Quality of Failure Data – The Good, the Bad, and the Ugly

Katerina Goševa-Popstojanova
Lane Department of Computer Science and Electrical Engineering
West Virginia University, Morgantown, WV 26506-6109
Katerina.Goseva@mail.wvu.edu

Abstract

Experimenting with large, realistic applications allows existing theoretical results to be tested and new domains to be explored, which over time will enable evolution of the knowledge and increase the prediction ability in software reliability engineering. One important fact should not be overlooked in this process - the quality of the reliability predictions depends not only on the methods used, but also on the quality of the failure data. One reason for low data quality is due to the fact that in most cases problem and change tracking repositories used today were not designed with failure analysis in mind. Thus, currently collected data should be augmented with empirical observations of other explanatory factors such as for example testing effort, code coverage, and operational usage which will allow more complete prediction models. Another major reason of low data quality is the lack of consistency and discipline in the process of recording the data. For example, studies of both open source and commercial applications have faced difficulties in distinguishing changes made to fix faults from other changes such as enhancements and adding new requirements. We believe that (1) improving the process of collecting and recording the failure data, (2) making real failure data from variety of sources publicly available, and (3) coupling the academic research with practical problems of interest to industry will contribute towards closing the gap in failure analysis.

1. Some open issues and conflicting results

Although there is a positive trend of publishing empirical software reliability studies of open source and commercial operating systems and other large applications, many myths, open issues, and conflicting results require further exploration. One myth explored widely is that *size metrics (such as LOC) are good predictors of a module's fault density*. More than 20 years ago Basili and Pericone [1] reported that fault density decreases with module size, which was supported by more recent study by Ostrand and Weyuker [8]. On the other side, Fenton and Ohlsson did not find evidence that module size has a significant impact on fault density [3]. Even more, they claimed that the strong results presented in [1] may be due to inappropriate analysis. The same myth was recently revisited by Murphy [7] and was found not to hold true.

Another controversial issue worth mentioning is whether at the module level *a higher incidence of faults in prerelease testing implies higher incidence of faults in operation.* Based on analysis of four releases of a large, mature application with thousands of modules Biyani and Santhanam [2] concluded that the higher number of prerelease faults in modules predicts a higher number of faults in field usage. This finding may mean that modules are fault prone from some fundamental reason and remain fault prone through the life cycle. A subsequent study of a few hundred modules from two releases of a large telecommunication software conducted by Fenton and Ohlsson [3] led to an opposite result "Those modules which are the most fault prone prerelease are among least fault prone postrelease, while conversely, the modules which are most fault prone postrelease are among the least fault prone prerelease". One possible explanation of this counter–intuitive finding provided by the authors is that a high incidence of faults in a module prior to release may be due to the fact that such a module has been well tested and will, therefore, be reliable in operation. In a study of thirteen successive releases Ostrand and Weyuker [8] found an indication that the files that contain prerelease faults are not the most likely place where postrelease faults will occur. However, in their case no release contained more than 20 postrelease faults, that is, there was not enough data to draw strong conclusions about fault concentration in modules during prerelease and postrelease. The key explanatory data, such as for example the testing effort per module, that may shed some light on these conflicting results most likely was not available to the authors of these studies and therefore was not reported.

2. Quality of failure data

While the positive trend of conducting and publishing empirical studies is encouraging, it should not be overlooked that *the quality of the reliability predictions depends not only on the methods used, but also on the quality of the failure data.* One reason for low data quality is due to the fact that in most cases problem and change tracking repositories used today were not designed with failure analysis in mind. Thus, currently collected data should be augmented with empirical observations of other explanatory factors such as for example testing effort, code coverage, and operational usage which will allow more complete pre-

diction models.

Another major reason of low data quality is the lack of consistency and discipline in the process of recording the data. For example, studies of both open source [4] and commercial applications [8] have faced difficulties in distinguishing changes made to fix faults from other changes such as enhancements and adding new requirements. These observations are consistent with the results from the survey of several open source projects presented in [5]. Out of 119 individual responses to the survey only 11.77% claimed that the defect tracking system was very consistent, that is, no defect gets fixed without reporting. 45.22% of the respondents answered that the defect tracking system is almost consistent, while the remaining 37.81% that it is not very consistent or not consistent.

The problem of missing essential data in [8] was approached by using a rule of thumb - only changes made to one or two files are related to fixing faults. Instead of using some kind of simplifying heuristics, in our earlier work [4] we developed two automatic and two manual methods for more accurate identification of changes made to fix faults. The result of our analysis showed that 30.59% of failures required fixing more than two files [4]. Similarly to our results, the analysis of nearly two hundred anomalies from seven NASA spacecraft systems led to conclusion that some anomalies have multiple targets, that is, multiple corrections are made to fix the problem [6].

Obviously, using heuristics that are not justified may lead to significant errors in the analysis. On the other side, developing more accurate methods to infer the missing data is a tedious and time consuming process. The best solution, of course, is to add a mandatory input field in the change (i.e. modification) requests with a clear identification whether the change was made for fixing faults. The problems of data quality do not end with missing essential data. Other data related problems for example include [9] (1) routinely leaving the default value in place for mandatory field, (2) leaving non–mandatory fields blank and (3) choosing a value from a drop-down list more or less randomly.

3. Concluding remarks

The motivating examples given in this paper clearly prove the importance of the interaction between theoretical and experimental research. In order to progress further, software reliability engineering should go through cycles of building theories, testing them empirically, learning from the experiments, and refining the theories to capture the newly discovered phenomena.

The current state of art and practice in failure analysis and reliability predictions should be advanced in several directions.

- *Improve the process of collecting and recording the failure data.* Accurate and complete information is a precondition for better predictions and providing insights into many open issues and conflicting results. Therefore, it is crucial to develop and adopt better format for keeping track of problem reports and changes made to the source code.
- *Make real failure data from variety of sources publicly available.* Exploring large industrial and open source software systems from different domains, produced in different environments, is needed to explain or resolve the currently conflicting results and to enhance the knowledge about complex, mainly unexplored phenomena.
- *Couple the academic research with practical problems of interest to industry.* A series of dialogues between researchers from academia and industry should provide the answer to the question “Are we solving the right problems?”.

Acknowledgements

This work is funded by NASA OSMA SARP under grant managed through NASA IV&V Facility in Fairmont and by NSF under CAREER grant CNS-0447715.

References

[1] V. R. Basili and B. T. Perricone, “Software Errors and Complexity: An Empirical Investigation”, *Communicatons of ACM*, Vol.27, No.1, 1984, pp. 42–52.

[2] S. H. Biyani and P. Santhanam, “Exploring Defect Data from Development and Customer Usage on Software Modules over Multiple Releases”, *Proc. 9th IEEE Int’l Symp. Software Reliability Engineering*, 1998, pp. 316–320.

[3] N. E. Fenton and N. Ohisson, “Quantitative Analysis of Faults and Failures in a Complex Software System”, *IEEE Transactions on Software Engineering*, Vol.26, No. 8, August 2000, pp. 797–814.

[4] K. Goševa–Popstojanova, M. Hamill, and R. Perugupalli, “Large Empirical Case Study of Architecture–based Software Reliability”, *Proc. 16th IEEE Int’l Symp. Software Reliability Engineering*, 2005, pp. 43–52.

[5] A. Gunes Koru and J. Tian, “Defect Handling in Medium and Large Open Source Projects”, *IEEE Software*, July/August 2004, pp. 54–61.

[6] R. R. Lutz and I. C. Mikulski, “Empirical Analysis of Safety Critical Anomalies During Operation”, *IEEE Transactions of Software Engineering*, Vol. 30, No.3, March 2004, pp. 172–180.

[7] B. Murphy, “Estimating the Risk of Releasing Software”, *Supplemental Proc. 17th IEEE Int’l Symp. Software Reliability Engineering*, 2006.

[8] T. J. Ostrand and E. J. Weyuker, “The Distribution of Faults in a Large Industrial Software System”, *Proc. ACM International Symposium on Software Testing and Analysis*, 2002, pp. 55–64.

[9] T. J. Ostrand and E. J. Weyuker, “Difficulties Encountered Doing Empirical Studies in an Industrial Environment”, *Supplemental Proc. 15th IEEE International Symposium on Software Reliability*, 2004, pp. 17–18.