Community Codes: How They Develop and What it Takes to Develop One – Part I

Anshu Dubey

With slides from
Brian O'Shea (Enzo)
Cecelia DeLuca (ESMF)
Matt Turk (Enzo and yt)

The Development and Funding Models

Examples from Three Different Community Codes

“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
  - Scientists tend to be skeptical of software engineering
  - 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

Open Source Benefits

- Nobody can pull the plug on you.
  - You have the source code, free to use and modify, in perpetuity.
  - That includes me
- You don’t have to pay
  - But you might be asked to help generate funding
  - You pay with your time and attention and what you give back.
  - Stakeholders are power (if you can figure out how to tap it)
    - Not all stakeholders are equivalent
    - User count may not be as helpful as vocal collaborators

Object-Oriented Programming Helps

- Strong interfaces and encapsulation (enforced by the language or build system) enables community participation.
  - Users can try derived classes and get their code running without too much direct hand-holding.
  - Open-source means they can change interfaces locally.

- Design-by-Contract (DbC)

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

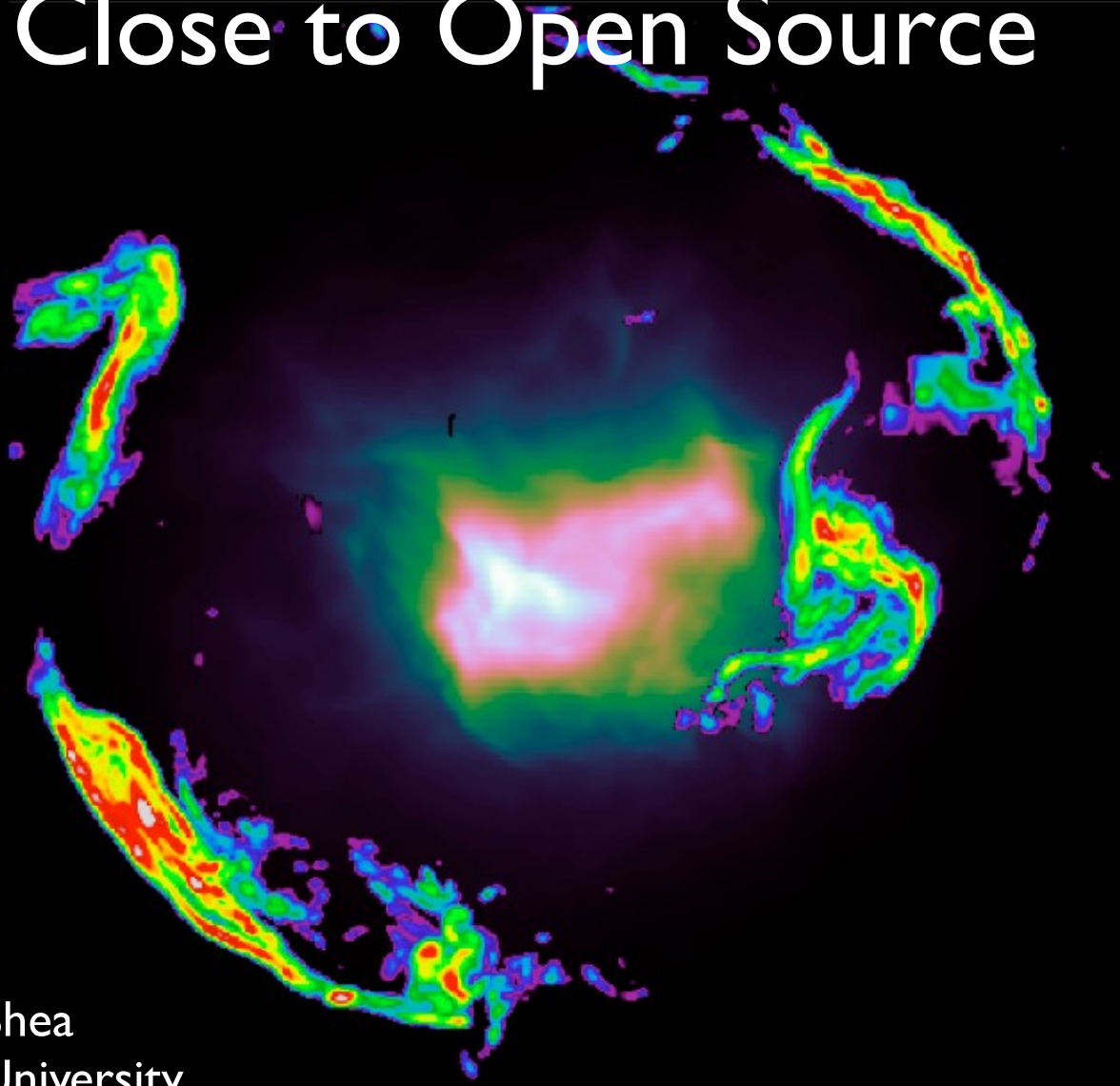
A More Sustained Path

- 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

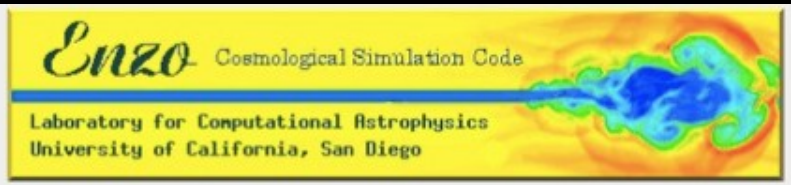
Examples: Enzo Transitioning from Close to Open Source



Brian W. O'Shea
Michigan State University

1996-2003: closed-source!

March 2004: Enzo 1.0



General

- Description
- Features
- Authors
- Acknowledgements
- News

Software

- License
- Requirements
- Download
- Code Development

Documentation

- User Guide
- Tutorial
- Cookbook
- Developer's Guide
- Change Log
- Notes

Publications

- Refereed Articles
- Popular Articles
- Press Releases
- Images
- Animations

General Information

Description

Enzo is an adaptive mesh refinement (AMR), grid-based hybrid code (hydro + N-Body) which is designed to do simulations of cosmological structure formation. It uses the algorithms of Berger & Collela to improve spatial and temporal resolution in regions of large gradients, such as gravitationally collapsing objects. The Enzo simulation software is incredibly flexible, and can be used to simulate a wide range of cosmological situations with the physics packages described below (see Features).

Enzo has been parallelized using the MPI message-passing library and can run on any shared or distributed memory parallel supercomputer or PC cluster. Simulations using as many as 1024 processors have been successfully carried out on the San Diego Supercomputing Center's Blue Horizon, an IBM SP.

Enzo was released to the public on March 1, 2004. To obtain a copy of the code go to the download page. For more information please contact enzo-l@ucsd.edu at the Laboratory for Computational Astrophysics.

Features

The public release of Enzo includes (but is not limited to) the following physics:

- N-body gravitational dynamics using the particle-mesh method
- hydrodynamics using both the piecewise parabolic method (PPM) and the finite-difference method used in the Zeus

The Enzo community today

- 25 contributors (~12 active developers) at >10 institutions
- ~200 people on enzo-users mailing list (~50% active?)
- ~80 million SUs devoted to Enzo simulations in 2011 from NSF, NASA, DOE (with more in 2012)
- Financial support from NSF (AST, OCI, PHY), NASA, and DOE
- Complementary community: **yt** (http://yt-project.org)

Development model

- Entirely distributed development model: small number of devs per institution!
- Code distribution using mercurial (BitBucket)
- Use code forks / pull requests to move features from development branches into main branch of
  the code
- Almost all development discussion takes place on archived, public mailing list and on Google DOcs (meeting notes emailed out)

Community support

- Most developers are astrophysicists "scratching their own itch" (and funded to do science!)
- Development spurred by ~1.5 workshops/year + periodic task-oriented "code sprints"
- Active mailing lists for users and developers
- Development funded by many streams: universities, federal agencies, postdoctoral fellowships
- Complementary yt development has helped to spur usage of Enzo!

Impact

- Enthusiastic and heavily-involved user/developer community
- Enzo is widely known in astrophysics - strongly represented in code comparisons, conference talks/posters - and highly trusted
- Code is flexible and extensible: high science/dollar!
- Has spurred development of open-source science
- Involvement in this community has strongly affected young scientists' career trajectories

Challenges

- No "Fearless Leader" of development process: hard to make major code revisions (esp. user-facing)
- Part-time developers: distractions, hard to do "boring but important" infrastructure projects
- Significant work required to build consensus and keep community together!

Conclusions

- Conversion of a code from closed-source to an open-source community code is not without technical and sociological challenges.
- For the Enzo collaboration, this transition has been worth it:
  - Enhanced transparency/reproducibility (more trust in the code)
  - Larger user base: more eyes on the code, wider adoption

  More and **better** science per dollar!

Example 3 : ESMF Community infrastructure for building and coupling high performance climate, weather, and coastal models

Vision

- Earth system models that can be built, assembled and reconfigured easily, using shared toolkits and standard interfaces.
- A growing pool of Earth system modeling components that, through their broad distribution and ability to interoperate, promotes the rapid transfer of knowledge.
- Earth system modelers who are able to work more productively, focusing on science rather than technical details.
- An Earth system modeling community with cost-effective, shared infrastructure development and many new opportunities for scientific collaboration.
- Accelerated scientific discovery and improved predictive capability through the social and technical influence of ESMF.

Evolution

Phase 1: 2002-2005

NASA's Earth Science Technology Office ran a solicitation to develop an **Earth System Modeling Framework** (ESMF).

A multi-agency collaboration (NASA/NSF/DOE/NOAA) won the award. The core development team was located at NCAR.

A prototype ESMF software package (version 2r) demonstrated feasibility.

Phase 2: 2005-2010

New sponsors included Department of Defense and NOAA.

A multi-agency governance plan including the CRB was created:
http://www.earthsystemmodeling.org/management/paper_1004_projectplan.pdf

Many new applications and requirements were brought into the project, motivating a complete redesign of framework data structures (version 3r).

Phase 3: 2010-2015 (and beyond)

The core development team moved to NOAA/CIRES for closer alignment with federal models.

Basic framework development completed with version 5r (ports, bugs, feature requests, user support etc. still require resources).

The focus is on increasing adoption and creating a community of interoperable codes.

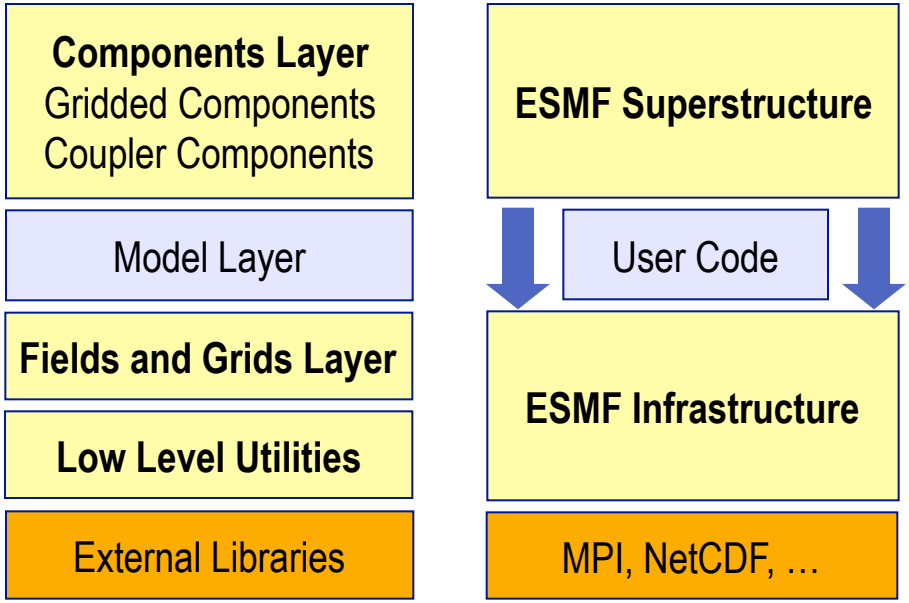
Computational Context

- Teams of specialists, often at different sites, contribute scientific or computational **components** to an overall modeling system
- Components may be at multiple levels: individual physical processes (e.g. atmospheric chemistry), physical realms (e.g. atmosphere, ocean), and members of same or multi-model ensembles (e.g. "MIP" experiments)
- Components contributed from multiple teams **must be coupled** together, often requiring transformations of data in the process(e.g. grid remapping and interpolation, merging, redistribution)
- Transformations are most frequently 2D data, but 3D is becoming more common
- There is an increasing need for cross-disciplinary and inter-framework coupling for climate impacts
- Running on **tens of thousands of processors** is fairly routine; utilizing hundreds of thousands of processors or GPUs is less common
- Modelers will tolerate virtually **no framework overhead** and seek **fault tolerance and bit reproducibility**
- **Provenance collection** is increasingly important for climate simulations

Architecture

- The Earth System Modeling Framework (ESMF) provides a component architecture or **superstructure** for assembling geophysical components into applications.
- ESMF provides an **infrastructure** that modelers use to
  - Generate and apply interpolation weights
  - Handle metadata, time management, I/O and communications, and other common functions

The ESMF distribution does not include scientific models



Summary of Features

- Components with multiple coupling and execution modes for flexibility, including a web service execution mode
- Fast parallel remapping with many features
- Core methods are scalable to tens of thousands of processors
- Supports hybrid (threaded/distributed) programming for optimal performance on many computer architectures; works with codes that use OpenMP and OpenACC
- Time management utility with many calendars, forward/reverse time operations, alarms, and other features
- Metadata utility that enables comprehensive, standard metadata to be written out in standard formats
- Runs on 30+ platform/compiler combinations, exhaustive nightly regression test suite (4500+ tests) and documentation
- Couples Fortran or C-based model components
- Open source license

Major Users

ESMF Components:

- NOAA National Weather Service operational weather models (GFS, Global Ensemble, NEMS)
- NASA atmospheric general circulation model GEOS-5
- Navy and related atmospheric, ocean and coastal research and operational models – COAMPS, NOGAPS, HYCOM, WaveWatch, others
- Hydrological modelers at Delft hydraulics, space weather modelers at NCAR and NOAA

ESMF Regridding and Python Libraries

- NCAR/DOE Community Earth System Model (CESM)
- Analysis and visualization packages: NCAR Command Language, Ultrascale Visualization - Climate Data Analysis Tools (UV-CDAT), PyFerret users
- Community Surface Dynamics Modeling System

Usage Metrics

- Updated ESMF component listing at: http://www.earthsystemmodeling.org/components/
- Includes 85 components with ESMF interfaces, 12 coupled cross-agency modeling systems in space weather, climate, weather, hydrology, and coastal prediction, for operational and research use
- About 4500 registered downloads

Values and Principles

- Community driven development and community ownership
- Openness of project processes, management, code and information
- Correctness
- Commitment to a globally distributed and diverse development and customer base
- Simplicity
- Efficiency
- User engagement
- Environmental stewardship

Web link for detail: http://www.esmf.ucar.edu/about_us/values.shtml

Making Distributed Co-Development Work

Hinges on asynchronous, all-to-all communication patterns: everybody must have information

- Archived email list where all development correspondence gets cc'd
- Minutes for all telecons
- Web browsable repositories (main and contributions), mail summary on check-ins
- Daily, publicly archived test results
- Monthly archived metrics
- Public archived trackers (bugs, feature requests, support requests, etc.)

Discouraged: IMing, one-to-one correspondence or calls – the medium matters

Change Review Board

- CRB established as a vehicle for **shared ownership** through user task prioritization and release content decisions
- Consists of technical leads from key user communities
- Not led by the development team!
- Sets the schedule and expectations for future functionality enhancements in ESMF internal and public distributions
  - Based on broad user community and stakeholder input
  - Constrained by available developer resources
  - Updated quarterly to reflect current realities
- CRB reviews releases after delivery for adherence to release plan

Governance Highlights

Management of ESMF required governance that recognized social and cultural factors as well as technical factors

Main practical objectives of governance:

- Enabling stakeholders to fight and criticize in a civilized, contained, constructive way
- Enabling people to make priority decisions based on resource realities

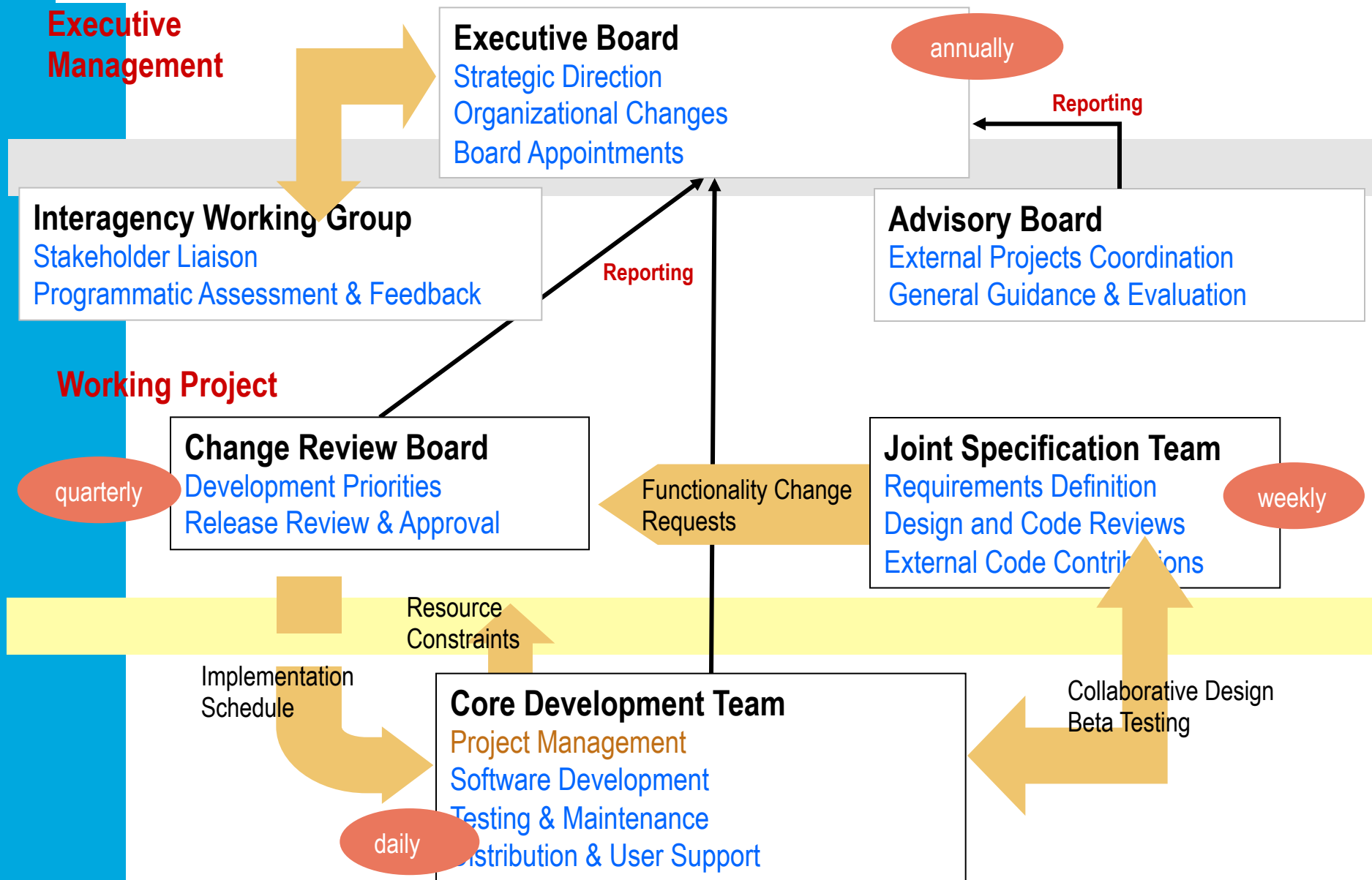
Observations:

- Sometimes just getting everyone equally dissatisfied and ready to move on is a victory
- Thorough, informed criticism is the most useful input a project can get
- Governance changes and evolves over the life span of a project

Governance Functions

- **Prioritize development tasks** in a manner acceptable to major stakeholders and the broader community, and define development schedules based on realistic assessments of resource constraints (CRB)
- **Deliver a product** that meets the needs of critical applications, including adequate and correct functionality, satisfactory performance and memory use, ... (Core)
- **Support users** via prompt responses to questions, training classes, minimal code changes for adoption, thorough documentation, ... (Core)
- Encourage **community participation in design and implementation decisions** frequently throughout the development cycle (JST)
- **Leverage contributions** of software from the community when possible (JST)
- Create frank and constructive **mechanisms for feedback** (Adv. Board)
- Enable stakeholders to **modify the organizational structure** as required (Exec. Board)
- **Coordinate and communicate at many levels** in order to create a knowledgeable and supportive network that includes developers, technical management, institutional management, and program management (IAWG and other bodies)

Governance


Executive Management
Executive Board
Strategic Direction
Organizational Changes
Board Appointments
annually
Reporting
Interagency Working Group
Stakeholder Liaison
Programmatic Assessment & Feedback
Reporting
Advisory Board
External Projects Coordination
General Guidance & Evaluation
Working Project
Change Review Board
Development Priorities
Release Review & Approval
quarterly
Functionality Change Requests
Joint Specification Team
Requirements Definition
Design and Code Reviews
External Code Contri ions
weekly
Resource Constraints
Implementation Schedule
Core Development Team
Project Management
Software Development
Testing & Maintenance
istribution & User Support
daily
Collaborative Design
Beta Testing

Example : FLASH Developed and Distributed by One Institution

- Under sustained funding from the ASC alliance program
- One of the expected outcomes was a public code
  - Use the same code for many different applications
    - All target applications were for reactive flows
- Diverging camps from the beginning
  - Camp 1: Produce a well architected modular code
  - Camp 2: Yes, but also use it soon for science
- Both goals hard to meet in the near term
- Two parallel development paths started
  - Not enough resources to sustain both
  - Camp 2 won out
- First release FLASH1.6 – three iterations of refactoring

Version 1

- Smashed together from three distinct existing codes
  - PARAMESH for AMR
  - Prometheus for Hydro
  - EOS and nuclear burn from other research codes
- F77 style of programming; Common blocks for data sharing
- Inconsistent data structures, divergent coding practices and no coding standards
- Concept of alternative implementations brought in with a script for plugging different EOS
- Beginning of inheriting directory structure

Version 2 : Data Inventory

- Centralized database
  - Common blocks eliminated
  - All data inventoried
  - Different types of variables identified
- Testing got formalized
  - Test-suite version 1
  - Run on multiple platforms
  - Policies about monitoring
- Not much else changed in the architecture

Version 3 : the Current Architecture

- Kept inheriting directory structure, inheritance and customization mechanisms from earlier versions
- Defined naming conventions
  - Differentiate between namespace and organizational directories
  - Differentiate between API and non-API functions in a unit
  - Prefixes indicating the source and scope of data items
- Formalized the unit architecture
  - Defined API for each unit with null implementation at the top level
- Resolved data ownership and scope
- Resolved lateral dependencies for encapsulation
- Introduced subunits and built-in unit test framework

Transition to Version 2

- The bias at the time – keep the scientists in control
- Keep the development and production branches synchronized
  - Enforced backward compatibility in the interfaces
  - Precluded needed deep changes
  - Hugely increased developer effort
  - High barrier to entry for a new developer
- Did not get adopted for production in the center for more than two years
  - Development continued in FLASH1.6, and so had to be brought simultaneously into FLASH2 too.
  - Database caused performance hit and IPA could not be done, so slower

Transition to Version 3

- Controlled by the developers
- Sufficient time and resources made available to design and prototype
- No attempt at backward compatibility
- No attempt to keep development synchronized with production
- All focus on a forward looking modular, extensible and maintainable code

Two very important factors to remember:
The scientists had a robust enough production code
The developers had internalized the vagaries of the solvers

The Methodology

- Build the framework in isolation from the production code base
- Infrastructure units first implemented with a homegrown Uniform Grid.
  - Helped define the API and data ownership
- Unit tests for infrastructure built before any physics was brought over
- Hydro and ideal gas EOS were next with Sod problem
- Next was PARAMESH: the Sod problem and the IO implementation were verified
- Test-suite was started on multiple platforms with various configurations (1/2/3D, UG/PARAMESH, HDF5/PnetCDF)
- This took about a year and a half, the framework was very well tested and robust by this time

The Methodology Continued …

- In the next stage the mature solvers (ones that were unlikely to have incremental changes) were transitioned to the code
  - Once a code unit became designated for FLASH3, no users could make a change to that unit in FLASH2 without consulting the code group.
- The next transition was the simplest production application (with minimal amount of physics)
- Scientists were in the loop for verification and in prioritizing the units to be transitioned
- FLASH3 was in production in the Center long before its official 3.0 release
  - More trust between developers and scientists
  - More reliable code; unit tests provided more confidence, and it was easier to add capabilities

Interdisciplinary Interactions

Prioritization

- whether good long term design or meet short term science objectives
- Both have their place
- Initial stages driven by science objectives
  - Too early for long term software design
  - Quick and dirty solutions with an eye to learning about code components and their interplay
- Once there is useable code, long term planning and design should occur
  - Willingness to make wholesale changes to the code at least once in necessary
  - At no stage should one lose sight of science objectives

The Management and Governance Model

- Licensed and distributed from the Flash Center at the University of Chicago
  - License allows modification and customization but not redistribution
  - External contributions are welcome but they go out in the release tarball from the Flash Center
  - In a new model there are code add-ons that are available as-is
    - Not verified and/or guaranteed by the core team
- A dedicated code group of with many roles
  - Develop and maintain the code
  - Manage the various production and project branches
  - Support simulation campaigns at the center
- The code group is not a consumer of its own code
  - But supports the consumption

Community Building

- Took several years
- Started with collaborations with the Center scientists
- Alumni of the center took the culture and the code with them
  - Their students and post-docs adopted the code
- We started holding Tutorials on-site and at scientific conferences
  - Tutorials had hands-on sessions and help for user's specific problems
- Easy customizability built into the infrastructure helped
  - As did the included ready to run examples

The greatest impact in popularizing the code though was relative ease in getting started, quick turn-around for user's questions and hand holding provided through the mailing lists

Variety of User Expertise

- Novice users – execute one of included applications
  - change only the runtime parameters
- Most users – generate new problems, analyze
  - Generate new Simulations with initial conditions, parameters
  - Write alternate API routines for specialized output
- Advanced users – Customize existing routines
  - Add small amounts of new code where their application resides
- Expert – new research
  - Completely new algorithms and/or capabilities
  - Can contribute to core functionality

Distribution Policies

- The licensing agreement
- Distribution control
- What is included in the release
- How often to release

FLASH Example

- A custom licensing agreement
- Source code is included, can be modified, but cannot be redistributed
- More than 3/4 of the usable code base is distributed
- Once or twice a year full releases, patches in-between

Contribution Policies

- Balancing contributors and code distribution needs
  - Contributors want their code to become integrated with the code so it is maintained, but may not want it released immediately
    - Not exercised enough
    - 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 not included with the distribution, but work with the code

Contribution and Attribution Policies: FLASH Example

- Code accepted with the understanding that it will eventually be distributed
- Pre-negotiated period of time when the code exists in FLASH repo but is not released
- The contributor provides user support also for negotiated time (usually that doesn't stop)
- The contribution does need to include the makefile snippet and appropriate tests that can be included in the test suite
- At least one example setup for users and its appropriate documentation is needed if it is a new capability
- If it is an alternative implementation of a new capability then the documentation only for the code is sufficient
- All contributions are acknowledged in user's guide and release notes. The contributors can also provide publications to be cited if their code is used

Community Building

- Popularizing the code alone does not build a community
- Neither does customizability – different users want different capabilities
  - The Center's research priorities do not align much with a large fraction of the user community

So What Did it Take ?

- Enabling contributions from users and providing support for them
- Including policy provisions for balancing the IP protection with open source needs
- Relaxing the redistribution policies – groups of users can modify the code and share among themselves as long as they have the license
- Group licenses also became available

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

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

Building a community …

The technical and social aspects

The yt Project

Growing & Engaging a community of practice

Matthew Turk
Columbia University

Traditional View of Scientific Development

"Users"

"Developers"

Most Scientific Development

Community of Practice

"Devusers"

"Developers"

Inspection and verification
Tracking modifications
Sharing information
Doing new and interesting things

"Users"

Uncritical acceptance of code?

"Users"

"These are the people we give the code to that don't care how it works."

Challenges

Academic Reward Structure

de facto & de jure

Utilization of developed tools

Respect from community

Project involvements

Invitations and opportunities to speak

de facto & de jure

Funding

Publications

Citation count

Influence

Traditional astrophysics does not favor tool builders.

Chores

Documentation,

testing,

outreach,

infrastructure development.

Chores

Tasks not fully-aligned with reward structure present great motivational challenges.

Co-opetition

(Funding
Publications
Citation count
Influence)

The "citation economy" for community codes is broken, and this disproportionately impacts new and junior contributors.

The "citation economy" for community codes is broken, and this disproportionately impacts new and junior contributors.

(It's bad for scientists, but even worse for infrastructure.)

How developer community engagement, cohesion, excitement and energy is affected by funded improvements remains unclear.

Strategies

Design the community you want.

Design the community you want.

Diversity. Tone. Enthusiasm. Congeniality.

Design the community you want.

This is an investment.

Technical
& Social

Reduce barrier to entry
Test on every push
Review every changeset

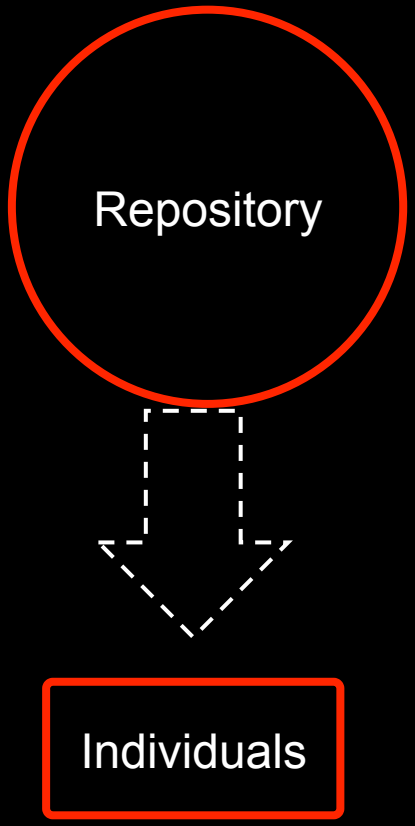
Reduce barrier to entry

Test on every push

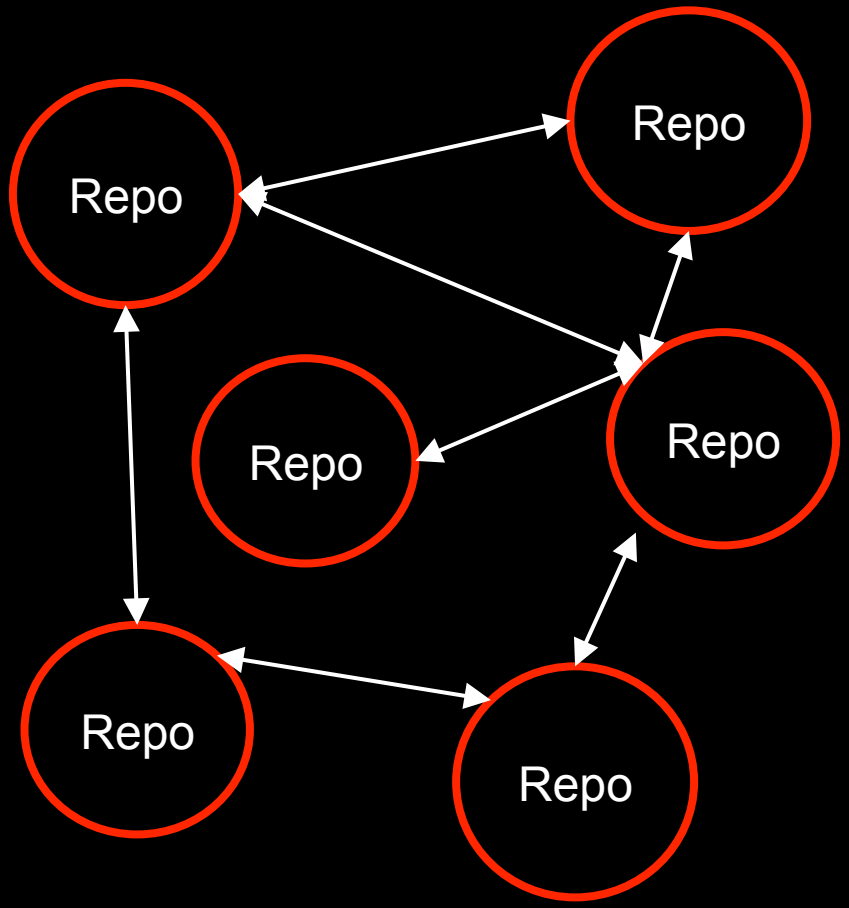
Review every changeset

Everything comes in the box: version control, extensions, sample data, dependencies, and tutorials.

CVCS



DVCS



Reduce barrier to entry

Test on every push

Review every changeset

Shining Panda for unit tests & small data answer tests, ReadTheDocs.org, and an auto-deployed ReST blog.

Reduce barrier to entry

Test on every push

Review every changeset

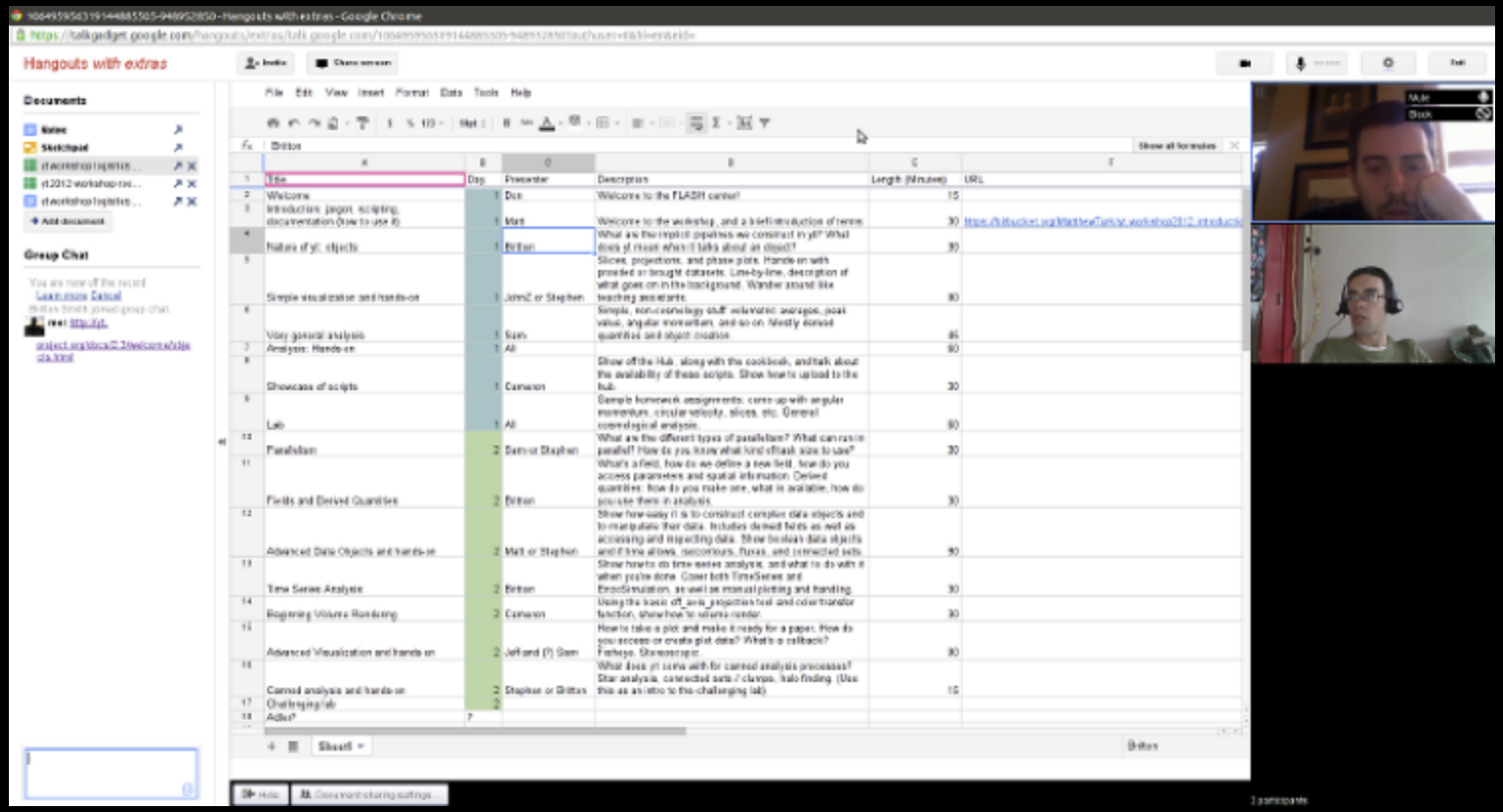
Pull requests and mentoring of new developers, through IRC, mailing list, and code comments.

Accept contributions of data, scripts, images, projects

Communication

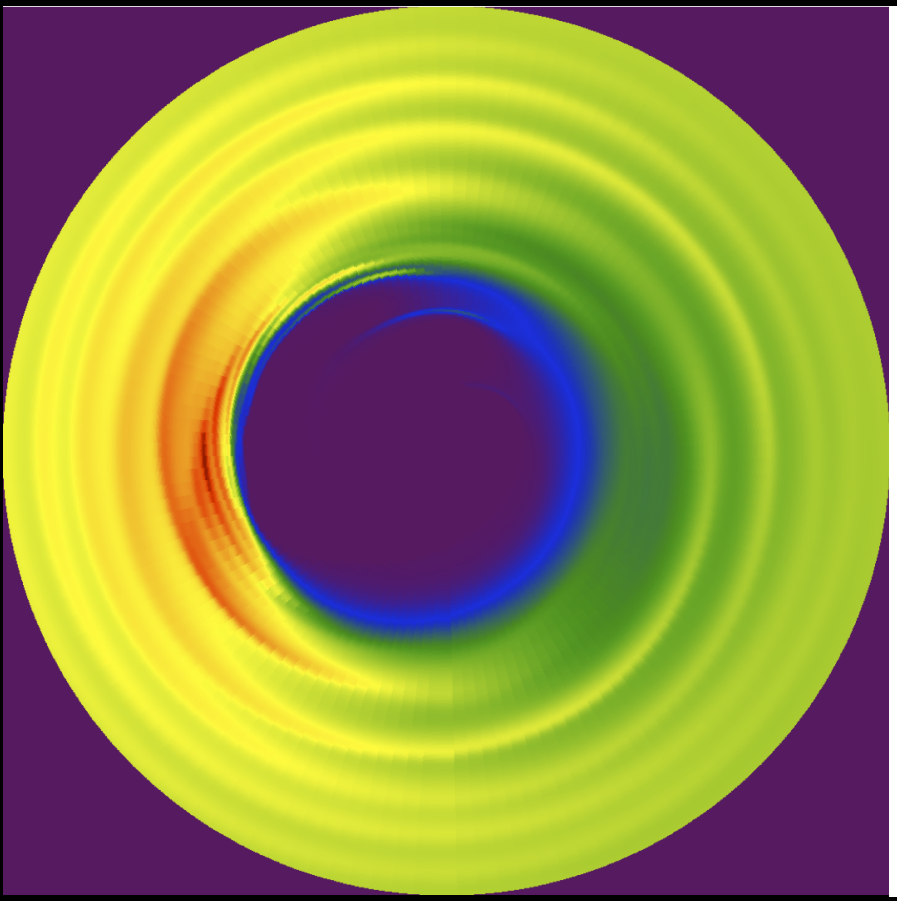
All project business should be conducted openly.

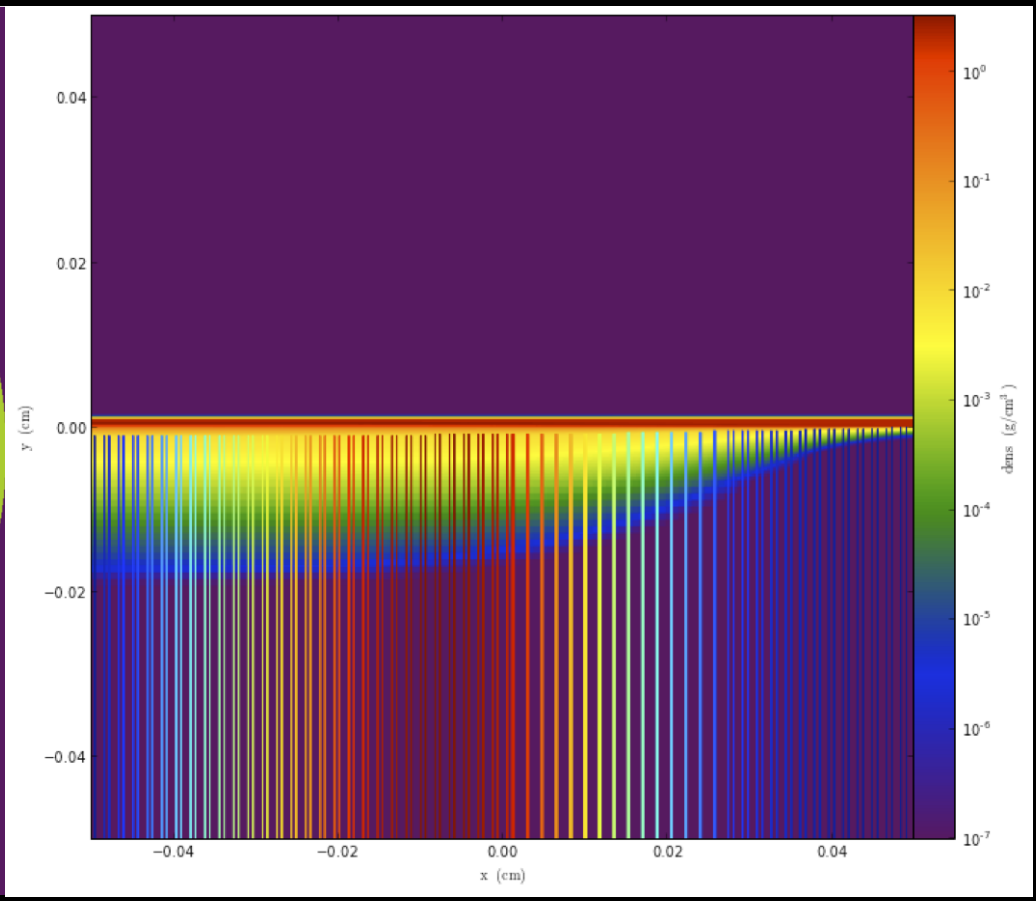
Immediate


Hangouts with extras
Documents
Group Chat
Title
Day
Presenter
Description
Length (Minutes)
URL
Welcome
Introduction: jargon, scripting, documentation (how to use it)
Nature of yt: objects
Simple visualization and hands-on
Very general analysis
Analysis: Hands-on
Showcase of scripts
Lab
Parallelism
Fields and Derived Quantities
Advanced Data Objects and hands-on
Time Series Analysis
Beginning Volume Rendering
Advanced Visualization and hands-on
Cannot analysis and hands-on
Challenging lab
Advice?

Immediate

Immediate




0.04
0.02
0.00
−0.02
−0.04
y (cm)
−0.04
−0.02
0.00
0.02
0.04
x (cm)
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
dens (g/cm^3)

Low-Latency

```
 yt: analysis and viz.  home: http://yt-project.org/ (and still not in any app stores!)
23:49:55 < CIA-62> yt: Nathan Goldbaum <goldbaum@ucolick.org> * 358092443a92 r5992
                   /yt/visualization/plot_modifications.py:
23:49:55 < CIA-62> yt: Fixing a bug in convert_to_pixel, which I've renamed to
23:50:12 < CIA-62> yt: convert_to_plot since it should convert to plot coordinates (not
23:50:12 < CIA-62> yt: necessarily the same as pixel coordinates).
23:50:12 < CIA-62> yt: Nathan Goldbaum <goldbaum@ucolick.org> * 5c7b2095ee5a r5993
                   /yt/visualization/plot_window.py: Need to cast this to a string
23:50:12 < CIA-62> yt: Matthew Turk <matthewturk@gmail.com> * 148b51ad39af r5994 /yt/ (3 files in 2 dirs):
                   Merged in ngoldbaum/yt-ngoldbaum (pull request #194)
23:50:19 < ngoldbaum> awesome, thanks matt
Day changed to 11 Jul 2012
00:22:00 < mjturk> np
00:22:03 < mjturk> thank you for the changes
00:22:52 < xarthisius> mjturk: is this a typo or there's some magic behind that I don't understand?
                       http://paste.lugons.org/show/2824/
00:24:00 < xarthisius> without that patch I get weird axis labels for non-square domains
00:27:06 < xarthisius> oh, ngoldbaum that ^^ should be directed to you :)
00:27:33 < ngoldbaum> it's a type
00:27:41 < ngoldbaum> thanks for testing on non-square domains
00:27:55 < ngoldbaum> if anything doesn't work it's a bug (and probably a typo)
00:28:06 < ngoldbaum> thanks xarthisius
 [08:00] [mjturk(+Zi)] [2:#yt(+cnt)]
```

High-Latency

August 2012 Archives by thread

- **Messages sorted by:** [subject] [author] [date]
- More info on this list...

Starting: *Wed Aug 1 06:33:13 PDT 2012*
Ending: *Fri Aug 31 12:30:40 PDT 2012*
Messages: 99

- [yt-users] error in light_ray periodic ray creation *Britton Smith*
- [yt-users] Quiver normalization *Massimo Gaspari*
  - [yt-users] Quiver normalization *Jean-Claude Passy*
  - [yt-users] Quiver normalization *Massimo Gaspari*
    - [yt-users] Quiver normalization *Jean-Claude Passy*
- [yt-users] yt 2.4 release announcement *Nathan Goldbaum*
- [yt-users] YT installation *Sherwood Richers*
  - [yt-users] YT installation *Matthew Turk*
    - [yt-users] YT installation *Sherwood Richers*
      - [yt-users] YT installation *j s oishi*
      - [yt-users] YT installation *j s oishi*
    - [yt-users] YT installation *Sherwood Richers*
      - [yt-users] YT installation *Nathan Goldbaum*
      - [yt-users] YT installation *Sherwood Richers*
      - [yt-users] YT installation *Matthew Turk*
- [yt-users] yt update error *Latif*
  - [yt-users] yt update error *Matthew Turk*
    - [yt-users] yt update error *Latif*
    - [yt-users] yt update error *Matthew Turk*

Technical
& Social

Culture self-propagates.
So, it must be seeded directly.

Foster a community of peers,
not a community of elites.

Humility

Respect

Trust

Humility

I think there might be a bug in ...

It's like that for a good reason. Don't touch it.

I think there might be a bug in ...

It behaves that way because ...

Respect

I've noticed something is acting strangely with ...

You're probably doing it wrong.

I've noticed something is acting strangely with ...

Can you tell us how you'd expect it to act?

Trust

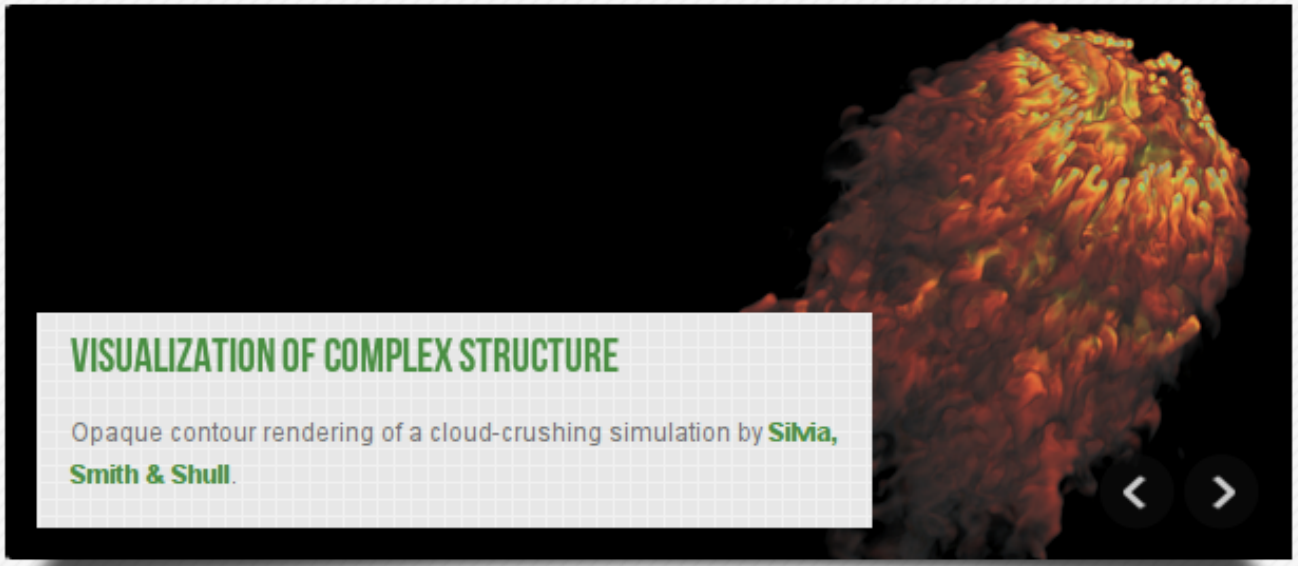

Letting
go...

By emphasizing pride over ownership, we've found projects can move between people without smothering through control.

Success

THE YT PROJECT

ASTROPHYSICAL SIMULATION ANALYSIS AND VIZ



VISUALIZATION OF COMPLEX STRUCTURE

Opaque contour rendering of a cloud-crushing simulation by **Silvia, Smith & Shull**.

DETAILED DATA ANALYSIS AND VISUALIZATIONS, WRITTEN BY WORKING ASTROPHYSICISTS AND DESIGNED FOR PRAGMATIC ANALYSIS NEEDS.

DATA-DRIVEN

Inspect your data

yt is designed to provide a consistent, cross-code interface to analyzing and visualizing

COMMUNITY

Participants welcome!

yt is composed of a friendly community of users and developers. We want to make it

FREE SOFTWARE

Open Source, Open Science

yt is developed completely in the open, released under the GPL license. The developers are

In a Nutshell, yt...

...has had 7,345 commits made by 42 contributors
representing 113,588 lines of code

...is mostly written in Python
with an average number of source code comments

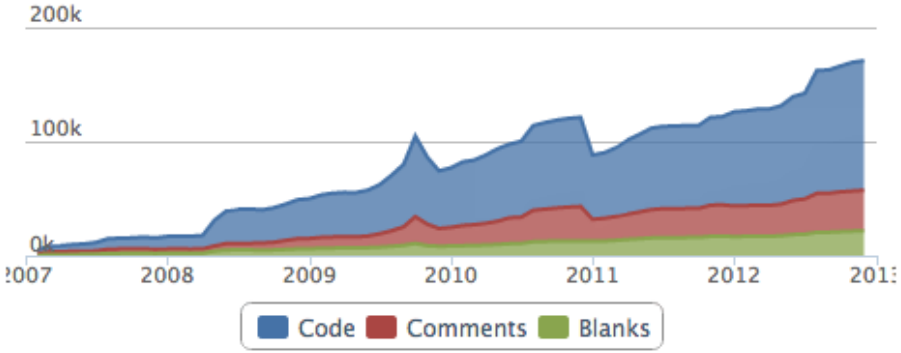
...has a well established, mature codebase
maintained by a large development team
with stable year-over-year commits

...took an estimated 29 years of effort (COCOMO model)
starting with its first commit in February, 2007
ending with its most recent commit about 17 hours ago

Languages

Python	77%	C	12%
JavaScript	5%	8 Other	6%

Lines of Code



Activity

30 Day Summary

Nov 14 2012 — Dec 14 2012

219 Commits
12 Contributors
including 1 new contributor
1 New Language :
TeX/LaTeX added Dec 13

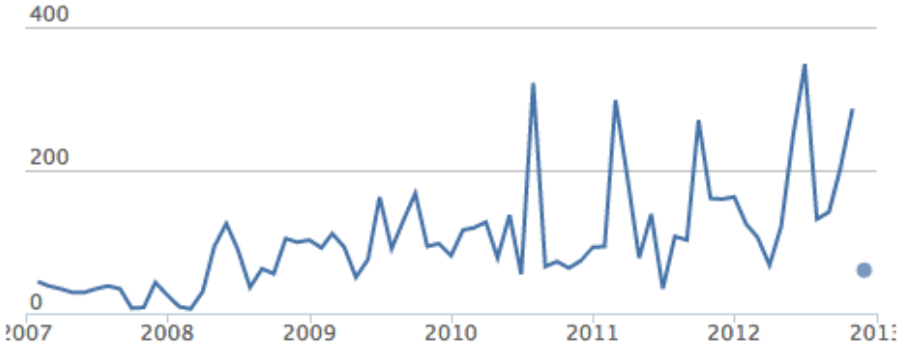
12 Month Summary

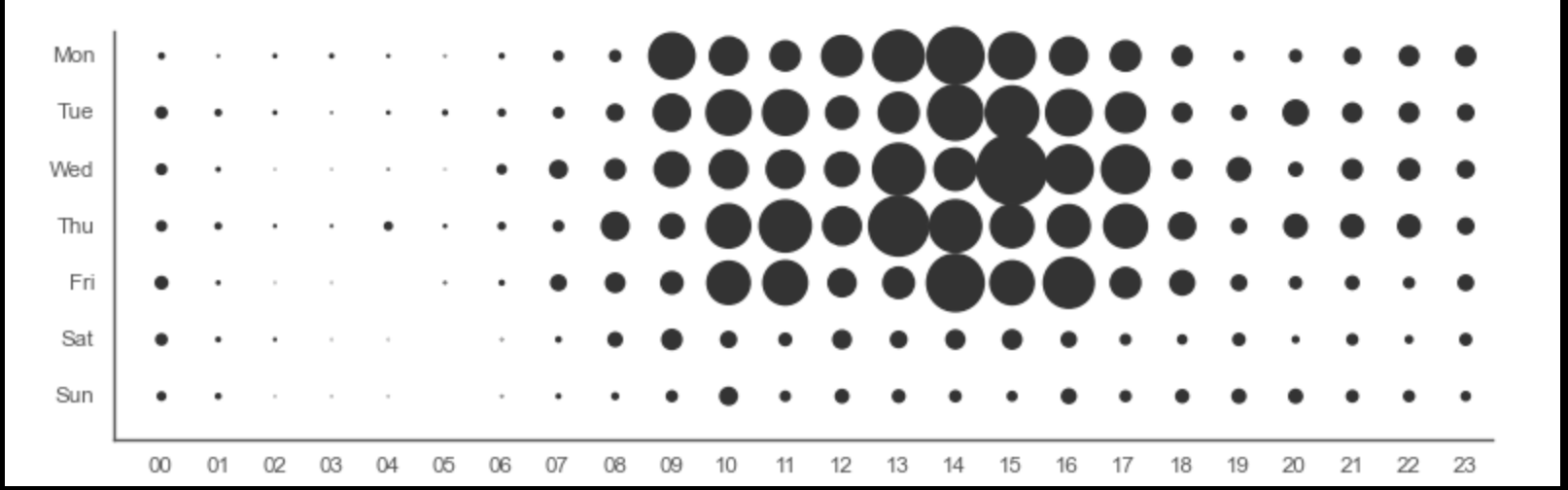
Dec 14 2011 — Dec 14 2012

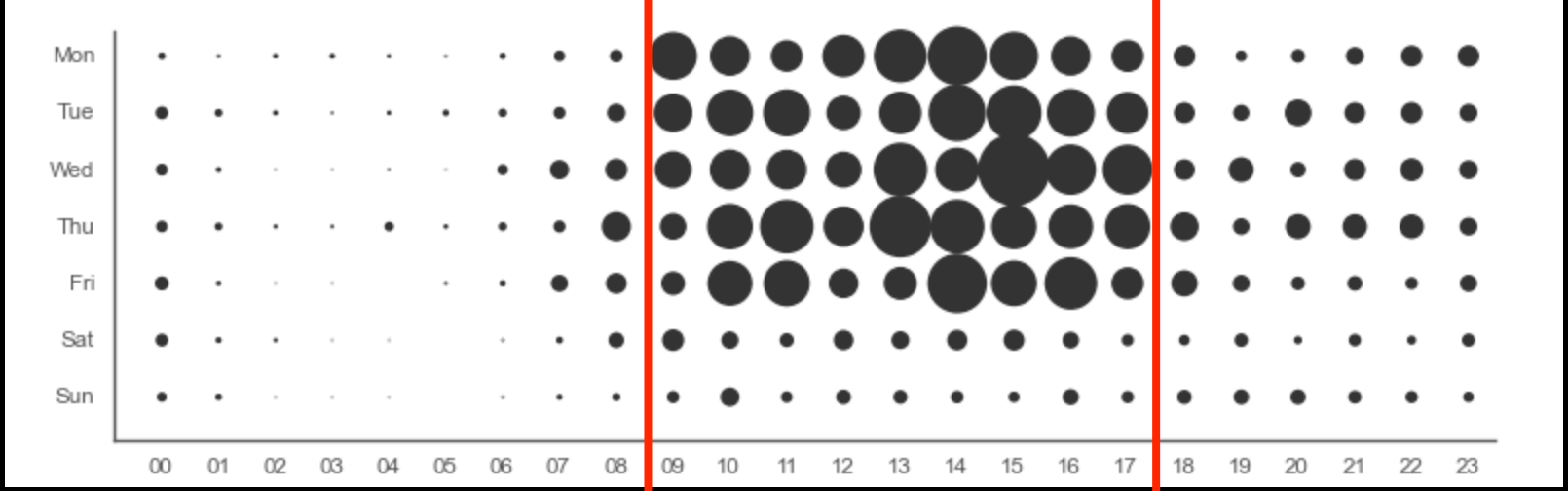
2057 Commits
Up +384 (22%) from previous 12 months

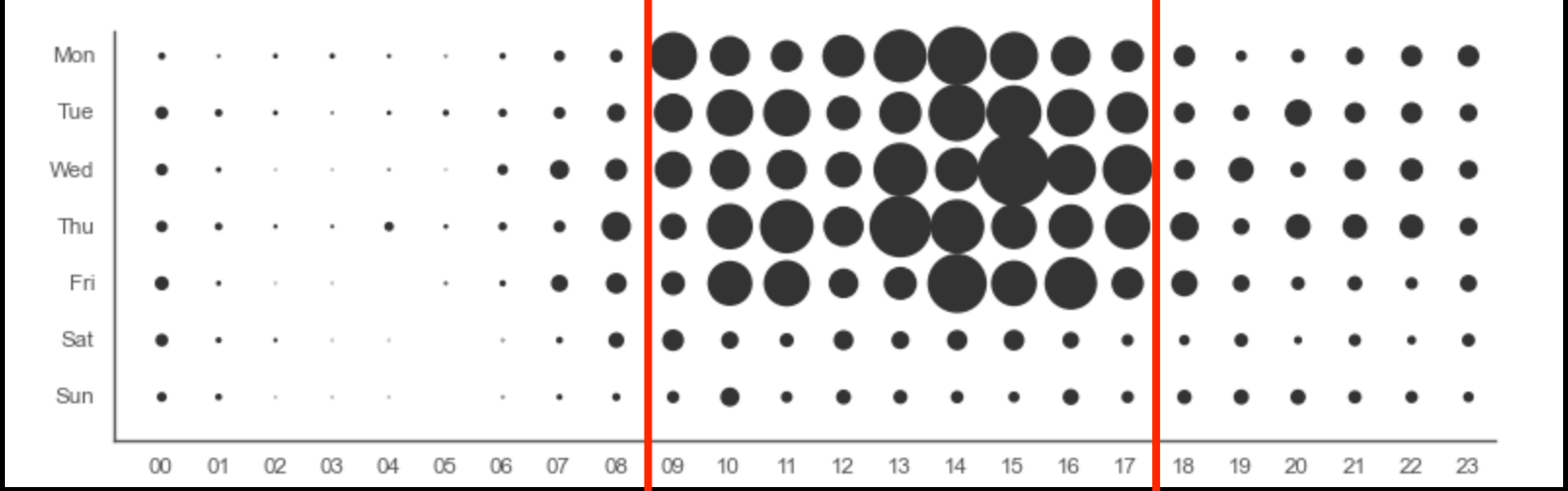
26 Contributors
Up +3 (13%) from previous 12 months

Commits per Month

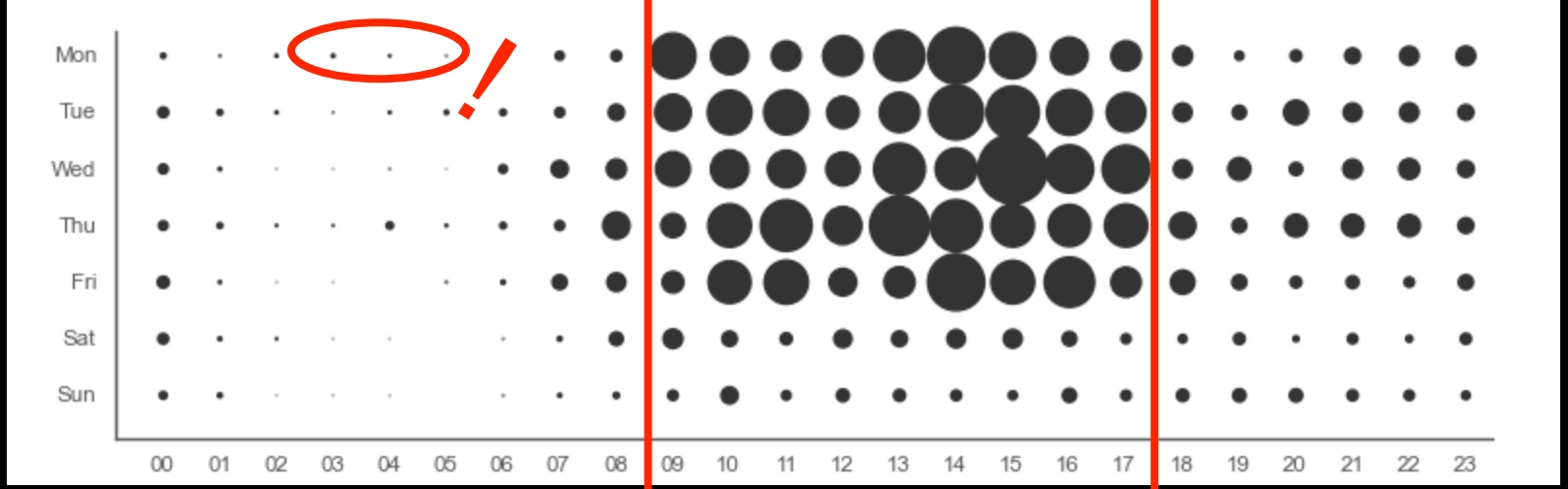



Mon
Tue
Wed
Thu
Fri
Sat
Sun
00
01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20
21
22
23

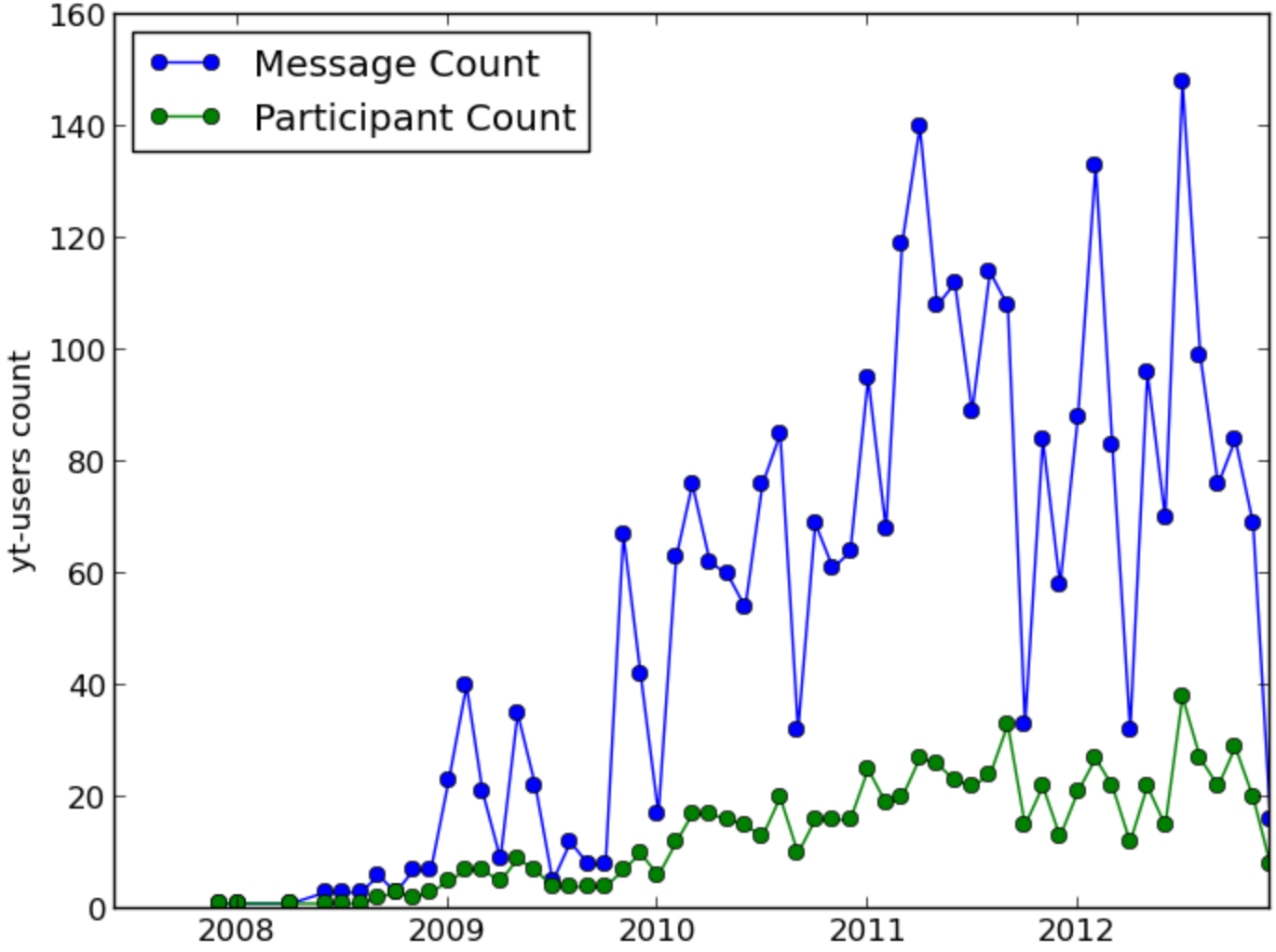

Mon
Tue
Wed
Thu
Fri
Sat
Sun
00
01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20
21
22
23

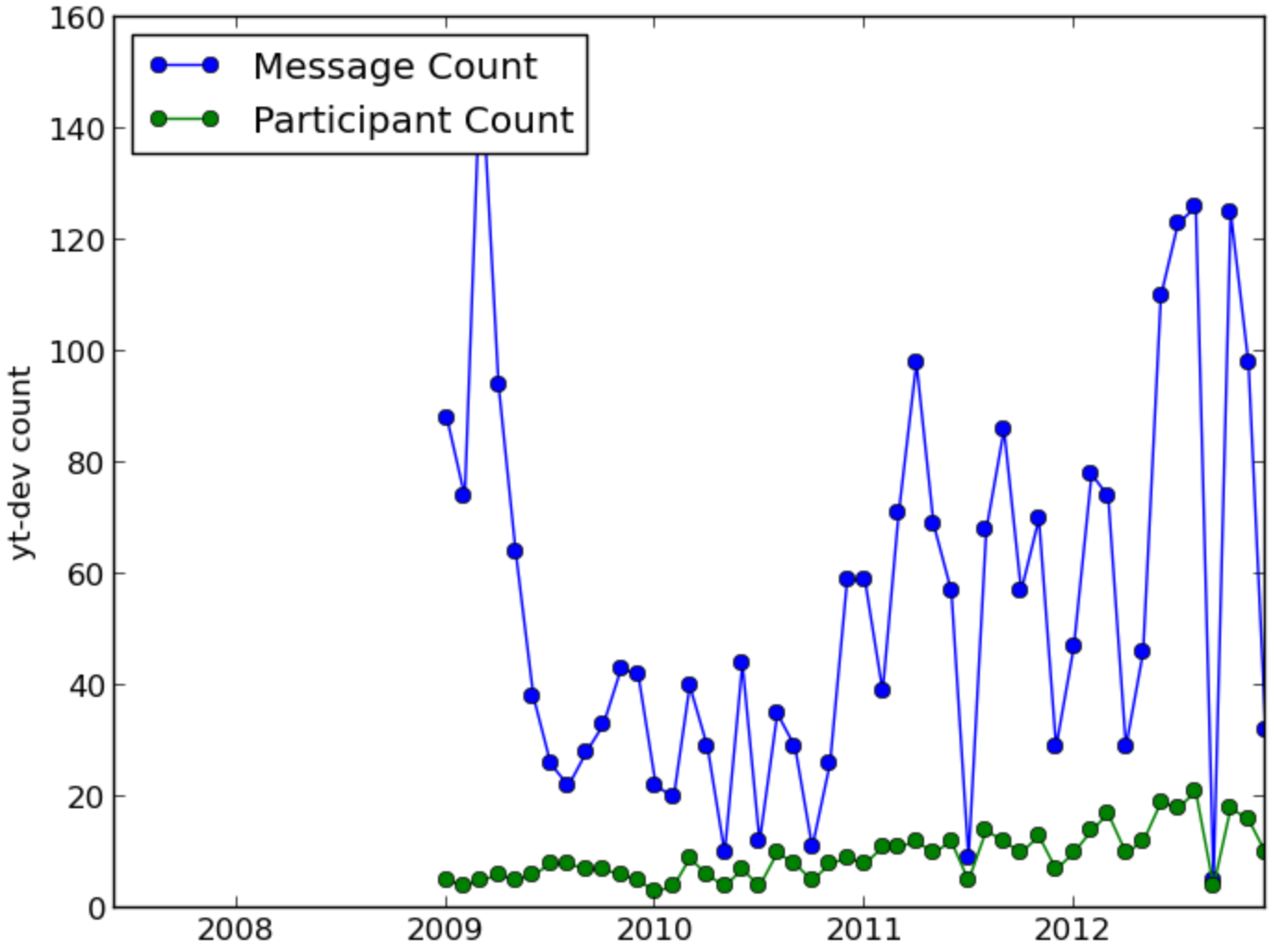


Developed by working astrophysicists.



Developed by working astrophysicists.


Message Count
Participant Count
yt-users count
0
20
40
60
80
100
120
140
160
2008
2009
2010
2011
2012


Message Count
Participant Count
yt-dev count
160
140
120
100
80
60
40
20
0
2008
2009
2010
2011
2012

	count	CPU hours	users	projects
matlab	24852	464244.83	14	8
cctm	3577	210446.84	1	2
cdo	1592	101810.32	1	1
gcn.v4	1588	391359.18	3	1
cam	1559	45894.90	11	5
yt	1231	121396.69	13	12
sigma	641	24622.21	3	2
grads	541	161510.58	3	1
mm5	400	459.18	4	2
cdcn	384	42798.52	6	4
grib	346	87726.47	6	2
milc	326	2377.34	3	2
grmhd	303	531167.83	2	3
ncl	259	2675.59	2	2
sscs	184	9948.52	1	1
paraview	177	34682.84	8	3
swift	169	2325.34	4	2
visit	101	4869.59	8	8
pop	100	209565.51	3	3
wrf	100	398.94	4	3
enzo	87	24014.62	4	3
tsc	74	105.90	1	1
R	70	24628.78	2	2
partadv	69	2062.75	1	1
a_out	56	158.65	4	4
hsi	48	2716.69	7	5
hmc	45	5.08	2	1
cactus	42	157.10	1	1
ior	42	79503.02	2	2
hy3s	39	53.02	4	3
idl	38	1971.62	6	6
music	29	7032.03	1	1

Usage on XSEDE Nautilus

Szczepanski et al, 2012

"... it seems likely that significant software contributions to existing scientific software projects are not likely to be rewarded through the traditional reputation economy of science. Together these factors provide a reason to expect the over-production of independent scientific software packages, and the underproduction of collaborative projects in which later academics build on the work of earlier ones."

Howison & Herbsleb (2011)

(Short) Bibliography

"The Art of Community" by Jono Bacon

"Producing Open Source Software" by Karl Fogel

"Team Geek" by Brian Fitzpatrick & Ben Collins-Sussman

"Organizing Simulation Code Collectives" by Mikaela Sundberg

"Scientific Software Production" by James Howison & James Herbsleb

"Your Community is your Best Feature" by Gina Trapani

"The Proof of the Pudding" by John Allsopp

"Standing Out in the Crowd" by Skud

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

For presentations about various community codes

Workshop to be held with SC13 on sustainable software for science

http://wssspe.researchcomputing.org.uk/