Editor: **Jeffrey C. Carver**
University of Alabama
carver@cs.ua.edu

Automatic Program Repair

Jeffrey Carver, Ricardo Colomo-Palacios, Xabier Larrucea, and Miroslaw Staron

FOLLOWING ALONG WITH the theme of this issue of *IEEE Software*, this column reports on papers about automatic program repair (APR) from the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE20), the 35th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW20), and the 13th IEEE International Conference on Software Testing, Validation and Verification (ICST20). Feedback or suggestions are welcome. In addition, if you try or adopt any of the practices included in the column, please send us and the authors a note about your experiences.

In addition, if you try or adopt any of the practices included in the column, please send us and the authors a note about your experiences.

Digital Object Identifier 10.1109/MS.2021.3071567
Date of current version: 18 June 2021

Antipatterns for Java

"Antipatterns for Java Automated Program Repair Tools" by Yi Wu analyzes plausible patches, that is, patches that produce correct outputs for all inputs in the test suite but may fail beyond the test suite, for Java code generated by automated repair tools such as SimFix, CapGen, and LSRepair to identify deficiencies in these patches. The author manually identifies antipatterns, a set of forbidden code transformations, in these plausible patches and applies antipatterns to improve repair performance. The paper integrates antipatterns in jGenProg2 and evaluates them on the Defects4J benchmark. Concerning the number of plausible patches, the original jGenProg2 and the jGenProg2 with antipatterns integrated produced 67 and 29 patches respectively for 14 Defects4J bugs, showing a reduction of 38 plausible patches. The average repair time for these 14 Defects4J bugs is reduced by 22.6%. The study provided evidence about the effectiveness of applying antipatterns in future Java automated repair tools. This paper appears in the ASE20 conference proceedings. Access it at http://bit.ly/PD-2021-July-01.

Automating Patches: Dynamic and Static

"Automated Patch Correctness Assessment: How Far are We?" by Shangwen Wang and colleagues presents the results of an empirical study on the effectiveness of automated patch correctness assessment techniques, including both static and dynamic approaches. This paper addresses plausible patches that are considered overfitting patches, that is, they do not fix the target bug. APR tools face the overfitting problem because they generate more overfitting patches than correct patches,

resulting in low precision. The authors assessed 902 patches automatically generated by 21 APR tools. This analysis shows that while static approaches (for example, ssFix, CapGen, and S3) are appropriate for identifying some overfitting patches (53.5%), dynamic approaches help with identifying and solving these patches and have higher precision (93.3%). More specifically, the authors analyzed dynamic tools requiring an oracle (for example, Evosuite, Randooop, DiffTGen, and Daikon) and without an oracle (for example, Patch-sim, e-patch-sim, R-Opad, and e-pad). Finally, authors designed a strategy to integrate static code features via learning and then combine those results with others through majority voting. This paper appears in the ASE20 conference proceedings. Access it at http://bit.ly/PD-2021-July-02.

Dealing With Strings

"No Strings Attached: An Empirical Study of String-Related Software Bugs" by Aryaz Eghbali and Michael Pradel argues the lack of knowledge about string-related bugs can lead developers to repeat the same mistakes. This problem is relevant primarily in languages where strings play a critical role such as JavaScript. However, it is also critical in other contexts like building database queries or in reflection-like access of an object property based on the property name. This paper describes a study of 204 string-related bugs in JavaScript from 13 popular open source projects. The results of this study show almost all of these bugs (95.6%) result from one or more recurring root causes, that is, bugs in string literals and bugs in regular expressions (42% and 37%, respectively). There are other root causes such as the incorrect usage of string APIs (13%) and comparison and operations involving strings (6%). These bugs result in incorrect output (30%) or file corruption (5%). Only 11% of the bugs generate an error message. In addition, the authors suggest that clever test oracles must be defined and automated code analysis tools must be used at different stages. String-related bugs are spread over the entire software system, and 53% of the bugs are affecting the core functionality of the projects. Their empirical study reveals that 61% of the bugs can be solved by modifying a single line of code, and 25% by using tokens found close to the bug location. This paper appears in the ASE20 conference proceedings. Access it at http://bit.ly/PD-2021-July-03.

Unified Debugging

"On the Effectiveness of Unified Debugging: An Extensive Study on 16 Program Repair Systems" by Samuel Benton and colleagues describes the results of a study on unified debugging, that is, a new debugging methodology that unifies fault localization and repair. More specifically, unified debugging utilizes the patch-execution results from repair systems to help improve state-of-the-art fault localization. In this way, unified debugging not only improves fault localization for manual repair but also extends the application scope of automate repair to all bugs. This study of 16 APR systems, including jKali, SimFix, and PraPR, reveals various practical guidelines for unified debugging: 1) nearly all of the 16 studied repair systems can positively contribute to unified debugging despite their varying repair capabilities; 2) repair systems targeting multiedit patches can introduce extraneous noise into unified debugging; 3) repair systems with more executed/plausible patches tend to perform better for debugging; and 4) unified debugging effectiveness does not rely on the availability of correct patches in automated repair. This paper appears in the ASE20 conference proceedings. Access it at http://bit.ly/PD-2021-July-04.

Coding Patterns and Code Review

"Characterizing Colocated Insecure Coding Patterns in Infrastructure as Code Scripts" by Farzana Ahamed Bhuiyan and Akond Rahman presents an empirical study of over 7,000 Puppet Scripts from Mozilla, OpenStack, and Wikimedia to understand insecure coding patterns. Examples of insecure coding patterns that suggest potential weaknesses include use of HTTP without TLS/SSL or using hard-coded or default passwords. Understanding how these insecure coding patterns spread in the infrastructure as code will help practitioners prioritize which code to review. The approach in this paper uses unsupervised machine learning and association rule mining, to find pairs (or triplets) of insecure coding patterns. The results show a significant number of colocated insecure patterns (on the order of thousands). The approach in this paper helps developers identify problematic coding patterns and then focus the manual review effort appropriately. The next step in this work is to automate these reviews and fix the problems, for example, by replacing the HTTP protocol with the HTTPS protocol. The paper also describes which source code metrics can automatically identify the insecure coding patterns—for example, hard-coded strings, the number of attributes, the number of includes or even the simplistic lines-of-code metric.

ABOUT THE AUTHORS

JEFFREY CARVER is a professor in the University of Alabama's Department of Computer Science, Tuscaloosa, Alabama, 35487, USA. Further information about him can be found at http://carver.cs.ua.edu. Contact him at carver@cs.ua.edu.

RICARDO COLOMO-PALACIOS is a professor in the Department of Computer Sciences, Østfold University College, Halden, 1757, Norway. Contact him at ricardo.colomo-palacios@hiof.no.

XABIER LARRUCEA is a main researcher at TECNALIA, Basque Research and Technology Alliance, Derio, Bizkaia, E-48160, Spain. Contact him at xabier.larrucea@tecnalia.com.

MIROSLAW STARON is a professor in the software engineering division, Chalmers University of Technology, and the University of Gothenburg, Gothenburg, SE-412 96, Sweden. Contact him at miroslaw.staron@cse.gu.se.

This paper appears in the ASEW20 conference proceedings. Access it at http://bit.ly/PD-2021-July-05.

Maximizing Oracle Accuracy

"Human-in-the-Loop Automatic Program Repair" by Marcel Böhme, Charaka Geethal, and Van-Thuan Pham presents Learn2Fix, a technique that automatically repairs function bugs even if no automated code exists. Indeed, Learn2Fix's novelty is that it can automatically learn a condition under which the bug is replicated by asking questions to the bug-reporting user. Learn2Fix uses an unbiased committee of automatically created oracles to generate test cases automatically and determine whether these tests pass. The promising results show that Learn2Fix can predict the test case label (that is, whether the automatically generated test case passes or fails) with over 75% accuracy after seeing only one failing test from a labeled test suite. In addition, Learn2Fix can obtain similar proficiency in identifying failing test cases while requiring the human to review fewer of them. This learning approach helps users identify/suggest test cases to trigger user-reported bugs before fixing these bugs. The software developer and the user can then more easily verify the fix. Learn2Fix can be used both by experienced programmers, who know the system well, and junior programmers, who are just learning the system under test. While the technique currently works only on numerical inputs, it should be extensible to other types. This paper appears in the ICST20 conference proceedings. Access it at http://bit.ly/PD-2021-July-06.