Argonne NATIONAL LABORATORY

WHAT COMMUNITY CODES DO: CASE STUDIES

ANSHU DUBEY
Mathematics and Computer Science Division
Argonne National Laboratory

August 9, 2016
ATPESC
St Charles, IL

OUTLINE

- Motivation & Logistics
- Community Development
- Best Practices

MOTIVATION & LOGISTICS

WHY COMMUNITY CODES ?

- Scientists can focus on developing for their algorithmic needs instead of getting bogged down by the infrastructural development
- Graduate students do not start developing codes from scratch
  - Look at the available public codes and converge on the ones that most meet their needs
  - Look at the effort of customization for their purposes
  - Select the public code, and build upon it as they need

Important to remember that they still need to understand the components developed by others that they are using, they just don't have to actually develop everything themselves. And this is particularly true of pesky detailed infrastructure/solvers that are too well understood to have any research component, but are time consuming to implement right

- Researchers can build upon work of others and get further faster, instead of reinventing the wheel
  - Code component re-use
  - No need to become an expert in every numerical technique
- More reliable results because of more stress tested code
  - Enough eyes looking at the code will find any errors faster
  - New implementations take several years to iron out the bugs and deficiencies
  - Different users use the code in different ways and stress it in different ways
- Open-source science results in more reproducible results
  - Generally good for the credibility

MODELS: "CATHEDRAL AND THE BAZAAR", ERIC S. RAYMOND

- **The *Cathedral* model**
  - Code is available with each software release
  - Development between releases is restricted to an exclusive group of software developers.
    - GNU Emacs and GCC are presented as examples.
  - Central control models

- **The *Bazaar* model**
  - Code is developed over the Internet in view of the public.
  - Raymond credits Linus Torvalds, leader of the Linux kernel project, as the inventor of this process.
  - Distributed control models

SCIENTIFIC CODES

- Mostly follow the cathedral model
- Many reasons are given, some valid, others spring from bias
- The valid ones
  - The code quality becomes hard to maintain
  - Hard to find financial support for gate keeping and general maintenance
  - Typical user communities are too small to effectively support the bazaar model
  - The reward structure for majority of potential contributors is incompatible
- The not so valid ones
  - Codes are far too complex
  - Competitive advantage from owning the code

The real reason many times is simply the history of the development of the code and the pride of ownership

THE BENEFITS OF THE BAZAAR MODEL

- Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix will be obvious to someone
  - More varied test cases that demonstrate bugs
  - Debugging can be effectively parallelized.
  - The infrastructure limitations are quickly exposed
- Capability addition is rapid, codes can do more
  - A corollary to that is a good extensible design
  - Users always want something more and/or something different from what is available
  - Greater knowledge pool operating together, more possibility of innovation

THE PITFALLS OF THE BAZAAR MODEL

- Many of the benefitting reasons can equally easily go the other way
  - Bigger knowledge pool can also mean more conflicting opinions
  - Prioritizations can become extremely challenging
- Gatekeeping can become a huge challenge for maintaining software quality
  - Scientific codes have their own peculiarities for verification and validation that can be extremely challenging
  - The orchestration of capability combination is harder when there is physics involved because many times it just won't play well together

SCIENTIFIC COMMUNITY CODES CAN FOLLOW SEVERAL DIFFERENT PATHS :

- The most common path
  - Someone wrote a very useful piece of code that several people in the group started using
  - Collaborations happened
  - People moved and took the code with them
  - Critical mass of users achieved, code becomes popular
- No focused effort to build the code
  - Usually very little software process involved
  - For the whole code, limited shelf life

A MORE SUSTAINED PATH

- Sometimes enough like minded people take it a step further
  - Some long term planning resulting in better engineered code
  - Thought given to extensibility and for future code growth
  - As the code grows so does its community supported model
- This model is still relatively rare.
  - The occurrences are increasing

A DESIRABLE PATH

- Explicit funding to build a code for a target community
- Implied support for the design phase
- The outcome is expected to be long lasting and well engineered
- The occurrences are even rarer
- And it is getting increasingly harder
- When it works outcome is more capable and longer lasting codes

COMMUNITY DEVELOPMENT

Argonne NATIONAL LABORATORY

WHAT COMMUNITIES ?

- Community/open-source approach more common in areas which need multi-physics and/or multi-scale
- A visionary sees the benefit of software re-use and releases the code
- Sophistication in modeling advances more rapidly in such communities
- Others keep their software close for perceived competitive advantage
  - Repeated re-invention of wheel
  - General advancement of model fidelity slower

Let us examine what does it take to build a community code

COMMUNITY BUILDING

- Popularizing the code alone does not build a community
- Neither does customizability – different users want different capabilities

So what does it take ?

- Enabling contributions from users and providing support for them
- Including policy provisions for balancing the IP protection with open source needs
- Relaxed distribution policies – giving collective ownership to groups of users so they can modify the code and share among themselves as long as they have the license

More inclusivity => greater success in community building
An investment in robust and extensible infrastructure, and a strong culture of user support is a pre-requisite

CONTRIBUTION POLICIES

- Balancing contributors and code distribution needs
  - Contributor may want some IP protection
- Maintainable code requirements
  - The minimum set needed from the contributor
    - Source code, build scripts, tests, documentation
- Agreement on user support
  - Contributor or the distributor
- Add-ons: components not included with the distribution, but work with the code

SURVEY OF IDEAS USE-CASES

IDEAS scientific software productivity project: www.ideas-productivity.org

- Five application codes and four numerical libraries
- All use version control, and all but one use distributed version control
- Builds are evenly divided between GNU make and CMake
- All provide documentation with some form of user's guide, many use automated documentation generation tools
- All have testing in some form, a couple do manual regression testing, the rest are automated
- Roughly half make use of unit testing explicitly
- Majority are publicly available

SUMMARY FROM COMMUNITY CODES WORKSHOP (2012)

http://flash.uchicago.edu/cc2012/

- Codes – FLASH, Cactus, Enzo, ESMF, Lattice QCD code-suite, AMBER, Chombo, and yt
- Software architecture is almost always in the form of composable components
  - Need for extensibility
- All codes have rigorous auditing processes in place
- Gatekeeping for contributions, though models are different
- All codes have wide user communities, and the communities benefit from a common highly exercised code base

COMMUNITIES HEADED TOWARDS COMMUNITY CODES

- Climate modeling
  - DOE effort – ACME
    - Started in 2014
    - Community Model – many groups
    - Many practices in place

- Accelerator
  - People thinking about it
    - Whitepapers
  - Objective – avoid duplication, get some convergence,
    - Also more believable results

COMMON THREADS

- Open source with a governance structure in place
- Trust building among teams
- Commitment to transparent communications
- Strong commitment to user support
- Either an interdisciplinary team, or a group of people comfortable with science and code development
- Attention to software engineering and documentation
- Understanding the benefit of sharing as opposed to being secretive about the code

BEST PRACTICES

Argonne NATIONAL LABORATORY

SOFTWARE PROCESS

Baseline

- ❑ Invest in extensible code design
- ❑ Use version control and automated testing
- ❑ Institute a rigorous verification and validation regime
- ❑ Define coding and testing standards
- ❑ Clear and well defined policies for
  - ❑ Auditing and maintenance
  - ❑ Distribution and contribution
  - ❑ Documentation

Desirable

- ❑ Provenance and reproducibility
- ❑ Lifecycle management
- ❑ Open development and frequent releases

Many of these practices have been covered in earlier lectures

A USEFUL RESOURCE

https://ideas-productivity.org/resources/howtos/

- **‘What Is’ docs**: 2-page characterizations of important topics for SW projects in computational science & engineering (CSE)
- **‘How To’ docs**: brief sketch of best practices
  - Emphasis on ``bite-sized'' topics enables CSE software teams to consider improvements at a small but impactful scale
- We welcome feedback from the community to help make these documents more useful

OTHER RESOURCES

http://www.software.ac.uk/

http://software-carpentry.org/

http://flash.uchicago.edu/cc2012/

http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1001745

http://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=4375255

http://www.orau.gov/swproductivity2014/SoftwareProductivityWorkshopReport2014.pdf

http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6171147

INTERDISCIPLINARY INTERACTIONS

A partnership model that works

- Science users treat the code as a research instrument that needs its own research
- Developers and computer scientists interested in a product and the science being done with the code
  - Helps to have people with multidisciplinary training
- Comparable resources and autonomy for the developers
  - And recognition of their intellectual contribution to scientific discovery
- Careful balance between long term and short term objectives

COMMUNITY CODES: SUMMARY

- Open source with a governance structure in place
- Trust building among teams
- Commitment to transparent communications
- Strong commitment to user support
- Either an interdisciplinary team, or a group of people comfortable with science and code development
- Attention to software engineering and documentation
- Understanding the benefit of sharing as opposed to being secretive about the code

IT IS EXTREMELY IMPORTANT TO RECOGNIZE THAT SCIENCE THROUGH COMPUTING IS ONLY AS GOOD AS THE SOFTWARE THAT PRODUCES IT