



Editor: **Les Hatton**
Oakwood Computing Associates
lesh@oakcomp.co.uk



Editor: **Michiel van Genuchten**
VitalHealth Software
genuchten@ieee.org

Computational Reproducibility

The Elephant in the Room

Les Hatton and Michiel van Genuchten

SINCE 2010, WE have been editing articles from all over the world for this column. The theme has essentially been the same—where does software appear, in what quantity, and using what technologies? We have had a vast range of responses, from the realms of the very large (space exploration) to the very small (the search for the Higgs boson). In between, we have touched on many practical systems, such as air-traffic control, banking in emerging nations, navigation, and general automobile systems.

One common factor repeatedly emerges, surprising even us, who have had careers working with software-based systems: the sheer size—or whatever other word you would choose—of software that has been deployed. Thirty years ago, systems of more than a million lines of source code were comparatively rare. They were rare for good reason, because they present formidable challenges to development and, later in their life, to their maintenance and particularly their reliability. We have repeatedly tried to present the economic consequences of such systems.³ As these

columns have shown, however, million-line systems written in various technologies are now commonplace. Indeed, with the growth of software stacks that combine rich functionality, we are seeing systems of tens of millions of lines of code.^{8,15}

This begs an interesting question. Have our testing procedures and methods of defect elimination kept up with the inexorable growth of around 20% per year?⁴ We will discuss this in one particular manifestation, the problem of reproducibility, because it appears to be exacerbating an existing crisis in the scientific world. Reproducibility is a very simple concept. In essence, all we ask in science is that the results can be independently repeated to some acceptable level of significance. This is how trust in a result grows. Reproducibility causes sufficient problems in science on its own, even without including the effects of computation, but we are now adding a deep and unquantifiable layer of uncertainty to this with the growing influence of computation in scientific discovery.

The Scientific Method

The scientific method emerged in the last 200 years or so, and it led to a

revolution in science. However, it is widely misunderstood and is now in danger of being trampled underfoot by the growth of social media platforms, with their up-and-down voting systems and pseudonumeric statistically illiterate conclusions, the relentless effects of which have elevated opinion to the same level as evidence. Because social media is used increasingly to discuss science, there is a particular danger in this.

Perhaps the cleanest insight into the scientific method came from the philosophers of the 20th century, such as Popper¹⁴ and Kuhn.⁷ In particular, Popper¹⁴ reiterated the essence of the scientific method, which is to promote falsifiability. A scientific experiment develops a theory to predict some outcome and then attempts to falsify it by using experimental data. Failure to falsify it means the theory lives to fight another day. If the results cannot be independently reproduced to some acceptable level of statistical significance, the theory is falsified. Over subsequent years and decades, good theories continue because additional experimental evidence fails to falsify them. Bad theories get broken and fall by the wayside. This does not, of course, mean that the

Digital Object Identifier 10.1109/MS.2018.2883805
Date of publication: 22 February 2019

“good” theories are correct—it was centuries before Newtonian physics was found wanting by relativistic arguments—but it does mean that they accumulate trust. Newtonian physics is a good enough approximation of the world we live in, and nearly all our engineering structures depend on it in some way. It fails only when our world intrudes on the very large or very small (e.g., as occurs with GPS satellite navigation systems), where the general relativistic effects and time dilation have to be taken into account.

Even without considering software, science has a problem with reproducibility. The estimated cost of irreproducible biomedical research is US\$28 billion/year,² and “Currently, many published research findings are false or exaggerated, and an estimated 85% of research resources are wasted.”⁶ Irreproducible results can have profound effects in the medical world, where they may lead to questionable protocols and, in some cases, significant loss of life.⁵

So what further complications does software introduce?

Computational Reproducibility

Like all of the disparate products we have covered in these columns, software written to support scientific enterprise has been growing rapidly for exactly the same reasons—cheap and extremely powerful hardware; the explosive growth of open source software, including compilers for languages, such as C, C++, Ada, and Fortran, and interpreters for the new(er) ones on the block, Python, Perl, and Java; and the widespread availability of excellent libraries. Parsing a protein sequence or analyzing an email now is a matter of a few hundred lines of Python. Added to this, we have higher-level and very sophisticated open source systems for handling databases, such as MySQL (itself written in

mountains of C), so we can store and apply computation to vast quantities of data. Perhaps even more significantly, we have open source statistical analysis programs of great sophistication, such as R (also written in mountains of C), which include the ability to produce the kind of wonderfully elaborate 3D plots that have become de rigueur in scientific publications. As a result of this effort by volunteers around the world, writing such analysis and data capture software is precisely what scientists have been doing for the last decade or two—and in huge quantities—in fields as diverse as space research^{10,11} and the search for the Higgs boson.¹³ There is a problem, however.

The Problem With Software

Perhaps 30 years ago, we had a wide understanding of Popperian falsifiability in the field of software testing. Indeed, the whole point of testing was not to affirm that software behaved correctly but to find those circumstances and conditions under which it failed to perform correctly—to falsify its premise of working correctly. As Myers¹⁰ states, “Testing is the process of executing a program with the intention of finding errors.” The whole idea was to break it and hand the pieces back to the developer (trying very hard not to be smug about it).

We seem to have taken a step backward from this. Perhaps overwhelmed by the massive and continuing growth of software, we have invented agile and other methodologies whose intention is primarily to converge on an acceptable solution for the end user, which may have little to do with the end user’s original aspirations, either in its delivered or its future behavior. This may be acceptable when the software supports functions, such as publishing a picture on a timeline or sending a message from one user to

another. It is not good enough if the function is to control a robot operating on a patient or an autonomous car in city traffic. Software technologies have unfortunately been burdened with an arcane vocabulary all their own, obscuring the clinical essence of falsifiability. As a result, many still believe—erroneously—that testing, insofar as it exists at all, is there to prove that the software works.

We need to be quite blunt about this. Computer science has no technology to guarantee the absence of defect; we never have had such a technology, and in all likelihood we never will. We don’t even have any easily applicable technologies to quantify the impact of residual defects, which we inevitably introduce and fail to remove before delivery, and yet the amount of software source code generally continues to grow by about 20% per year. We can therefore assert that scientific experiments that rely heavily on computation cannot follow the scientific method unless the complete means to reproduce the computational results independently is provided as an essential part of the whole. Paraphrasing the wise words of the geophysicist Jon Claerbout at Stanford, without the means to reproduce them, the results are merely an advertisement of scholarship and not the scholarship itself.¹ To be really blunt about it, they are not science in the sense of the scientific method.

What does *the complete means to reproduce* actually mean? It seems to us that unless a scientific article allows the reader to reproduce every table, diagram, and statistical result using code, for which the source and means to run it are provided,¹⁶ then the results are unquantifiably in error. Of course, nobody expects every line to be pored over for every scientific article, but for really critical

results, we might have to do exactly that. The whole stack needs to be open—the knock-on effects of incorrect results can be hugely expensive.

This is no mere quibble. Even with an open stack, it is inconceivable for one reader to verify a scientific result. Instead, readers must confine themselves to inspecting some part of the analysis or statistics code, leaving everything deeper to others. Figure 1 gives some idea of this, although it is not to scale. The bit that most scientists have to write is typically a few thousand up to a million or so lines of code and is shown in yellow. The next bit down is the applications we use, such as the language interpreters or compilers, the database software, the graphics, or the statistical analysis software. This is colored in green and will typically be around 1–5 million lines of code. The purple section at the bottom is tens of millions of lines of code, corresponding to the operating system and all of the C libraries; although larger, software lower down the stack tends to be more reliable because its usage is proportionately much higher. The most we can reasonably ask for is that enough people read the yellow parts. This is the science.

We conclude by saying that science has a serious and growing problem. Science is beginning to appreciate the size of the problem,¹² but it is shared by all consumers of software, as the numbers of failures in established systems will testify. (The reader might like to type “software failures” into a browser of choice—be prepared for a shock.) The humble and amusing bug of 30 years ago has grown into a system- and career-wrecking monster. It is within our powers to reign back this inexorable

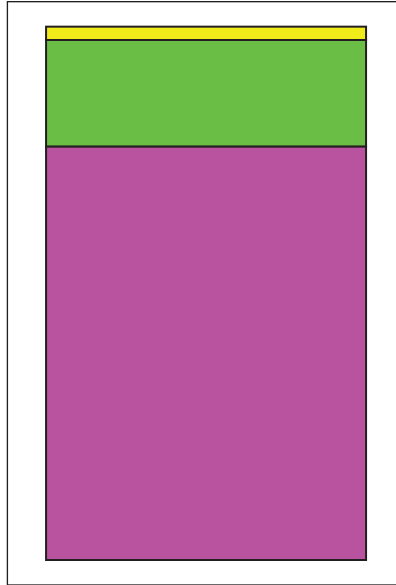



FIGURE 1. An illustration of a typical software stack in computational science.

growth, but not until we remember what *falsifiable* really means.

A Call for Columns

At the start of the 10th year of the “Impact” department in *IEEE Software*, we would like to invite you to discuss the impact of your own software. We have published more than 40 “Impact” departments so far, and you will be in good company with authors from large and small institutions who went before you. There are only three requirements: an interesting read for the *IEEE Software* audience and some metrics about size, volume of shipment if relevant, and how you address the maintenance. We have published departments from all continents except Antarctica, but we would welcome more contributions from software powerhouses in the East. Also, financial applications have eluded us so far, so don’t be shy. Your software does not have to be millions of lines of code or be associated with a high-profile phenomenon, such as the Mars Rover,

the Higgs boson discovery, or “Dieselgate.” We have a short and painless review process (no endless cycles; we are too busy for that), and although we can’t promise fame, we can promise that your experiences will help enrich the practical knowledge of how software systems perform in myriad ways we ask them to. 

References

1. J. F. Claerbout, “Seventeen years of super computing and other problems in seismology,” National Research Council Meeting on High Performance Computing in Seismology, Oct. 2, 1994. [Online]. Available: <http://sepwww.stanford.edu/sep/jon/nrc.html>
2. L. P. Freedman, I. M. Cockburn, and T. S. Simcoe, “The economics of reproducibility in preclinical research,” *PLOS Biology*, vol. 13, no. 6, pp. e1002165, 2015. doi: 10.1371/journal.pbio.1002165.
3. M. van Genuchten and L. Hatton, “Software mileage,” *IEEE Softw.*, vol. 28, no. 5, pp. 24–26, Sept.–Oct. 2011.
4. L. Hatton, D. Spinellis, and M. van Genuchten, “The long-term growth rate of evolving software: empirical results and implications,” *J. Software: Evolution Process*, vol. 29, no. 5, 2017. doi: 10.1002/smr.1847.
5. L. Huston, “New England Journal of Medicine declines to retract papers from disgraced research group,” 2014. [Online]. Available: <https://www.forbes.com/sites/larryhusten/2014/09/26/new-england-journal-of-medicine-declines-to-retract-papers-from-disgraced-research-group/#7f8405a17228>
6. J. P. A. Ioannidis, “How to make more published research true,” *PLOS Medicine*, vol. 11, no. 10, pp. e1001747, 2014. doi: 10.1371/journal.pmed.1001747.

(continued on page 144)

IEEE Software (ISSN 0740-7459) is published bimonthly by the IEEE Computer Society. IEEE headquarters: Three Park Ave., 17th Floor, New York, NY 10016-5997. IEEE Computer Society Publications Office: 10662 Los Vaqueros Cir., Los Alamitos, CA 90720; +1 714 821 8380; fax +1 714 821 4010. IEEE Computer Society headquarters: 2001 L St., Ste. 700, Washington, DC 20036. Subscribe to *IEEE Software* by visiting www.computer.org/software.

Postmaster: Send undelivered copies and address changes to *IEEE Software*, Membership Processing Dept., IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854-4141. Periodicals Postage Paid at New York, NY, and at additional mailing offices. Canadian GST #125634188. Canada Post Publications Mail Agreement Number 40013885. Return undeliverable Canadian addresses to PO Box 122, Niagara Falls, ON L2E 6S8, Canada. Printed in the USA.

Reuse Rights and Reprint Permissions: Educational or personal use of this material is permitted without fee, provided such use: 1) is not made for profit; 2) includes this notice and a full citation to the original work on the first page of the copy; and 3) does not imply IEEE endorsement of any

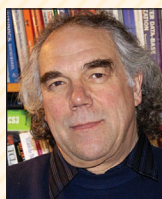
third-party products or services. Authors and their companies are permitted to post the accepted version of IEEE-copyrighted material on their own webservers without permission, provided that the IEEE copyright notice and a full citation to the original work appear on the first screen of the posted copy. An accepted manuscript is a version that has been revised by the author to incorporate review suggestions, but not the published version with copyediting, proofreading, and formatting added by IEEE. For more information, please go to: http://www.ieee.org/publications_standards/publications/rights/paperversionpolicy.html. Permission to reprint/republish this material for commercial, advertising, or promotional purposes or for creating new collective works for resale or redistribution must be obtained from IEEE by writing to the IEEE Intellectual Property Rights Office, 445 Hoes Lane, Piscataway, NJ 08854-4141 or pubs-permissions@ieee.org. Copyright © 2019 IEEE. All rights reserved.

Abstracting and Library Use: Abstracting is permitted with credit to the source. Libraries are permitted to photocopy for private use of patrons, provided the per-copy fee is paid through the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923.

IMPACT *(continued from page 139)*

7. T. S. Kuhn, *The Structure of Scientific Revolutions*. Chicago, IL: Univ. of Chicago Press, 1962. doi: 10.1086/ahr/68.3.700.
8. J. M. Mossinger, "Software in automotive systems," *IEEE Softw.*, vol. 27, no. 2, pp. 92–94, Mar.–Apr. 2010.
9. G. J. Myers, *The Art of Software Testing*. New York: Wiley, 1979.
10. J. Nagy, K. Balajthy, S. Szalai, B. Sódor, I. Horváth, and C. Lipusz, "Obstanovka: Exploring nearby space," *IEEE Softw.*, vol. 33, no. 4, pp. 101–105, July–Aug. 2016.
11. S. P. Zwart and J. Bédorf, "Creating the virtual universe," *IEEE Softw.*, vol. 33, no. 5, pp. 25–29, Sept.–Oct. 2016.
12. Perkel J., "A toolkit for data transparency takes shape," 2018. [Online]. Available: <https://www.nature.com/articles/d41586-018-05990-5>
13. D. Rousseau, "The software behind the Higgs boson discovery," *IEEE Softw.*, vol. 29, no. 5, pp. 11–15, Sept.–Oct. 2012.

ABOUT THE AUTHORS



LES HATTON is an emeritus professor of forensic software engineering at Kingston University, London. Contact him at esh@oakcomp.co.uk.



MICHIEL VAN GENUCHTEN is a member of the management team of VitalHealth Software. Contact him at genuchten@ieee.org.

14. K. Popper, *The Logic of Scientific Discovery*. New York: Routledge, 1959.
15. R. Wester and J. Koster, "The software behind Moore's law," *IEEE Softw.*, vol. 32, no. 2, pp. 37–40, Mar.–Apr. 2015.
16. L. Hatton and G. Warr, Full-computational reproducibility in biological science: methods, software and a case study in protein biology. 2016. [Online]. Available: <https://arxiv.org/abs/1608.06897>