

The curse of growing scales: from inception to successful community-driven software development

Vijay Mahadevan Andrew Siegel

CSESSP Challenges, Oct 15-16 2015



Progressive scales in scientific software

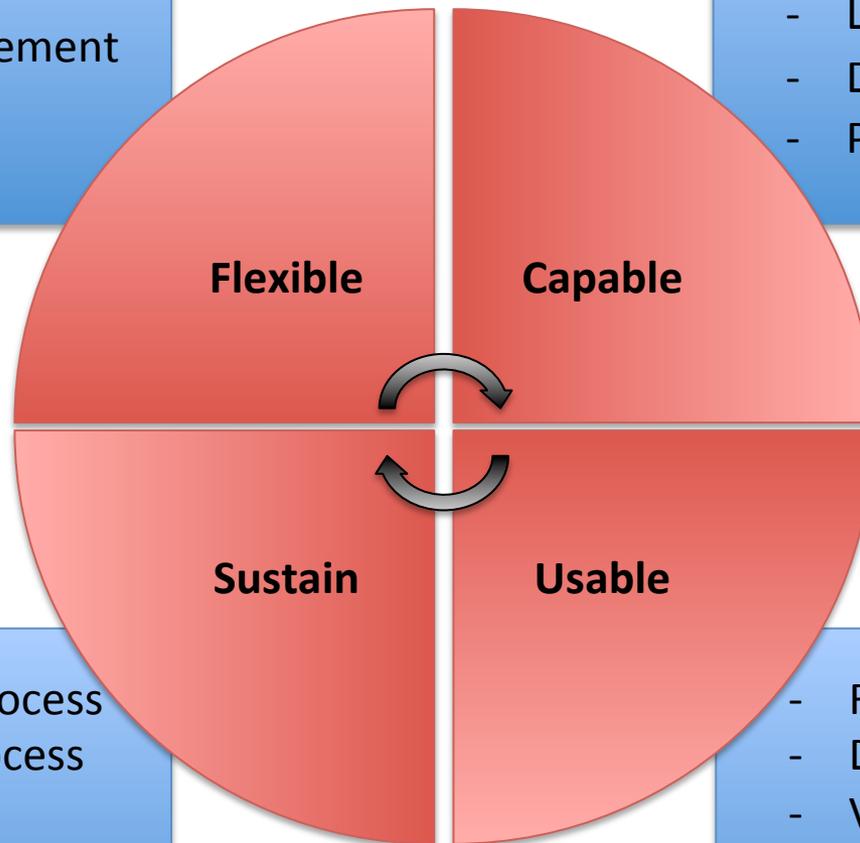
As physical resolution and discrete problem complexity increases, demands on scientific software escalate!

1. **Pilot scale** (*design and prototype of vision*)
 - rapid code changes and design abstractions
2. **Mini scale** (*capability and feature addition*)
 - increased range of applicability of library/application use-cases, SQA
3. **Macro scale** (*usability and scientific productivity*)
 - wide adoption, user support, scalable execution on large-scale machines
4. **Distributed scale** (*sustainability and extensibility*)
 - large teams, coherent source management and review, portable software, resource management

Sustaining software lifecycle

- Flexible abstractions
- Build management
- Configuration management
- Rapid TDD
- Static analysis

- High resolution capability
- Language interoperability
- Dependency management
- Profiling and optimization



Portability
Mailing-Lists

Documentation

CMake

Autotools

Tutorials

Examples

Profiling

valgrind

cppcheck

Jenkins Subversion

unit Buildbot

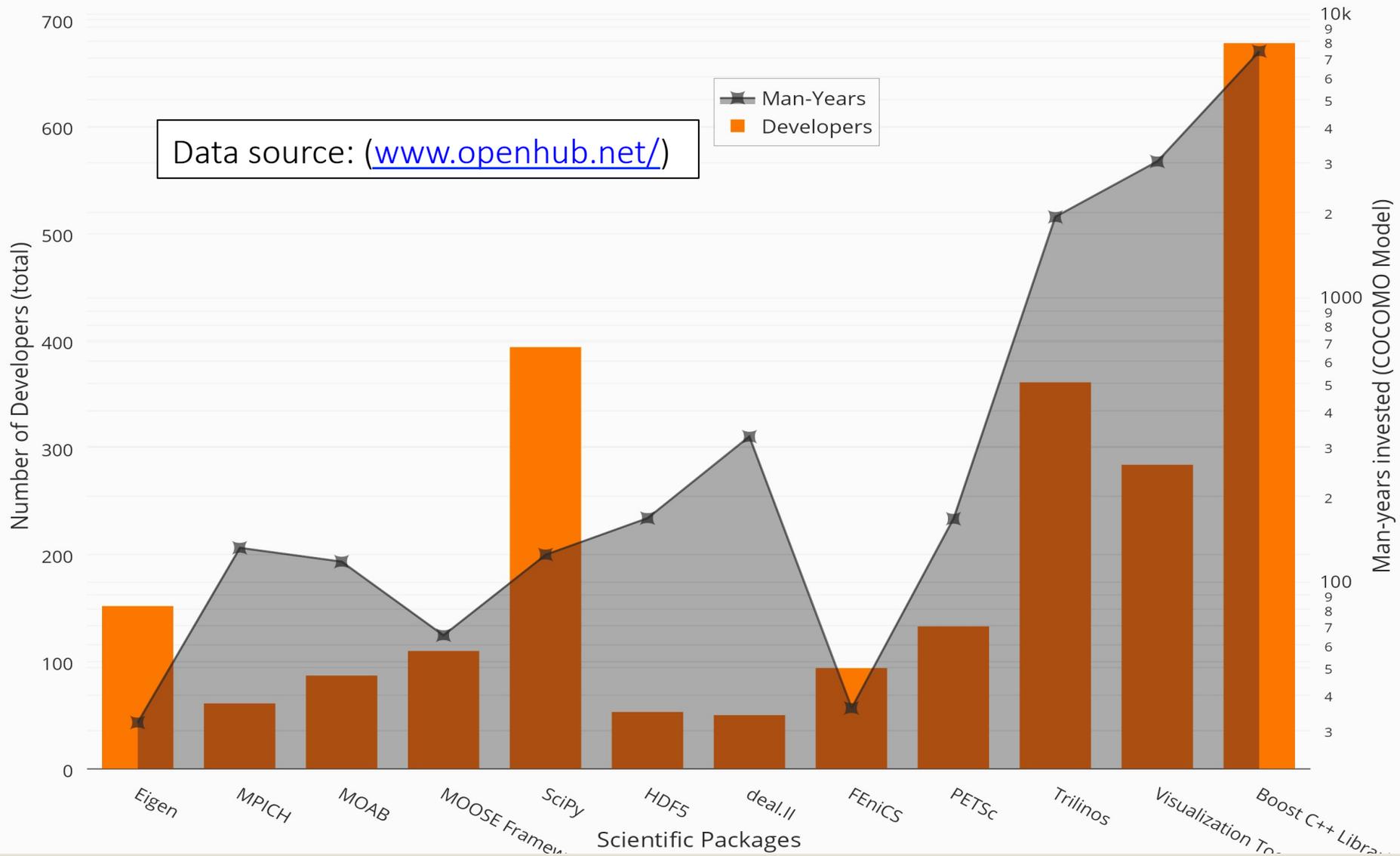
Git ROSE

Github integration
Bitbucket

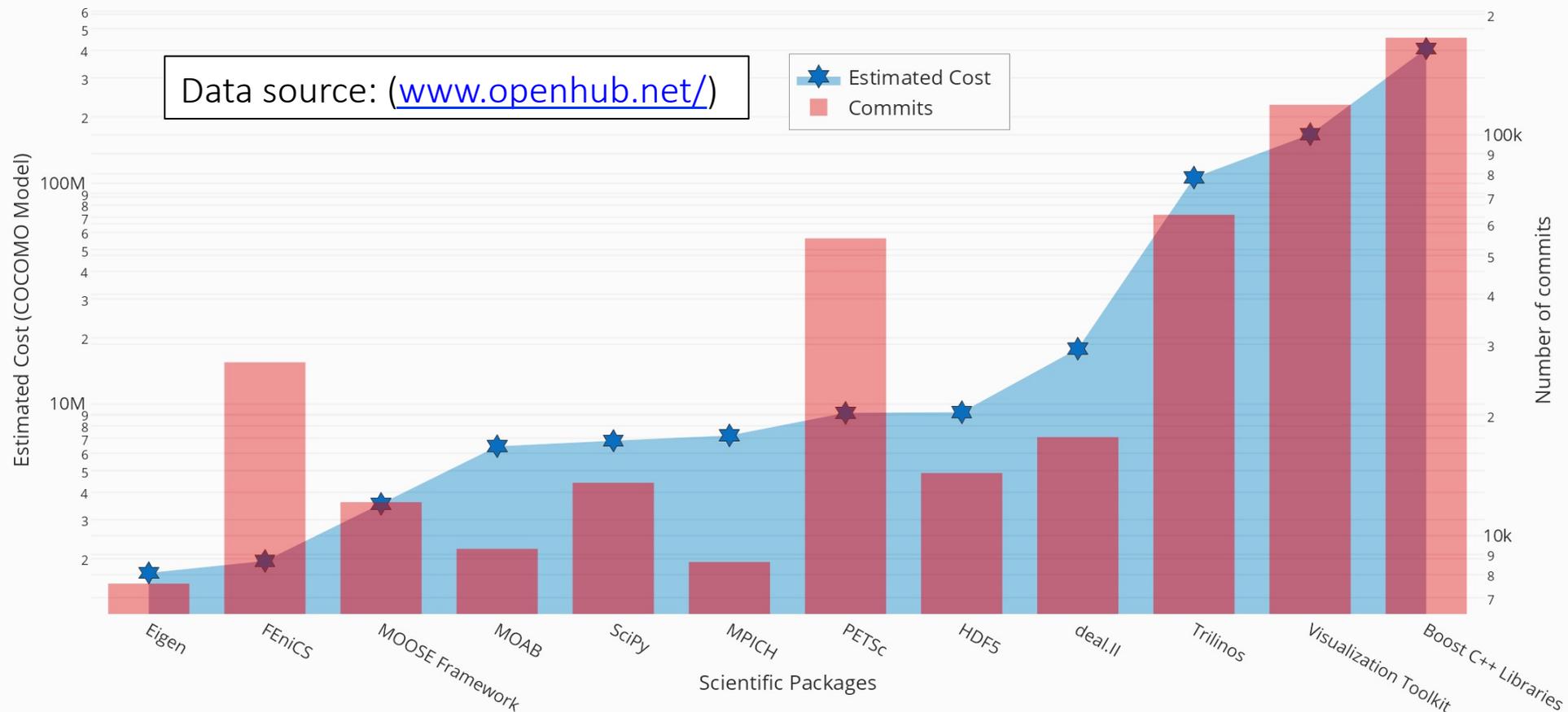
- Distributed review process
- Formalize release process
- Ease collaboration
- Resource management
- Extend scope

- Reproducibility
- Documentation
- Verification test suites
- User group support
- Licensing policy

Lessons from mature scientific software



Lessons from mature scientific software



- achieving distributed scale development requires rigorous software verification and open access to developers
- maintaining and sustaining scientific software requires increasing resources

Relevant and interesting references

1. Software Sustainability Institute (<http://www.software.ac.uk>)
2. Katz, D. S., et al. (2014) Summary of the First Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE1), Journal of Open Research Software, 2: (1).
3. Senyard, A., Michlmayr, M. (2004) How to Have a Successful Free Software Project, in Proceedings of Software Engineering Conference, Dec 2004.
4. Godfrey, M. W., Tu, Q. (2000) Evolution in Open Source Software: A Case Study, in Proceedings of the International Conference on Software Maintenance (ICSM'00), 131-142.
5. Brooks, F.P. (1987) No Silver Bullet – essence and accident in software Engineering, Computer, 20: (4), 10–19.
6. Software engineering for large scale systems (<http://www1.rmit.edu.au/courses/041259>)
7. Trajectory of a software engineer
<https://michaelochurch.wordpress.com/2012/01/26/the-trajectory-of-a-software-engineer-and-where-it-all-goes-wrong>