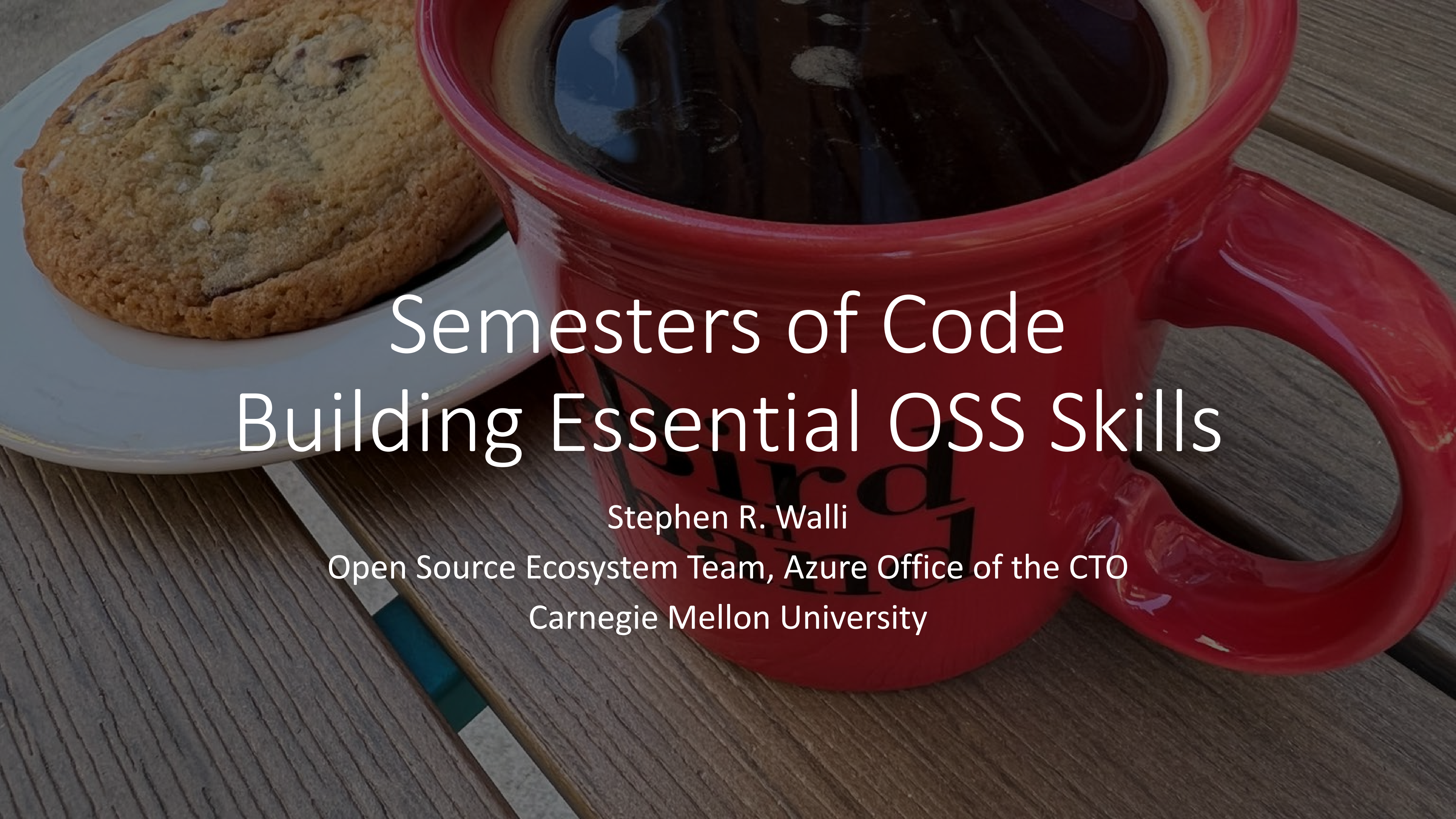


Semesters of Code Building Essential OSS Skills

Stephen R. Walli
Open Source Ecosystem Team, Azure Office of the CTO
Carnegie Mellon University

A history of collaborating in standards & OSS & nonprofits with other engineers

- IEEE Founding member P1003 Project Management Committee (1990-1999)
- Founder and VP Engineering in a VC-backed 'open source' startup (1995-1999) acquired by Microsoft in an asset acquisition in 1999
- Technical Director, Outercurve Foundation (2010-2013)

  "The Rise and Evolution of the Open Source Software Foundation" (with Paula Hunter)
- Linux Foundation
  - Microsoft Board & Board alternate for a number of years
  - Founding member, Open Container Initiative, (2015-2017)
  - Founding member, Board Chair, Confidential Computing Consortium, (2019-2022)
  - Founding member, eBPF Foundation, (2021-2022)
  - Mentoring Microsoft members in LF Energy, the Green Software Foundation, …
  - Member of the OpenSSF Governance Committee (2023-2024)
- Eclipse Foundation
  - Microsoft Board member (2021-2024)
  - Founding member, Steering Committee Chair, Software Defined Vehicle WG, (2021-2022)
- IEEE WG Chair, P3190, Recommended Practices for OSS Projects (2022)
- Along the way ISO, ECMA, OCP, LF Cloud Foundry, OpenStack, LF Open Manufacturing Platform (and JDF), and OMG/Digital Twin Consortium …

The Problem

“Software is eating the world.”

—Marc Andreesen, 2011

2012 Octoverse	4.6M+ repositories
2016 Octoverse	19.4M+ repositories
2018 Octoverse	96M+ repositories
2019 Octoverse	140M+ repositories
2020 Octoverse	200M+ repositories

“We are drowning in software, most of it mediocre, duplicative, and bad.”

—Not Marc Andreesen

The Industry Problem Statement

1. There is a startling lack of understanding of open source software in production
2. The knowledge of software engineering practice is being diluted and narrowed to tool practices across the industry (when anyone can be a net producer of software now)

AND

3. Software 'maintenance' (sustainability, security*, …) is an accelerating problem

*Software Transmitted Disease is the new new STD

The A-ha Moment!

- Well-run open source licensed projects are natural labs for software engineering experience
- What if we create an undergraduate course that taught:
  - Basic Software Engineering Theory
  - Healthy Open Source Software Project Practices
  - Intellectual Property Basics for Software Engineers
- Then create student projects in active open source licensed software projects with mentors as the lab/homework for the course
- Semesters of Code is born!

There is educational history …

- 20,000 students have come through Google Summer of Code
- Academics have taught “open source” for 15-20 years

Johns Hopkins University

The First Experiments

Choose Your Own Open Source Adventure

Johns Hopkins Intersession – January 2021 (COVID Lockdown)

- Lab 1: Build 125 year's worth of software value in an hour (httpd)
  https://github.com/jhu-ospo-courses/JHU-EN.601.210/tree/main/labs/1#lab-1-build-125-years-worth-of-software-value-in-an-hour-or-so
- Healthy open source software projects (On-ramps and practices)
  https://github.com/jhu-ospo-courses/JHU-EN.601.210/tree/main/lessons/3
- Lab 2: Evaluating Projects (Perl, Python, Node, Semester.ly)
  https://github.com/jhu-ospo-courses/JHU-EN.601.210/tree/main/labs/2#lab-2-evaluating-projects

The First Course (Fall 2021, 2022)

EN.601.270 Open Source Software Engineering (Semesters of Code I) (E, 3 credits)

The course will provide students a development experience focused on learning software engineering skills to deliver software at scale to a broad community of users associated with open source licensed projects. The class work will introduce students to ideas behind open source software with structured modules on recognizing and building healthy project structure, intellectual property basics, community & project governance, social and ethical concerns, and software economics.

The practical side of the course will engage and mentor students directly in OSI-licensed project communities to provide hands-on learning experiences of practices covered in the classroom modules, and team building experience working in the project.

Prerequisites: EN.601.220 Intermediate Programming & EN.601.226 Data Structures (see appendix)

Time: TuTh 10:30-11:45a ET

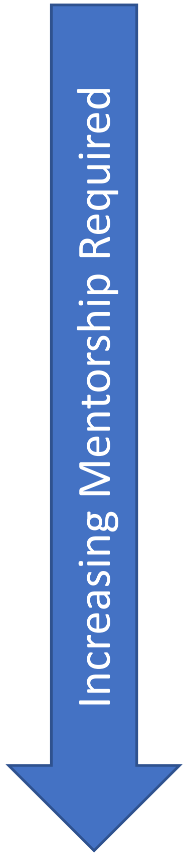
Limit: 35, CS majors only

Sizing Student Projects

Assumptions:

- 30-45 students
- Projects:
  - Fall 2021: PASS, Lutece, Powershell, Semester.ly, OpenCRAVAT
  - Fall 2022: PASS, Lutece, Powershell, enarx, OHDSI (“Odyssey”), .NET, PatternFly
- 2+ mentors define 5+ student projects per open source project
- There are ~14 weeks (includes a reading week break)
- 4-5 hours/week of student time **means ~50-70 hours per student project**

Ideas for Student Projects



- **Low-hanging fruit:** These projects require minimal familiarity with the codebase and basic technical knowledge. They are relatively short, with clear goals.
- **Fun/Peripheral:** These projects might not be related to the current core development focus but create new innovations and new perspective for your project.
- **Core development:** These projects derive from the ongoing work from the core of your development team. The list of features and bugs is never-ending, and help is always welcome.
- **Infrastructure/Automation:** These projects are the code that your organization uses to get its development work done; for example, projects that improve the automation of releases, regression tests and automated builds. This is a category in which a student can be really helpful, doing work that the development team has been putting off while they focus on core development.
- **Risky/Exploratory:** These projects push the scope boundaries of your development effort. They might require expertise in an area not covered by your current development team. They might take advantage of a new technology. **There is a reasonable chance that the project might be less successful, but the potential rewards for everyone make it worth the attempt.**

The General Evaluation

- **50% The Student Project**
  - **25% for each half term (25 October)**
  - **The first half term grade can’t be improved**
- 20% x2 in class mid-terms
- 10% In Class Participation
  - Discussions around regular short reading assignments (most weeks)
  - There’s one simple homework around project evaluations

The Mid-term Mentor Evaluation

- Did they get to a successful build in a reasonable way/time?
- Do they check in regularly via whatever project channels exist?
- Do they show progress towards goals (even artificial sub-goals) at a regular cadence?
- Are they accomplishing work – not just about the learning process, do they get work done?
- Does it feel like they will reach the original goal by term’s end? (Have expected outcomes changed?)
- Other notable observations
- Excellent? Good? Needs Improvement?

Semesters-of-Code Is Not GSoC

- It is a single course in a normal school semester – not a summer job
- GSoC plans for ~150 hours per student over the summer for a student project – we have a reasonable expectation of ~50-70 hours
- GSoC projects (being larger) tend to be 1:1 mentor to student project
- GSoC has a longer student matching period & ramp up period for larger student projects
- GSoC mentors expect to spend about 2-3 hours a week for each student project – GSoC projects are bigger in scope
- Google pays students stipends – <u>a key student outcome is a check!</u>

New in Fall 2022

- Doubled the student population from 20 to 40
- Experimenting with concurrent distance learning by adding students at University of Galway (with local administrative support)
- Added more student projects from industry (as opposed to research)
- Experimenting with multiple students working in the same student project (side-by-side, <u>not directly collaborating</u>)
- Moving to an active learning style in the classroom

Pedagogy

The Basics (from a Professional Teacher)

- Structuring of the lessons and lesson plans
- Every lesson has a set of learning objectives (which is very useful when building tests)
- Building the course curriculum flow
- Building the rubric and thinking about grading

All Learning Is Social Learning

- Lave, ‘Cognition in Practice’, 1988
- Lave & Wenger, ‘Situated Learning’, 1991, (Five Apprentice Situations Studied)
- **Legitimate Peripheral Participation Requirements**
  - Legitimate – Real opportunity for learning
  - Peripheral – The learner was outside learning their way in-group
  - Participation – The learning strategy involved the learner doing something

So, apprenticeship and mentorship

- Eventually leads to Wenger’s ‘Communities of Practice’ work, 1998

Active Learning & Carl Wieman

"Nearly all techniques labeled as active learning include those features known to be required for the development of expertise; in this case, thinking like an expert in the discipline. The **active learning methods are designed to have the student working on tasks that simulate an aspect of expert reasoning** and/or problem-solving **while receiving timely and specific feedback from fellow students and the instructor** that guides them on how to improve."

… **So, mentorship and social learning**

Large-scale comparison of science teaching methods sends clear message (2014)
www.pnas.org/cgi/doi/10.1073/pnas.1407304111

Learnings after the First Two Iterations at Hopkins

- Scaling for more students means scaling mentors & student projects
    - Scaling the mentoring workshops to set the bar
    - Automate student project onboarding and student project matching
    - Smooth the student on-ramp into the open source project
- Scaling for more schools/programs means scaling instructors
    - There is still ongoing curriculum tuning
    - Building a pipeline of industry-based instructors & instructor workshops
- **Mentors notice a growing problem with student time commitments**
    - **Students all have 4-5 other courses banging for their attention**
    - **70-80 hours is normal for project work and students are starting to struggle**

Carnegie Mellon University

Evolving the Experiments

Summer 2023 “Internship” Course

- A set of students lost their internships in the economic downturn
- We created a full internship experience within CMU SCS S3D
- Working with OpenStack (OIF) and a couple of Pittsburgh area startups we defined large projects for teams of 4-5 students to tackle as a team
- Students are each working as a team 20-40 hours per week (for 12 weeks)
- 2 project mentors per team, and weekly coaching time with the instructors
- LOTS of curriculum tuning for more active learning opportunities
- Student teams present status out to class regularly through summer
- CMU co-teaches classes (I get to learn with a professing professional!)

Remarkably Better Student Outcomes!

- Undergraduate students can solve big problems together with steep learning curves and complex toolchains
- Students mentor one another (without guidance) on the tool chain complexity! THIS WAS SURPRISING & APPARENTLY BACKED BY SCIENCE*
- Student confidence grows dramatically through the summer
- Mentors are excited tackling "unbudgeted" work in their open source projects
- Re-ran the course Summer 2024 with CMU-Qatar in a 10-week format,OpenStack & Eclipse projects, front packing the lessons in the first 6 weeks, and full remote project work for the last 4 weeks **with consistent excellent student outcomes†**

*Leveraging Learning Collectives: How Novice Outsiders Break into an Occupation, Ece Kaynak, 2023 https://doi.org/10.1287/orsc.2020.14214
† A CMU student team write-up from Summer 2024 on LinkedIn, and their final presentation at the close of course (16-min video).

Why is this exciting?

Why Are These Ideas Exciting?

- Peter Naur, “Programming as Theorem Building”, 1985 https://doi.org/10.1016/0165-6074(85)90032-8
- Jean Lave, “Cognition in Practice: Mind, Mathematics and Culture in Everyday Life”, 1988
- Jean Lave & Etienne Wenger, “Situated Learning: Legitimate Peripheral Participation”, 1991
- Carl Wieman’s work at Stanford and UBC in 2000s
- Leveraging Learning Collectives paper (previous slide)
- Successful modern project and non-profit cultures in open source and standards (IETF, IEEE, Eclipse, OpenStack)

Peter Naur and ‘Programming as Theory Building’, 1985

http://pages.cs.wisc.edu/~remzi/Naur.pdf

“… the programmer’s knowledge transcends that given in documentation …”

1. The programmer knows how the real-world maps to the program, and which parts of the world are relevant to the program or not.
2. The programmer can explain all design decisions. “The justification is and must always remain the programmer’s direct, intuitive knowledge or estimate.”
3. The programmer knows how best to modify the program to meet new requirements. This depends on recognizing similarities between new and old situations.

The Consequences If Naur Is Correct Are Everywhere

- RTFM … as long as the manual is current/correct
- “Comment your program” vs “comments can’t be trusted”
- “Your comments should reflect and explain your design”
- “When a design debate ends, the document is a record of when the shooting stopped, not a clear explanation.” [It’s a ceasefire line.]
- Literate programming systems (Knuth and programs as literature) [1990s]
- Formal methods (Z notation/VDM/TLA+ and programs as math) [1990s]
- TDD [2003] is predicated on a re-statement of the program’s theory as test assertions developed before the program
- Naur: If you don’t understand the theory of the program, changes create debt, until the program becomes unmaintainable – you create the proverbial Big Ball of Mud.

Naur’s Solution

“What is required is that the new programmer has the opportunity to work in close contact with the programmers who already possess the theory…. This problem of education of new programmers in an existing theory of a program is quite similar to that of the education problem of other activities where the knowledge of how to do certain things dominates over the knowledge that certain things are the case, such as writing and playing a music instrument. **The most important educational activity is the student’s doing relevant things under suitable supervision and guidance.”**

i.e., Legitimate Peripheral Participation, and Jean Lave’s pedagogical theory that all learning is social learning

All Software is Social Software

- Consider the well-run, OSI-licensed project communities you know
- Project governance is a practice – not just its documentation
  - The development process through PRs and Issues/email is mentorship
  - The release/delivery process is a social practice of managing the release
  - Engaging new community members through forums, meet-ups, email is a social practice
  - Running a nonprofit to remove risk requires establishing social norms beyond the charter for how decisions in committee happen so as to support the project work

N.B. This isn't just about OSI-licensed software project communities

Three On Ramps for Community Building

How do you encourage people to use your project?

(Because that's where you'll find bugs reports & tutorials & developers)

(How do you make it easy to install/configure/use the software?)

How do you encourage people selfishly to experiment?

(Because these are your future contributors)

(How do you make it easy to build/test/experiment?)

How do you encourage people to share their work?

(Because contribution flow is the growth and success of your project)

(How do you make it easy to contribute?)

Three On Ramps for Community Building

How do you teach/mentor people to use your project?

(Because that's where you'll find bugs reports & tutorials & developers)

(How do you make it easy to install/configure/use the software?)

How do you teach/mentor people selfishly to experiment?

(Because these are your future contributors)

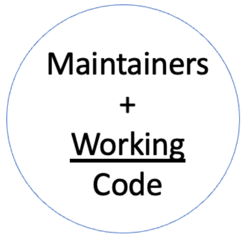
(How do you make it easy to build/test/experiment?)

How do you teach/mentor people to share their work?

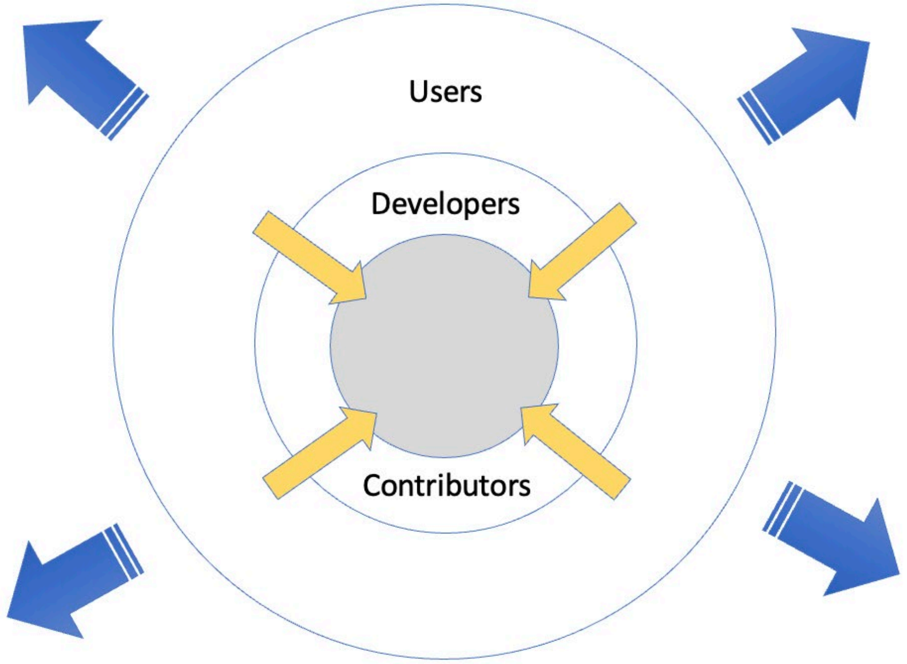
(Because contribution flow is the growth and success of your project)

(How do you make it easy to contribute?)

Frameworks for Building Software/Community/Nonprofit



Building the Software
(Sharing Innovation Outbound)



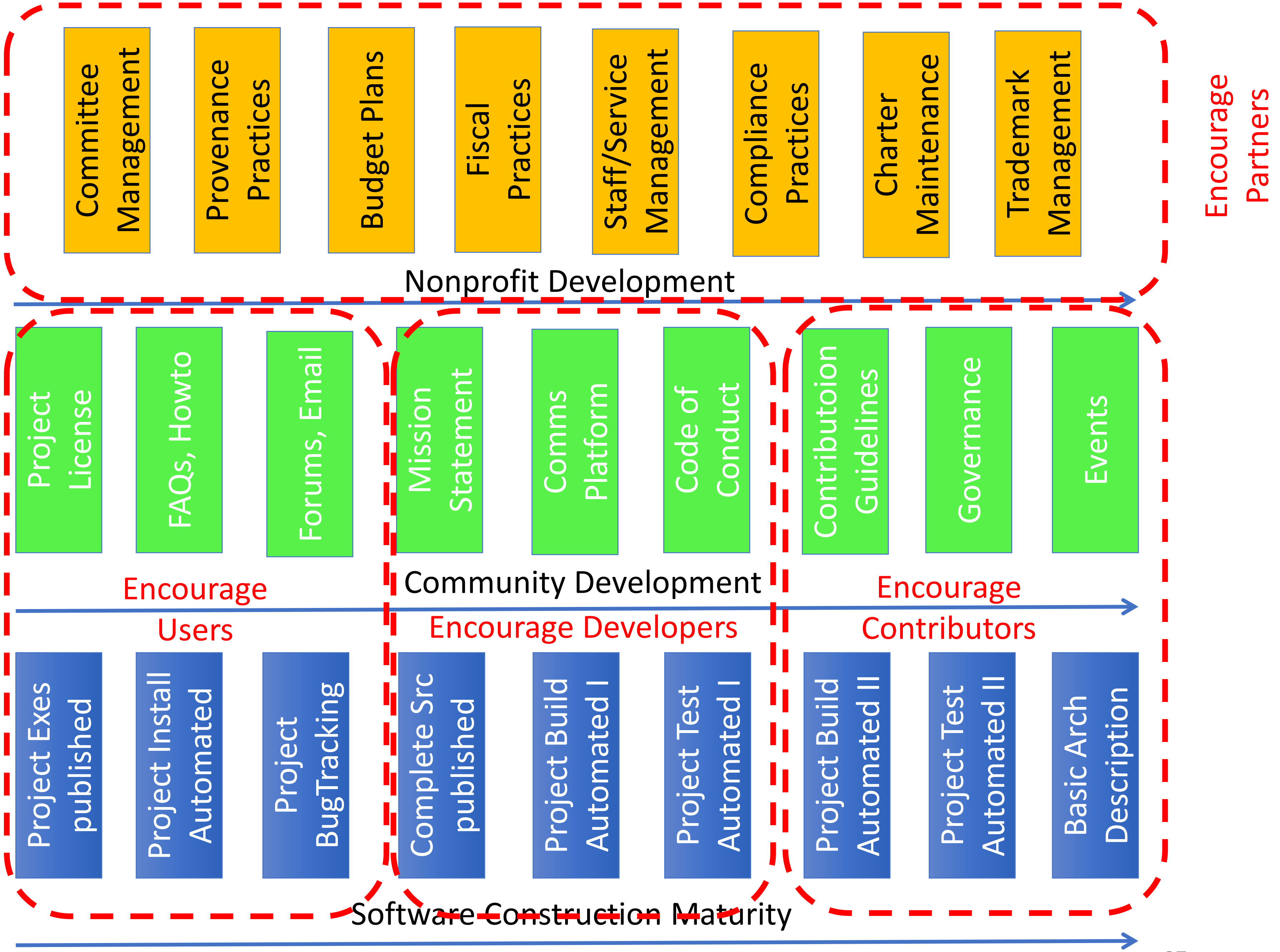
Building the Community
(Capturing Innovation Inbound)

Building the Non-profit
(Remove Risk, Hold Assets, Collect and Distribute Funds, Anchor the message)

Nonprofit
Activities

Community
Activities

Project
Activities



Continuing the Evolution

- Continuing the university experiments
- Software Engineering Bootcamps – Stepping outside tradition university settings (CMU-Africa, Co-Develop, Open Source Community Africa, etc.)
- Bringing the Summer Internship in-house – secure software practice!

Q&A