2013 IEEE Sixth International Conference on Software Testing, Verification and Validation

Leveraging Light-Weight Analyses to Aid Software Maintenance

Zachary P. Fry
zpf5a@virginia.edu

Westley Weimer
weimer@cs.virginia.edu

I. Introduction

Software maintenance can account for up to 90% of a system's life cycle cost [1]. Anecdotal evidence from real-world developers suggests that for real systems, maintenance teams often cannot keep up with the high rate at which faults are found and reported [2]. As a result, many automated techniques have been developed to reduce the overall effort necessary to sustain software over time. While many of these tools work well under certain circumstances, we believe that they could be improved by taking advantage of large untapped sources of unstructured information that result from natural software development. Software maintenance remains a costly problem in practice; the proposed work will identify weaknesses in common maintenance processes and attempt to address them to reduce both human and computational costs.

The proposed research will design lightweight analyses to extract latent information encoded by humans in software development artifacts and thereby reduce the costs of software maintenance. We will design analyses that apply throughout the maintenance process, focusing on three areas: (1) the human cost associated with manually triaging large collections of automatically exposed defects; (2) the computational cost and expressive power of evolving automatic defect repairs; and (3) ensuring the continued consistency of system documentation to reduce the difficulty (and thus cost) of understanding systems over time. In each case, we hope to concretely show that specific maintenance costs can be reduced by extracting and analyzing previously unused information thereby easing the total maintenance burden.

The key insight is that many existing maintenance techniques could be improved by leveraging the less-structured, human-created artifacts that are inherent in software development. For example, we have previously shown that natural language clues can dramatically improve automatic defect localization from human-written defect reports [3]. We hypothesize that light-weight techniques can be used to extract and analyze actionable information latent in software artifacts, thus reducing maintenance costs overall.

II. Proposed Research

We will evaluate our techniques and tools on large, real-world systems, comprising tens of millions of lines of code and thousands of defects. The proposed work will attempt to reduce the cost of three specific maintenance tasks: triaging automatically-generated defect reports, automatically synthesizing defect repairs, and automatically identifying out-of-date or incomplete system documentation. We hope to address bottlenecks in each of these three areas and show concrete time and effort savings for each process. The rest of this section describes each maintenance process and our proposed improvements in each case.

A. Clustering Duplicate Automatically-Generated Defect Reports.

Large software systems contain so many faults that automatic defect-finding tools are commonly used to expose them [4] — and in large systems, many faults share the same symptoms or the same causes. Unfortunately, automatic defect finders rarely recognize such similarity. We have found, however, that over 30% of such defect reports (over 2,600 actual instances in our study), were similar enough that time and effort could be saved by handling them aggregately. While some of these defect reports share syntactic similarities, in many cases the similarities are only visible at a semantic level. We propose to cluster such reports by extracting both syntactic and semantic information and using lightweight metrics to model their similarity. The success metrics for such a technique are accuracy (i.e., clustering only defects that are related) and effort savings (i.e., reducing the number of clusters that must be triaged separately). While there are no directly-comparable tools, we adapt three established code clone detectors to the task of clustering automatically-generated defects to use as baselines when assessing our technique's ability to save maintenance effort. Preliminary results show that our technique is both capable of producing clusters that perfectly match our annotated data set (while code clone

978-0-7695-4968-2/13 $26.00 © 2013 IEEE
DOI 10.1109/ICST.2013.77

tools are not) and can save more developer effort at nearly all levels of accuracy.

B. Improved Fitness Functions for Automatic Program Repair.

Even after similar defect reports have been triaged, so many defects remain that there has been increasing interest in automated program repair techniques to reduce the maintenance burden [5]. However, quickly evolving functionally correct repairs remains a challenge. Current techniques typically measure the fitness or "quality" of a candidate repair only in terms of test cases, treating all tests equally. Unfortunately, this is a very noisy signal in practice [6] — not all test cases are created equal [7] and internal program state remains an untapped resource [8]. We propose to build a light-weight model that can guide the search to a repair. By weighting test cases and including information about run-time program state, we propose to develop a fitness function with a higher fitness-distance correlation [9]. The goal of this work is to allow program repair tools to generate more eventual defect fixes and to do so faster than they would with more naive fitness functions, on an established ([5]) set of real-world defects.

C. Ensuring Documentation Consistency.

Changing a program, either to fix a fault or to add a feature, may lead to inconsistent or incorrect documentation. Previous researchers have shown that comments rarely co-evolve with code in real-world systems [10], which can lead to severely inconsistent comments [11], [12]. We propose to create a general, lightweight model of comment quality with respect to consistency and completeness. We will do so by synthesizing structured "template" documentation that includes key concepts but not extraneous natural language. We will then extract similar information from existing natural language comments and perform a structured comparison. This proposed work is the most speculative and represents a higher-risk higher-reward research trade-off. We propose to evaluate our technique with a set of human studies, focusing on two evaluation metrics: (1) how often human annotators agree with our model's judgments of comment consistency and completeness; and (2) how much the use of our tool increases humans' speed and accuracy when identifying inconsistent and incomplete comments.

III. Acknowledgment

I wish to thank my research advisor, Westley Weimer, for the guidance he has provided throughout the dissertation process.

References

[1] R. C. Seacord, D. Plakosh, and G. A. Lewis, *Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[2] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in *International Conference on Dependable Systems and Networks*, pp. 52–61, 2008.

[3] Z. P. Fry and W. Weimer, "Fault Localization Using Textual Similarities," *ArXiv e-prints*, 2012.

[4] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. R. Engler, "A few billion lines of code later: using static analysis to find bugs in the real world," *Communications of the ACM*, vol. 53, no. 2, pp. 66–75, 2010.

[5] C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer, "A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each," in *International Conference on Software Engineering*, pp. 3–13, 2012.

[6] E. Fast, C. Le Goues, S. Forrest, and W. Weimer, "Designing better fitness functions for automated program repair," in *Genetic and Evolutionary Computation Conference*, pp. 965–972, 2010.

[7] K. Walcott, M. Soffa, G. Kapfhammer, and R. Roos, "Time-aware test suite prioritization," in *International Symposium on Software Testing and Analysis*, 2006.

[8] B. Liblit, A. Aiken, A. X. Zheng, and M. I. Jordan, "Bug isolation via remote program sampling," in *Programming Language Design and Implementation*, pp. 141–154, 2003.

[9] T. Jones and S. Forrest, "Fitness distance correlation as a measure of problem difficulty for genetic algorithms," in *International Conference on Genetic Algorithms*, pp. 184–192, 1995.

[10] B. Fluri, M. Wursch, and H. C. Gall, "Do code and comments co-evolve? on the relation between source code and comment changes," WCRE '07, pp. 70–79, 2007.

[11] L. Tan, D. Yuan, G. Krishna, and Y. Zhou, "/* iComment: Bugs or bad comments? */," in *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP07)*, October 2007.

[12] S. H. Tan, D. Marinov, L. Tan, and G. T. Leavens, "@tComment: Testing javadoc comments to detect comment-code inconsistencies," in *Proceedings of the 5th International Conference on Software Testing, Verification and Validation*, April 2012.