SAND2021-7334C

Encouraging pride in the tools we use

By Jacob Moxley and Reed Milewicz

In this paper we will discuss the concept of viewing tools as an extension of the people who build and use them, and why pride in those tools is a desirable cultural value for scientists and engineers. Scientists and Engineers use tools and experimental apparatus to accomplish their jobs every day, and for the majority of us the tools we use most often are our computers and software. One admirable aspect of many technical fields is the professional pride that practitioners take in the tools they use, even if they did not build them. A farmer is proud of his tractor and works to understand how to repair and customize it to his or her own needs. The electrician or a mechanic is proud to show their tools off and view their choices about which tool manufactures they choose as an expression of their professional selves.

As scientists and engineers, we should take no less pride in the attention to detail and craftmanship that goes into building the software we use each day. Why did you choose to use one python library over another, or one CI/CD pipeline as opposed to a competitor? Surely availability was one important aspect, but often the simple answer is “because it worked”. This is not always a poor answer, but we can and should encourage people to spend parts of their time and money into valuing software quality.

Documentation is the Key

Now the question becomes if we are to encourage pride in the scientific software we produce and use which aspects are admirable and why, so that we are not instigating dogma. The most uncontroversial point in my opinion is using software with well produced open source documentation. Open source documentation is the first piece towards a formal mathematical proof of the correctness and completeness of any results. This has major implications for the producers and users of all software, and scientific software in particular. Computer aided proofs are now being used broadly for sensitive applications in aerospace, cryptography and the web, with perhaps the most ambitious attempt in Project Everest.[1] Furthermore, proper technical documentation helps to drive the production of more well documented software built on top it. This is where human beings come into the iterative loop. If a developer or user starts with certain aspects of their workflow proven for them, think of the proof of correctness for numpy’s matrix multiply[2], then they are immediately prompted to test and prove their own new code or use case. This virtuous spiral also encourages the inclusion of a diverse skillset into any team that wishes to generalize their results. Technical writers, mathematicians, engineers, and scientists can bring different points of view to gather a consensus around which proofs hold up under scrutiny. Open source documentation then evolves not just computer aided proofs, but also grows the number of teams who can understand and make good use of those proofs.

Open Source for Everyone

Another way to improve the tools we build ourselves is to model them after the best tools currently available. At all levels in scientific and engineering fields we can encourage practitioners to be interested in the design choices of the people who build the tools they use so they can reproduce the best

[1] https://project-everest.github.io/

[2] https://home.cs.colorado.edu/~srirams/courses/csci3104-spr15/lectures/l2Note.pdf

features. At a high level, programing languages have different views on how when and where to teach and implement packaging. In my opinion the gold standard for integrating proper packaging and incorporating it early into the learning process is Rust.[3] The second piece to encourage the instinct to open source in engineering disciplines as rigorously as we do in computer science. This does not need to mean open sourcing every project produced, but it does mean building software in way that you could do so every single time. By encouraging both open source code and quality documentation as a default, rather than an eventual goal, we can hope to instill pride in the sources and methods of producing scientific software as a cultural norm in both producers and consumers.

Agnosticism All Around

The third piece that we can try to generate pride in is the concept of hardware, operating system, and operator agnosticism. For scientists and engineers it may not be immediately obvious why this is so important. The ability for software to preform the same computations on multiple machines is fundamental to the scientific concept of reproducibility. The scientific community has increasingly emphasized open source data sets as a manner to ensure that people have produced their results in good faith. Producing software that is hardware and operating system agnostic allows for consumers of data to efficiently reproduce results to evaluate for themselves. Recent consequential examples of this include the OAS report on the 2019 Bolivian election and some of the first primarily data based prosecutions of insider trading by the SEC.[4][5] The OAS report on the 2019 Bolivian election is particularly heartening because it shows how the scientific process for software can sharpen and guide confidence in even the most controversial of situations. The OAS dataset and software that was used to analyze it was completely open source, and keen observers from a variety of viewpoints helped to find errors and contribute revisions to the original paper and software which was periodically updated to represent the most up to date information. This was done in a rapid fashion only made possible by the interoperability of the published software on the variety of machines used by distributed researchers across the globe. In practice we can encourage the use of tools like Docker, and Binder, along with methods like code compliance and testing to make agnosticism the default for projects that may be run by people from a variety of backgrounds.

Modularity and Interoperability

The final concepts that we can enculturate as worthy goals are modularity and interoperability. Modularity and interoperability are both valued in technical fields in a variety of applications. Everyone wants the tool that handles more events correctly, but no one wants to lug around extra gear, be it a mechanic that is digging through their toolbox or an engineer deploying a software package. Modularity ought to be a goal for all engineering projects from both a software and a human perspective. Software modularity allows for flexible rapid deployment by allowing for a full accounting of dependencies, which flows directly into operating system agnosticism, and by helping with efficient deployment across a wider variety of hardware. From a user and developer perspective modularity helps to leverage the powerful human capacity for translational learning.[6] We all have a finite space for tools in our bag and

---

[3] https://doc.rust-lang.org/cargo/

[4] https://www.oas.org/en/media_center/press_release.asp?sCodigo=E-109/19

[5] https://www.reuters.com/article/bc-finreg-data-analytics/secs-advanced-data-analytics-helps-detect-even-the-smallest-illicit-market-activity-idUSKBN19L28C

[6] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4383335/

limited space in our brains for how to use them. Allowing for the repurposing of those tools dramatically increases our list of solvable problems. The same set of differential equations may describe a variety of physical phenomena and the software we use to simulate them is likely to be used in ways we cannot foresee. By enabling both machines and humans to repurpose specific parts of our processes we can help contribute to a richer knowledge ecosystem and a deeper software stack. Lastly, interoperability is a nearly universal scientific and technical goal. Number theory was an obscure branch of pure mathematics for hundreds of years until it suddenly became of a topic of intense discussion as a result of the proliferation of cryptography.[7] Likewise we cannot know where our contributions might end up, but we can make them as widely understood as possible. Interoperability is the enabling piece that allows for software to fuse ideas over time and humans to move across teams.

Just as plumbers can be interested in the grade of tool steel their wrenches are made from scientists and engineers can take the time to evaluate software quality. It is incumbent upon the producers of scientific software to explain which software characteristics they have chosen to emphasize in their products and how these characteristics can help teams of people function more effectively. Open source methods, agnosticism, modularity and interoperability are goals that every technical team can strive for and prize in the tools they utilize. Every producer and consumer of scientific software can then play a part in the virtuous spiral improving software quality and incentivizing the production of quality software.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DENA0003525. (SAND2020-6634 C)

[7] https://www.britannica.com/science/number-theory