Autom Softw Eng (2012) 19:137–140
DOI 10.1007/s10515-011-0095-y

Guest editorial: learning to organize testing

Ayse Bener · Tim Menzies

Received: 7 September 2011 / Accepted: 8 September 2011 / Published online: 7 October 2011

At the start of the decade, two publications (Shull et al. 2002; Boehm and Basili 2001) described the start-of-the art in defect reduction. Since then, there has been considerable research into data mining of defect data; e.g. Menzies et al. (2007). The data mining work has become less about *defect reduction*, and more about how to *organize a project's test resources* in order to *improve product quality* by (say) defining a procedure such that the modules most likely to contain defects are inspected first (Menzies et al. 2010).

After a decade of intensive work into data mining to make best use of testing resources, it is time to ask: what have we learned from all that research? Some of that research offers success stories with (e.g.)

- Reducing the costs to find defects (Menzies et al. 2010);
- Generalizing defect predictors to other projects (Tosun et al. 2011);
- Tuning those predictors to different business goals (Turhan et al. 2009).

But other research offers the cautions that:

- Defect predictors may not generalize to other projects (Zimmermann et al. 2009);
- Despite much effort on data mining and defects, most of that work achieves similar conclusions (Lessmann et al. 2008);
- Data mining data is fundamentally less important than discussing those effects with the users (Fenton et al. 2008).

A. Bener (✉)
TRSITM, Ryerson University, Toronto, Canada
e-mail: ayse.bener@ryerson.ca

T. Menzies
West Virginia University, Morgantown, USA
e-mail: tim@menzies.us

The above references sample just a small subset of the research performed this decade on data mining and software defects. In this special issue we wanted to focus on research that documents, reviews, and extends this work. Do the insights from the start of the decade still hold? Has anything extra really been learned in the meanwhile? If we wrote an article "What We Have Learned About Organizing Testing Resources" in 2010, what would we write in such an article, that has been *verified using publicly available data sets*?

Therefore, this special issue aimed to address the research community's progress (or lack of progress) in using data mining to organize test resources to fight defects with verifiable results based on *public-domain data sets*. This special issue has three papers whose results verify previous studies:

1. The first paper titled "Applying the Mahalonobis-Taguchi Strategy for Software Defect Diagnosis" uses a well known technique in other domains for the first time in the software engineering domain. It is a data driven technique that clusters defective modules. It also gives insights to the most relevant metrics that explain defective classes at the design and development stages of the product development. The technique is more like a heuristic than a classical statistical method, and the basic idea is that faulty classes do not belong to different population; rather, each is a unique case. The authors conducted their experiments on 10 different NASA datasets. The technique promises to be quite useful in practice since it makes expert-like assumptions about distributions. The data-centric nature of this approach offers much insight into the relationship between metrics and defects.
2. The second article, titled "An Investigation on the Feasibility of Cross-Project Defect Prediction", is another data-driven approach in software defect prediction. The research in this article focuses on choosing the best training data and offers significant extensions to previous studies in the literature. The authors confirm that using cross project data achieves good results since similar projects are the best indicators of defects. This study further emphasizes that in real life projects obtaining historical data may be difficult and may not be so meaningful either due to heterogeneity of the environment. However, obtaining data from other projects would be relatively easier, and hence produce better results, provided that training data is carefully selected. Therefore, they emphasize better understanding of data in constructing effective defect prediction models.
3. The third paper titled "Sample-based Software Defect Prediction with Active and Semi-supervised Learning" is also a data-driven approach that repeats and confirms previous studies in the literature. As with the previous paper, the basic assumption is the lack of historical data . Accordingly, the authors seek to understand the minimum size requirements for a data set being used to build a good defect prediction model. They employ various sampling techniques and conclude that effective defect predictors can be built with smaller data samples. Their proposed semi-supervised learning model ACoForest performs better than the conventional machine learning methods, because it inherently learns more from the available data.

In this special issue we are pleased to observe that recent work in defect prediction studies produce repeatable results and verify the findings of previous studies,

although they use different datasets, techniques, and experiment designs. Such results are important since software engineering is a domain with many random factors and relatively limited data. Nevertheless, in the software domain, remarkably effective predictors for software products have been generated using data mining methods. The success of these models seems unlikely considering all the factors involved in software development:

- For example, organizations can work in different domains, have different process, and define/ measure defects and other aspects of their product and process in different ways.
- Furthermore, most organizations do not precisely define their processes, products, measurements, etc.

Nevertheless, it is true that very simple models suffice for generating approximately correct predictions for software development time, the location of software defects. One candidate explanation for the strange predictability in software development is that, despite all the seemingly random factors influencing software construction, the net result follows very tight statistical patterns. Building oracles to predict defects and/or effort via data mining is also an inductive generalization over past experience.

Going forward, this may lead this community to focus on "how we can use predictive models to define policies in software development organizations?", and "how can we generalize our results to make policies?". Perhaps descriptions of software modules only in terms of static code attributes can overlook some important aspects of software including:

- the type of application domain;
- the skill level of the individual programmers involved in system development;
- contractor development practices;
- the variation in measurement practices;
- and the validation of the measurements and instruments used to collect the data.

For this reason in order to be able talk about policy making and generalization, future research might consider augmenting, or even replacing static code measures with repository metrics such as past faults or changes to code or number of developers who have worked on the code, etc. In building oracles we have successfully modelled product attributes (static code metrics, repository metrics, etc.), and process attributes (organizational factors, experience of people, etc). However, in software development projects people (developers, testers, analysts) are the most important pillar, but very difficult to model. It is inevitable that we should move to a model that considers Product, Process, and People (3Ps). Future models should also be able to proactively guide practitioners in effective resource allocation to organize testing.

References

Boehm, B., Basili, V.: Software defect reduction top 10 list. IEEE Comput. **34**(1), 135–137 (2001). http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/82.78.pdf

Fenton, N.E., Neil, M., Marsh, W., Hearty, P., Radlinski, L., Krause, P.: On the effectiveness of early life cycle defect prediction with Bayesian nets. J. Empir. Soft. Eng. 499–537 (2008)

Lessmann, S., Baesens, B., Mues, C., Pietsch, S.: Benchmarking classification models for software defect prediction: a proposed framework and novel findings. In: IEEE TSE, July 2008, pp. 485–496 (2008)

Menzies, T., Greenwald, J., Frank, A.: Data mining static code attributes to learn defect predictors. In: IEEE TSE 2007, January, pp. 2–13 (2007)

Menzies, T., Milton, Z., Turhan, B., Cukic, B., Jiang, Y., Bener, A.: Defect prediction from static code features: current results,imitations, new approaches. J. Autom. Softw. Eng. (2010). http://menzies.us/pdf/10which.pdf

Shull, F., Basili, V.R., Boehm, B.W., Brown, A.W., Costa, P., Lindvall, M., Port, D., Rus, I., Tesoriero, R., Zelkowitz, M.V.: What we have learned about fighting defects. In: IEEE Metrics 2002, p. 249 (2002). http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.20.1413

Tosun, A., Bener, A., Kale, R.: AI-based software defect predictors:applications and benefits in a case study. Artif. Intell. (2011)

Turhan, B., Menzies, T., Bener, A.B., Stefano, J.D.: On the relative value of cross-company and within company data for defect prediction. J. Empir. Softw. Eng. 278–290 (2009). http://menzies.us/pdf/08ccwc.pdf

Zimmermann, T., Nagappan, N., Gall, H., Giger, E., Murphy, Brendan: Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In: ESEC/SIGSOFT FSE, pp. 91–100 (2009)