WarningsGuru: Integrating Statistical Bug Models with Static Analysis to Provide Timely and Specific Bug Warnings

Louis-Philippe Querel
Department of Computer Science
Concordia University, Montreal, QC, Canada
l_querel@encs.concordia.ca

Peter C. Rigby
Department of Software Engineering
Concordia University, Montreal, QC, Canada
peter.rigby@concordia.ca

ABSTRACT

The detection of bugs in software systems has been divided into two research areas: static code analysis and statistical modeling of historical data. Static analysis indicates precise problems on line numbers but has the disadvantage of suggesting many warning which are often false positives. In contrast, statistical models use the history of the system to suggest which files or commits are likely to contain bugs. These course-grained predictions do not indicate to the developer the precise reasons for the bug prediction. We combine static analysis with statistical bug models to limit the number of warnings and provide specific warnings information at the line level. Previous research was able to process only a limited number of releases, our tool, WarningsGuru, can analyze all commits in a source code repository and we currently have processed thousands of commits and warnings. Since we process every commit, we present developers with more precise information about when a warning is introduced allowing us to show recent warnings that are introduced in statistically risky commits. Results from two OSS projects show that CommitGuru's statistical model flags 25% and 29% of all commits as risky. When we combine this with static analysis in WarningsGuru the number of risky commits with warnings is 20% for both projects and the number commits with new warnings is only 3% and 6%. We can drastically reduce the number of commits and warnings developers have to examine. The tool, source code, and demo is available at https://github.com/louisq/warningsguru.

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**;

KEYWORDS

Static Analysis, Statistical Bug Models, WarningsGuru

ACM Reference Format:
Louis-Philippe Querel and Peter C. Rigby. 2018. WarningsGuru: Integrating Statistical Bug Models with Static Analysis to Provide Timely and Specific Bug Warnings. In *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18), November 4–9, 2018, Lake Buena Vista, FL, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3236024.3264599

ESEC/FSE '18, November 4–9, 2018, Lake Buena Vista, FL, USA

ACM ISBN 978-1-4503-5573-5/18/11…$15.00
https://doi.org/10.1145/3236024.3264599

1 INTRODUCTION

Static analysis tools find defects by examining the code, data-flow, and control-flow for problematic patterns. To make static checking tractable in practice, FindBugs and Jlint make simplifications and abstractions that lead to a large number of warnings many of which can be false positives, including trivial and unlikely warnings [1, 2]. The advantage of static analysis is that the warnings are specific, *e.g.*, a null pointer on a specific source code line. The disadvantage is the overwhelming number of reported warnings [7, 13] and a disconnect between field defects and warnings [3].

In contrast, statistical bug models use historical development information to indicate risky, *i.e.* potentially defective, files or commits [4, 5]. The predictors used in the models include both product measures, such as the number of lines changed in a commit, and the process measures, such as the expertise of the developers modifying a file. The advantage of statistical bug models is that they provide reasonable predictions of field defects in commits and files [4]. The disadvantage is that the prediction is not fine-grained, *i.e.* an entire file or commit is flagged as risky.

We developed a tool called WarningsGuru which combines the two approaches, using statistical models to identify which commits are likely to contain defects and showing the specific line numbers of the new static analysis warnings that have been introduced for the commit. We also develop a technique which traces the warning introducing commit which we use to retroactively assign warnings to failed builds.

2 WARNINGSGURU FEATURES AND ARCHITECTURE

We give an overview of the features of WarningsGuru illustrated by figures. Each feature is implemented as part of the pipeline in Figure 1. WarningsGuru builds on an existing statistical bug modeling tool, CommitGuru [10]; static analysis tools, including FindBugs and Jlint; and static analysis warning integration tool, TOIF [6].

(1) Figure 2 shows for each commit the total number of warnings and new warnings and CommitGuru's statistical risk prediction.
(2) Figure 3 shows the number of existing and new warnings associated with each file for a commit and which measures indicate risk in CommitGuru's model.
(3) Figure 4 shows the warnings per line reported by FindBugs and JLint. WarningsGuru reports the commit in which the warning first appeared and the line which it is presently on. WarningsGuru can retroactively assign warnings to commits that do not build.
(4) Each warning is clickable taking the developer to the highlighted problematic line on GitHub. By clicking the commit hash of the originating commit, the developer will instead be

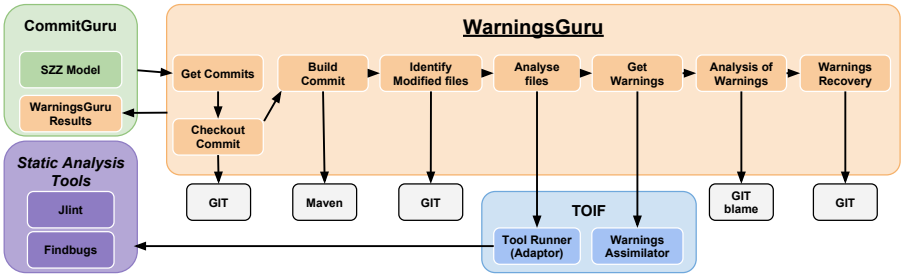

Figure 1: Architecture of the WarningsGuru integration

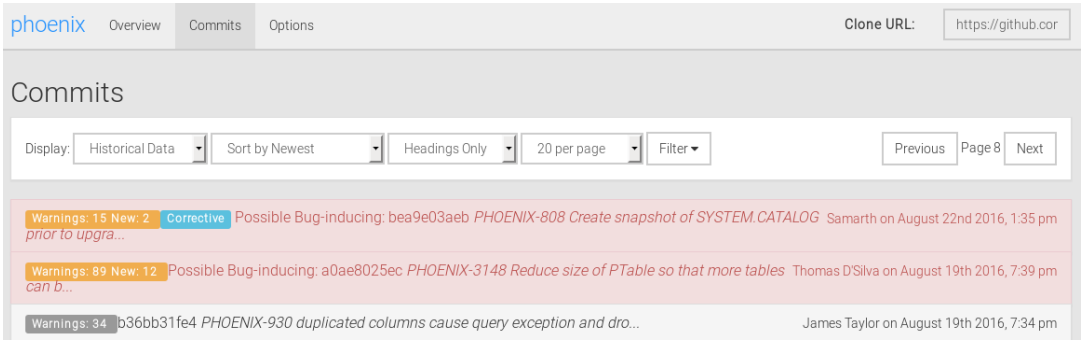

Figure 2: Commits that are considered by the statistical model to be risky are highligted. The number of new and existing warnings are also shown. For example, commit 'b36bb31fe4' is not considered risky even though the changed files have 34 existing warnings. There are no new warnings introduced in this commit.

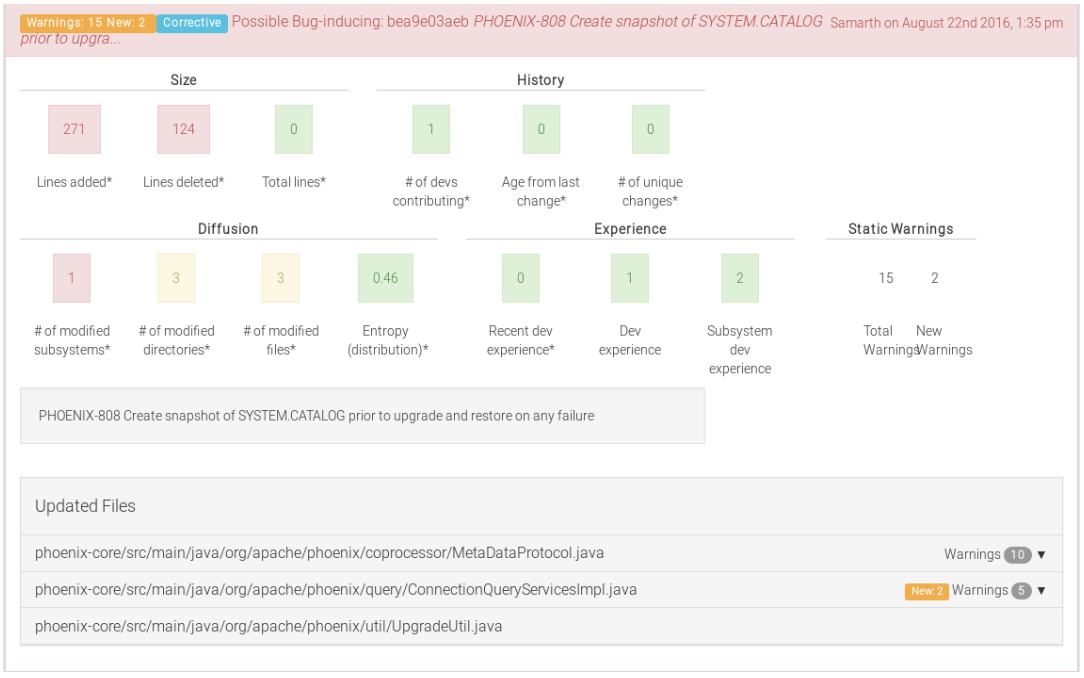

Figure 3: The number of new and existing warnings are shown for the commit and for each modified file. The CommitGuru measures are shown with anomalous values highlighted in red for the commit. For example, "Lines added" is abnormally high and large commits tend to be risky.

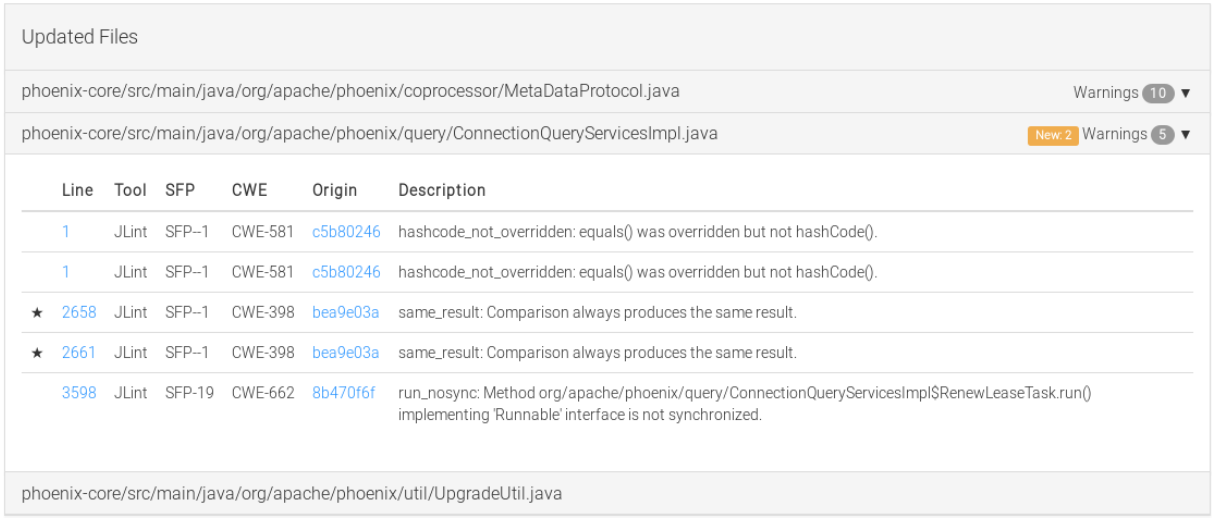

Figure 4: When a file is selected, the warnings are shown for each line number. The star indicates new warnings. The commit that the warning was introduced in is also shown. Clicking on a line takes the user to that line on GitHub.

taken to the original problematic commit on GitHub which introduced the warning (not shown in a figure).

(5) New projects can be added simply by submitting the Git URL of a Maven project hosted on GitHub to WarningsGuru (not shown in a figure).

In the remainder of this paper, we describe each component in detail, provide examples of our integration of statistical bug models with static analysis and discuss planned future work. We conclude the paper with preliminary results that show less than 9% of commits contain new static analysis warnings and less than 6% of statistically risky commits contain new warnings.

3 BUILDING THOUSANDS OF COMMITS

Recent works have shown that combining static analysis with statistical models can increase the defect finding effectiveness of both techniques [9, 11]. However, these research works did not provide developers with a usable tool and were performed on a small number of releases. For example, Rahman *et al.* [9] study between 5 and 8 release for five open source projects for a total of 34 releases. They state that the effort to build and run JLint and FindBugs on the 34 versions took six person-months of effort. One of our major contributions is a Maven based technique that allows us to build and run JLint and FindBugs on thousands of project versions. In this demonstration we process 1.6k and 3.5k commits for the Phoenix and Kylin projects respectively. By processing all the commits for a project, WarningsGuru gives precise and timely information about when a static analysis warning first appeared simplifying future comparisons of statistical and static bug predictions.

Static analysis tools typically require a buildable system to run. To build a each historical commit, WarningsGuru requires a Maven POM file [12]. A POM file stores the build configurations of the project including version numbers of required dependencies allowing us to build historical versions. We automatically modify the POM file to retroactively add static analysis as part of the build process using Maven's *exec-maven-plugin* plugin. We determine which files have been added or modified and run static analysis only on these files making the process scalable across thousands of commits.

Researchers and developers can add new projects to WarningsGuru provided that a Maven POM file exists. For the two sample projects in this demonstration, we find that we process a median of one commit every 69 seconds running on a standard desktop. The processing time depends on the project build time and the number of files that changed. Commits can be processed by in parallel.

Warnings on broken builds. We found that 11% and 15% of commits for Phoenix and Kylin could not be built. We cannot directly generate warnings on a commit that does not build; however, on subsequent commits that are buildable we use Git blame to retroactively assign warnings to lines that originated in files that did not initially build. In practice we can generate static analysis warnings for all commits, even those that do not build. An interesting preliminary finding is that commits that do not build tend to introduce a higher percentage of static analysis warnings. For example, broken builds are 1.8 and 1.6 times more likely to introduce warnings than buildable commits for Phoenix and Kylin, respectively.

4 STATIC ANALYSIS INTEGRATION - TOIF

Each static analysis tool has its own execution flow and method of reporting warnings. We use TOIF, an open source framework developed by our partner KDM Analytics to integrate static analytics tools [6]. TOIF provides a common execution interface for the static analysis tool and parses their results to convert them into a common format. We currently run JLint and FindBugs, which are Java static analysis tools.

TOIF also enriches the warnings by mapping them to software security warnings including the *common weakness enumeration (CWE)* [8] and *software fault pattern (SFP)*[6] categories. This mapping is security focused, allowing developers and future researchers interested in security to ignore warnings that rarely lead to security problems. Figure 4 shows the integrated and additional warning fields in a GUI in WarningsGuru.

5 VERSION CONTROL - GIT

Version control system are used to manage the source code of software projects where incremental changes are stored. WarningsGuru uses Git to retrieve the state of the system at each commit. The system state at each commit is then analyzed for the following four purposes. First, we walk the Git DAG to checkout the state of the system at each commit. WarningsGuru then builds and runs static analysis tools on each commit. Second, running static analysis tool is computationally expensive. We use Git to identify only the files that have been added, removed, or modified for a commit. From these files we are able to determine the new and removed warnings. Third, we use Git blame to determine the commit in which a line in a file was last modified. This allows us to determine the historical commit in which the warning was introduced. Figure 4, developers can click on the "Origin" commit of older warnings or view the lines with new "starred" warnings. Fourth, CommitGuru works at the commit level allowing us to combine its statistical risk measures with the warnings for each commit.

6 STATISTICAL MODELS WITH COMMITGURU

Statistical models indicate when a change may introduce a bug by predicting when a commit or file is prone to being buggy using historical measures such as churn, entropy of change and experience of the developer [5]. By being able to identify bug fixing commits from either an issue tracker or commit message it is possible to determine which commits might have introduced the faulty change. The measures from these candidate buggy commits are then used to identify other commits which may also be bug introducing based on historical patterns.

CommitGuru implements the SZZ/ASZZ algorithm [5]. Bug fixing commits are identified based on keywords in their commit message. Using Git blame, the fixing commit is traced back to the commit that last changed the fixed lines. The lines that have been changed are deemed to be bug introducing. Measures are extracted at each commit and used as predictors in a logistic regression with ten-fold cross validation. This logistic regression model is used to predict whether or not a commit is likely to introduce a bug, *i.e.* is risky.

In Figure 2 we see for each commit whether the statistical model has determined if a change is likely to introduce a bug. We see which measures contribute additional risk. While this allows CommitGuru to determine which commits may be the most risky, it does not give additional details of where the issue might be in the commit as the predictions are not at the line level. For example, this figure shows that the model predicts a change to be risky because many new lines are being added in the commit. WarningsGuru supplements this knowledge with specific static analysis warnings that have been introduced in a change.

7 TOOL EFFECTIVENESS AND CONCLUSIONS

To illustrate the effectiveness of our approach we ran WarningsGuru on two Apache foundation projects: Phoenix and Kylin. Our study is retrospective as neither project uses static analysis or statistical bug prediction. We study 1,602 commits over 23 month period for Phoenix and 3,568 commits over a 14 month period of Kylin. We found that 65% and 49% of commits contain warnings, while only 6% and 9% introduced new warnings. By progressively examining new warnings as changes are made developers need examine only a small number of warnings and commits.

By using a statistical model alone, we find that CommitGuru flags 25% and 29% of commits as risky. When we combine this with static analysis in WarningsGuru the number of risky commits with warnings is 20% for both projects and the number of new warnings is 3% and 6%, respectively.

In conclusion, we can drastically reduce the number of commits and warnings developers have to examine by combining statistical models with static analysis. Future work is necessary to determine if the warnings and commits that we select help developers find defects. We are working with KDM Analytics and Defense Research and Development Canada to determine if WarningsGuru is effective in practice. Our tool is also publicly available for use by researchers and developers.[1]

ACKNOWLEDGEMENT

The authors would like to thank KDM Analytics, Defense Research and Development Canada, and NSERC for funding this tool development. We also thank Dr. Shihab's team for providing feedback as we integrated our code with CommitGuru.

REFERENCES

[1] Nathaniel Ayewah, William Pugh, J. David Morgenthaler, John Penix, and YuQian Zhou. 2007. Evaluating Static Analysis Defect Warnings on Production Software. In *Proceedings of the 7th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE '07)*. ACM, 1–8. https://doi.org/10.1145/1251535.1251536

[2] M. Beller, R. Bholanath, S. McIntosh, and A. Zaidman. 2016. Analyzing the State of Static Analysis: A Large-Scale Evaluation in Open Source Software. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. 470–481. https://doi.org/10.1109/SANER.2016.105

[3] Cesar Couto, João Eduardo Montandon, Christofer Silva, and Marco Tulio Valente. 2013. Static correspondence and correlation between field defects and warnings reported by a bug finding tool. *Software Quality Journal* 21, 2 (2013), 241–257. https://doi.org/10.1007/s11219-011-9172-5

[4] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. 2012. A Systematic Literature Review on Fault Prediction Performance in Software Engineering. *IEEE Transactions on Software Engineering* 38, 6 (Nov 2012), 1276–1304. https://doi.org/10.1109/TSE.2011.103

[5] Yasutaka Kamei, Emad Shihab, Bram Adams, Ahmed E. Hassan, Audris Mockus, Anand Sinha, and Naoyasu Ubayashi. 2013. A Large-Scale Empirical Study of Just-in-Time Quality Assurance. *IEEE Trans. Softw. Eng.* 39, 6 (June 2013), 757–773. https://doi.org/10.1109/TSE.2012.70

[6] KDM Analytics. 2016. Blade Tool Output Integration Framework (TOIF). http://www.kdmanalytics.com/toif/.

[7] Ugur Koc, Parsa Saadatpanah, Jeffrey S. Foster, and Adam A. Porter. 2017. Learning a Classifier for False Positive Error Reports Emitted by Static Code Analysis Tools. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL 2017)*. ACM, 35–42. https://doi.org/10.1145/3088525.3088675

[8] MITRE Corporation. 2016. Common Weakness Enumeration (CWE). https://cwe.mitre.org/.

[9] Foyzur Rahman, Sameer Khatri, Earl T. Barr, and Premkumar Devanbu. 2014. Comparing Static Bug Finders and Statistical Prediction. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. ACM, New York, NY, USA, 424–434. https://doi.org/10.1145/2568225.2568269

[10] Christoffer Rosen, Ben Grawi, and Emad Shihab. 2015. Commit Guru: Analytics and Risk Prediction of Software Commits. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015)*. ACM, New York, NY, USA, 966–969. https://doi.org/10.1145/2786805.2803183

[11] Hao Tang, Tian Lan, Dan Hao, and Lu Zhang. 2015. Enhancing Defect Prediction with Static Defect Analysis. In *Proceedings of the 7th Asia-Pacific Symposium on Internetware (Internetware '15)*. ACM, New York, NY, USA, 43–51. https://doi.org/10.1145/2875913.2875922

[12] The Apache Software Foundation. 2016. Maven - POM Reference. https://maven.apache.org/pom.html.

[13] Omer Tripp, Salvatore Guarnieri, Marco Pistoia, and Aleksandr Aravkin. 2014. ALETHEIA: Improving the Usability of Static Security Analysis. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*. ACM, New York, NY, USA, 762–774. https://doi.org/10.1145/2660267.2660339

[1] WarningsGuru source code: https://github.com/louisq/warningsguru