What Is the Price of Open-Source Software?

ARTICLE *in* JOURNAL OF PHYSICAL CHEMISTRY LETTERS · JULY 2015

Impact Factor: 7.46 · DOI: 10.1021/acs.jpclett.5b01258 · Source: PubMed

Anna I Krylov
University of Southern California
179 PUBLICATIONS **7,033** CITATIONS

Peter James Knowles
Cardiff University
135 PUBLICATIONS **14,486** CITATIONS

Roland Lindh
Uppsala University
164 PUBLICATIONS **6,921** CITATIONS

Frederick R Manby
University of Bristol
96 PUBLICATIONS **3,931** CITATIONS

The Journal of Physical Chemistry Letters
Viewpoint
pubs.acs.org/JPCL

What Is the Price of Open-Source Software?

The notion that all scientific software should be open-source and free has been actively promoted in recent years, mostly from the top down via mandates from funding agencies[1] but occasionally from the bottom up, as exemplified by a recent Viewpoint in this journal.[2] A commonly articulated rationale is that the results of scientific research funded by government grants should be free for society and that the scientific community benefits from free access. The purpose of this Viewpoint is to examine the consequences of these opinions.

What Is Scientific Software? Modern computational chemistry software is an extremely complex product based on advanced scientific ideas (models and theories) and sophisticated algorithms that transform these ideas from equations into useful tools. The development of practical software that can be used by nonexperts to solve contemporary research problems requires considerable technical effort to produce and maintain robust, efficient, and validated code. Unlike the development of, for example, a smart-phone app, where the code base is small[3] and a relatively large community can easily write extensions and add-ons, production of scientific software involves the curation of millions of lines of source code. The complexity of this code demands long-term user and developer support to maintain its integrity and performance while keeping up with new computer architectures, fixing bugs, and adding features. Recognizing the importance of these ideas, various funding agencies in the U.S. have made "sustainable software" a key priority in the distribution of research support.[1] Sustainability is a critical goal, but one that can be realized in various ways.

Good Software Is Important to Science. Computational chemistry software is an essential scientific instrument that facilitates discovery and innovation far beyond the laboratories in which it is created, an achievement that was recognized by the 1998 and 2013 Nobel Prizes in Chemistry.[4] Focusing on quantum chemistry software in this Viewpoint, we note that today any chemist can (with very little training) use numerous quantum chemistry programs as teaching and research tools that aid in the design and interpretation of experiments.

A software package should be more than just a tool for end users, however; it should also be a platform to develop and test new models and algorithms. Maintaining a code base requires extensive validation, and given the complexity of modern computational methods, even testing of "pilot code" or a "proof-of-principle" implementation requires access to basic software infrastructure, for example, an integrals library, a self-consistent field procedure, efficient I/O and memory management, tools for manipulating tensors, and so forth. Modularity is a laudable goal, but in reality, "interoperability" often comes at the expense of performance. In high-performance codes, the aforementioned components are tightly interwoven, to the extent that expert help is often required to modify key components or to develop nonstandard interfaces to them. As such, the ability to innovate along either applied or theoretical lines depends crucially on the quality of the software and the availability of documentation and expert support.

As examples, consider two widely used electronic structure programs, Q-Chem[5] and Molpro.[6] These codes consist of ~5.5 and ~2.5 million lines of source code, respectively, written in multiple languages and each in continuous development over several decades. Q-Chem incorporates scientific advances reported in more than 300 peer-reviewed scientific publications, whereas methods implemented for the first time in Molpro have led to 20 high-impact papers that have each been cited over 300 times. Neither code is static: more than 70 scientists are actively contributing to Molpro, and the Q-Chem developer base numbers more than 100. Such agile innovation comes at a price, however. Significant effort is required to keep the code robust, efficient, and sound and to provide the documentation that ensures the usability of new methods and the extensibility of older ones.

Software from academia is often developed with an emphasis on ideas rather than implementation, fed by the need for timely peer-reviewed journal publications that provide ongoing grant support and future jobs for graduate students. To bring new ideas to the production level, with software that is accessible to (and useful for) the broader scientific community, contributions from expert programmers are required. These technical tasks usually cannot—and generally should not—be conducted by graduate students or postdocs, who should instead be focused on science and innovation. To this end, Q-Chem employs four scientific programmers. Other quantum chemistry codes (e.g., Molpro,[6] Turbomole,[7] Jaguar,[8] Molcas,[9] PQS,[10] and ONETEP[11]) face the same challenges and adopt similar models to ensure sustainability.

It is important to distinguish these academically led software ventures from purely commercial endeavors. The large majority of the code in a package like Q-Chem is funded by the government, either through grants to academic groups or, in some cases, through technology grants to the company itself. The role of the company programmers is to enable sustainability through bug fixes, user support, release management, and the addition of features that academic developers either cannot or will not add themselves. Programmers employed by the company place emphasis on functionality, robustness, and performance, more so than scientific innovation. They are directly addressing the "reproducibility problem".[2] Sales revenue cannot support the entire development cost of an academic code, but it contributes critically to its sustainability. The cost that the customer pays for a code like Q-Chem reflects this funding model: it is vastly lower than the development cost, particularly for academic customers but also for industry. It primarily reflects the sustainability cost.

Software Is Not Data. In his Viewpoint,[2] Gezelter argues that both software and data should be open, yet it is important not to conflate the two. Software is not data, and simply because it is *feasible* to put software on the Internet does not imply that it *should* be posted. Software is a product that contains an intellectual component (models and algorithms) but owes its

Published: July 16, 2015

existence to additional technical efforts. Such efforts include implementation of minor but useful (or requested) features that are not publishable in the peer-reviewed literature. This is not to say that details should be withheld as proprietary information. Just as models and algorithms should be described in full detail in scientific publications, so too should implementation details be specified, along with performance metrics (timings and scaling data) and benchmarks (energies and other computed properties). Nevertheless, the software itself is a product, not a scientific finding, more akin to, say, an NMR spectrometer—a sophisticated instrument—than to the spectra produced by that instrument.

Consider an analogy from the field of photovoltaics. Scientific findings concerning the mechanistic details of charge generation and exciton propagation in a given material are results that merit discussion in the peer-reviewed literature. However, creating a new solar panel based on this research requires significant additional engineering effort, which is most commonly conducted in an industrial setting. This is a common mechanism for technology transfer, by means of which society benefits from academic research. Likewise, new telecommunication technologies, information storage media, computer chips, and so forth are products that build upon—but are not equivalent to—scientific findings. Going from a journal article to a product in one's home or office requires a significant investment of resources that is often impossible to achieve in the absence of a commercial platform. Software is not different.

There Is No Free Software. The creation of scientific software is a labor-intensive process, and its support and curation even more so. How do we pay for these labor costs? The answer is clear in the case of commercial software, where license fees are used to defray the costs of development and support. In this model, users buy the software that fits their research needs and affords them the highest productivity. The decision mechanism and price-versus-deliverables choices are similar to those faced when purchasing a computer or other lab equipment. Just like a new laser system may enable the pursuit of new science, software that offers a competitive advantage is a sensible investment of research funds.

Interestingly, Gezelter[2] specifically mentions the Quantum Chemistry Program Exchange (QCPE), an early repository for open-source software, as having contributed greatly to the growth of the field. It is therefore telling to note that QCPE was supported initially by the Air Force Office of Scientific Research and then by the National Science Foundation before subsequently becoming a fee-for-software service with paid employees to do the time-consuming work of testing, documenting, and distributing the contributed programs.[12] Gezelter acknowledges the cost of maintaining scientific software and suggests alternative models to defray these costs including selling support, consulting, or an interface, all the while making the source code available for free.[2] These suggestions strike us as naïve, something akin to giving away automobiles but charging for the mechanic who services them. Such a model creates a financial incentive to release a less-than-stellar product into the public domain, then charge to make it useful and usable. It is better to release a top-of-the-line product for a nominal fee.

Is "free" software genuinely free of charge to individual researchers? Consider software developed in the U.S. national laboratories. These ventures are supported by full-time scientific programmers employed specifically for the task, and the cost to support and develop these products is subtracted from the pool of research funding available to the rest of the community. The individual researcher pays for these codes, in a sense, with his rejected grant proposals in times of lean funding. In contrast to using one's own performance metrics to guide software purchases, within this system, one has no choice in what one pays for. In other words, "free software" is not free for you; the only sense in which it is "free" is that you are freed from making a choice about how to spend your research money.

Computational chemistry software must balance the needs of two audiences: users, who gauge their productivity based on the speed, functionality, and user-friendliness of a given program; and developers, who may be more concerned with whether the structure "under the hood" provides an environment that fosters innovation and ease of implementation. As a quantitative example, consider that the cost of supporting a postdoctoral associate (salary plus benefits) is perhaps \$4,800/month. If the use of well-supported commercial software can save 2 weeks of a postdoc's time, then this would justify an expense of $\gtrsim$\$2,000 to purchase a software license. This amount exceeds the cost of an academic license for many computational chemistry programs. Given the choice between a free product and a commercial one, a scientist should make a decision based on her own needs and her own criteria for doing innovative research.

What Is "Open Source"? The term "open source" is ubiquitous but its meaning is ambiguous. Some codes are "free" but are not open,[13] whereas others make the source code available, albeit without binary executables, so that responsibility for compilation and installation is left to the user. Insofar as the use of commercial quantum chemistry software is a mainstay of modern chemical research and teaching, there exists a broad consensus that the commercial model offers the stability and user support that the community desires. Strict coding guidelines can be enforced within a model where source code access is limited to qualified developers, and this kind of stability offers one counterbalance to the "reproducibility crisis".[2] To the extent that such a crisis exists, it has occurred *in spite of* the existence of open-source electronic structure codes such as GAMESS,[14] NWChem,[15] and CP2K.[16]

Occasionally the open-source model is touted on the grounds that one can use the source code to learn about the underlying algorithms, but this hardly seems relevant if the methods and algorithms are published in the scientific literature. Source code itself rarely constitutes enjoyable reading, and using source code to learn about an algorithm is a last resort forced by poorly written scientific papers. Better peer review is a more desirable solution.

A more practical use of openly available source code is to reuse parts of it in other programs, provided that the terms of the software license allow this. Often, they do not. Some ostensibly "open" chemistry codes forbid reuse, or even redistribution.[13,17] Others, such as CP2K,[16] use the restrictive General Public License[18] that requires any derivative built on the original code to be open-source itself. Variation in design structure from one program to the next also severely hampers transferability, even if the license terms are amenable.

Access to source code allows developers to introduce their own innovations, but this is distinct from reuse. To facilitate innovation by developers, source code needs only to be available to people who intend to build upon it. This is commonly accomplished in the framework of "closed-source" software projects by granting academic groups access to the source code for development purposes. Given the large number of developers for many quantum chemistry codes, this developer community should not be envisaged as some small, secluded cabal but rather as a rich, diverse community of academic scientists.

Let us analyze accessibility using Q-CHEM as a specific example. This is commercial software whose source code is not in the public domain because that would eliminate the product that Q-Chem, Inc., is selling. Consider, however, how many developers benefit from Q-CHEM as a platform for their own innovation: this community exceeds 100 scientists from at least 12 countries,[19] for whom the code is open. We call this model *open teamware*.[5] Moreover, any licensed user can obtain access to the source code upon signing a nondisclosure agreement. Given the size of the user base, this is likely a significantly larger group than the number of people who care to look at many existing open-source packages. Does an "open-source" code that serves just a few people offer more benefit to the scientific community than a "closed-source" code that fosters a community of 100+ active developers and thousands of users? What would the impact be on computational chemistry of destroying other teamware projects such as MOLPRO,[6] TURBOMOLE,[7] JAGUAR,[8] MOLCAS,[9] PQS,[10] or ONETEP,[11] in the interest of satisfying some "open-source" mandate?

Practical Consequences of an Open-Source Mandate. One of the pillars of science funding in the U.S. and elsewhere is a merit-based funding model that distributes resources based on intellectual merit, productivity, and impact. In the long run, more-competitive ideas are selected over less-competitive ones, and investigators are rewarded for a track record of productivity. Research that is judged to have higher impact is ranked as more meritorious and more deserving of support. This model has proven successful in fostering innovation and discovery, so it is worth considering what consequences a blanket requirement that software be free and open-source might engender.

Such a requirement would, in our view, detract from the merit-based review process. When evaluating grant proposals that involve software development, the questions to be asked should be:

1. What will be the quality of the software in terms of the new science that it enables, either on the applications side or on the development side?
2. How will the software foster productivity? For example, how computationally efficient is it for a given task? How usable will the software be, and how quickly will other scientists be able to learn to use it for their own research?

A rigid, mindless focus on an open-source mantra is a distraction from these more important criteria. It can even be an excuse to ignore them, and creates an uneven playing field in which developers who prefer to work with a commercial platform are put at a disadvantage and potentially forced to adopt less efficient practices.

Although Gezelter[2] suggests that online code repositories would help young researchers to showcase their work, an open-source *requirement* actually puts young researchers at a particular disadvantage. The demands of tenure and promotion place special burdens to bring ideas before the community rapidly and also to secure funding quickly. Open-source requirements potentially force a scientist to choose between pursuing a funding opportunity versus implementing an idea in the quickest, most efficient, and highest-impact way. A strictly open-source environment may furthermore disincentivize young researchers to make new code available right away, lest their ability to publish papers be short-circuited by a more senior researcher with an army of postdocs poised to take advantage of any new code. This would contribute directly to the scenario that Gezelter wishes to avoid, namely, one where students leave behind "orphaned" code that will never be incorporated into mainstream, production-level software. Viewed in these terms, an open-source mandate degrades, rather than enhances, cyberinfrastructure.

How should the impact of software be measured? Scientific publications are a more sound metric than either the price of a product or whether its source code is available in the public domain. Software is meant to serve scientific research, in the same way that any other scientific instrument is intended. As such, the question should not be whether software is free or open source, but rather, what new science can be accomplished with it? Let us not allow political rhetoric to dictate how we are to do science. Let different ideas and different models (including open source!) compete freely and flourish, and let the community focus instead on the most important metric of all: what is good for scientific discovery.

Anna I. Krylov*,†
John M. Herbert*,‡
Filipp Furche§
Martin Head-Gordon‖
Peter J. Knowles⊥
Roland Lindh¶
Frederick R. Manby▽
Peter Pulay○
Chris-Kriton Skylaris▲
Hans-Joachim Werner□

†Department of Chemistry, University of Southern California, Los Angeles, California 90033, United States
‡Department of Chemistry and Biochemistry, The Ohio State University, Columbus, Ohio 43210, United States
§Department of Chemistry, University of California, Irvine, California 92697, United States
‖Department of Chemistry, University of California, Berkeley, California 94720, United States
⊥School of Chemistry, Cardiff University, Cardiff CF10 3AT, United Kingdom
¶Department of Chemistry—Ångström Laboratory, Uppsala University, Uppsala 751 05, Sweden
▽Center for Computational Chemistry, School of Chemistry, University of Bristol, Bristol BS8 1TH, United Kingdom
○Department of Chemistry and Biochemistry, University of Arkansas, Fayetteville, Arkansas 72701, United States
▲School of Chemistry, University of Southampton, Highfield, Southampton SO17 1BJ, United Kingdom
□Institut für Theoretische Chemie, Universität Stuttgart, Stuttgart 70174, Germany

■ AUTHOR INFORMATION

Corresponding Authors

*E-mail: krylov@usc.edu.
*E-mail: herbert@chemistry.ohio-state.edu.

Notes

The authors declare the following competing financial interest(s): A.I.K., J.M.H., and M.H.-G. are part owners of Q-Chem, Inc. P.P. is a part owner of PQS, Inc. F.F. has an equity interest in Turbomole, GmbH, and serves as a Scientific Coordinator. The terms of this arrangement have been reviewed and approved by the University of California, Irvine, in accordance with its conflict of interest policies. H.J.W., P.J.K., and F.R.M. are co-authors of Molpro. R.L. is a lead developer of Molcas. C.-K.S. is a founding author of ONETEP.

■ ACKNOWLEDGMENTS

The authors thank Axel Becke, Georg Kresse, Todd Martínez, and John Stanton for useful comments and supportive feedback.

■ REFERENCES

(1) National Science Foundation, "A Vision and Strategy for Software for Science, Engineering, and Education: Cyberinfrastructure Framework for the 21st Century" (http://www.nsf.gov/publications/pub_summ.jsp?ods_key=nsf12113); "Software Infrastructure for Sustained Innovation" (http://www.nsf.gov/funding/pgm_summ.jsp?pims_id=503489&org=NSF); and Department of Energy "Scientific Discovery through Advanced Computing" (http://science.energy.gov/ascr/research/scidac) (accessed April 2015).

(2) Gezelter, J. D. Open Source and Open Data Should Be Standard Practices. *J. Phys. Chem. Lett.* **2015**, *6*, 1168−1169.

(3) http://www.informationisbeautiful.net/visualizations/million-lines-of-code (accessed April 2015).

(4) http://www.nobelprize.org/nobel_prizes/chemistry/laureates/1998 (accessed April 2015); http://www.nobelprize.org/nobel_prizes/chemistry/laureates/2013 (accessed April 2015).

(5) Krylov, A. I.; Gill, P. M. W. Q-Chem: An Engine for Innovation. *Wiley Interdiscip. Rev.: Comput. Mol. Sci.* **2013**, *3*, 317−326.

(6) Werner, H.-J.; Knowles, P. J.; Knizia, G.; Manby, F. R.; Schütz, M. Molpro: A General-Purpose Quantum Chemistry Program Package. *Wiley Interdiscip. Rev.: Comput. Mol. Sci.* **2012**, *2*, 242−253.

(7) Furche, F.; Ahlrichs, R.; Hättig, C.; Klopper, W.; Sierka, M.; Weigend, F. Turbomole. *Wiley Interdiscip. Rev.: Comput. Mol. Sci.* **2014**, *4*, 91−100.

(8) Bochevarov, A. D.; Harder, E.; Hughes, T. F.; Greenwood, J. R.; Braden, D. A.; Philipp, D. M.; Rinaldo, D.; Halls, M. D.; Zhang, J.; Friesner, R. A. Jaguar: A High-Performance Quantum Chemistry Software Program with Strengths in Life and Materials Sciences. *Int. J. Quantum Chem.* **2013**, *113*, 2110−2142.

(9) Aquilante, F.; De Vico, L.; Ferré, N.; Ghigo, G.; Malmqvist, P.-Å; Neogrády, P.; Pedersen, T. B.; Pitonak, M.; Reiher, M.; Roos, B. O.; Serrano-Andrés, L.; Urban, M.; Veryazov, V.; Lindh, R. MOLCAS 7: The Next Generation. *J. Comput. Chem.* **2010**, *31*, 224−247.

(10) Baker, J.; Wolinski, K.; Malagoli, M.; Kinghorn, D.; Wolinski, P.; Magyarfalvi, G.; Saebo, S.; Janowski, T.; Pulay, P. Quantum Chemistry in Parallel with PQS. *J. Comput. Chem.* **2009**, *30*, 317−335.

(11) Skylaris, C.-K.; Haynes, P. D.; Mostofi, A. A.; Payne, M. C. Introducing ONETEP: Linear-Scaling Density Functional Simulations on Parallel Computers. *J. Chem. Phys.* **2005**, *122*, 084119:1−10.

(12) Boyd, D. B. Quantum Chemistry Program Exchange, Facilitator of Theoretical and Computational Chemistry in Pre-Internet History. Chapter 8 of *Pioneers of Quantum Chemistry*, ACS Symposium Series vol. *1122*, pp 221−273 (2013).

(13) See, for example, the Orca license. https://orcaforum.cec.mpg.de/license.html (accessed April 2015).

(14) Schmidt, M. W.; Baldridge, K. K.; Boatz, J. A.; Elbert, S. T.; Gordon, M. S.; Jensen, J. H.; Koseki, S.; Matsunaga, N.; Nguyen, K. A.; Su, S.; Windus, T. L.; Dupuis, M.; Montgomery, J. A., Jr. The General Atomic and Molecular Electronic Structure System. *J. Comput. Chem.* **1993**, *14*, 1347−1363.

(15) Valiev, M.; Bylaska, E. J.; Govind, N.; Kowalski, K.; Staatsma, T. P.; van Dam, H. J. J.; Wang, D.; Nieplocha, J.; Apra, E.; Windus, T. L.; de Jong, W. A. NWChem: A Comprehensive and Scalable Open-Source Solution for Large Scale Molecular Simulations. *Comput. Phys. Commun.* **2010**, *181*, 1477−1489.

(16) Hutter, J.; Iannuzzi, M.; Schiffmann, F.; VandeVondele, J. CP2K: Atomistic Simulations of Condensed Matter Systems. *Wiley Interdiscip. Rev.: Comput. Mol. Sci.* **2014**, *4*, 15−25. For the cp2k license, see http://www.cp2k.org (accessed April 2015).

(17) See, for example, the GAMESS license: http://www.msg.ameslab.gov/gamess/License_Agreement.html (accessed April 2015).

(18) http://www.gnu.org/licenses/gpl-3.0.en.html (accessed April 2015).

(19) Shao, Y.; Gan, Z.; Epifanovsky, E.; Gilbert, A. T. B.; Wormit, M.; Kussmann, J.; Lange, A. W.; Behn, A.; Deng, J.; Feng, X.; et al. Advances in Molecular Quantum Chemistry Contained in the Q-Chem 4 Program Package. *Mol. Phys.* **2015**, *113*, 184−215.