Defexts: A Curated Dataset of Reproducible Real-World Bugs for Modern JVM Languages

Samuel Benton, Ali Ghanbari, Lingming Zhang
Department of Computer Science, The University of Texas at Dallas, USA 75080
Email: {samuel.benton1, ali.ghanbari, lingming.zhang}@utdallas.edu

***Abstract*—Software engineering studies, such as bug detection, localization, repair, and prediction, often require benchmark bug datasets for their experiments. Few publicly available reproducible bug datasets exist for research consumption. Such datasets which publicly exist tend to be applicable exclusively towards the most popular traditional programming languages (e.g., Defects4J for Java and CoreBench for C). Thus, the creation and widespread usage of bug datasets for other popular modern JVM (Java Virtual Machine) programming languages serve to provide vital resources for software engineering research. This paper introduces Defexts, a family of bug datasets currently containing child datasets for Kotlin (Defexts$_{Kotlin}$) and Groovy (Defexts$_{Groovy}$). Each dataset contains reproducible real-world bugs and their corresponding patches scraped from real-world projects. Our introductory versions of Defexts$_{Kotlin}$ and Defexts$_{Groovy}$ include 225 Kotlin and 301 Groovy bugs and patches. As development of Defexts continues, we aim to include other JVM languages, notably Scala. A video demonstration of Defexts is located at following link: https://youtu.be/lenYcVzRGGQ**

I. Introduction

Public access to consistent, reproducible datasets is critical to furthering reproducible software engineering research. Fields such as automated bug detection [1], localization [2], repair [3], and prediction [4] benefit from benchmark buggy programs. Lacking access to public bug datasets, researchers must either collect reproducible bugs from existing projects or inject artificial bugs into projects [5]. The first option can be time-consuming, may not yield bugs pertinent to the research, or may contain insufficient bugs. The second option can be equally time-consuming and the fabricated bugs may not reflect real-world developer mistakes. For both options, the crafted bug datasets will significantly vary between different research teams and the respective teams' needs. This lack of consistent bug datasets hampers research progress within the software engineering research community.

This problem is exacerbated when these benchmarks need distribution within the research community, whether for the purposes of replicating experiments or comparing different research techniques / tools. Researchers wishing to replicate experiments should use the original experiments' benchmark(s), but these datasets can be inadvertently lost or modified over time. Furthermore, comparing different techniques generally requires said techniques to execute against common datasets. For these reasons, we believe there is a significant need to create new bug datasets and to encourage the widespread usage of such benchmarks within the software engineering research community.

Traditional programming languages, such as Java, already have multiple bug datasets, such as Defect4J [6], Bug.Jar [7], and BugSwarm [8]. In particular, Defect4J, a dataset currently with 395 reproducible Java bugs, has been directly cited in 200+ software engineering papers since its release in 2014, and has had a significant impact on software engineering research. Despite their popularity, other modern JVM languages, such as Kotlin [9], often lack even one widespread bug dataset. Thus, we find it increasingly critical to create bug datasets for these unrepresented JVM languages and share such datasets with the research community to further facilitate software engineering research within these languages.

In this paper, we present (1) a systematic methodology for collecting project bugs and their repair patches from version control repositories such as GitHub and (2) Defexts, a family of real-world bug datasets tailored towards modern JVM languages. Due to its minimalist syntax, Kotlin has become one of the two official languages for Android [10] and has been routinely used in practice (e.g., Uber, Square, Coursera, and Twitter apps). Similarly, Groovy [11] has been proposed as a Java enhancer which provides greater flexibility [12], e.g., by providing Ruby/Python-like constructs. Therefore we decide to curate real-world bug datasets for these two JVM languages for Defexts' initial version, Defexts$_{Kotlin}$ (a Kotlin bug dataset) and Defexts$_{Groovy}$ (a Groovy bug dataset). This paper provides the following contributions:

- Defexts, a family of bug datasets focused towards modern JVM-based programming languages (though extendable to non-JVM languages) complete with an interface allowing users to easily access Defexts' projects.
- Defexts$_{Kotlin}$, a Kotlin-specific subset of Defexts which contains 225 reproducible bugs and patches from 152 real-world Kotlin projects.
- Defexts$_{Groovy}$, a Groovy-specific subset of Defexts which contains 301 reproducible bugs and patches from 170 real-world Groovy projects.

II. Collecting the Defexts Bug Datasets

A. Scraping Public GitHub Projects

Due to our familiarity with GitHub's Search API, we chose to exclusively search for projects within GitHub for Defexts' initial version, though our process can be extended to

other repository hosting services such as BitBucket [13]. By using GitHub's Search API, we examined all public projects hosted by GitHub categorized as Groovy for Defexts_{Groovy} and Kotlin for Defexts_{Kotlin} at the time of collection. For Defexts_{Groovy}, we collected GitHub Groovy projects created from December 31$^{\text{st}}$, 2009 to September 4$^{\text{th}}$, 2018. For Defexts_{Kotlin}, we collected GitHub Kotlin projects created from December 31$^{\text{st}}$, 2009 to September 16$^{\text{th}}$, 2018. We used these date ranges to ensure the capture all respective projects at the time of collection. Due to how GitHub identifies a project's primary language, we additionally capture projects which are currently tagged as one of our target languages but were previously composed of another language at the time of commit (e.g. was exclusively composed of Java in March 2012 and is currently tagged as primarily Kotlin-based).

B. Searching for Potentially Bug-fixing Project Commits

During our collection process, we determined it infeasible to process every commit for every collected project. Thus, we reduced the commit search space by searching for commits matching specific criteria. Following prior work on bug collection [14], we specifically searched projects for commits which (1) contain any of the following repair-related keywords in the commit title or description: `error`, `issue`, `fix`, `repair`, `solve`, `remove`, `problem`, (2) exclusively modified files ending in one the following extensions: `.kt`, `.kts`, `.scala`, `.groovy`, `.java`, and (3) modified exactly one source file with six or less modifications (as prescribed by the commit diff) and modified any number of test files (with no restriction on modification amount). We define source files as any file with '/src/main/' in the filename and test files as any file with '/src/test/' in the filename. Any commit which (1) does not modify a source file or (2) modifies any file not categorized as a source file or test file are additionally discarded.

We now present the reasoning for our collection process. First, since we are looking for commits which fix bugs, we search for the aforementioned repair-related keywords to significantly decrease the number of commits unlikely to patch some bug. Second, we acknowledge that, though some systems are composed entirely in one programming language, most real-world systems are composed of multiple programming languages. As Defexts is focused towards bugs for real-world modern JVM-language systems, we decide it necessary to look for hybrid systems of the most popular JVM languages (Java, Kotlin, Groovy, and Scala) and thus consider the changes in all those JVM languages. Lastly, we limit the number of source file modifications to six and search for modifications in exactly one source file to help ensure modifications directly pertain to patching an underlying bug (e.g., commits with huge changes may intertwine bug fixes with benign changes). While these criteria do limit the type of captured bugs, they also sufficiently limit the commit search space to a reasonably processable number of bug patches. We defer a further explanation and justification of our collection constraints to the Defexts website [15].

In total, we extracted 49,982 buggy commits from 93,321 Kotlin projects and 26,438 buggy commits from 43,576 Groovy projects based on this search criteria (see Table I).

C. Verifying Patches

1) Automated Patch Verification: For each repair commit across project P, we tested the commit (denoted as P_C) and the commit immediately preceding it (denoted as P_{C-1}). We tested each commit via the Gradle (version 4.8) test task (with no daemon) or Maven (version 3.3.9) test task, dependent on the commit's underlying build system. Each commit was also executed against Java JDK 1.8. Thus, commits incompatible with JDK 1.8, Gradle 4.8, Maven 3.3.9, or utilizing other build systems were automatically excluded from Defexts. Due to time considerations, we additionally excluded from our initial dataset any commit which took longer than 60 minutes to clone, compile, or test.

We followed Defects4J's strategy when testing commits, by importing any modified test files within P_C into P_{C-1}. This provides both P_{C-1} and P_C with the same test suite. From this, should the test suite status change from fail to pass at P_{C-1} to P_C, we know the difference between P_C and P_{C-1}, diff $\underline{\text{Diff}}_C$, must successfully repair program P. Commits which are not **fail-pass** patches (that fail the test task at P_{C-1} and pass the test task at P_C) are further excluded from Defexts.

2) Manually Removing False Positive Patches: The final step in our collection process was to verify each fail-pass commit failed due to a failing test suite as opposed to environment misconfigurations or unresolvable build issues. For each entry in Defexts, we (1) executed each project's fail-pass status at least 5 times - removing projects which are never fail-pass and (2) manually inspected that every entry in Defexts fails due test failures - resolving build issues where possible and removing projects with unresolvable build issues.

With our best efforts, all entries within our final dataset contain a failing test suite at P_{C-1} and exhibit a fully passing test suite at P_C.

D. Process Limitations and Specialized Datasets

Though our process captures several hundreds of bug and patches, we recognize our approach is merely a best effort attempt in collecting bug and patches as opposed to a comprehensive attempt. Manipulating the process search criteria and constraints can lead to bug datasets specialized for particular studies and techniques. Allowing commits to modify multiple source files, for example, may prove useful to studies which aim to analyze inter-repair interactions in bug patches. We leave such expansions as future works.

III. Dataset API Interface Details

Defexts is shipped with an interface that allows the users to checkout projects or query project information. Similar to Defects4J [6], we use *bug identifiers* to uniquely identify bugs. A bug identifier is a pair of the form `projectName-bugNo`, where `projectName` is the name of the project in which the

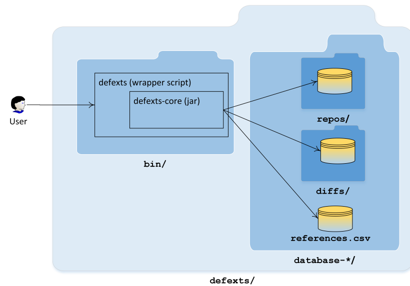

Fig. 1: Architecture and directory structure of Defexts

bug resides, and `bugNo` is an integer identifying a specific bug for the project. In the following subsections, we dissect this interface's architecture and introduce the interface.

A. Architecture and Directory Structure

Fig. 1 depicts a simplified overview of Defexts's architecture in which each component is annotated with the name of the subdirectory that the rcomponent resides in. The dataset for each language is contained within separate "database" directories.

Defexts' interface is a Python script compatible with Python 2+ and Python 3+. Executing Defexts' interface works as follows. The user invokes the script and, based on the passed parameters, the script can show information about the bug(s) or the fix(es), print project diff information, or checkout the appropriate version of a project. All the information about the projects are automatically fed to the interface through a CSV file. This mandatory CSV file contains supplementary information about each Defexts commit, such as each commit's bugID, fix url, and underlying build system(s). As Defexts grows, we aim to increase the details contained in this CSV file.

In an effort maximize the accessibility of the interface, we initially intended to ship the whole Defexts dataset (including all repositories) alongside the interface. However, we realized that some repositories are quite large even after compression. Thus, we followed an approach similar to that of Bugs.Jar [7]. That is, we stored one repository per project (under the subdirectory `defexts/database-*/repos`) and created two branches for each bug: one branch for the bug itself, and the other for its corresponding fix. The branches are named by the commit ID for the fix followed by `-b`, for the buggy version branch, or by `-f`, for the fixed version.

For each previously mentioned buggy branch, we checkout P_{C-1} and checkout all test files modified between P_{C-1} and P_C. The branch for the bugs contains an automatically generated text file containing supplementary information such as the commit URL for the fix and the list of source and test file(s) that are modified between the buggy and fixed versions. To make these modifications persistent, we have made at least one more commit (i.e. `git commit -am "Defexts changes"`) in each buggy branch.

Software engineering researchers sometimes need the exact details of the changes between versions of the same files. Thus, we have included two diff files for each patch: one diff storing the diff of source file(s), and the other containing the diff of

	Defexts_{Kotlin}	Defexts_{Groovy}
Scraped Projects	93,321	43,576
Suitable Commits	49,982	26,438
Fail-Pass Commits	225	301
Unique Projects	152	170

TABLE I: Defexts Collection Statistics

the test file(s). These files are stored under the subdirectory `defexts/database-*/diffs`. We distinguish the files by naming them as `projectName-shaHash.src.patch` and `projectName-shaHash.test.patch`, respectively.

B. Using Defexts' Interface

The users access Defexts' interface through the command-line. The interface to Defexts can be invoked using a command of the following form.

```
python defexts.py -h| <dataset> <options>
```

Where `<dataset>` specifies the dataset to be accessed. Currently, it may be `kotlin` for the Kotlin dataset, or `groovy` for the Groovy dataset. The part denoted by `<options>` can be any of the following switches:

a) `-a` or `--all-projects`: Shows the list of all projects available in the dataset.

b) `-l` or `--list-bugs`: Shows the list of all bugs available in the dataset.

c) `-c` or `--checkout`: Given a bug identifier, this command checks out a bug or its corresponding fix to the current working directory. The user must indicate if they want to checkout the buggy version (using the switch `-b` or `--buggy`) or the fixed version (using the switch `-f` or `--fixed`).

d) `-d` or `--diff`: Given a bug identifier, this command prints out the diff for files modified between the buggy and fixed versions. The user must specify whether they want to print out the diff of the source file(s) (using the switch `-s`) or the test files (using the switch `-t`).

C. Minimum System Requirements

Both the interface and the subject dataset programs are verified to work with JDK 1.8. The interface also needs `git`, Perl 5+, and Python 2.7+ / Python 3+ with the 'prettytable' module to be installed. Finally, all subject programs are verified to work with Gradle 4.8 or Maven 3.3.9. Defexts users may use other versions of these build systems but successful project execution may not be guaranteed.

D. Dataset Statistics

Table I displays the collection statistics for Defexts_{Kotlin} and Defexts_{Groovy}. Row "Scraped Projects" describes the number of projects (`Kotlin` for Defexts_{Kotlin} and `Groovy` for Defexts_{Groovy}) we pulled from GitHub. Row "Suitable Commits" describes the number of commits we collected after performing the filtering processes as described in §II-B. Row "Fail-Pass Commits" describes the number of commits remaining after (1) verifying each "suitable commit" for a fail-pass test suite and (2) filtering out unsuitable commits, described in Section II-C. Row "Unique Projects" describes

the number of distinct projects within each dataset after full completion of the filtering described in Section II.

IV. Related Work

The impact of reproducible bug datasets can be far-reaching within software engineering. Research in bug prediction, detection, localization, and repair all benefit from bug datasets. To the best of our knowledge, no other bug datasets have been designed to collect real bugs for the family of modern JVM languages.

Defects4J [6] is a publicly available Java bug dataset initially published in 2014. Various studies [16], [17] have utilized Defects4J as their bug dataset since Defects4J's publication, allowing for the effective comparison of different techniques and tools. Bugs.Jar [7] is a Java exclusive bug dataset composed of real-bugs, similar to Defects4J. Bugs.Jar contains 1100+ bugs pulled from 8 open-source Java projects. QuixBugs [18] is yet another dataset for Java that has recently attracted the attention of APR research community [19]. The recent BugSwarm dataset [8] includes both Java and Python bugs. Other widely used datasets mainly focus on C/C++, including ManyBugs [20], IntroClass [20], CodeFlaws [21], CoreBench [22], DbgBench [23]. None of the aforementioned datasets focus on modern JVM languages.

V. Future Work and Extensions

While Defexts$_{Kotlin}$ and Defexts$_{Groovy}$ contain a significant number of real-world bugs and their respective patches, researchers can always create their own specialized datasets. The current search criteria provide a strong basis for capturing isolated bug patches, but these criteria may not necessarily be the most optimal. As an example, our current methodology and search criteria exclusively captures bug repairs enclosed within one source file. By weakening the search constraints or even by using alternative criteria, researchers can create alternative versions of Defexts$_{Kotlin}$ and Defexts$_{Groovy}$ with new entries more suitable for their needs. In addition to encouraging researchers to create their own specialized datasets, we hope to allow researchers the ability to directly contribute to existing Defexts datasets.

In our process, we decided to scrape projects from GitHub. Repositories from other version control systems, e.g. BitBucket [13], can replace or coexist alongside GitHub repositories. We leave the integration of additional version control systems as a future work.

Our process for collecting and processing fail-pass patches is completely independent of the queried programming language. Thus, other researchers can utilize the process to capture other programming languages, such as Scala [24], as they see fit.

VI. Conclusion

This paper presents Defexts, a family of reproducible real-world bug datasets for modern JVM languages. In particular, we present a bug dataset for Kotlin (Defexts$_{Kotlin}$) and a bug dataset for Groovy (Defexts$_{Groovy}$) each containing bugs collected from public GitHub projects. For Defexts$_{Kotlin}$, we ultimately collected 225 repairable bugs from 152 projects. For Defexts$_{Groovy}$, we ultimately collected 301 repairable bugs from 170 projects.

Though several bug datasets already exist for traditional programming languages, Defexts aims to supply software engineering researchers the bug datasets necessary to further their research (in fields such as bug prediction, detection, localization, and repair) and publish more reliable results. Further information about Defexts can be found online [15].

References

[1] S. Wang, D. Chollak, D. Movshovitz-Attias, and L. Tan, "Bugram: bug detection with n-gram language models," in ICSE, 2016.

[2] S. Pearson, J. Campos, R. Just, G. Fraser, R. Abreu, M. D. Ernst, D. Pang, and B. Keller, "Evaluating and improving fault localization," in ICSE, 2017.

[3] C. L. Goues, T. Nguyen, S. Forrest, and W. Weimer, "Genprog: A generic method for automatic software repair," IEEE TSE, January 2012.

[4] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems," IEEE TSE, April 2005.

[5] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," IEEE TSE, Sept 2010.

[6] R. Just, D. Jalali, and M. D. Ernst, "Defects4j: A database of existing faults to enable controlled testing studies for java programs," in ISSTA, 2014.

[7] R. K. Saha, Y. Lyu, W. Lam, H. Yoshida, and M. R. Prasad, "Bugs.jar: A large-scale, diverse dataset of real-world java bugs," in MSR, 2018.

[8] D. University of California, "Bugswarm." http://www.bugswarm.org/.

[9] JetBrains, "Kotlin programming language." https://kotlinlang.org/.

[10] Jetbrains, "Kotlin on android. now official." https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/, May 2017.

[11] Apache, "Groovy." http://groovy-lang.org/.

[12] "The apache groovy programming language - ecosystem." http://groovy-lang.org/ecosystem.html.

[13] Atlassian, "Bitbucket - the git solution for professional teams." https://bitbucket.org/.

[14] S. Kim, T. Zimmermann, E. J. Whitehead Jr, and A. Zeller, "Predicting faults from cached history," in ICSE, IEEE Computer Society, 2007.

[15] "Defexts." http://www.github.com/defexts/defexts.

[16] T. Durieux, M. Martinez, M. Monperrus, R. Sommerard, and J. Xuan, "Automatic repair of real bugs: An experience report on the defects4j dataset," CoRR, 2015.

[17] V. Sobreira, T. Durieux, F. Madeiral, M. Monperrus, and M. A. Maia, "Dissection of a bug dataset: Anatomy of 395 patches from defects4j," in SANER, 2018.

[18] D. Lin, J. Koppel, A. Chen, and A. Solar-Lezama, "Quixbugs: A multi-lingual program repair benchmark set based on the quixey challenge," in OOPSLA, pp. 55–56, ACM, 2017.

[19] H. Ye, M. Martinez, and M. Monperrus, "A comprehensive study of automatic program repair on the quixbugs benchmark," CoRR, 2018.

[20] C. Le Goues, N. Holtschulte, E. K. Smith, Y. Brun, P. Devanbu, S. Forrest, and W. Weimer, "The ManyBugs and IntroClass benchmarks for automated repair of C programs," IEEE TSE, December 2015.

[21] S. H. Tan, J. Yi, Yulis, S. Mechtaev, and A. Roychoudhury, "Codeflaws: a programming competition benchmark for evaluating automated program repair tools," in ICSE, May 2017.

[22] M. Böhme and A. Roychoudhury, "Corebench: Studying complexity of regression errors," in ISSTA, 2014.

[23] M. Böhme, E. O. Soremekun, S. Chattopadhyay, E. Ugherughe, and A. Zeller, "Where is the bug and how is it fixed? an experiment with practitioners," in FSE, pp. 117–128, 2017.

[24] "The scala programming language." https://www.scala-lang.org/.