# No scale without discipline (Software Engineering 101)

Stephen R. Walli

# Objectives

- Understand the difference between programming-in-the-small and large systems programming
- Be able to explain the basic practices that are needed to deliver software at scale, and why they are needed
- Understand how to apply these ideas back to open source software projects as learning experiences, and as criteria for judging project health

Notes

https://github.com/jhu-ospo-courses/JHU-EN.601.210/tree/main/lessons/4#notes

# IEEE Definition (IEEE Std 610.12-1990)

**software engineering.**

(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

(2) The study of approaches as in (1).

**engineering.**

The application of a systematic, disciplined, quantifiable approach to structures, machines, products, systems, or processes.

# Oxford Definition

**engineering.**

the activity of applying scientific knowledge to the design, building and control of machines, roads, bridges, electrical equipment, etc.

# Science vs. Engineering

# $1 \rightarrow 10 \rightarrow 100 \rightarrow 1000 \rightarrow \ldots$

- You need to know how to build the program.
- You need a release process.
- You need to document how to install and run the program.
- You may want to tell users how to tell you about successes, failures, bugs, and new features that may be useful.

# Cooking Software

We All Know How To Fry An Egg

We May Get Good Enough To Cook For Friends

We May Tackle The Holiday Meal.

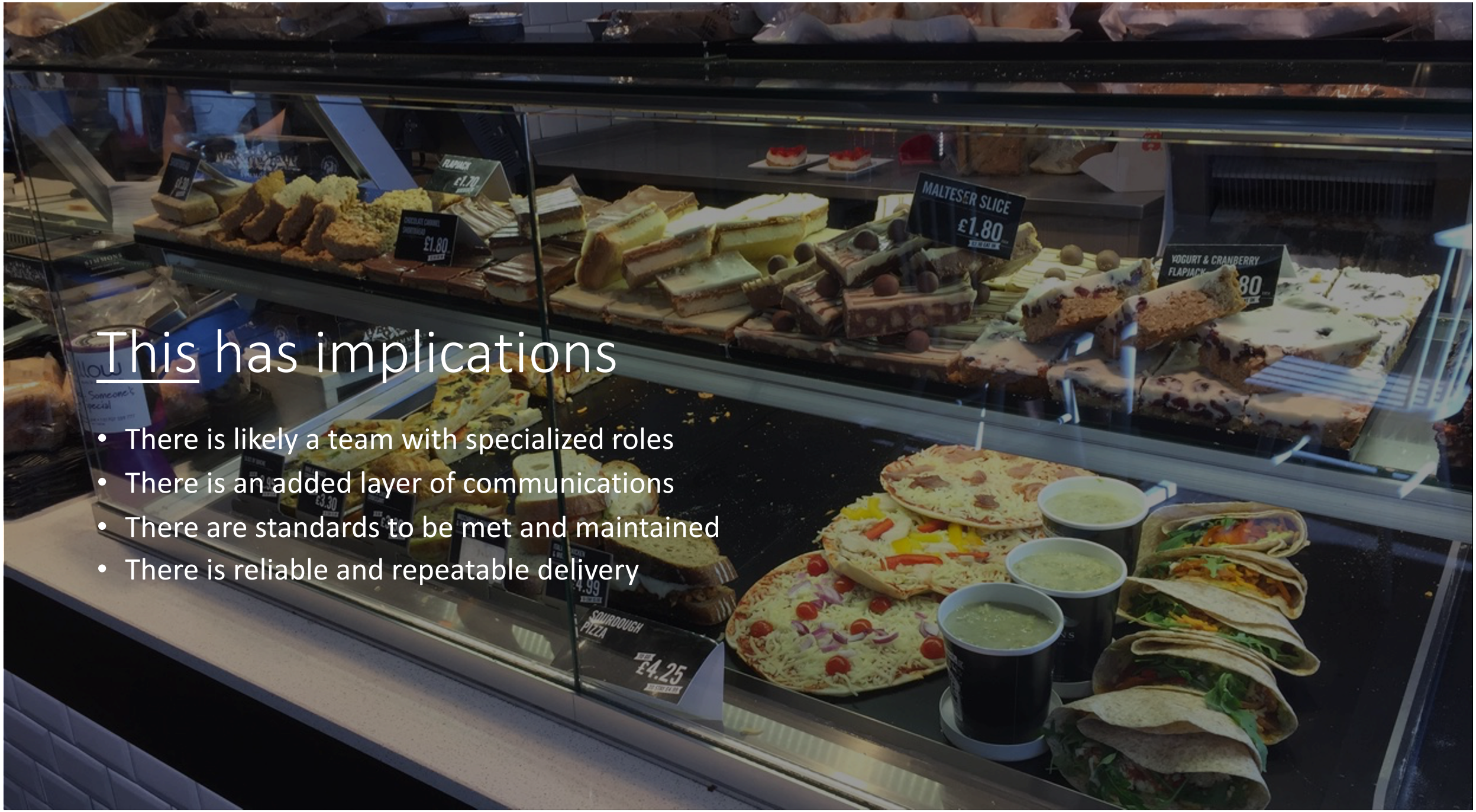
We May Get Good At A Particular Type of Cooking.

# Restaurant

# Taco Trucks and Michelin Experiences

We go from this …


… to this.




# This has implications

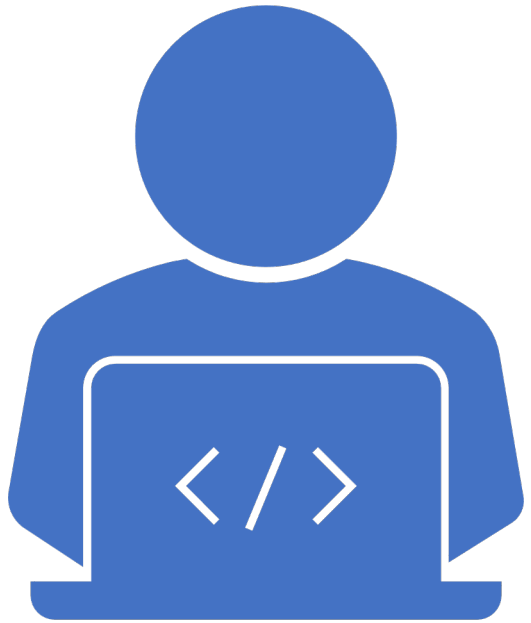
- There is likely a team with specialized roles
- There is an added layer of communications
- There are standards to be met and maintained
- There is reliable and repeatable delivery

# Scaling Problems = Engineering Problems

# users

software size and complexity

# developers



# Software is Inherently Dynamic

- Mistakes are discovered in the program in the way the solution is expressed.
- Mistakes are discovered in the program based on how it solves the initial requirements.
- New requirements are realized, and earlier requirements are retired.
- Genuinely new uses for the solution are discovered.
- Every useful application outlives the platform on which it was developed and deployed. (*Walli's First Law of Applications Portability*)
- Useful applications seldom live in a vacuum. (*Walli's Second Law of Applications Portability*)

# Stress

- Software system’s architecture
- On the team and the economics of change

# Technology Debt vs Refactoring

- Cost of a patch
- Mounting debt when fixing one problem creates more problems
- Restructuring/Refactoring for growth
- EVERY OSI-licensed project of note hits a refactoring point

# Lientz (1979) and Meyer (1988)

Breakdown of Maintenance Costs [Lientz 1979]

| **Reason for Change** | **% of cost** |
|---|---|
| Changes in User Requirements | 41.8 |
| Changes in Data Formats | 17.4 |
| Emergency fixes | 12.4 |
| Routine Debugging | 9.0 |
| Hardware Changes | 6.2 |
| Documentation | 5.5 |
| Efficiency Improvements | 4.0 |
| Other | 3.4 |

# An Attribute of Large Software Systems

- Bespoke solutions deployed in a company by IT development staff
- Large complex software product developed and sold by a company
- Long term project that supporting a collection of research projects
- A software system published under an OSI-approved open source license that has broad use and a large community

# Scale Problems in Large [open source] Software Systems

# users

software size and complexity

# developers

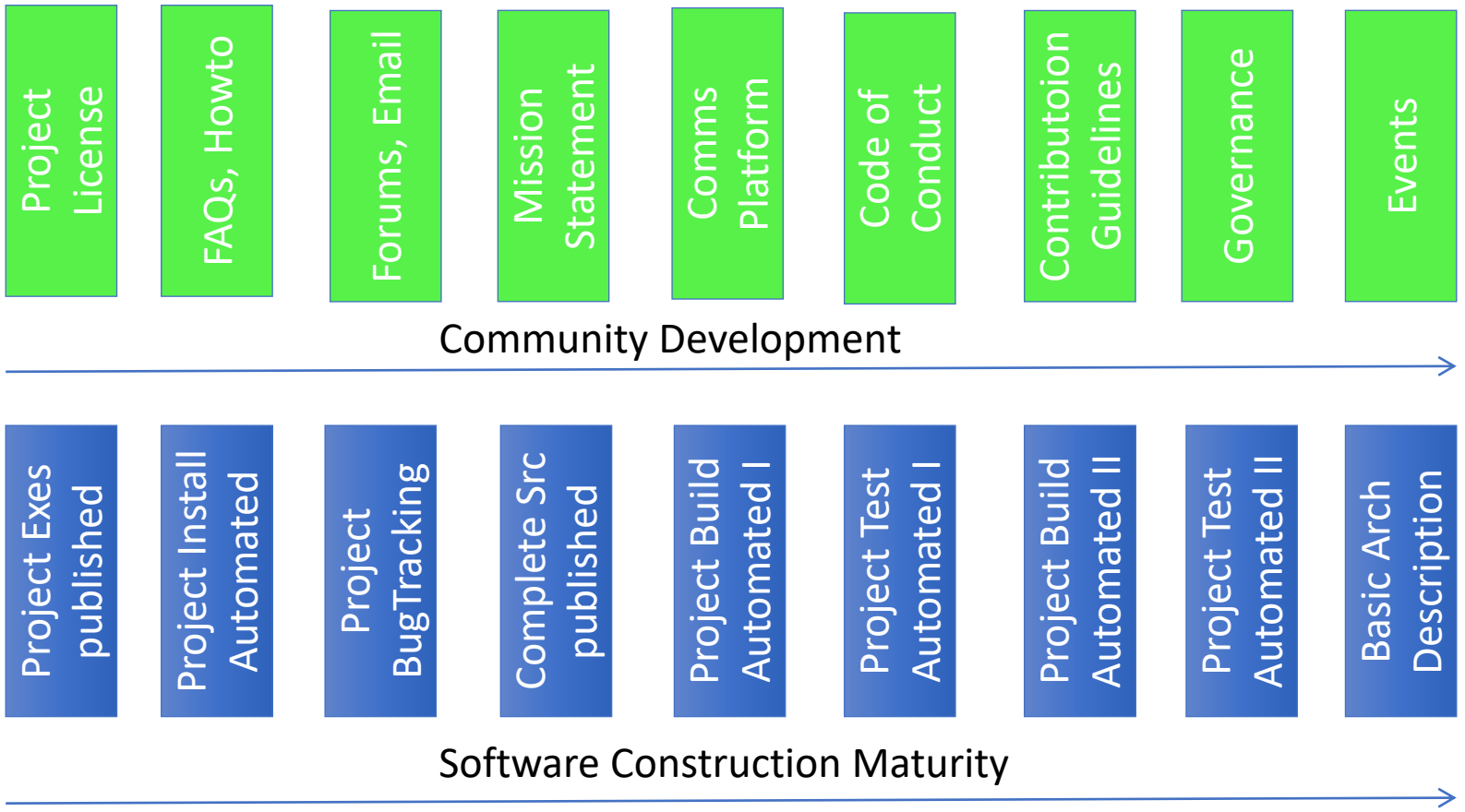
# Tillbrook & Software Hygiene

*This paper is concerned with software hygiene, which we feel is similar to personal hygiene, where software hygiene is the science concerned with maintaining healthy software. Like personal hygiene, software hygiene is most conspicuous in its absence. We do not claim that good software hygiene will help you win friends and influence people, but poor hygiene will certainly cause you to lose clients and discourage users.*

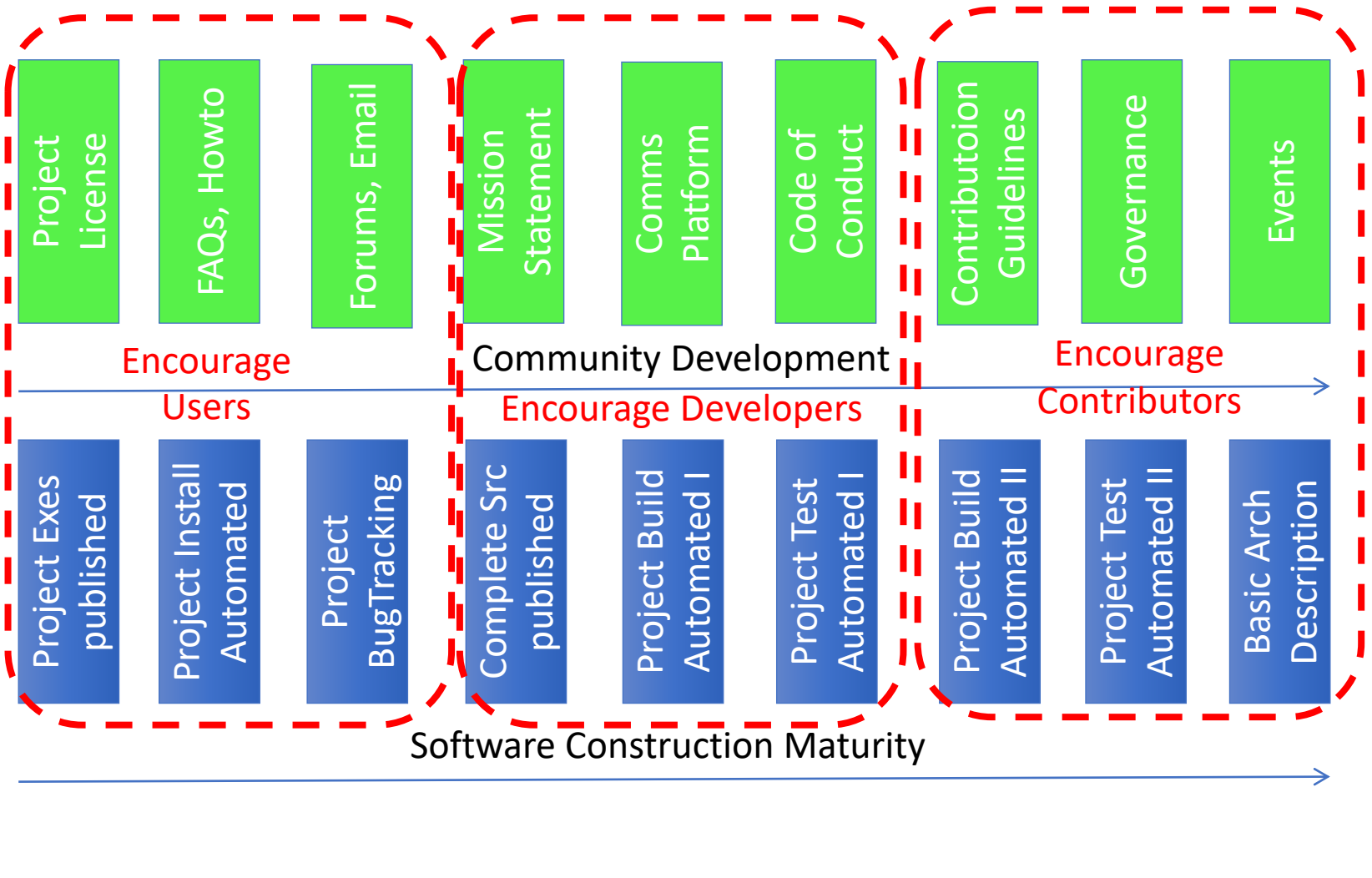
*Like personal hygiene, software hygiene is largely a matter of simple practices, the programming equivalent of washing behind your ears and flossing your teeth regularly.*

https://grosskurth.ca/bib/1990/tilbrook.pdf

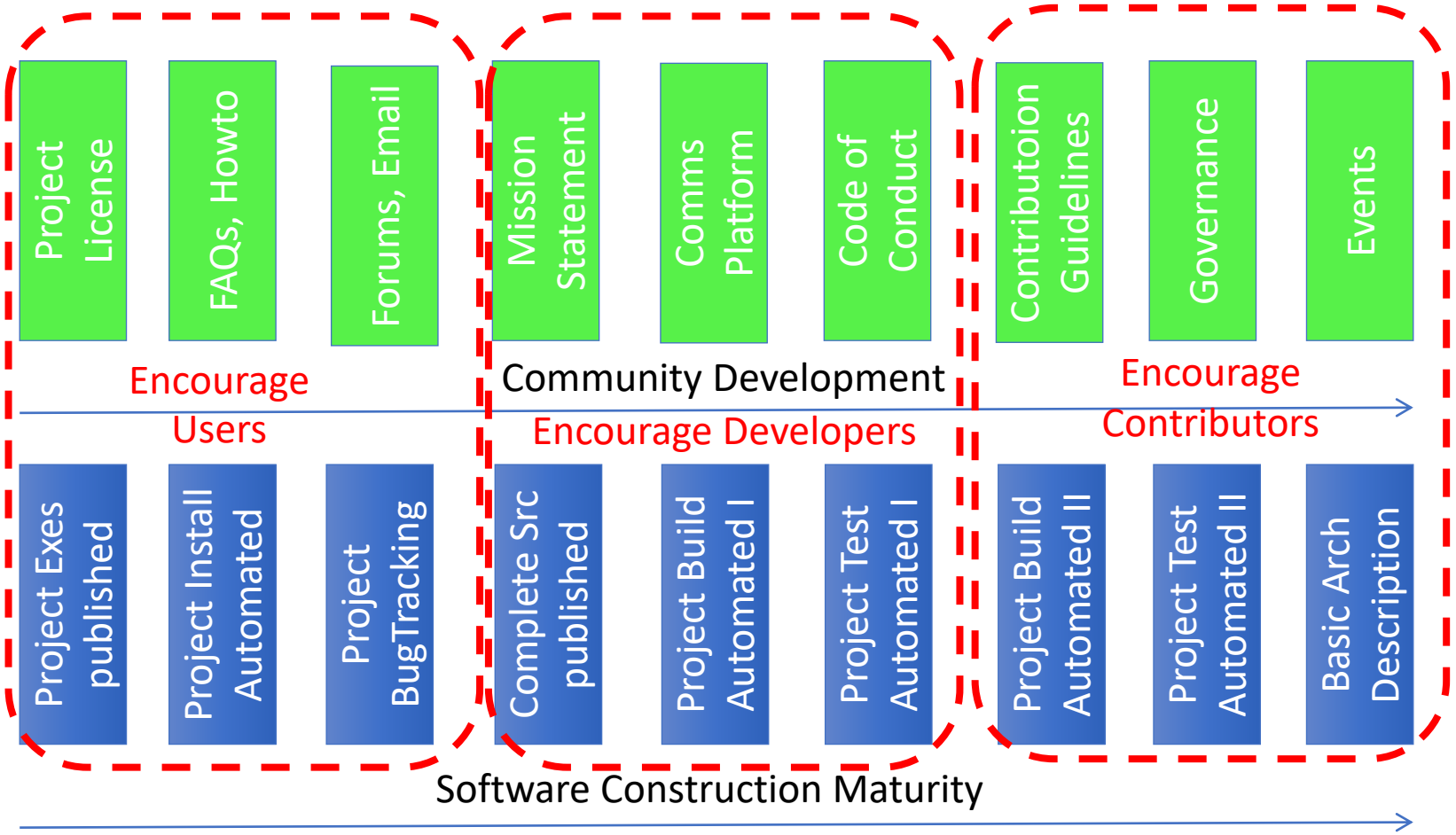
# Open Source Community Practices



# Open Source Community On-ramps



# Software Engineering Practices Enabling Scale



# Large OSI-Licensed Projects Manage

- Scale of participants so everyone understands the expectations and practices for their role (maintainer, contributor)
- Scale of the software system where automation and architecture reduces errors in consistent deliverables (across platforms)
- Scale in the community of use so everyone understands the expectations and practices for their role (user, developer)

# Software Construction Activities – I

- Consistent executables are built and available on known platforms.
- Project build is automated.
- Project has an automated installer for known platforms.
- Complete source is published and easy to download.
- Software source can be navigated to aid understanding.
- Project can be tested to a known state for known platforms.
- Bugtracking or issue tracking is available.
- Procedures for bug reports, bug triage, new feature requests, contributions (code, docs, tests, etc.) are documented, up-to-date, so everyone engaged has their expectations set.

# Software Construction Activities – II

For larger communities with a larger contribution flow:

- Project build pipeline support(e.g., linting, compliance scanning, etc.).
- Sophisticated project testing as part of a build pipeline.
- Architectural documentation exists as well as roadmap discussions, and how roadmap decisions are made is transparent and clear.

# ‘Community’ Development Activities

- The project license is easy to find as this is the outbound statement on how they share.
- There is easy on-boarding documentation (e.g., FAQ, How-to, startup tutorials).
- There is an easy engagement mechanisms (e.g., IRC, email distribution, forums).
- There is a mission statement.
- There is a Code-of-Conduct.
- It is clear which communications channels to use for what purpose.
- There are contribution guidelines.
- The project governance is well documented.
- There are real world events (e.g., conference BoF, Meet-ups).