R^3 – Repeatability, Reproducibility and Rigor

Jan Vitek
Purdue University, USA

Tomas Kalibera
University of Kent, UK

Abstract

Computer systems research spans sub-disciplines that include embedded systems, programming languages and compilers, networking, and operating systems. Our contention is that a number of structural factors inhibit quality systems research. We highlight some of the factors we have encountered in our own work and observed in published papers and propose solutions that could both increase the productivity of researchers and the quality of their output.

1. Introduction

> "One of the students told me she wanted to do an experiment that went something like this ... under certain circumstances, X, rats did something, A. She was curious as to whether, if she changed the circumstances to Y, they would still do, A. So her proposal was to do the experiment under circumstances Y and see if they still did A. I explained to her that it was necessary first to repeat in her laboratory the experiment of the other person — to do it under condition X to see if she could also get result A — and then change to Y and see if A changed. Then she would know that the real difference was the thing she thought she had under control. She was very delighted with this new idea, and went to her professor. And his reply was, no, you cannot do that, because the experiment has already been done and you would be wasting time." — *Feynman, 1974, Cargo Cult Science*

Publications are the cornerstone of academic life. Computer science is in a unique position amongst scientific disciplines as conferences are the venue of choice for our best work. With the perceived benefit of shortening time to publication due to a single-stage reviewing process, conferences have had profound impact on the way science is conducted. To appear competitive, researchers are trapped in an arms race that emphasizes quantitative measures. The pressure to publish novel ideas at an ever increasing rate favors perfunctory feasibility studies over the kind of careful empirical evaluation that is the hallmark of great research. To make matter worse, few publications venues are willing to accept empirical papers that evaluate a previously formulated theory on the ground of insufficient novelty.

> "When we ignore experimentation and avoid contact with the reality, we hamper progress." — *Tichy*

[0]*A version appeared as an invited contribution in the proceedings of EMSOFT'11.* Correspondence to: J. Vitek, Dept of Computer Sciences, Purdue University, 305 N University St, West Lafayette, IN 47907. `jv@cs.purdue.edu`

The essence of the scientific process consists of (*a*) positing a hypothesis or model, (*b*) engineering a concrete implementation, and (*c*) designing and conducting an experimental evaluation. What is the value of an unevaluated claim? How much work is needed to truly validate a claim? What is reasonable to expect in a paper? Given the death march of our field towards publication it is not realistic to expect much. Evaluating a non-trivial idea is beyond the time budget of any single paper as this requires running many benchmarks on multiple implementations with different hardware and software platforms. Often a careful comparison to the state of the art means implementing competing solutions. The result of this state of affairs is that papers presenting potentially useful novel ideas regularly appear without a comparison to the state of the art, without appropriate benchmarks, without any mention of limitations, and without sufficient detail to reproduce the experiments. This hampers scientific progress and perpetuates the cycle.

> "In the exact sciences observation means the study of nature. In computer science this means the measurement of real systems." — *Feitelson, 2006, Experimental Computer Science*

Systems research, ranging from embedded systems to programming language implementation, is particularly affected due to the inherent difficulties of experimental work in the field. Unlike many other sub-disciplines of computing, getting realistic software that can be used to conduct measurements is critical. There are limits to publicly available software repositories and companies are often tight-fisted with their source code. Thus researchers often have to (re)invent "representative" applications from scratch to evaluate their claims. To make matters worse applications can be tied to specific features of the operating environment — be that hardware or software. Finally, the basic properties of interest are not always clear cut. Power consumption or response time are difficult to measure non-intrusively and there is not even agreement on what metrics to use to present results.

> "An inherent principle of publication is that others should be able to replicate and build upon the published claims. Therefore, a condition of publication is that authors are required to make materials, data and associated protocols available to readers promptly on request." — *Nature Methods, Author's guide*

Important results in systems research should be *repeatable*, they should be *reproduced*, and their evaluation should be carried with adequate *rigor*. Instead, the symptoms of the current state of practice include the following quartet:

- Unrepeatable results,
- Unreproduced results,
- Measuring the wrong thing,
- Meaninglessly measuring the right thing.

The enabling factors for this state of affairs, beyond the sheer pressure to publish, include the following trio:

- Lack of benchmarks,
- Lack of experimental methodology,
- Lack of understanding of statistical methods.

This paper argues that we, as a community, can do better without hindering the rate of scientific progress. In fact, we contend that adopting our recommendations will lead to more opportunities for publication and better science overall.[1]

2. Deadly Sins

"In industry, we ignore the evaluation in academic papers. It is often wrong and always irrelevant." — *Head of a major industrial lab, 2011*

We list some common mistakes. While not always deadly in the sense of voiding the scientific claims of the research, they make the published work much less useful.

Unclear goals. Without a clear statement of the goal of an experiment, and of what constitutes a significant improvement, there is no point in carrying out any evaluation. Too often, authors assume that an improvement on any metric, however small, is sufficient. This is not so, as there are trade-offs (e.g. speed vs. space or power) and it is necessary to report on all relevant dimensions. Wieringa et al. [31] recommend each paper be accompanied by an explicit problem statement describing: the research questions (what do we want to know?), the unit of study (about what?), relevant concepts (what do we know already?), and the research goal (what do we expect to achieve?).

Implicit assumptions. Authors must describe their experimental setup and methodology. Mytkowicz et al. [24] show how innocuous aspects of an experiment can introduce measurement bias sufficient to lead to incorrect conclusions; out of 133 papers from ASPLOS, PACT, PLDI, and CGO, none adequately considered measurement bias. They suggest setup randomization, i.e. running each experiment in many different experimental setups and using statistical methods to mitigate measurement bias. In general, all assumptions on which a claim relies should be made explicit. Clarke et al. [8] uncovered that performance of the Xen virtual machine critically relied on SMP support being turned off. The original paper's authors had not realized that this was a key assumption of their work, and made unwarranted conclusions.

[1] Academic integrity is not one of our major consideration. In our experience, while fraud does occur it is sufficiently infrequent that it can be merely viewed as an extreme case of mediocre research. We believe, without evidence, that the impact of the latter dominates the former.

Proprietary data. One of the hallmarks of research done in industry is access to proprietary benchmarks and data sets. But what is the value of publishing numbers obtained with unknown inputs? As far as the reader is concerned, there is little point in showing data on such experiments as nothing can be learned. Consider for instance the Dacapo benchmark suite for Java applications [6]. The suite arose out of the observed deficiencies of the previously accepted SPEC JVM98 benchmark suite. Without access to the code and data of the SPEC benchmark it would have been difficult to identify its deficiencies. As a recent exercise in forensic bioinformatics by Baggerly and Coombes [4] demonstrates, access to data is essential to uncover, potentially life-threatening, mistakes.

Weak statistics. In 2011, 39 of 42 PLDI papers reporting execution time did not bother to mention uncertainty. Every student learns about uncertainty in measurements and how to estimate this uncertainty based on statistical theory [20, 28]. Reporting on an experiment without giving a notion of the variability in the observations may make weak results appear conclusive or make statistical noise appear like an actual improvement. More sophisticated statistical methods are available but very few authors appeal to them.

Meaningless Measurements. A danger associated with the complex phenomena being measured is to attribute an observation to the wrong cause. In our own work on real-time systems [18], we wanted to measure the impact of compiler optimization of high-level language features on a representative platform. We failed to realize that our platform was emulating floating point operations, and the two compilers being compared were emitting different numbers of those operations. Thus, we ended up measuring the quality of floating point optimizations instead of what we had in mind. Another common error is to use training data to evaluate profile-based techniques or to measure the cost of a feature in isolation without taking into account its impact on a complete system. In both cases, the results are likely to be either wrong or meaningless.

No baseline. Establishing a credible baseline is crucial to meaningful results. Many authors use as baseline their own implementation with and without some optimization. But to be thorough, they should compare with the state of the art. This means, e.g. in the field of compilers, comparing against a production compiler. Sometimes coming up with the appropriate baseline requires re-implementing previously described algorithms, a time consuming but necessary task.

Unrepresentative workloads. Unrepresentative workloads are perhaps the greatest danger for empirical work. Either the benchmark over-represents some operations (such as micro-benchmarks tend to do, or are order of magnitudes smaller than real programs), the distribution of inputs is unlike real application (as is often the case in simulations), or when the external environment is modeled in an inappropriate fashion (e.g. running on an over-powered configuration, or failing to capture the complexity of the external world). In our research on JavaScript performance we have observed that industry standard benchmarks do not accurately represent the behavior of websites [26] and shown that performance improvement claims do not generalize to real-world usage.

3. Case Studies

We illustrate our discussion with examples.

Bad benchmarks can stifle progress. Research on trace-based compilation was motivated by JavaScript. In 2009, A. Gal et al. [11] published a highly regarded paper which showed that speed ups ranging from 2× to 20× could be achieved by integrating a trace-based just-in-time compiler in Firefox. They evaluated their work using SunSpider, an industry standard JavaScript benchmark suite consisting of 26 short-running programs. Unfortunately, Richards et al. showed in [27] that SunSpider has very little in common with the JavaScript code found in real web sites. Then, Richards et al. [26] constructed representative benchmark programs by automatically extracting them from popular web sites. The performance improvement of the trace-based just-in-time compiler on those benchmarks was very small. As it turns out, real-world JavaScript lacks the regularity needed to amortize trace-based compilation. It is our understanding that trace-based compilation has been abandoned in recent version of Firefox. Figure 1 (from [26]) illustrates our observations by contrasting speed improvements obtained on SunSpider vs. a popular website (Amazon). Firefox 3.5 introduced trace-based compilation. On SunSpider, this led to an almost 10× improvement, whereas there was no perceptible speed up on Amazon.

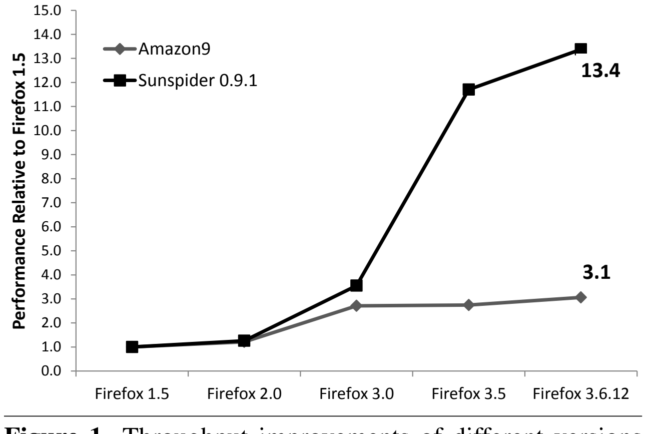

Figure 1. Throughput improvements of different versions of Firefox. (`sunspider`, `amazon`; FF1.5 - FF 3.6.12) Measurements on an HPZ800, dual Xeon E5630 2.53Ghz, 24GB memory, Windows 7 64-bit Enterprise. Numbers normalized to FF 1.5.

Good benchmarks can be an enabler for research. Garbage collection (GC) is a mature research area, about 40 years old, which involved heavy experimentation from the beginning. Over the years the quality of experiments has improved as members of the GC community have come together to form the Dacapo group and standardized best practice reporting and methodology: the Dacapo benchmark suite [6], which includes a range of highly non-trivial benchmarks built using open-source highly used Java libraries (close to a million lines of code). The suite has recently been updated by more recent workloads, which make heavier use of multiple cores. To support reproduction studies, the Memory Management Toolkit (MMTk) [7] provides an open-source library of memory managers and garbage collectors that can be evaluated with Jikes RVM [3], an open-source, community supported Java Virtual Machine. The fact that many GCs are implemented within a common platform, JikesRVM/MMTk, allows comparisons against a base-line without duplication of efforts. The number of benchmarks in the Dacapo suite is small enough to make looking at individual benchmarks possible, and as they are widely known to the community, a seasoned analyst can infer information simply based on which benchmarks a GC performed well and on which it did not. Quantitative characteristics of the individual benchmarks have been published. Comparisons of two systems are common, with the main metric of interest being the ratio of mean execution times. Although there is some understanding in the community that confidence intervals should be used, they rarely are and if so then almost always for individual systems only (and not say for the ratio of means of the two systems [10, 22]). Statistical tests or analysis of variance (ANOVA) seem to be used only in papers about how statistics should be applied in the field [12, 13]. Experiment design is still primitive — the best studies in systems research repeat benchmark runs as well as iterations in benchmarks, but summarize the results with a single arithmetic mean, as if they were all independent and identically distributed. They are not, and hence, if provided, a confidence interval or standard deviation are computed incorrectly. The best studies use multiple platforms or compilers, but do not attempt to summarize data over these (e.g. using ANOVA). Except for varying platforms or compilers, little is being done to avoid bias.

The right metrics are not obvious. The main metric in quantitative studies of GCs is execution time of complete systems. This is what users experience, what is easy to interpret, and what can be measured on benchmark applications. But there is a well known trade-off between maximum memory size and execution time: the smaller the memory, the more often the GC runs, and hence the longer the execution. To address this trade-off the GC community devised a reporting best practice which is to show results for multiple heap sizes. The workings of the GC algorithm are often intimately intertwined with the application. This makes GC benchmarking hard: one has to worry about non-obvious factors, such as code inserted by the compiler into the application (so called barriers) to make a particular GC work, or overhead of the memory allocator. As the experiments involve execution of the complete systems, factors not directly related to GC or memory have also to be taken into account. One such factor is the just-in-time compiler of the Java VMs which picks methods to optimize. Java VMs make this decision based on how hot the methods are. Due to statistical nature of sampling, re-running the experiment can lead to different sets of methods optimized, with different performance. Hence, we have a random factor that influences whole VM executions, and thus we need to repeat VM executions many times. Such a repetition is needed for other reasons as well, such as to reliably randomize memory placement. Sometimes it is possible to enforce the same sets of methods to compile to both systems [13] and perform paired experiments, which can be more precise within given time budget for experiments. This is, however, only possible when the code of the systems differs only slightly, and thus these sets of methods to compile are defined and realistic for both systems.

The current practice for summarization of results is to calculate geometric mean over different benchmarks. There is an ongoing debate which mean is more appropriate (arithmetic, harmonic, or geometric) and a general understanding that a single number is not enough, and that the analysis should look at individual benchmarks separately. This is mainly because we have still a relatively small set of benchmarks that is unlikely to cover all applications, and that we are nowhere near heaving weights that would express how common particular benchmarks are in real applications. These weights would otherwise be excellent for weighted means. Given the lack of such weights, the mean calculated over benchmarks of a suite depends very much on the outlying benchmarks in the suite. It is not uncommon that one or several benchmark dominate all the others according to some metric, e.g. allocation rate or lock acquisition rate. Adding or removing one such benchmark then could have big impact on the calculated means.

So, while not perfect, the state of affairs in measuring garbage collection algorithms has acquired some rigor to make understanding (and trusting) reported results possible.

4. How to move forward

We identify five areas in need of improvement.

4.1 Statistical methods

The phenomena we observe, postulate hypotheses about, and later measure, are influenced by numerous factors. In systems research, a common phenomenon is performance, measured by execution time. Factors involved are many and stem from the architecture, operating system, compiler, and the application. Some factors are controlled by the experimenter (i.e. architecture, compiler options, or time of day of running the experiment). They are either fixed for the whole experiment and become assumptions of the hypothesis, or they are systematically varied. There are uncontrolled factors, some of which can be observed and some of which cannot. All uncontrolled factors need to be randomized. Some parts of the experiment can be out of the control of the experimenter. Many systems respond to stimuli from the real world, which themselves have random nature. Given these complexities, it is difficult to decide how many repetitions of what combinations of factor configurations to run (experiment design) and how to summarize the results, separating the randomness from the properties of interest (statistical inference). Both experimental design and statistical inference are mature fields of statistics. Pointers to literature and basic principles of the scientific method can be found in [16, 21, 32]. Few advanced statistical methods are actually used in computer science.

It is crucial that the factors relevant to our field are known and well understood. In contrast to natural and social sciences, which focus on objects that have not significantly changed over the last few hundred years, computers are new and change rapidly. Factors influencing them are hard to find and not well studied. For example, memory placement is a known factor influencing execution time, through the number of cache misses. Some factors that in turn influence memory placement are less obvious: linking order, size of the space for environment variables in UNIX [24], symbol names [14], mapping of virtual to physical pages [15, 17], or randomly generated symbol prefixes during compilation [19]. If we miss a factor, the results we get are biased and only a reproduction study can find this.

There are also certain limits to which we can readily adopt all statistical methods used in natural and social sciences. These methods are based on the normal distribution, which is common for a wide range of phenomena observed in nature, but less so in computer systems. The normal distribution can be viewed as a model where many independent sources of error can add as well as each remove a constant from the true quantity. A good example of violation of this principle is execution time. The sources of "error" here (such as cache misses at all levels, slow-paths, etc.) often can add much more to execution time than they could remove. Also, they are often not independent. Consequently, execution time usually has a multi-modal distribution, where each fragment is highly skewed to the right (things can go exceptionally slow, but not so much exceptionally fast). Still, we get asymptotic normality through repetitions of the same, independent, measurements, when we summarize via arithmetic mean. This can partially justify the use of some methods that do require normality, e.g. we can get asymptotic confidence interval for the mean, perform a statistical test, or ANOVA. The asymptotic normality does not justify the use of these methods fully, additional assumptions need to be met that are beyond the scope of simple applied statistics and that cannot be easily checked anyway as we do not know much about the real distributions of our data [5, 23, 25]. However, using these methods that require normality when the data is in fact not normal is better than nothing. And, for additional rigor, there are also non-parametric methods that do not rely on the normality assumptions, and particularly the bootstrap methods are intuitively simple [9].

4.2 Documentation

> ..."must include instructions for building and installing the software from source, and any dependencies on both proprietary and freely available prerequisites. For software libraries, instructions for using the API are also required. Test data and instructions. The test dataset(s) must enable complete demonstration of the capabilities of the software, and may either be available from a publicly accessible data archive, or may be specifically created and made available with the software. The results of running the software on the test data should be reproducible, and any external dependencies must be documented." — *PLoS Computational Biology author's guide.*

The smallest bar we have to clear is *documentation* and archival of all experiment artifacts for future reference. The authors thus can look up later exactly under what conditions the earlier results were obtained, to confront the results with and validate against new findings. Good archival and documentation allows this even long after the actual hardware or software infrastructures to repeat the experiment become unavailable. Indeed, this can also lead to negative results, such as finding an error in the earlier experiments. The community should provide means of corrections of published papers for these instances.

4.3 Repetition

Repetition is the ability to re-run the exact same experiment with the same method on the same or similar system and obtain the same or very similar result. While this is needed for authors to ensure stability of their results, we argue that it is also needed for the community at large. The ability to repeat an experiment gives a baseline against which to evaluate new ideas. Thus, supporting repetition makes systems research easier. For reviewers, requiring submissions to be repeatable (e.g. by requiring executables or web interfaces) allows them to vary the input to the program and test the robustness of the proposed method, at least in cases when this does not require special hardware or software infrastructure. And generally, it helps to gain confidence as to the lack of random errors on the experimenter side and sufficient statistical methods for the random effects in the underlying system. Of course there is a cost in repetition — submitting a paper with enough supporting material for repeatability takes more time and may prevent authors from publishing results early. A good thing in our opinion. Is a paper for which the author feels that it is not worth making the code available worth reviewing? Support for repetition (access to data and protocols) was crucial in uncovering recent mistakes in biological studies [4].

4.4 Reproducibility

Independent confirmation of a scientific hypothesis through reproduction by an independent researcher/lab is at the core of the scientific method. The reproductions are carried out after a publication, based on the information in the paper and possibly some other information, such as data sets, published via scientific data repositories or provided by the authors on inquiry. Some journals (e.g. PLOS and Nature Methods) and some government institutions (e.g. NIH) require authors to archive all data and provide it on demand to anyone interested, so as to allow reproduction and promote further research. While there is an ongoing discussion of what should be mandatorily disclosed, as there is a trade-off between the confidence we get into scientific theories and duplication of efforts, the need for independent reproduction is accepted as a matter of course, and reproductions are in addition to new research and reviews part of the scientist's job. Reproductions are published in journals, no matter whether they end up confirming or disapproving the original work. This is not happening in computer "science". Notable attempts to change this include Reproducible Research Planet [2] and the Evaluate Collaboratory [1]. Reproducible Research Planet is a webspace for scientists to archive and publish data and code with their papers. It was promoted in several scientific papers in various fields of scientific computing. The motivation for authors to disclose data should be, apart from good practice, increased chance of being cited. The Evaluate Collaboratory then, in addition to organizing workshops on experimental evaluation of software and computer systems, initiated a petition to program committee chairs of conferences and workshops that called for acceptance of reproduction studies as first class publications. Although the petition got notable support, reproduction studies are not yet being published at major conferences. The reasons for the lack of reproductions is not just the lack of will, but also lack of knowledge, methods, and tools that would allow repeatability of experimental studies in computer science.

Reproduction is more powerful that repetition as it can uncover mistakes (or fraud) more readily. Repetition is important as it provides a baseline and facilitates extending and building on previous work.

4.5 Benchmarks

Experimental studies in systems are based on benchmark applications. A benchmark is a factor in an experiment as much as anything else. If we run just one benchmark, we have measured the phenomenon of interest just for this benchmark. If the benchmark is actually the application the user wants to run, then this is fine (modulo inputs). Otherwise, to draw more general conclusions, we need to randomize the benchmark factor as well: run using many different benchmarks that are representative of real applications, and statistically summarize the results. Despite a journey of improvement from instruction mixes, kernel benchmarks, and microbenchmarks, today's benchmarks are often inadequate. In some domains, they do not exist. Application benchmarks, which mimic the real applications to the highest extent possible, are essential. Ideally, application benchmarks are simply instrumented real applications with realistic data sets. When evaluating systems, we usually want the system to perform best/well for any application from a particular domain. Then, we need not one but many application benchmarks, as diverse as possible, to rule out bias through some factors hidden in most of the benchmarks. Micro-benchmarks can also be useful, as they simplify complex experiments by splitting them into several simpler experiments and allow the experimenter to vary a given factor of interest. Given the complexity of computer systems and the limited knowledge of factors that influence performance, hypotheses formed based on results with micro-benchmarks need to be validated by application benchmarks.

5. Recommendations

We propose the following changes to our practices:

- **Develop open source benchmarks:** We need benchmarks. Good, documented, open-source, and thoroughly evaluated benchmarks should be fully accepted as first class contributions for publications at premier conferences and journals, and should be worthy of support by governmental research agencies. This is important both to validate them through the review process and to create a reward system. So far, published benchmarks have been an exception rather than a rule and the authors are not aware of any funding being available for that thankless task.
- **Codify best practice documentation, methodologies and reporting standards:** We need to agree on minimal standards for documenting experiments and reporting results. This can be done through community effort and dedicated working groups. Software artifacts should be made available.[2] We need to understand the factors that influence measurable properties of computer systems, and we need to have better understanding of their statistical properties. Good observational studies on these topics have appeared at premier conferences, and this needs to continue. Reviewers should be encouraged to recognize the value of statistical methods and educated in their proper use. Lectures on statistical methods should be incorporated in the curriculum and statisticians should be consulted when designing new experiments.
- **Require repeatability of published results:** Repeatability should be part of the publication reviewing process. Based on the paper and supplementary material on the experiments (documentation, configuration, source code, input data sets, scripts), the reviewers should verify that the experiments are repeatable. This includes checking the documentation and reporting standards, including appropriate use of statistics, but does not mean that reviewers should be expected to re-run the experiments. Lack of repeatability can then be discovered by a reproduction study.
- **Encourage reproduction studies:** Thorough reproduction studies should be fully accepted as first class contributions for publications at premier conferences and journals. Researchers should be expected to have some publications of this kind on their curricula. Reproduction studies should be supported by governmental research agencies. The standards for good reproduction studies should require high level of rigor, repeatability and thoroughness, no matter if they approve or disapprove the original work. The credit of publications on new ideas should increase when independently confirmed in published reproduction study. Students should be expected to carry out a reproduction study early in their PhD.

What we propose will lead to higher quality of research. And, pragmatically, while there is a cost to the additional requirements, authors should think of this as "pay it forward". The benefits kick in when building on earlier work that provided artifacts to facilitate follow ons. Although it would be slightly harder to publish new ideas, one can get credit for publishing reproductions or benchmarks or observational studies.[3] Also, to be clear, we are not arguing against publishing "idea papers", papers that put forward a novel idea or theory. A good but unevaluated idea should be published, but an ill-evaluated idea shouldn't.[4]

"Beware of bugs in the above code; I have only proved it correct, not tried it." — *Don Knuth*

Acknowledgments. This paper would not have been without Sebastian Fischmeister's encouragements. We thank Phil Wadler for suggesting that we publish a version of this paper in SIGPLAN Notices. We greatly benefited from discussions with Richard Jones, Matthias Hauswirth, Tony Hosking, James Noble, Filip Pizlo, Olga Vitek and Mario Wolzcko. Many of the references used in this paper are available at the excellent Evaluate Collaboratory web site (`http://evaluate.inf.usi.ch`) where a letter to PC chairs awaits signatures.

[2] The `http://www.researchwithoutwalls.org` effort is arguing that "research that is reviewed by volunteers and often funded by the public should be freely available to all." The ACM Digital Library now supports the addition of ancillary material, `http://www.acm.org/publications/policies/dlinclusions/`, without requiring the authors to relinquish copyrights to the ACM. We would like to add meta-data to all ACM papers: *Code Complete* could indicate papers with source code, and *Data Complete* could denote papers that come with all necessary experimental data.

[3] The continuing discussion on the role of conferences and journals and the state of publishing in computer science is not directly related to our concerns [29, 30], but the pressure to publish clearly shoulders part of the blame for the state of experimental research.

[4] We would argue that "idea papers" should be in a category of their own, one that does not require experimental data. A follow up to an "idea paper" that evaluates the idea rigorously ought to be valued and publishable in the same venue.

References

[1] Evaluate collaboratory: Experimental evaluation of software and systems in computer science. `http://evaluate.inf.usi.ch/`, 2011.

[2] Reproducible research planet. `http://www.rrplanet.com/`, 2011.

[3] B. Alpern, C. R. Attanasio, J. J. Barton, M. G. Burke, P. Cheng, J.-D. Choi, A. Cocchi, S. J. Fink, D. Grove, M. Hind, S. F. Hummel, D. Lieber, V. Litvinov, M. F. Mergen, T. Ngo, J. R. Russell, V. Sarkar, M. J. Serrano, J. C. Shepherd, S. E. Smith, V. C. Sreedhar, H. Srinivasan, and J. Whaley. The Jalapeno virtual machine. *IBM Syst. J.*, 39(1):211–238, 2000.

[4] K. Baggerly and K. Coombes. Deriving chemo sensitivity from cell lines: Forensic bioinformatics and reproducible research in high-throughput biology. *Annals of Applied Statistics*, 2008.

[5] S. Basu and A. DasGupta. Robustness of standard confidence intervals for location parameters under departure from normality. *Annals of Statistics*, 23(4):1433–1442, 1995.

[6] S. Blackburn, R. Garner, K. S. McKinley, A. Diwan, S. Z. Guyer, A. Hosking, J. E. B. Moss, D. Stefanović, et al. The DaCapo benchmarks: Java benchmarking development and analysis. In *Conference on Object-Oriented Programing, Systems, Languages, and Applications (OOPSLA)*, 2006.

[7] S. M. Blackburn, P. Cheng, and K. S. McKinley. Oil and Water? High Performance Garbage Collection in Java with MMTk. In *Proceedings of the 26th International Conference on Software Engineering (ICSE)*, pages 137–146, 2004.

[8] B. Clark, T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne, and J. N. Matthews. Xen and the art of repeated research. In *USENIX Annual Technical Conference*, 2004.

[9] A. C. Davison and D. V. Hinkley. *Bootstrap Methods and Their Applications*. Cambridge University Press, 1997.

[10] E. C. Fieller. Some problems in interval estimation. *Journal of the Royal Statistical Society*, pages 175–185, 1954.

[11] A. Gal, B. Eich, M. Shaver, D. Anderson, D. Mandelin, M. R. Haghighat, B. Kaplan, G. Hoare, B. Zbarsky, J. Orendorff, J. Ruderman, E. W. Smith, R. Reitmaier, M. Bebenita, M. Chang, and M. Franz. Trace-based just-in-time type specialization for dynamic languages. In *Proceedings of the Conference on Programming Language Design and Implementation (PLDI)*, pages 465–478, 2009.

[12] A. Georges, D. Buytaert, and L. Eeckhout. Statistically rigorous Java performance evaluation. In *Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA)*, 2007.

[13] A. Georges, L. Eeckhout, and D. Buytaert. Java performance evaluation through rigorous replay compilation. In *Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA)*, 2008.

[14] D. Gu, C. Verbrugge, and E. Gagnon. Code layout as a source of noise in JVM performance. In *Component And Middleware Performance Workshop, OOPSLA*, 2004.

[15] M. Hocko and T. Kalibera. Reducing performance non-determinism via cache-aware page allocation strategies. In *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering*, pages 223–234, 2010.

[16] R. Jain. *The Art of Computer Systems Performance Analysis*. Wiley, 1991.

[17] T. Kalibera, L. Bulej, and P. Tuma. Automated detection of performance regressions: The Mono experience. In *Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2005.

[18] T. Kalibera, J. Hagelberg, P. Maj, F. Pizlo, B. Titzer, and J. Vitek. A family of real-time Java benchmarks. *Concurrency and Computation: Practice and Experience*, 2011.

[19] T. Kalibera and P. Tuma. Precise regression benchmarking with random effects: Improving Mono benchmark results. In *Third European Performance Engineering Workshop (EPEW)*, 2006.

[20] L. Kirkup. *Experimental Methods: An Introduction to the Analysis and Presentation of Data*. Wiley, 1994.

[21] D. J. Lilja. *Measuring Computer Performance: A Practitioner's Guide*. Cambridge University Press, 2000.

[22] Y. Luo and L. K. John. Efficiently evaluating speedup using sampled processor simulation. *Computer Architecture Letters*, 4:22–25, 2004.

[23] S. E. Maxwell and H. D. Delaney. *Designing experiments and analyzing data: a model comparison perspective*. Routledge, 2004.

[24] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney. Producing wrong data without doing anything obviously wrong! In *Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2009.

[25] D. Rasch and V. Guiard. The robustness of parametric statistical methods. *Psychology Science*, 46(2):175–208, 2004.

[26] G. Richards, A. Gal, B. Eich, and J. Vitek. Automated construction of JavaScript benchmarks. In *Conference on Object-Oriented Programing, Systems, Languages, and Applications (OOPSLA)*, 2011.

[27] G. Richards, S. Lesbrene, B. Burg, and J. Vitek. An analysis of the dynamic behavior of JavaScript programs. In *Proceedings of the ACM Programming Language Design and Implementation Conference (PLDI)*, June 2010.

[28] B. N. Taylor and C. E. Kuyatt. Guidelines for evaluating and expressing the uncertainty of NIST measurement results. Technical Note 1297, National Institute of Standards and Technology, 1994.

[29] M. Y. Vardi. Conferences vs. journals in computing research. *Commun. ACM*, 52(5):5, 2009.

[30] D. S. Wallach. Rebooting the CS publication process. *Commun. ACM*, 54(10):32–35, 2011.

[31] R. Wieringa, H. Heerkens, and B. Regnell. How to read and write a scientific evaluation paper. In *Requirements Engineering Conference (RE)*, 2009.

[32] E. B. Wilson. *An Introduction to Scientific Research*. McGraw Hill, 1952.