Defects4J as a Challenge Case for the Search-Based Software Engineering Community

Gregory Gay[1] and René Just[2]

[1] Chalmers and the University of Gothenburg, Gothenburg, SE, greg@greggay.com
[2] University of Washington, Seattle, WA, USA, rjust@cs.washington.edu

Abstract. Defects4J is a collection of reproducible bugs, extracted from real-world Java software systems, together with a supporting infrastructure for using these bugs. Defects4J has been widely used to evaluate software engineering research, including research on automated test generation, program repair, and fault localization. Defects4J has recently grown substantially, both in number of software systems and number of bugs. This report proposes that Defects4J can serve as a benchmark for Search-Based Software Engineering (SBSE) research as well as a catalyst for new innovations. Specifically, it outlines the current Defects4J dataset and infrastructure, and details how it can serve as a challenge case to support SBSE research and to expand Defects4J itself.

Keywords: Software Faults, Research Infrastructure, Research Benchmarks

1 Introduction

Each year, the Symposium on Search-Based Software Engineering (SSBSE) hosts a Challenge Track. This track presents a series of challenge cases, often centered around particular software systems or research domains, and tasks researchers with applying their tools, techniques, and algorithms to those challenge cases. The Challenge Track has attracted great attention and competition, and has been a powerful mechanism for highlighting the applicability of state-of-the-art SBSE research to complex, real-world problems [5,6,7,13,17,23].

This report proposes that Defects4J can serve as a compelling challenge case for future editions of SSBSE. Defects4J is a collection of reproducible bugs, extracted from real-world Java software systems, together with a supporting infrastructure for using these bugs [9]. The current version (v2.0.0) contains 835 bugs from 17 Java software systems as well as supporting infrastructure for conducting experiments in software testing and debugging research. For example, Defects4J has been used to evaluate automated test generation [18,20], automated program repair [11,12], and fault localization [16] research. Furthermore, past contributions to the SSBSE Challenge Track have expanded the Defects4J dataset [2,5,6].

The inclusion of Defects4J in the SSBSE Challenge Track can serve as a benchmark for SBSE research as well as a catalyst for new innovations. This report outlines the current version of Defects4J (Section 2), and details how it can support SBSE research and inspire extensions to Defects4J itself (Section 3).

2 Defects4J

Defects4J is an extensible collection of reproducible bugs from Java software systems, together with a supporting infrastructure, and aims at advancing software engineering research [9]. Defects4J is available at: `https://defects4j.org`

The Bugs: The current version of Defects4J (v2.0.0) targets Java 8 and consists of 835 reproducible bugs from 17 projects: Chart (26 bugs), Cli (39), Closure (174), Codec (18), Collections (4), Compress (47), Csv (16), Gson (18), JacksonCore (26), JacksonDatabind (112), JacksonXml (6), Jsoup (93), JxPath (22), Lang (64), Math (106), Mockito (38), and Time (26). The 835 bugs span more than a decade of development history, and the 17 projects span a wide range of domains, including compilers, parsers, testing infrastructure, and a variety of libraries.

Each bug in Defects4J has the following three properties:

1. Each bug consists of a buggy and a fixed source code version. The fixed version is explicitly labeled as a fix to an issue reported in the project's issue tracker, and the changes imposed by the fix must be to source code, not to other project artifacts such as configuration or build files.
2. Each bug is reproducible: all tests pass on the fixed version and at least one of those tests fails on the buggy version, thereby exposing the bug.
3. Each bug is isolated: the buggy and the fixed version differ only by a minimal set of changes, all of which are related to the bug. That is, the difference is free of unrelated code changes, such as refactoring or feature additions.

For each bug, Defects4J provides the following artifacts and metadata:

- A pair of source code versions—the buggy and the fixed version.
- A set of classes and source-code lines modified by the patch that fixes the bug.
- A set of developer-written tests that expose the bug—called "trigger tests".
- A stacktrace for each trigger test, when executed on the buggy version.
- A set of classes loaded by the classloader during execution of the trigger tests.
- A set of tests that are relevant to the bug—tests that load at least one class modified by the patch that fixes the bug.

Supporting Infrastructure: Defect4J offers a command-line utility to execute a set of common tasks for each bug, including the following: print information about a bug, checkout a buggy or fixed source code version, compile a source code version, execute tests and (optionally) monitor the classloader, perform coverage or mutation analysis, export metadata such as classpaths, directories, or sets of tests, and a utility that queries the metadata to support automated analyses.

By default, Defects4J commands use the developer-written tests that come with each project. However, each command can be executed for an arbitrary JUnit test suite, including those created by automated test generation tools. Defects4J offers a uniform interface for automated test generation. Concrete instantiations of that interface are provided for EvoSuite [4], a search-based test generator, and Randoop [14], a feedback-directed random test generator. Defects4J provides a template to ease incorporation of additional test generators into the infrastructure. Coverage and mutation analyses are provided through Cobertura [1] and the Major mutation framework [8], respectively.

Expanding Defects4J: Defects4J can be expanded along different dimensions. First, testing and debugging tools can be integrated into the supporting infrastructure, through well-defined interfaces.

Second, a semi-automated process[3] supports mining candidate bugs from an existing project's version control system and issue tracker. This process requires the creation of a meta build file that can compile any source code version of the project (generally by calling that version's existing build script). An automated step mines candidate bugs by cross-referencing the project's version control history with its issue tracker, identifying commits that fix a reported and closed issue. This step also compares each fix commit with its predecessor commit to determine whether at least one trigger test exists that reliably passes on the fixed version and fails on the buggy version. Each reproducible bug is then subject to a manual minimization process that eliminates irrelevant code changes (i.e., refactoring or feature additions). Finally, an automated step adds all reproducible, minimized bugs to the dataset and computes their metadata.

3 Research Challenges

Defects4J can serve as a challenging and diverse benchmark for SBSE research as well as a catalyst for new innovations. Past research has successfully used Defects4J to evaluate software testing and debugging approaches. For example, past research used Defects4J to assess the effectiveness of automated test generation and corresponding fitness functions [20], automated program repair [11], and fault localization [16].

This section outlines three concrete areas in which Defects4J can serve as a challenge case. These areas and their corresponding challenges do not form an exhaustive set, but are intended to provide inspiration. Defects4J can be used to validate and extend work presented in prior editions of SSBSE or to explore new SBSE-based approaches.

3.1 Empirical Validation

Genetic Improvement: Search-based approaches have been used to improve system performance [17]. These performance improvements should be semantics preserving—that is, not alter the functional behavior of a system. Defects4J can support the empirical validation of performance improvement research by providing a supporting infrastructure for integrating tools and a thorough set of tests for assessing generated patches.

Hyper-Parameter Tuning: Search-based approaches often have many parameters that can, and need to, be tuned to increase effectiveness on particular problem instances. Examples include crossover and mutation rates of a genetic algorithm [22]. A fair and comprehensive assessment of automated tuning processes requires a well-defined dataset. Defects4J provides such a dataset, fosters replicability of experiments, and supports validation of in-project and cross-project generalization of hyper-parameter settings.

Longitudinal Studies and Software Evolution: Software evolves over time, and Defects4J's artifacts capture this evolution. With a diverse set of projects and multiple artifacts per project, spanning multiple years of development history, Defects4J supports longitudinal studies that, e.g., investigate the effectiveness and generalizability of SBSE approaches over time.

[3] The entire process is documented at `https://github.com/rjust/defects4j/tree/master/framework/bug-mining/README.md`.

Test Suite Diversity: Prior work hypothesized that diverse test suites are more effective than those that contain similar tests [21]. Defects4J supports controlled experiments that can assess the impact of increased or decreased diversity on fault detection. The use of a common dataset also offers common grounds for researchers to compare different diversification techniques.

3.2 Novel SBSE Techniques

Predictive Modeling: Search-based approaches can be used to tackle prediction problems, such as defect prediction [19]. The 835 bugs in Defects4J correspond to over 1000 buggy Java classes. Features of those classes and the isolated bugs themselves can be used to develop or train predictive models.

State Space Exploration in Program Repair: Search-based approaches for automated program repair often have difficulty traversing the search space due to a costly fitness evaluation. Recent work addressed this challenge by capturing dependencies between source code statements and high-level differences between patch variants [3]. Defects4J is a natural benchmark for novel program repair research, as the dataset includes both buggy and fixed versions of a variety of complex bugs.

Topic Modeling: Recent work examined the extraction of information from textual artifacts, such as bug reports [10,15]. Defects4J provides many examples for topic modeling and can serve as a basis for linking code and textual artifacts, since each bug in Defects4J is linked to a bug report and the projects and classes themselves often include detailed documentation.

The topics above capture only a portion of the research presented at SSBSE 2019. Defects4J can also support other research areas, such as **crash reproduction** (Defects4J provides trigger tests, detailed stacktraces, and isolated bugs, all of which can be used to assess crash-reproduction approaches) or **mutation testing** (Defects4J's real bugs can serve as templates for evolving new mutation operators and generation techniques).

3.3 Extending Defects4J

In addition to providing a challenging benchmark for assessing SBSE research, extending Defects4J also poses interesting challenges that may inspire new approaches. In particular, researchers may wish to consider the following:

- **Automated build script creation and repair:** Adding bugs to Defects4J currently requires the manual creation of a meta build file. Genetic programming could be used to automate this step. Some of the associated challenges include gathering dependencies and inferring necessary properties to build each source code version.
- **New projects and bugs:** The set of software systems in Defects4J is diverse, but many domains are not accounted for. New projects and additional bugs would increase the range of research that could be supported. Of particular interest are AI-based systems, concurrent systems, and systems in difficult-to-test domains.
- **New test generation approaches:** Defects4J includes a standardized interface for automated test generation. This interface allows researchers to quickly integrate and evaluate new approaches.
- **Interface for automated patch generation:** Similar to the standardized interface for test generation, Defects4J would benefit from a well-defined interface for tools

that generate patches, including tools for automated program repair and genetic improvement. For example, tools for automated program repair are routinely benchmarked on Defects4J [11]. A standardized interface for executing such tools would facilitate reproducibility and comparability.
– **Support for newer Java versions:** The current version of Defects4J supports Java 8. Newer version of Java may result in unexpected compilation errors, test failures, or metadata inconsistencies. Novel approaches for automatically migrating source code and tests to newer Java versions would be a welcome contribution.

In addition to the examples above, researchers may identify other needs and challenges that require attention to support different research areas or experimental protocols.

4 Conclusions

This report proposes that Defects4J can serve as a challenge case for SBSE research as well as a catalyst for new innovations. Among the topics explored in work published at the 2019 Symposium on Search-Based Software Engineering, Defects4J could benefit research in genetic improvement, hyper-parameter tuning, test suite diversity and generation, predictive modeling, state space exploration, and topic modeling. Furthermore, extending Defects4J poses additional challenges that can be tackled through SBSE.

Acknowledgements

This material is based upon work supported by the National Science Foundation under grants CNS-1823172 and CCF-1942055.

References

1. Cobertura. https://cobertura.github.io/cobertura/
2. Almulla, H., Salahirad, A., Gay, G.: Using search-based test generation to discover real faults in Guava. In: Proceedings of the Symposium on Search-Based Software Engineering. SSBSE 2017, Springer Verlag (2017)
3. Dantas, A., de Souza, E.F., Souza, J., Camilo-Junior, C.G.: Code naturalness to assist search space exploration in search-based program repair methods. In: Nejati, S., Gay, G. (eds.) Search-Based Software Engineering. pp. 164–170. Springer International Publishing, Cham (2019)
4. Fraser, G., Staats, M., McMinn, P., Arcuri, A., Padberg, F.: Does automated unit test generation really help software testers? a controlled empirical study. ACM Trans. Softw. Eng. Methodol. 24(4), 23:1–23:49 (Sep 2015), http://doi.acm.org/10.1145/2699688
5. Gay, G.: Challenges in using search-based test generation to identify real faults in mockito. In: Search Based Software Engineering: 8th International Symposium, SSBSE 2016, Raleigh, NC, USA, October 8-10, 2016, Proceedings. pp. 231–237. Springer International Publishing, Cham (2016), http://dx.doi.org/10.1007/978-3-319-47106-8_17
6. Gay, G.: Detecting real faults in the Gson library through search-based unit test generation. In: Proceedings of the Symposium on Search-Based Software Engineering. SSBSE 2018, Springer Verlag (2018)
7. Harman, M., Jia, Y., Langdon, W.B.: Babel pidgin: Sbse can grow and graft entirely new functionality into a real world system. In: Le Goues, C., Yoo, S. (eds.) Search-Based Software Engineering. pp. 247–252. Springer International Publishing, Cham (2014)

8. Just, R.: The major mutation framework: Efficient and scalable mutation analysis for java. In: Proceedings of the 2014 International Symposium on Software Testing and Analysis. pp. 433–436. ISSTA 2014, ACM, New York, NY, USA (2014), http://doi.acm.org/10.1145/2610384.2628053
9. Just, R., Jalali, D., Ernst, M.D.: 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. pp. 437–440. ISSTA 2014, ACM, New York, NY, USA (2014), http://doi.acm.org/10.1145/2610384.2628055
10. Just, R., Parnin, C., Drosos, I., Ernst, M.D.: Comparing developer-provided to user-provided tests for fault localization and automated program repair. In: Proceedings of the International Symposium on Software Testing and Analysis (ISSTA). pp. 287–297 (July 2018)
11. Martinez, M., Durieux, T., Sommerard, R., Xuan, J., Monperrus, M.: Automatic repair of real bugs in java: a large-scale experiment on the defects4j dataset. Empirical Software Engineering 22(4), 1936–1964 (2017), https://doi.org/10.1007/s10664-016-9470-4
12. Motwani, M., Soto, M., Brun, Y., Just, R., Le Goues, C.: Quality of automated program repair on real-world defects. IEEE Transactions on Software Engineering (June 2020)
13. de Oliveira Barros, M., de Almeida Farzat, F.: What can a big program teach us about optimization? In: Ruhe, G., Zhang, Y. (eds.) Search Based Software Engineering. pp. 275–281. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
14. Pacheco, C., Ernst, M.D.: Randoop: Feedback-directed random testing for java. In: Companion to the 22Nd ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications Companion. pp. 815–816. OOPSLA '07, ACM, New York, NY, USA (2007), http://doi.acm.org/10.1145/1297846.1297902
15. Panichella, A.: A systematic comparison of search algorithms for topic modelling—a study on duplicate bug report identification. In: Nejati, S., Gay, G. (eds.) Search-Based Software Engineering. pp. 11–26. Springer International Publishing, Cham (2019)
16. Pearson, S., Campos, J., Just, R., Fraser, G., Abreu, R., Ernst, M.D., Pang, D., Keller, B.: Evaluating and improving fault localization. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). pp. 609–620 (2017)
17. Petke, J., Brownlee, A.E.I.: Software improvement with gin: A case study. In: Nejati, S., Gay, G. (eds.) Search-Based Software Engineering. pp. 183–189. Springer International Publishing, Cham (2019)
18. Rueda, U., Just, R., Galeotti, J.P., Vos, T.E.: Unit testing tool competition: round four. In: Proceedings of the International Workshop on Search-Based Software Testing (SBST). pp. 19–28 (May 2016)
19. Sarro, F.: Search-based predictive modelling for software engineering: How far have we gone? In: Nejati, S., Gay, G. (eds.) Search-Based Software Engineering. pp. 3–7. Springer International Publishing, Cham (2019)
20. Shamshiri, S., Just, R., Rojas, J.M., Fraser, G., McMinn, P., Arcuri, A.: Do automatically generated unit tests find real faults? an empirical study of effectiveness and challenges. In: Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). ASE 2015, ACM, New York, NY, USA (2015)
21. Vogel, T., Tran, C., Grunske, L.: Does diversity improve the test suite generation for mobile applications? In: Nejati, S., Gay, G. (eds.) Search-Based Software Engineering. pp. 58–74. Springer International Publishing, Cham (2019)
22. Zamani, S., Hemmati, H.: Revisiting hyper-parameter tuning for search-based test data generation. In: Nejati, S., Gay, G. (eds.) Search-Based Software Engineering. pp. 137–152. Springer International Publishing, Cham (2019)
23. Zhang, Y., Harman, M., Jia, Y., Sarro, F.: Inferring test models from kate's bug reports using multi-objective search. In: Barros, M., Labiche, Y. (eds.) Search-Based Software Engineering. pp. 301–307. Springer International Publishing, Cham (2015)