HPC Communities of Practice

Andy R. Terrel
University of Texas
Continuum Analytics, Inc
andy.terrel@gmail.com

Matthew Turk
Columbia University

ABSTRACT
Writing software for HPC systems is a task fraught with many challenges. We argue when developed in an open community of practice, many of these challenges can be captured and solved benefiting many. Additionally we call to the well established rule that software for a larger community is as a guide for building better tools.

1. INTRODUCTION
Two recent events have shaped our view on producing accessible and robust software. The first is a rant accidently posted by Steve Yegge of Google, Inc on the state of culture in Google versus Amazon [0]. The second is a recent report on the extremely low bug count of the highly open reference Python interpreter [1]. Both events highlight the need for coordination among communities to build their highly complicated tools rather than the more traditional siloed software teams.

Steve Yegge’s now infamous post referenced a mandate by the CEO of Amazon to “dog-food” APIs from the platform team. Dog-fooding, in a nutshell, is the practice forcing the use of an alpha-stage product to find bugs and required usability updates; by forcing teams to utilize software themselves for day-to-day tasks, limitations and failures are raised to the level of critical, concrete tasks. The practice was credited as the success of the now-transformative Amazon Web Services (AWS) being deployed and its ascendance to the de facto standard for cloud systems. Even Globus, the backbone connecting our beloved HPC machines, uses it extensively. While Amazon’s main source of income was selling books, anyone would be remiss to call them a typical bookseller – they self-describe their work more like a computer science department. Joining resources between the conceptual divisions of book retail and software development, resulting in a platform for selling books, provided a launching pad for Amazon to produce tools to fuel the entire cloud computing ecosystem. This interaction within their community can be credited with producing a broadly transformative product.

In our second example, a report by Coverity has shown the CPython project to have an exceptionally low bug count, fewer than 5 defects per 1000 lines of code, far below industry standards. CPython is another example of communities of practice. The CPython community is highly open and transparent with an intentional focus of mentoring a diverse set of contributors. The community includes mentorship and education groups devoted to encourage more participation, a quality rarely seen in software projects and discouraged by Brook’s law of software engineering. The technique of encouraging participation in large projects is often criticized in our community, citing challenges to publication and renewed funding, but this recent report provides a concrete example of highly open practices leading to better software.

Open communities of practice do exist in HPC Software and typically account for the largest use of hours on supercomputing resources. It is our belief that any pursuit of Extreme Scientific Software would have a multiplicity of benefits by adhering to these simple techniques of collaboration. The techniques of dog fooding and open participation can make HPC Software better. To this end, we highlight the Enzo and yt communities that we have worked with and some practical steps that can address the lack of use of this model.

2. OPEN HPC COMMUNITIES
Development can often be divided into two primary components: development in support of a specific application and development in support of the broader code base. Within HPC communities, these often take the form of “library-driven” development and “application-driven” development. Often this division results from the individuals conducting the definition of requirements, the design and implementation of those requirements, and the manner in which improvements to code are deployed to the broader community. Typically “application-driven” development is characterized by development in support of specific projects or grants, and is often heavily tied to a specific publication or class of projects; in contrast, “library” development is typically characterized by developing without specific applications in mind, or with broad-classes of applications in mind.

From our experiences within the HPC community, both approaches are necessary – however, for many domains of problems, “application-driven” development leads to more focused, if lower performing, libraries and a shorter “time to...
science.” Furthermore, infrastructure components are often designed by the individuals who will be applying them; this is a hallmark of a community of practice, where uptake of newly developed features can be quite high. The process of dog-fooding, of application-driven development and of working practitioners guiding resources and investment of time is characterized by a focus on utility. This is not exclusively positive, as it can also lead to shortcuts, compromises and a reinvention of existing components; however, it can also lead to higher productivity and synergistic technology transfer between groups.

Within the Enzo and yt communities [3], we have identified several processes that both benefit from and also struggle with the “need-driven development” structure encouraged by dog-fooding. On the positive side, what we have primarily seen is that time is spent developing features that are immediately relevant to individuals and researchers – however, the downside of this is that often, infrastructure improvements are put off, and once they do become necessary the time required to do so comes at the expense of further short-term development goals. As an example of this, within both Enzo and yt scaling issues for addressing large processor counts have long been identified; the solutions to these problems would be invasive, but with remarkable reward for a subset of computations. Because they require substantial commitment and investment not only from one or two individuals, but from the whole community, and also at the expense of short-term goals, they have not yet been addressed or implemented. While this may seem like a failing of the “dog-fooding” methodology, we instead view it as a mixed bag. It is true that longer term goals have been set aside for the time being, but ultimately the shorter-term scientific output from the community has not suffered; rather, the lack of invasive changes has instead fostered a growing community of individuals contributing to the code base and widening its applicability, and ultimately forming a large enough base of individuals to perform the necessary improvements for longer-term development. We therefore see the goals of the community as aligned with but not identical to the goals of overall code improvement. While in the long term this may lead to bottlenecks and ultimately the abandonment of existing code bases in favor of newer, different code bases, the resource which likely bore the greatest investment (community) can and may survive such transitions; however, it is likely to be tested, in a much similar way to how code bases are tested as new HPC system architectures are developed and deployed.

3. CONCLUSION: PRACTICAL POLICIES
To conclude, our message is simple. The better the communities using Extreme Science systems the better use we will see. By investing in policies and incentivizing communities of practice, these machines will return many times more results than small siloed groups of PIs. Finally we list a set of policies we would like to see incorporated in the workshops conclusion.

1. Grant cross cutting research projects across disciplines with both science and tool production goals.
2. Require community presence for duration of project, i.e. blogs, mailing lists, open code repositories.
3. Support collaborations with institutes such as Software Sustainability Institute and open source scientific software companies.
4. Support development of social capital between projects. For example see the DARPA XDATA program.
5. Increased focus on a long tail of science, bringing up tools for non-experts.

4. REFERENCES