Walking the talk: adopting and adapting sustainable scientific software development processes in a small biology lab.

Michael R. Crusoe[1]
C. Titus Brown[2,1*]
1 Microbiology and Molecular Genetics,
2 Computer Science and Engineering,
Michigan State University, East Lansing, MI, USA
* Corresponding author: ctb@msu.edu

July 2012

Abstract

There's a funny thing about arguing that scientific software should be better designed, better tested, and more openly developed: one's peers may reasonably hold you to this higher standard. This is the story of our attempt to figure out how to hold ourselves to these higher standards on the khmer project, a small, single-lab software project in biology.

1 Introduction

In July of 2013, Michael R. Crusoe (MRC) was hired in in C. Titus Brown's (CTB) Laboratory of Genomics, Evolution, and Development at Michigan State University. This brief recounts MRC's experience so far in the lab as an engineer and budding researcher, including his assessment of the group's main body of code; CTB's reaction and engagement with the resulting issues; and a collaborative discussion of how best to prioritize and pace infrastructure work, research work, and the associated reduction (or accumulation) of technical debt.

The GED lab is in a phase of tool building, and the main body of code is the khmer suite of nucleotide sequence analysis tools [cite], for which MRC is now responsible. This code is primarily aimed at the "Big Data" problem of efficiently preprocessing very large amounts of data being produced by sequencing machines that have become available in the last five years. It is now four years old and the basis for a published paper, a PhD thesis, an invited book chapter, and three submitted preprints, as well as about five papers in various stages of development. khmer grew out of specific analysis needs, and was developed primarily on startup funding and as part of a USDA grant. Its development has led to at least two additional grants, including the NIH BIG DATA grant that now supports MRC. Over its lifetime khmer has had 13 different contributors, seven of which are currently active. khmer is largely written in C++ with a thin, hand-coded Python wrapper; the unit and functional tests are entirely written in Python. (diagram@@),

Dramatis personae: CTB's background includes about 20 years of research software development, starting from his work on the Avida project (@@wikipedia page; 1993 Avida ALIFE paper). In addition to research software, he participated in a number of small open source projects, largely in Python; from 2004-2008 his primary software development focus was on testing, when he created several testing tools. However, his PhD is in developmental biology and his research focus has always been on the intersection of domain-specific problems and effective open source programming.

MRC's background includes computer systems engineering, free and open-source systems, and microbiology; this combined with his prior experience as a user of the khmer suite gave him plenty of standing

to generate opinions about how to implement CTB's goal of "better science through superior software." [cite]

Project history: The khmer project was started by CTB as part of his research when he arrived at MSU in 2008 as an Assistant Professor. It grew from a very small initial core of exact substring counting functionality to provide an implementation of a memory-efficient CountMin-Sketch data structure (manuscript in preparation); a sequence assembly graph implementation using Bloom filters [Pell et al., 2012]; and an implementation of a novel algorithm for lossy compression of data structure [Brown et al., 2012]. Its estimated user base is in the 100s, and it is currently being extended in a number of ways (see below). khmer started out under git version control and was developed using "stupidity-driven development (SDD)" a term coined by CTB in which test development is primarily driven by the need to prevent regressions to previously observed bugs.

Throughout the khmer project, there have been a number of conflicting development goals: first, khmer serves as a research testbed for the development of novel data structures and algorithms; second, khmer also provides novel end-user functionality that enables certain types of data analysis; and third, we are working to integrate khmer into a larger research ecosystem of tools. This last goal is the hardest and requires the most collaborative outreach, because khmer primarily provides a set of filters for preprocessing data: the input data is generated by upstream research instruments and software filters, and khmer's output is fed into downstream research software. Therefore, khmer must interact correctly at both a technical and scientific level with other software that is completely out of our control.

Specific plans for khmer's future development include significant improvements in its processing efficiency (@eric mcdonald posa chapter), the implementation of novel algorithms for scaling (@cite infinite assembly/career grant), the implementation of specific novel algorithms for data-driven biological discovery (@cite NIH R01), integration with existing pipelines and user interfaces (@cite USDA grant and R01), and the extension of khmer as a general research platform for rapid exploratory development of novel substring approaches to biological data analysis.

In sum, we plan to optimize and scale khmer, add specific new functionality, and integrate it with other software, all while preventing software regression and expanding its flexibility for as-yet unplanned future research. These plans are simply impossible without some sort of software development process, which (at the time of MRC's arrival) could at best be described as "haphazard."

2 Initial Evaluation and Short-term Plans

To guide our development of a rational software development process, we applied the Software Sustainability Institute's Criteria-based assessment checklist [cite] to the khmer project shared the results with the community [cite, storify the tweets?]. The summary from that report was grim: khmer met 17 of 42 (or 40%) of of the SSI's criteria for Usability, and 43 of 119 (or 36%) of the criteria for Sustainability, for an overall fulfillment of 43 of the 119 items, or 36%.

The low fulfillment of the SSI's criteria is slightly embarrassing for the group for several reasons: CTB is a collaborator and co-author with SSI; quotes from him are featured in case study presentations [cite]; he is cited as someone who is doing things the correct way [recomputation manifesto citation]; and he regularly criticizes the field for poor software process. This is essentially the point, however: because these goals were highly valued within the lab, and because the software filled a novel niche, CTB successfully argued for resources to address these issues [cite R01 blog post]. Making significant progress towards these ends isn't merely a matter of budgets and hiring: even with a full time hybrid software engineer/bioinformatician on the team it is not straightforward to prioritize infrastructure work, let alone navigate between the core research work, user support, community engagement, and collaborations.

Our guiding principle, as eloquently put by MRC's first mentor, is "What is best for science?" /cite-Nagy2007. The (re)usability of khmer suite by others and the group's own ability to continue to extend and innovate with the codebase are two concrete areas of concern. On the usability front the most pressing issue is the lack of versioned releases and the fragility of the complicated build procedure due to a mixture of C++, C, and Python. There is an existing body of tests but we do not know how well they cover the C++ code. Moreover, there is no regular continuous integration (automated extension of the existing tests) as code is committed.

We are working towards a versioned release process and we are setting up a continuous integration system using Jenkins. This system already builds the codebase and runs the test; soon it will measure and report statement coverage of both Python and C++ code. Once we we have a measurement of how much of the core functionality is not covered by automated tests, we will fix those deficiencies and be able to consider more ambitious refactoring.

3 Medium-term Plans

We hope to release a "1.0" version in April 2014, and during this period we plan to adopt a number of new practices.

First and foremost, we will move to a "pull request" model for contributions, also known as "github flow," in which both small and large changes are started on branches and then discussed, debated, and updated in a persistent conversational form using GitHub's Pull Request functions. This can be used to educate contributors, require new members of the lab to adopt good coding practice, and (perhaps most importantly) enforce code review prior to merge. Preliminary experience suggest that this model is easy for long-time software developers to adopt.

Secondly, we are working on a series of integration tests in which khmer's functionality is tested in concert with a number of externally developed packages. This serves both an inward facing goal, where we can verify that our software plays correctly with other codebases, and an outward facing goal, where we can essentially place other codebases on which we depend "under test." While this latter goal might seem a bit passive aggressive, in practice it is useful because so few bioinformatics packages have any automated tests whatsoever.

One additional challenge that we could address by instrumenting the integration tests is to detect performance regressions in the khmer code base. Because khmer may run for hours, days, or weeks on very large data sets with properties that we cannot yet model accurately, running khmer on real data is the only way to observe the consequences of code changes on performance.

Finally, we want to put in place a set of explicit citation requests. We do not know how to measure and report khmer usage, although altmetrics is beginning to provide measures of online impact. Conservatively, paper citations seem likely to be the most practical and realizable way of defining khmer's impact on science, and so we will lay out distinct guidelines for appropriately citing algorithms, software, and protocols.

4 Persistent Challenges in Research Software Development

In the long term, we expect to face three major challenges in continuing to develop khmer.

First, we need to show the value of the process, so that we and others can convince colleagues and funding agencies that software development processes should be a first-class participant in publication and funding considerations. This will almost certainly have to be done by doing a better (faster, higher quality) job of tackling core scientific and biological questions, and will need to be measured in publications and citations.

Second, we must continually reevaluate how much and where to refactor and test. In the face of changing input data (due to instrument and experimental protocol changes), different expectations for output (bioinformaticians invent a new format every 5 minutes on average), competing algorithms with poor replicability, etc., we believe we could spend 100% of our time on quality control without developing anything truly new. However there is little or no reward in academia for merely continuing to produce functioning software: software maintenance grants are few and far between. So our core software maintenance efforts must instead be directed at enabling us to agilely tackle research problems, i.e. focus on issues that we don't yet understand or for which we don't have much intuition. While this may seem like a standard software engineering problem, we believe that the nature of pure research may lead to additional challenges here.

Third, we face systemic and severe cultural challenges in terms of recruiting software developers and researchers. Inevitably new lab members are undertrained in most of what we do, including testing, version control, good computational hygiene, data science, and/or the domain of biology. These are a lot of subjects to train new people in, and we have yet to establish an effective lab culture. On the converse

side, of course, we expect lab graduates to be increasingly employable in both academia and biotech; moreover, the lab reputation of caring about such things has started to attract people who are already somewhat better trained.

5 A brief dialogue

MRC: Before the evaluation was started, how mature did you think khmer was as a project?

CTB: Early adolescence. We'd demonstrated scientific value, and, between the automated tests and the version control, I felt that we were in the 99th percentile of bioinformatics codebases - yes, it's a nice, low bar. However, I knew the software was not as robust as it needed to be to support continual incremental development and refactoring, especially by new developers. I knew much more work needed to be done on the testing, in particular.

MRC: The work required to nurture a project into one that is more sustainable is non-trivial. How do you justify those resources?

CTB: I haven't entirely figured that out. The hope is that we can adapt and innovate faster because we are working off of a more stable code base. So far, so good. But can we scale this to a bigger group of developers? I don't know.

MRC: What particular challenges do you think life scientists face in making the software artifacts of the research process reusable and sustainable?

CTB: The two primary challenges are cultural (which was expected) and technical (which was unexpected). By cultural, I mean that there is virtually no culture of computational reproducibility or software development in biology, which makes it hard to discuss much less justify. Technically, the infrastructure and tools for enabling reproducibility are still quite young and do not fit the needs of subdomains terribly well. For example, I thought that other fields would have tools for provenance and workflow tracking that we could easily adapt; this is simply not the case. In fact, our most effective set of tools has emerged from open source and data science work, including the excellent Python and GitHub communities, IPython Notebook, and cloud computing infrastructure.

MRC: If you could change one thing about the project process from the past, what would it be?

CTB: With hindsight, I would place greater emphasis on putting some of the "trivial" little scripts we use for data manipulation under test and documentation. I've had several experiences where bugs in those simple scripts have caused days or weeks of stress, and the lack of good documentation for these scripts has been by far the biggest complaint of users.

MRC: What has been the most challenging aspect of the project, from a process perspective?

CTB: Choosing how to allocate finite time to development, testing, communication, fund-raising, and infrastructure – essentially what every startup goes through – but in a background of blank incomprehension or outright negativity from most of my colleagues and administrators.

MRC: What has been the most useful change since my arrival?

CTB: I expect the adoption of "github flow" (@cite) to have the biggest positive long-term impact; it really helps with visibility and discussion of features, and it has already helped train new contributors in our expectations.

Acknowledgments

MRC has been funded by AFRI Competitive Grant no. 2010-65205-20361 from the USDA NIFA and is now funded by the National Human Genome Research Institute of the National Institutes of Health under Award Number R01HG007513; both under C. Titus Brown.

References

[Brown et al., 2012] Brown, C., Howe, A., Zhang, Q., Pyrkosz, A. and Brom, T. (2012). A reference-free algorithm for computational normalization of shotgun sequencing data. In review at PLoS One, July 2012; Preprint at http://arxiv.org/abs/1203.4802.

[Pell et al., 2012] Pell, J., Hintze, A., Canino-Koning, R., Howe, A., Tiedje, J. and Brown, C. (2012). Scaling metagenome sequence assembly with probabilistic de bruijn graphs. Accepted at PNAS, July 2012; Preprint at http://arxiv.org/abs/1112.4193.