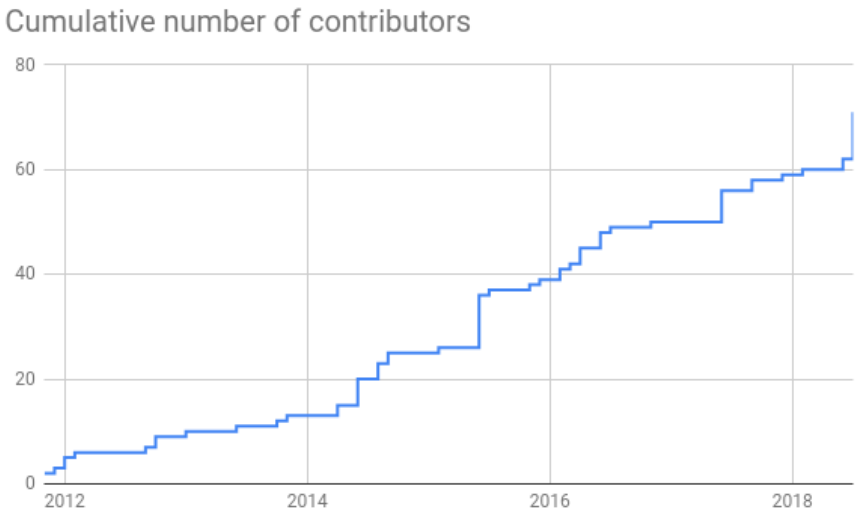
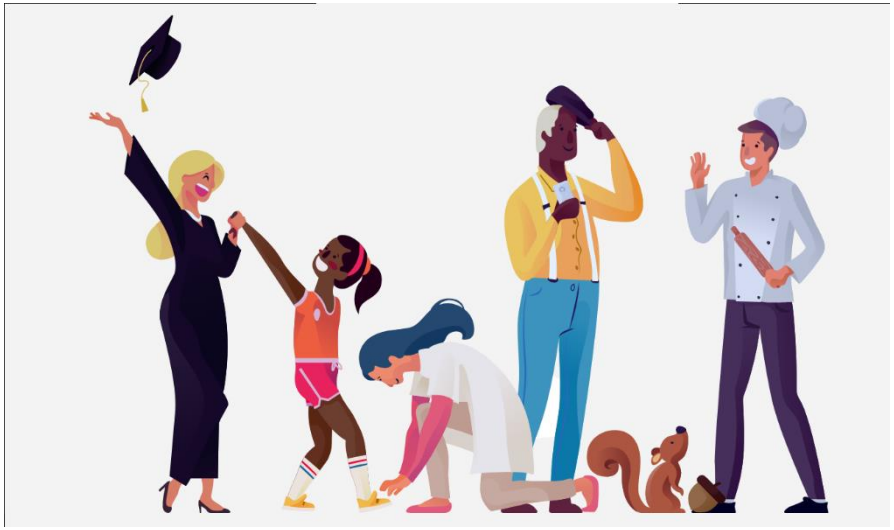
Better Scientific Software Communities

Rene Gassmoeller
University of California, Davis



Kellogg et al. (2018)



https://opensource.guide

better scientific software

IDEAS productivity

CIG COMPUTATIONAL INFRASTRUCTURE for GEODYNAMICS

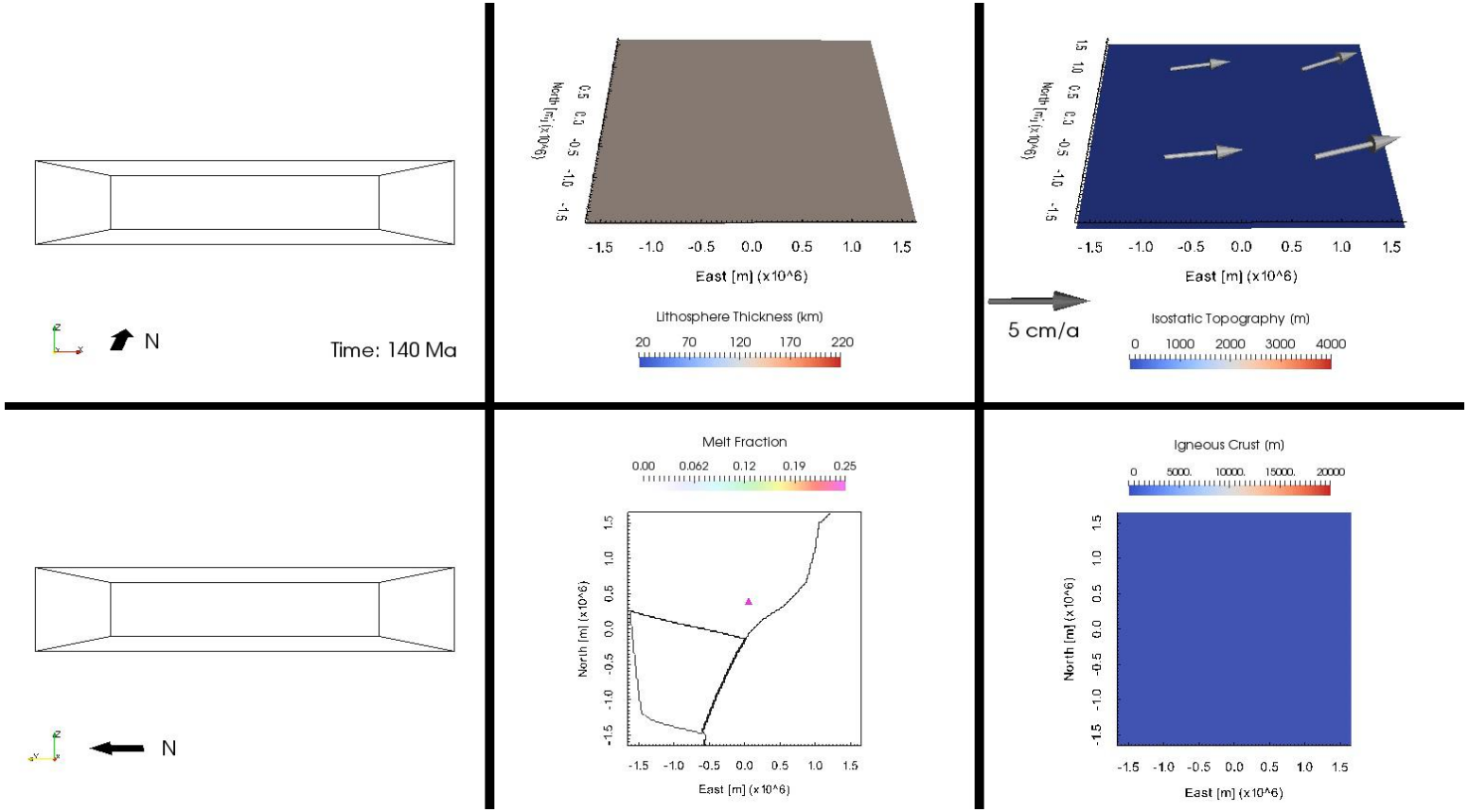
OUTLINE

Why talk about software communities?

What is BSSC?

MY BACKGROUND

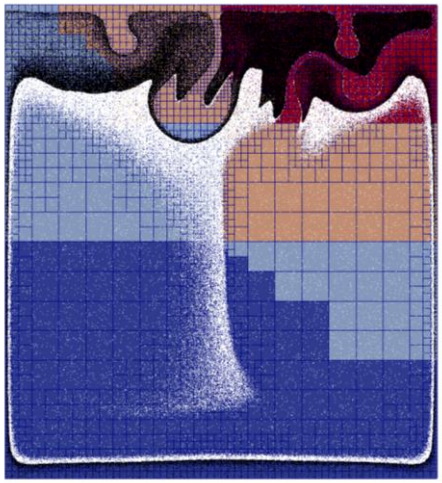
- I am a maintainer of ASPECT, a CFD solver for computational geodynamics

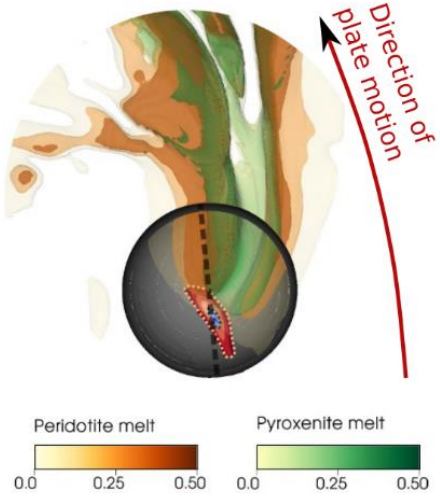


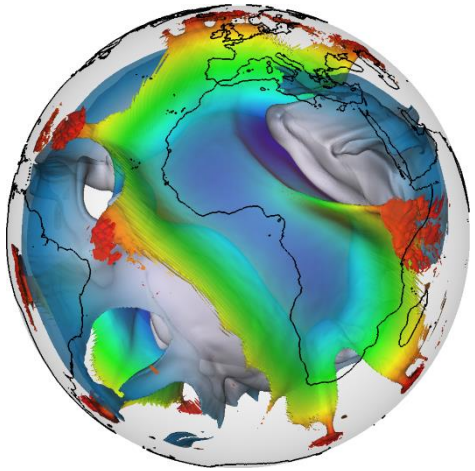
Gassmoeller et al, 2017

MY BACKGROUND

- Computational geodynamicist
- Transition from user to developer to maintainer of ASPECT
- Witnessed growth of the project (4 users ➡ >100 users)
- Now at Computational Infrastructure for Geodynamics (CIG), several scientific software projects (~5-10)
- 2019 BSSw Fellow of the IDEAS-ECP project
- My projects: https://github.com/gassmoeller







Gassmoeller et al, 2017, 2019
Dannberg & Gassmoeller, 2018

BEST PRACTICES FOR SUCCESSFUL SCIENTIFIC SOFTWARE

Plenty of tools for technical best practices:	Version Control (git, subversion, …)
	Code Review and Collaboration (github, gitlab, bitbucket)
	Testing (ctest, pytest, pyunit, testthat, …)
	Portability (cmake, autoconf, pip)
	Documentation (doxygen, sphinx, readthedocs)
	Reproducibility (docker, singularity, jupyter)
	Scalability (roofline)

BEST PRACTICES FOR SUCCESSFUL SCIENTIFIC SOFTWARE

- https://bssw.io/blog_posts
- https://ideas-productivity.org/events/hpc-best-practices-webinars/
- https://software-carpentry.org/lessons/
- https://geodynamics.org/cig/dev/best-practices/

- Wilson, G., et al. (2014). Best practices for scientific computing. *PLoS biology*, *12*(1).
- Heroux, M. A. & Willenbring, J. M. (2009). Barely sufficient software engineering: 10 practices to improve your CSE software. In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering* (pp. 15-21). IEEE Computer Society.
- Carver, J. C. (2012). Software engineering for computational science and engineering. *Computing in Science & Engineering*, *14*(2), 8.

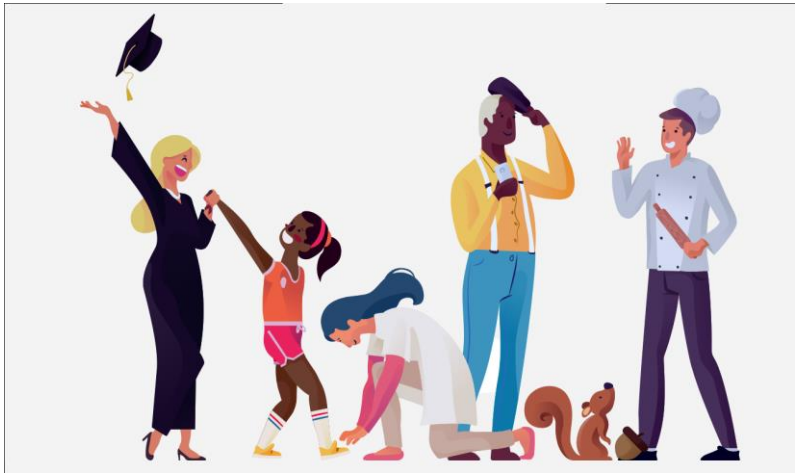
Lessons learned. What can possibly go wrong?

A lot of things!

As learned within ASPECT:

- A project is more than code and data, it is a community
- Communities are diverse and often unpredictable
- Maintainers must reevaluate and adjust policies and software architecture
- Also see recent BSSW blog by Wolfgang Bangerth:

https://bssw.io/blog_posts/leading-a-scientific-software-project-it-s-all-personal



https://opensource.guide

INTERLUDE: DEFINITIONS

User:
Uses software

Developer:
Changes software

Maintainer:
Cares for software

Community:
Everyone involved

Software:
Code, Tests, Doc

Software Project:
Software + Community

BREAK. QUESTIONS?

Lessons Learned: What Will Go Wrong?

1. Interactions between community and software
2. Tradeoffs between competing goals
3. Leadership and governance problems

Lessons Learned: What Will Go Wrong?

1. Interactions between community and software
   - Software architecture can support or hinder community growth
   - Community mood (competitive vs cooperative) influences size and quality of core architecture
   - Work on architecture and community is necessary, although not often acknowledged scientifically
2. Tradeoffs between competing goals
3. Leadership and Governance problems

Lessons Learned: What will go wrong?

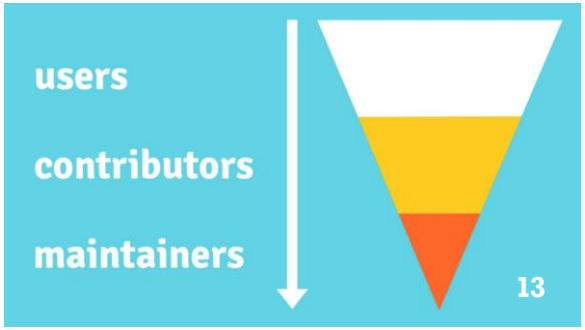
1. Interactions between community and software
2. Tradeoffs between competing goals
   - Good for one / bad for others (e.g. performance vs. flexibility)
   - Tradeoffs often resolveable using modern strategies (encapsulation, polymorphism, templates)
   - Community needs to align goals, e.g. at developer meetings
   - Maintainers might not be aware of user goals
3. Leadership and Governance problems

https://opensource.guide

LESSONS LEARNED: WHAT WILL GO WRONG?

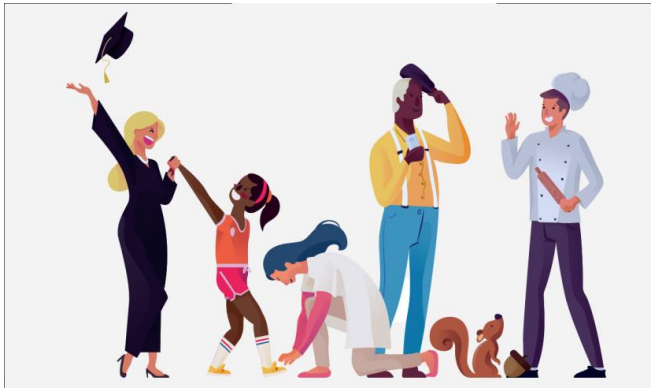
1. Interactions between community and software
2. Tradeoffs between competing goals
3. Leadership and Governance problems
   - Horizontal growth (user base) and vertical growth (user engagement) are necessary to prevent burnout of maintainers and maintain influx of new users
   - Design discussions and policy decisions must be communicated to a larger userbase
   - New users need to feel welcome in the community
   - Conflicts need to be managed, not ignored

https://opensource.guide



LESSONS LEARNED: WHAT WILL GO WRONG?

All these challenges are COMMUNITY challenges (either the interaction of community and software, or the interaction of community with community)



https://opensource.guide

← COMMUNITY

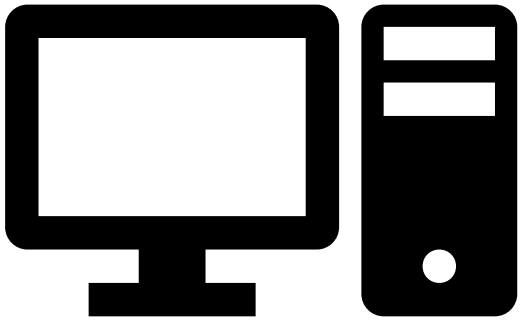
So why is COMMUNITY MANAGEMENT not in the list of best practices?

How to effectively manage software communities?

- For general open-source software widely recognized:
  - https://opensource.guide
  - Karl Fogel, "Producing OSS"
  - Richard Millington, "Buzzing Communities"
  - Fitzpatrick & Collins-Sussman "Debugging Teams"
  - Many blogs of OS maintainers

https://opensource.guide

- For scientific software often not acknowledged:
  - Concept of technical superiority
  - Problems of attribution leads to 'hero' codes
  - Scientists as community managers?



WHY COMMUNITY MANAGEMENT?

- The size of the community limits the activity of a software project
- The size of the community is bounded by:
  - Interest in software
  - Ease of access
  - Community support
  - Community atmosphere
- But: A large community creates work. More users mean:
  - More questions
  - More feature requests
  - More bugs discovered
  - More conflicts
- Efficiently managing a community creates success and saves time!

https://opensource.guide

BSSC - BETTER SCIENTIFIC SOFTWARE COMMUNITIES

Collect knowledge from successful scientific software projects

Distribute that knowledge as guides

Prepare new maintainers, help experienced maintainers

Form a community of practice

A software project consists of a collection of source-code and a community

Next: A tour through the guides that are currently under construction

BREAK. QUESTIONS?

1. Starting a project - the new maintainer

Define

Define your software's mission:

- E.g. ASPECT's mission: To provide the geosciences with a well-documented and extensible code base for their research needs.

Know

Know your audience:

- Aimed at application scientists? Developers? Which subdisciplines? Which career stage?

Find

Find capable and committed early users:

- Committed early users become maintainers later
- All but one of ASPECT's current principal developers were at the first user meeting in 2014

Work

Work towards their success (it's your own)

2. SPREADING OSS AMONG SCIENTISTS

Open work is important:

less bugs

more participation

more citations

increasingly required

see IDEAS webinar 24: "Software Licensing" by David E. Bernholdt

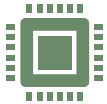
Common objections :

intellectual property

scientific reputation

scientific productivity

see IDEAS webinar 21: "Software Sustainability" by Neil Chue Hong



Modular architecture:

allows withholding for a certain time

allows easy merge later

allows multiple versions of algorithms

Finding the right pressure:

address fears, support courage

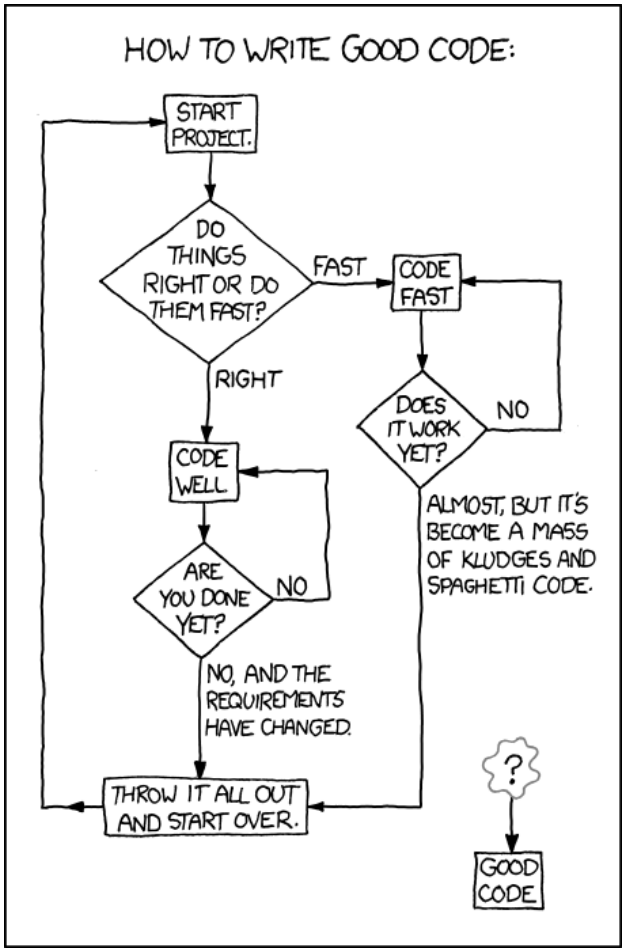
show rewards

be persistent

3. Balancing Structural and Scientific Work

Challenges:

- A good software architecture is critical
- But developing and maintaining a good architecture takes a lot of time
- Community expects growth
- Time pressure tempts to fudge things



https://xkcd.com/844/

https://opensource.guide

3. Balancing Structural and Scientific Work

Challenges:

- A good software architecture is critical
- But developing and maintaining a good architecture takes a lot of time
- Community expects growth
- Time pressure tempts to fudge things

Possible strategies:

- Combine structural and scientific work
- Delegate whenever possible
- Be as responsive and consistent as you can, **not more**

https://opensource.guide

3. BALANCING STRUCTURAL AND SCIENTIFIC WORK

As maintainers our responsibility is to provide a useful architecture, not to fulfil every wish from every user.

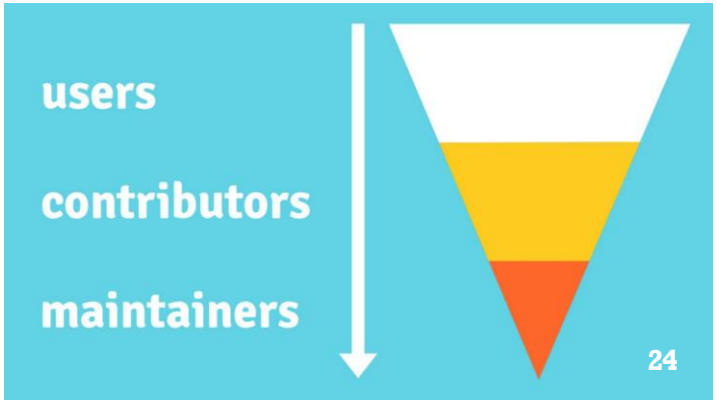
Time spent on building the proper architecture for a scientific study **is** useful time. It might provide unexpected windfalls.

Do not overengineer architecture! This is hard to define, but if there is no immediate and worthwhile application, do not build the infrastructure for it.

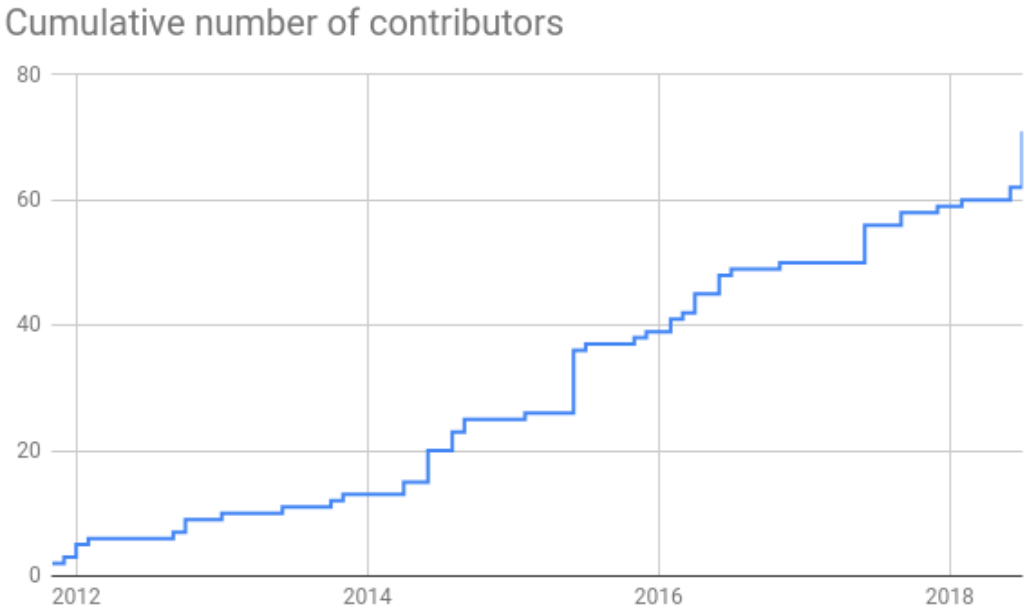
Know your limits. Burnout is a common threat for software maintainers. (https://opensource.guide/best-practices/#its-okay-to-hit-pause)

4. SOFTWARE QUALITY AND COMMUNITY CONTRIBUTIONS

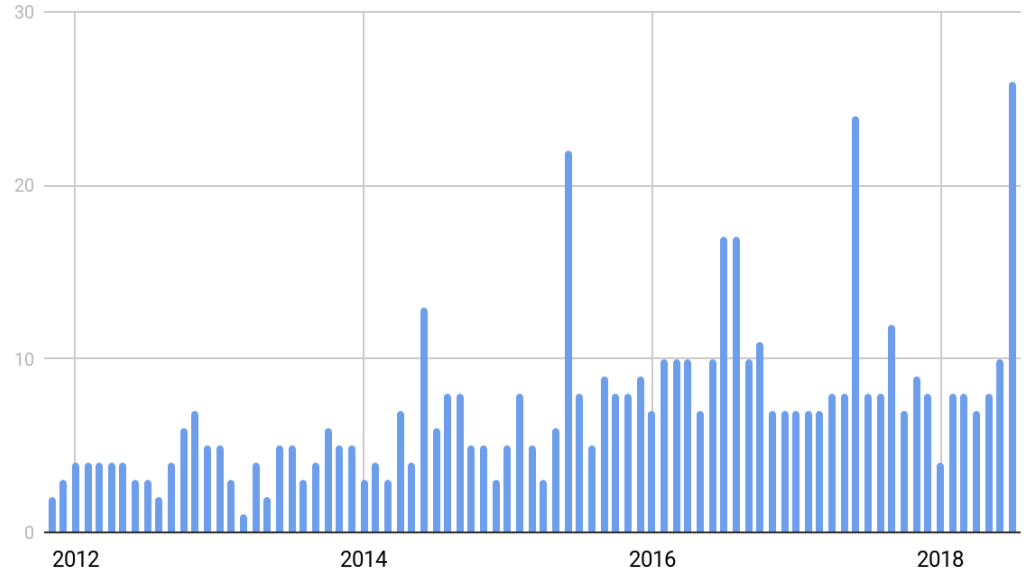
- The degree of community involvement is up to the project
- Members need the skills and tools to make contributions:
- Documentation and mentoring pays off in the long run
- Every contribution should be reviewed
  (See BSSw 2018 fellowship project by Jeff Carver)
- Guidance should be provided at
  the level of the contributor



The „contributor funnel“,
https://opensource.guide



Kellogg et al. (2018)



4. Developer Meetings as Tools to Build Community

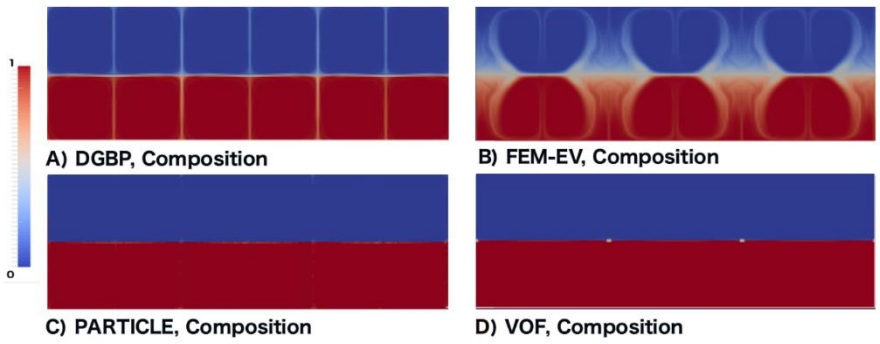
- ASPECT's yearly community meetings have grown the userbase and functionality (40% of yearly commits)
- Community meetings help onboard and mentor new members
- Novice users can learn and grow and must contribute
- Senior participants must advise, mentor, and review
- At least 20% of participants should be experienced developers
- A time of feature expansion, not fundamental rewrite

BREAK. QUESTIONS?

5. Mediating Feature Conflicts and Publications

Example 1: Competing techniques

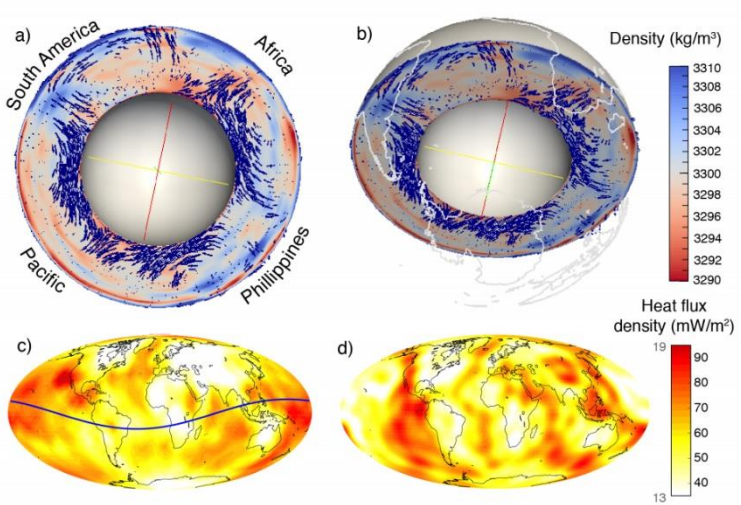
- Work on the same problem
- Provide different solutions
- No solution is universally superior
- A modular architecture allows both



Puckett et al., 2018

Example 2: Similar applications

- Work towards similar applications
- Need common technical solutions
- Implement method together, apply separately

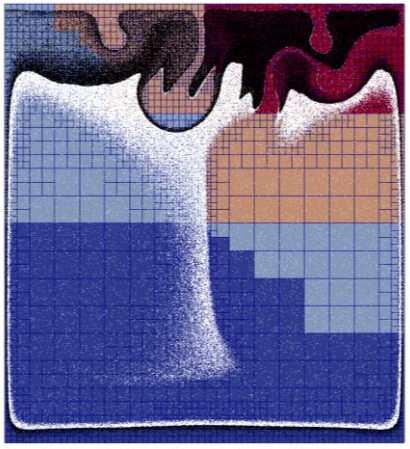


ASPECT manual, 2019

5. Mediating Feature Conflicts and Publications

Example 3: The same technique

- Work on the same technique
- Focus on different aspects of the technique
- E.g. structure and parallel scalability vs. mathematical accuracy and convergence



Gassmoeller et al., 2018

Takeaways:

- Feature conflicts are unavoidable
- Mitigating conflicts:
  - Early discovery and disclosure!
  - Efficient communication!
  - Diplomacy and Creativity! Find points for cooperation.
  - Share credit responsibly.

6. PROVIDING CREDIT

The question of credit

- Increasing importance of credit for software (see IDEAS webinar 17: "Software Citation Today and Tomorrow" by Daniel S Katz)
- More types of credit:
  - Perceived competence
  - Image as maintainer
  - Social network
  - Feel useful

Sharing credit

- Credit for developers is important motivation to contribute
- Use contributors/ developers/maintainers badges to reward members
- Still, a lot of motivation simply comes from working in a good team and needing the software (Lakhani & Wolf, 2003)

6. PROVIDING CREDIT

- Project specific, but needs to be publicly visible
- Possible ways:
  - Announce contributions as (automatic) newsletter:
    - see also https://github.com/gassmoeller/aspect_newsletter
  - Release announcements include all authors:
    - see https://aspect.geodynamics.org/doc/doxygen/changes_current.html
  - Release paper with main authors, e.g.:
    - Arndt, D., Bangerth, W., Clevenger, T. C., Davydov, D., Fehling, M., Garcia-Sanchez, D., ... & Kynch, R. M. (2019). The deal. II library, version 9.1. *Journal of Numerical Mathematics*.
  - Emphasize community members in talks
  - Hand over responsibility
    - Allow write-access to the repository for long-time developers

7. SUPPORTING GROWTH

- You need help, long-time contributors want credit … solve both by publicly assigning responsibility
- Allow gradual growth, let people prove their competence (e.g. answer questions, assign starter issues)
- Establish clear policies to show growth opportunities

Excerpt from ASPECT's CONTRIBUTING.md:

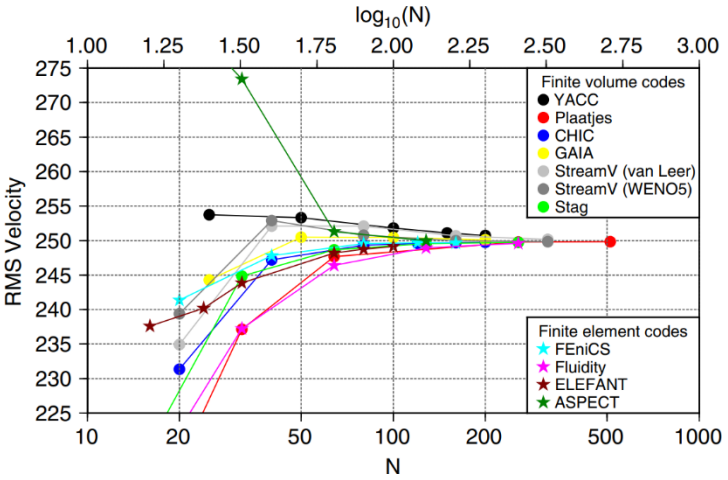
"Regularly, the Principal Developers of ASPECT come together and discuss [...] who should be invited to join the group of Principal Developers. Criteria that *Principal Developers* should match are:
- A profound understanding of ASPECT's structure and vision,
- A proven willingness to further the project's goals and help other users,
- Significant contributions to ASPECT (not necessarily only source code, also mailing list advice, documentation, benchmarks, tutorials),
- Regular and active contributions to ASPECT for more than one year, not restricted to user meetings.

The group of current Principal Developers is listed in the AUTHORS file in the main repository."

8. MANAGING RELATIONS WITH COMPETITORS

- Do not bash competing projects
- Avoid holy wars
- Compare facts not opinions
- Keep communications open (e.g. at conferences)
- Report problems, ask for advice
- Maybe offer help
- Having competitors is an advantage

https://opensource.guide



Tosi et al., 2015

WAYS TO HELP

Answer the survey and send me your response:
http://bit.ly/BSSC-community-survey

Suggestions for improvements:
https://github.com/gassmoeller/BSSC/issues

Open a pull request:
https://github.com/gassmoeller/BSSC/pulls

Send me your feedback:
rene.gassmoeller@mailbox.org

RESOURCES

- https://gassmoeller.github.io/BSSC/
- https://bssw.io/
- https://opensource.guide/
- https://producingoss.com/
- https://www.software.ac.uk/

- Bangerth, W., & Heister, T. (2013). What makes computational open source software libraries successful?. Computational Science & Discovery, 6(1), 015010.
- Smith, A. M., Katz, D. S., & Niemeyer, K. E. (2016). Software citation principles. PeerJ Computer Science, 2, e86.
- Kellogg, L. H., Hwang, L. J., Gassmöller, R., Bangerth, W., & Heister, T. (2018). The role of scientific communities in creating reusable software: Lessons from geophysics. *Computing in Science & Engineering*, *21*(2), 25-35.

TAKEAWAYS:

- **A software project is equally a collection of source-code and a social network.**
- **A successful software project needs a successful community.**
- **Effective community management can prevent or resolve many conflicts and help the project grow.**

- **Thanks for your attention!**