k	3	1691
PhpSpreadsheet	library for spereadsheet files	12.1k	30k	4.0k	80.01%	77.21%	52.25%	251k	12	14071
symfony	web application framework	28.2k	90k	63.9k	80.66%	65.67%	36.09%	1579k	188	41083
Total / Average		229.1k	570.9k	340.6k	76.43%	67.57%	42.31%	6.4M	513	139k

and fixed. We collected 653,606 bugs from 4483 PHP applications. Finally, we included a metafile that contains the commit ID, repository, changed line numbers, etc, for each data point.

4.2 Overview of the BugsPHP Dataset

We analyzed the size of the fix commits in terms of the number of line and file changes to understand the complexities of the fix commits. Most commits modify only 1 file, with 76.3% of the training dataset (498940 fix commits) and 92.2% of the testing dataset (473 bugs) representing single file changes. The percentage of fix commits with 2 and 3 file changes in the training dataset are 14.2% and 9.5%, respectively. In the test dataset, 5.8% of the dataset (30 bugs) have two file changes, whereas 2% of the dataset (10 bugs) have three file changes. A similar pattern exists for the number of lines of the commits. In the training dataset, 67.1% of commits have 1-10 line changes; in the testing dataset, 68% of bugs fall within this range. Also, the training dataset contains 101776 fix commits (15.6%) with 11-20 line changes, 50195 fix commits (7.7%) with 21-30 line changes, 31081 fix commits (4.8%) with 31-40 line changes, and 30760 fix commits (4.8%) 41-50 line changes.

In addition to the size of the patches, we also analyzed the types of bugs in our test dataset. We identified 462 bugs (90%) as functional errors, while the remaining bugs consist of 16 type errors, 15 security vulnerabilities, 13 compatibility issues, 5 usability issues and 2 performance bugs.

5 PRELIMINARY RESULTS WITH APR

With our training data, we train two learning-based APR models, CURE [7] and RewardRepair [17], and evaluate using 513 bugs in our test data. For each model, we generate 100 candidate patches per bug. Table 2 shows the patch result of each model. Columns N_C, N_V and N_P depicts # bugs a candidate patch was generated, failing test cases are fixed, and that pass all test cases, respectively. Similarly columns N_E and N_I depicts # bugs with a semantically equivalent patch and a identical patch, respectively.

Table 2: Efficacy of APR models in BugsPHP

Model	N_C	N_V	N_P	N_E	N_I
CURE	443	48	11	1	0
RewardRepair	513	103	43	13	6

RewardRepair generates a patch for 103 bugs that pass the failing test cases, but CURE generates patches only for 48 bugs. RewardRepair fixes 43 bugs out of these bugs, and CURE fixes 11 bugs, which passed all test cases. RewardRepair can generate 3-4 line patches, while CURE only generates single line patches. RewardRepair fixes 29 unique while CURE fixes four unique bugs. Both APR models commonly fixed seven bugs. However, none of the APR models could fix 473 bugs in our test dataset. We show that existing APR models can be evaluated using our new data set, and further research is needed to improve the capabilities of existing APR for PHP bugs.

6 CONCLUSION

PHP has been the most popular server-side language for over two decades, yet there is no dataset to study bugs appearing in PHP applications. Hence, we have curated a dataset of bug-fixing commits from the most popular open-source PHP applications to train learning-based repair tools and evaluate program repair techniques. BugsPHP contains a training dataset of 653,606 bug-fixing commits and a test dataset of 513 bugs, which maintains developer-written tests using the PHPUnit testing framework.

Data Set: Our dataset can be accessed via GitHub from the following repository: https://github.com/bugsphp/bugsPHP.git

ACKNOWLEDGEMENT

This work was partially supported by a Singapore Ministry of Education (MoE) Tier3 grant "Automated Program Repair", MOE-MOET32021-0001.

REFERENCES

[1] [n. d.]. W3Techs - World Wide Web Technology Surveys. https://w3techs.com/

[2] Jyoti Kaubiyal Ankit Kumar Jain, Somya Ranjan Sahoo. 2021. Online social networks security and privacy: comprehensive review and analysis. *Complex Intell. Syst.* 7 (october 2021), 2157–2177. https://doi.org/10.1007/s40747-021-00409-7

[3] Quang-Cuong Bui, Riccardo Scandariato, and Nicolás E. Díaz Ferreyra. 2022. Vul4J: A Dataset of Reproducible Java Vulnerabilities Geared towards the Study of Program Repair Techniques. In *Proceedings of the 19th International Conference on Mining Software Repositories* (Pittsburgh, Pennsylvania) *(MSR '22)*. Association for Computing Machinery, New York, NY, USA, 464–468. https://doi.org/10.1145/3524842.3528482

[4] Viktor Csuvik and László Vidács. 2022. FixJS: a dataset of bug-fixing JavaScript commits. In *Proceedings of the 19th International Conference on Mining Software Repositories*. 712–716.

[5] Claire Le Goues, Michael Pradel, and Abhik Roychoudhury. 2019. Automated Program Repair. *Commun. ACM* 62, 12 (nov 2019), 56–65. https://doi.org/10.1145/3318162

[6] Péter Gyimesi, Béla Vancsics, Andrea Stocco, Davood Mazinanian, Arpád Beszédes, Rudolf Ferenc, and Ali Mesbah. 2019. Bugsjs: a benchmark of javascript bugs. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 90–101.

[7] Nan Jiang, Thibaud Lutellier, and Lin Tan. 2021. Cure: Code-aware neural machine translation for automatic program repair. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1161–1173.

[8] René Just, Darioush Jalali, and Michael D Ernst. 2014. Defects4J: A database of existing faults to enable controlled testing studies for Java programs. In *Proceedings of the 2014 international symposium on software testing and analysis*. 437–440.

[9] Claire Le Goues, Neal Holtschulte, Edward K Smith, Yuriy Brun, Premkumar Devanbu, Stephanie Forrest, and Westley Weimer. 2015. The ManyBugs and IntroClass benchmarks for automated repair of C programs. *IEEE Transactions on Software Engineering* 41, 12 (2015), 1236–1256.

[10] Megan Leonhardt. 2019. EARN Equifax to pay $700 million for massive data breach. Here's what you need to know about getting a cut. https://www.cnbc.com/2019/07/22/what-you-need-to-know-equifax-data-breach-700-million-settlement.html

[11] Leonardo Mariani Luca Gazzola, Daniela Micucci. 2019. Automatic Software Repair: A Survey. *IIEEE Trans. Software Eng.* 45 (january 2019), 34–67. https://doi.org/10.1109/TSE.2017.2755013

[12] Fernanda Madeiral, Simon Urli, Marcelo Maia, and Martin Monperrus. 2019. Bears: An extensible java bug benchmark for automatic program repair studies. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 468–478.

[13] Devon H. O'Dell. 2017. The debugging mind-set. Communications of the ACM, 40–45. https://doi.org/10.1145/3052939

[14] Ripon K Saha, Yingjun Lyu, Wing Lam, Hiroaki Yoshida, and Mukul R Prasad. 2018. Bugs. jar: A large-scale, diverse dataset of real-world java bugs. In *Proceedings of the 15th international conference on mining software repositories*. 10–13.

[15] Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. 2019. An empirical study on learning bug-fixing patches in the wild via neural machine translation. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 28, 4 (2019), 1–29.

[16] Ratnadira Widyasari, Sheng Qin Sim, Camellia Lok, Haodi Qi, Jack Phan, Qijin Tay, Constance Tan, Fiona Wee, Jodie Ethelda Tan, Yuheng Yieh, Brian Goh, Ferdian Thung, Hong Jin Kang, Thong Hoang, David Lo, and Eng Lieh Ouh. 2020. BugsInPy: A Database of Existing Bugs in Python Programs to Enable Controlled Testing and Debugging Studies. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 1556–1560. https://doi.org/10.1145/3368089.3417943

[17] He Ye, Matias Martinez, and Martin Monperrus. 2022. Neural program repair with execution-based backpropagation. In *Proceedings of the 44th International Conference on Software Engineering*. 1506–1518.