Editorial, special issue, repeatable experiments in software engineering

Tim Menzies,
Lane Department of Computer Science and Electrical Engineering,
West Virginia University, WV, USA
tim@menzies.us

August 2, 2008

Welcome to the special issue of Empirical Software Engineering on repeatable experiments in software engineering. Earlier and shorter versions of the papers presented here first appeared at the PROMISE 2007 workshop in Minneapolis.

The PROMISE project has been running for 4 years now and aims to create large libraries of repeatable experiments in software engineering. PROMISE is somewhat different to other workshops that deal with learning from software data[1] in two ways. First, PROMISE emphasizes the data mining approach to generalization and, as such, is very concerned with the experimental methods used to generate the results. Second, PROMISE is more than just a workshop series. The project actually has three parts:

1. The PROMISE repository `http://promisedata.org/?cat=11`: Currently, the repository holds 76 data sets and is growing rapidly (around 40% more data sets each year). Where applicable, each data set is linked to papers describing current best results from that data (in many cases, those papers are PROMISE publications). All the data sets are stored in a content management system that allows registered users to add comments to the data. In this way, the community's experience with that data (tips, traps for beginners , etc) can be stored in a central web-accessible location.

2. The annual PROMISE conference: Each year, the PROMISE community meets to reflect on old results and examine new ones. Starting as a small workshop in ICSE 2005, PROMISE has now become a conference and PROMISE 2009 is a co-located event at ICSE 2009 (Vancouver, British Columbia).

[1] e.g. the Mining Software Repositories series co-located with ICSE (`http://www.msrconf.org/`) and the new DEFECTS series co-located at ISSTA (`http://pages.cpsc.ucalgary.ca/~zimmerth/defects-2008/`).

3. Journal special issues: Each year, the authors of the best PROMISE papers are invited to revise and extend their papers, then submit them to a peer-reviewed journal.

This special issue contains four such papers. Each paper generalizes some specific project experience to build general models. The range of models are quite diverse and the conclusions they draw from those models are fascinating.

For example, a widely-held view in the software engineering community is that module size is linear to defects; i.e. larger modules have more faults. According to this view, it makes sense to inspect larger modules before exploring smaller ones. In "Theory of Relative Defect Proneness: Replicated Studies on the Functional Form of the Size-Defect Relationship for Software Modules", Koru et.al. argues convincingly that this view is 100find a power-law relationship between module size and defects where smaller modules are proportionally more fault prone. Therefore, it is far more effective to focus verification efforts on smaller modules before moving to the larger modules. In their paper, Koru et.al. offers some speculations on why this power-law relationship holds and in this editorial, I offer one more. Perhaps in this modern era of refactoring and separation of concerns and auto-generated code, programmers must split their ideas into tiny pieces all across the code base. Once divided in this way it is hard to understand the interactions of all these tiny pieces.

Another widely-held view is that the best way to build defect models is by data mining. According to this view, since human experts do not understand all the nuances of their code, we must use automatic data mining to reveal the land mines buried in our systems. In this view, the construction of defect model is a 3-step process:

1. collecting domain knowledge (previous results, expert knowledge);

2. building a skeleton of the model based on step 1 including as yet unknown parameters;

3. estimating the model parameters using historical data.

Any practitioner in this area will report that step #3 can be extremely difficult: it often quite hard to obtain reliable data of the required granularity, or of the required volume with which we could later generalize our conclusions. In *On the effectiveness of early life cycle defect prediction with Bayesian Nets*, Fenton et al. offer an alternative approach that avoids step #3. Working with domain experts, Fenton et al. built a causal model (Bayesian net) for predicting the number of residual defects that are likely to be found during independent testing or operational usage. Note that this approach supports step #1 and #2, but does not require step #3. In two respects, their results are most impressive. Firstly, their Bayes nets makes very accurate defect predictions (an R^2 of 0.93 between predicted and actual defects). Secondly, since their method does not require detailed domain knowledge it can be applied very early in the process life cycle.

Yet another widely-held view is that "too many cooks spoil the broth"; i.e.. code units will be less fault-prone if they are written and maintained by only a few, or even just one, programmer. Another common belief is that a developer who works for the first time on a file that has previously been written or maintained by others is more likely to introduce faults into the software than programmers who have prior experience with the code. In *Do Too Many Cooks Spoil the Broth? Using the Number of Developers to Enhance Defect Prediction Models*, Weyuker & Ostrand explored the value of adding development team size and experience with specific code units to an existing defect prediction model. They found this information improved defect prediction, but only by a negligible amount.

Our final paper is concerned with the assessment of models, once they are built. There exists a large number of competing data mining methods for generating software defect models and deciding which model is "best" is a non-trivial task. In *Techniques for Evaluating Fault Prediction Models*, Jiang et al. comment that the comparison of fault-prone models is a multi dimensional problem. Rarely will one model or the modeling technique prove to be the "best" for all possible uses in software quality assessment. Overall model classification performance is not the ultimate goal in itself. Rather, optimizing the project cost and maximizing the efficiency of software verification procedures typically tops the agenda. This paper describes a methodological generalization of cost sensitive numerical performance indices called "cost curves" that offer a succinct graphical comparison of model performance across a wide range of module misclassification costs.

This special issue is the result of much work by a large group of people:

- First and foremost, I gratefully acknowledge the dedication of the reviewers of these papers. These reviewers were kind enough to review multiple versions of these papers, and to do so in record time.
- Also, special thanks are due to my fellow members of the PROMISE steering committee: Gary Boetticher (general chair), Tom Ostrand, and Gunther Ruhe.
- Last but not least, this issue would not have been possible without the support of the team at the Empirical Software Engineering Journal: Lionel Briand was kind enough to support this issue and Racquel Anievas was exceptionally helpful during the review and production process.

For more information on the PROMISE project, see `http://promisedata.org`. I hope that, soon, I will read your papers at a forthcoming PROMISE conference; or that other researchers use the data you contribute to the PROMISE repository.