Empirical Software Engineering manuscript No.
(will be inserted by the editor)

Forward to the Special Issue on Negative Results in Software Engineering

Richard Paige · Jordi Cabot · Neil A. Ernst

Received: date / Accepted: date

Welcome to this special issue on Negative Results in Software Engineering. First, what do we mean by negative results? Negative or null results—that is, results which fail to show an effect—are all too uncommon in the published literature for many reasons, including publication bias and self-selection effects. Such results are nevertheless important in showing the research directions that did not pay off. In particular, "replication cannot be meaningful without the potential acknowledgment of failed replications" [3].

Negative Results in Software Engineering We believe negative results are especially important in software engineering, in order to firmly embrace the nature of experimentation in software research, just like most of us believe industry should do. This means scientific inquiry that is conducted along Lean Startup [7] principles: start small, use validated learning and be prepared to 'pivot', or change course, if the learning outcome was negative. In this context, negative results are, given their methodology, failed approaches that are just as useful as successful approaches: they point out what hasn't worked, in order to redirect our collective

Richard Paige
Department of Computer Science,
University of York
Deramore Lane, York, YO10 5GH
Tel.: +44 1904 32 5170
Fax: +44 1904 32 5599
E-mail: richard.paige@york.ac.uk

Jordi Cabot
Internet Interdisciplinary Institute
Universitat Oberta de Catalunya
Av. Carl Friedrich Gauss, 5. Building B3
08860 Castelldefels - Spain
E-mail: jordi.cabot@icrea.cat

Neil Ernst
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15206
Tel.: +1 412-268-5800
Fax: +1 412-268-5758
E-mail: nernst@sei.cmu.edu

scientific efforts. As Walter Tichy writes in [9], "Negative results, if trustworthy, are extremely important for narrowing down the search space. They eliminate useless hypotheses and thus reorient and speed up the search for better approaches."

The software industry has long had a strong belief in the power of intuition: that productivity can vary by 10x among programmers, or that problems found doing requirements analysis cost dramatically less to fix than code bugs, among many others (see Glass [5] and Bossavit [2] for more details on this folklore). Part of our job as researchers must be to give good empirical weight for, or against, commonly held beliefs. Negative results are an important part of this discussion.

Lack of Negative Results Publication of negative results is rare, even more so in software engineering where, in contrast to life sciences, there are no specific tracks or journals to present such results. Data that led to negative results in software engineering is very rarely shared. Even open datasets are uncommon in software engineering, despite emerging efforts such as the PROMISE repository (`http://openscience.us/repo/`). Many researchers remain reliant on single sources like Github for a limited set of artefacts (predominantly code). By contrast, other fields in computing emphasise, as a community effort, the production and maintenance of open datasets, e.g., in machine learning (see, for example, the UCI machine learning repository at `https://archive.ics.uci.edu/ml/`). It seems that the software engineering community needs to consider developing a culture that accepts negative results as something that is as important to share as novel inventions.

How does this influence the papers we targeted for this issue? First and foremost, each of the papers selected adheres to the highest standards of the journal. Our view is that published negative results cannot sacrifice empirical rigour. A negative result due primarily to misaligned expectations or due to lack of statistical power (small samples) is not a negative results, rather a poorly designed experiment. The negative result should be a result of a lack of effect, not lack of methodological rigour.

Indeed, negative results often come *because* the investigation is so well conducted. For instance, insisting on sufficient power for an experiment (for example, by choosing larger numbers of subjects) can mean that the noisy result that confirmed your suspicions when N=10 (a 'positive' result) disappears when N=100, simply because there is always a random chance you will reject the null despite the true effect not existing. Statistician Andrew Gelman compares this to trying to measure the weight of a feather with a bathroom scale, and the feather is in the pouch of a jumping kangaroo [4].

Summary of Papers The six papers presented in this special issue cover a wide array of software engineering topics, and tend to tackle problems for which experimental approaches are more applicable. The topics range from experiments related to logging, the validity of metrics, and sentiment analysis through to productivity analysis, effort estimation and revisiting the fragile base class problem. We do not have any submissions reporting negative results using qualitative research approaches; we think this is partly because there is no such thing in a qualitative framework (vs. the erroneous but often-used "$p < 0.5$" approach, where a 'negative' result is when $p > 0.05$, which is not at all what p-values are telling you). The papers are as follows:

The first paper in this special issue, titled "Empirical Evaluation of the Effects of Experience on Code Quality and Programmer Productivity: An Exploratory

Study", by Dieste et al., examines the claim that expertise improves code quality and programming productivity. The authors conducted an experimental task with both industry and academia (students), focusing on an Iterative Test-Last assignment (developing tests and production code in parallel). They measured the experience of each subject, and assessed their performance on the task. Their results show that there is no evidence that experience is a significant factor in either quality or productivity.

Serebrenik et al. then consider what sentiment analysis tools are bringing to software engineering research, in "On Negative Results when Using Sentiment Analysis Tools for Software Engineering Research". Sentiment analysis tools claim to measure the positive (e.g., "great code") or negative opinions (e.g., "this code is not very good") expressed in text corpora. Since researchers are now using these tools to evaluate software artefacts, e.g. in discussions of pull requests for Github, the authors conducted an experiment to understand how well these tools worked, and whether they matched what developers actually thought of the artefact. They conclude that a) tools only weakly agree with each other, and b) replicating existing studies using these tools was highly sensitive to the choice of the tool used.

In "On the Correlation between Size and Metric Validity", Gil and Lalouche investigate the correlation between software metrics and code size (as measured in Source Lines of Code (SLOC)). Given the importance of software metrics in the field, and in a nice example of replications providing multiple data points, similar studies were conducted earlier (see [8,6]. This study adds more project data, but also examines the impact of both size and a given metric on external features, such as quality. They conclude that "one has to either decouple the metrics from size or show that the external features themselves are not correlated with size", since there is a strong correlation between metric and size.

Saban et al., in "Fragile Base-class Problem, Problem?", explores the well-accepted assumption that "misusing" inheritance and composition in object-oriented programming (e.g. in the context of framework reuse where these two mechanisms are extensively used) negatively affects software maintenance, since changes on the superclasses might cause faults in the subclasses (which therefore become more "fragile"). After a quantitative analysis, the authors conclude that fragile classes are not more fault prone than other classes. An additional qualitative study shows that the detected faults in those classes also were not caused by fragile base classes, showing that the fragile class problem may not be as problematic as previously thought in the literature.

Our fifth paper, "Negative Results for Software Effort Estimation", by Menzies et al., seeks to assess whether new software effort estimation methods are actually better than older COCOMO-based methods initially proposed many years ago. Menzies et al. show that Boehm's 2000 COCOMO II model [1] works as well (or better) than all approaches proposed since COCOMO II, for projects with enough data available to enable parametric estimation based on the 23 COCOMO attributes characterizing a software project. In short, in 2016 new innovations in effort estimation have not superseded parametric estimation.

Finally, in "To Log, or Not To Log: Using Heuristics to Identify Mandatory Log Events - A Controlled Experiment", King et al. highlight the challenge of deciding the Mandatory Log Events (user activities that must be logged to enable forensics) to track for a given security analysis goal. Authors conducted a controlled experiment to evaluate the effectiveness of three different methods (standards-

driven, resource-driven and heuristics-driven) to perform this task and show that none of them is significantly better than the other. This highlights the need for additional research in this area.

Acknowledgments We would like to acknowledge the rigour and dedication of the many reviewers of this special issue. One observation that we would make as editors of this special issue is that reviewing negative results is hard: not only are negative results papers different from the norm, they require a different way of thinking and critiquing. In many cases, reviewing the papers was a collaborative effort between reviewer and editor, to try to ensure that the negative result was as clearly expressed as possible. We thank our open-minded reviewers for their support in this. We are also thankful to the Empirical Software Engineering Editors in Chief, Lionel Briand and Thomas Zimmermann, for their support, help and patience throughout the process of preparing this special issue.

References

1. Barry Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald J. Reifer, and Bert Steece. "Software Cost Estimation with COCOMO II", Prentice-Hall (2000).
2. Bossavit, Laurent, "The Leprechauns of Software Engineering: How folklore turns into fact and what to do about it", LeanPub (2015).
3. Ferguson, Christopher and Moritz Heene, "A Vast Graveyard of Undead Theories: Publication Bias and Psychological Science's Aversion to the Null", Perspectives on Psychological Science, 7(6) pp. 555-561 doi: 10.1177/1745691612459059 (2012)
4. Andrew Gelman and J. Carlin. "Beyond power calculations: Assessing type S (sign) and type M (magnitude) errors". Perspectives on Psychological Science, 9:641651, (2014)
5. Glass, Robert L., "Facts and fallacies of software engineering", Addison-Wesley (2012).
6. Herraiz, I., Gonzalez-Barahona, J.M., Robles, G.: Towards a theoretical model for software growth. In: Proceedings of the Fourth International Workshop on Mining Software Repositories, (2007)
7. Ries, Eric, "The Lean Startup", Crown Publishing Group (2014).
8. Shepperd, M.: A critique of cyclomatic complexity as a software metric. Software Engineering Journal 3(2), 3036 (1988)
9. Tichy, Walter F., "Hints for Reviewing Empirical Work in Software Engineering", Empirical Software Engineering, 5, 309–312 (2000).