LF ENERGY

Working with upstream open source projects

Ibrahim Haddad, Ph.D.
VP Strategic Programs, Linux Foundation
Executive Director, LF AI Foundation

@IbrahimAtLinux

linkedin.com/in/ibrahimhaddad/

github.com/ibrahimhaddad

IbrahimAtLinux.com

Ibrahim@Linux.com

LF ENERGY

What is upstream?

upstream (noun)

- The originating open source software project upon which a derivative is built
- This term comes from the idea that water and the goods it carries float downstream and benefit those who are there to receive it.

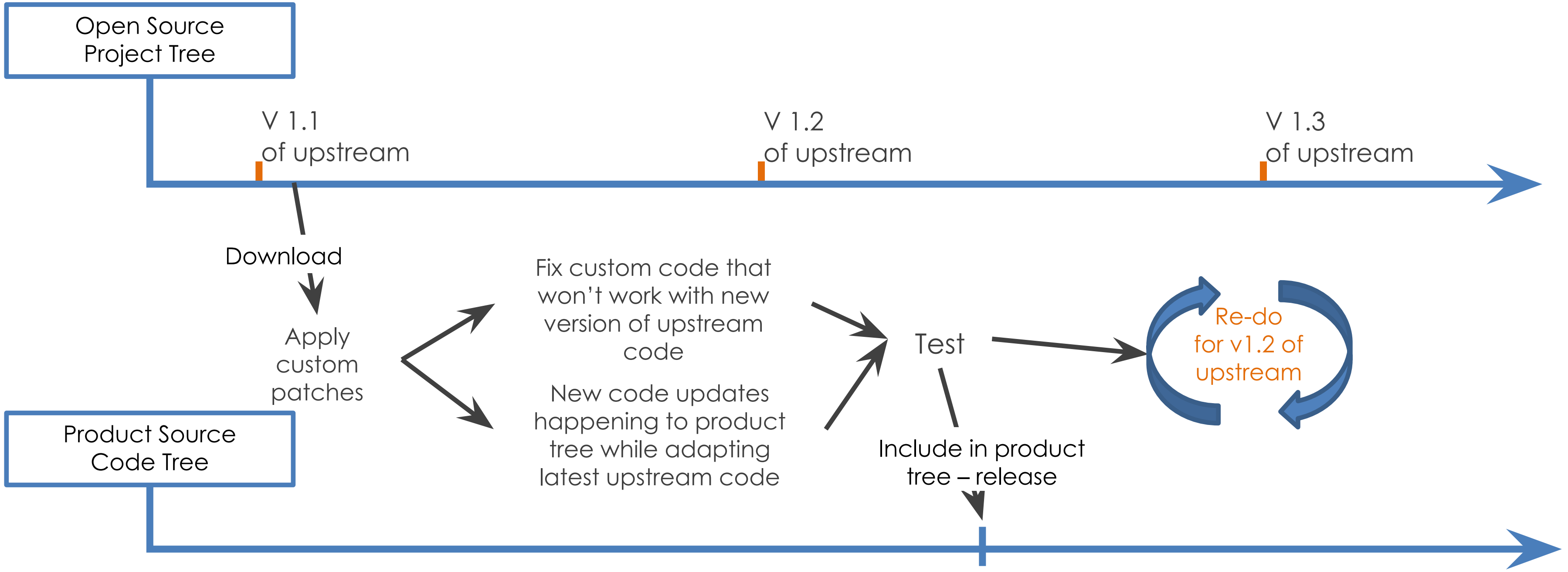
to upstream (verb)

- A shorthand way of saying "push changes to the upstream project“, i.e. contribute the changes you made to the source code program back to the original project dedicated for that program.

Motivations for contributing upstream

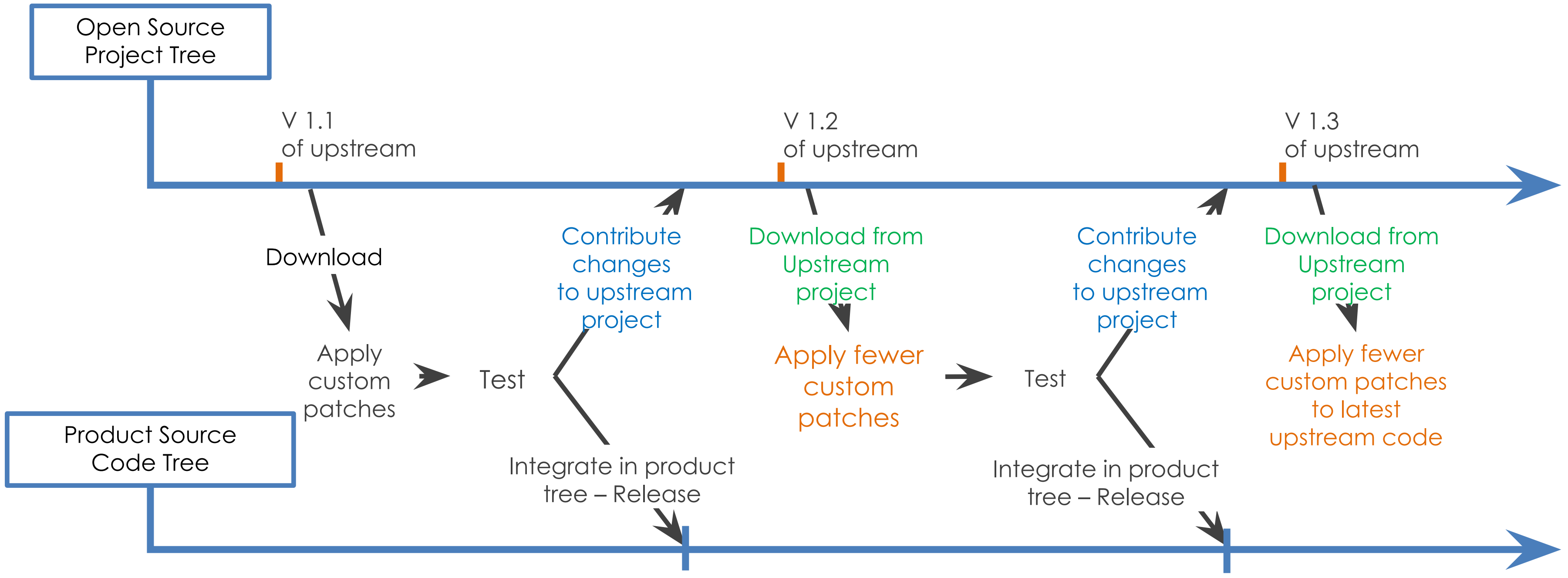
- Contributions strengthen and influence direction of the project
- Reduces the amount of effort to build and maintain open source software used in a product
- Contributed code is reviewed, improved
- Contributions may attract external developers into the project
- Integrating changes upstream means others are aware of them, and can plan for working with and around them
- Integrating changes reduces the risk of code breakage

Scenario 1: Development <u>without</u> upstreaming


Open Source Project Tree
V 1.1 of upstream
V 1.2 of upstream
V 1.3 of upstream
Download
Apply custom patches
Fix custom code that won't work with new version of upstream code
New code updates happening to product tree while adapting latest upstream code
Test
Re-do for v1.2 of upstream
Include in product tree – release
Product Source Code Tree

In-house derivatives of open source projects can delay development cycles.
High cost of maintenance.
Technical debt.

Scenario 2: Development with upstreaming


Open Source Project Tree
V 1.1 of upstream
V 1.2 of upstream
V 1.3 of upstream
Download
Apply custom patches
Test
Contribute changes to upstream project
Integrate in product tree – Release
Download from Upstream project
Apply fewer custom patches
Test
Contribute changes to upstream project
Integrate in product tree – Release
Download from Upstream project
Apply fewer custom patches to latest upstream code
Product Source Code Tree

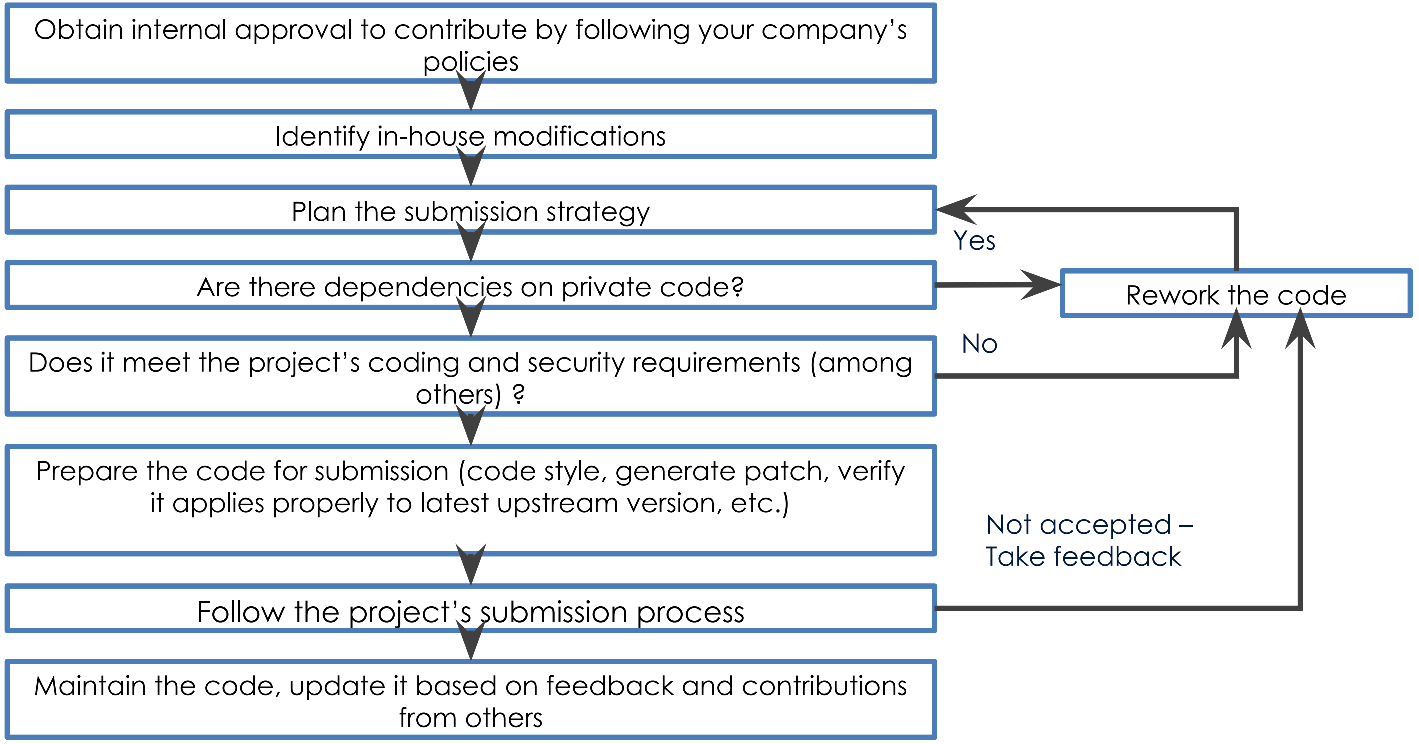
Upstreaming can accelerate development by reducing internal code maintenance.

Best practices

- Internally to your organization:
  - Upstream for the right reasons
  - Design and implement code with upstreaming in mind
  - Keep your developers involved in the open source project

- When submitting code:
  - Ensure the contribution is useful to others
  - Follow proper coding styles
  - Work within the submission processes for the project
  - Provide documentation
  - Listen to feedback, act upon it
  - Be patient and continue reworking you code until acceptance

Generic upstreaming process



Obtain internal approval to contribute by following your company's policies
Identify in-house modifications
Plan the submission strategy
Are there dependencies on private code?
Does it meet the project's coding and security requirements (among others) ?
Prepare the code for submission (code style, generate patch, verify it applies properly to latest upstream version, etc.)
Follow the project's submission process
Maintain the code, update it based on feedback and contributions from others
Yes
Rework the code
No
Not accepted – Take feedback

PS: This process is for illustration purposes. Several variations exist.

Characteristics of open source development

Characteristics

- Bottom up development
  - Those that do the most work get the most say
  - Personal relationships between developers are very important
- Release early, release often
  - Don’t wait to have a fully working version to release
- Peer review
  - Members of the community/project can comment
- Small, Incremental changes
  - Easier to understand small patches / changes
  - Very important because of lack of traditional test phase
  - A small change is less like to have unintended consequences
- Additional features should be small and non intrusive
- Features that ignore security concerns must be flagged

Characteristics

- Distributed development
  - Worldwide teams working in their own local time
  - Communication channels that reduce the impact of language barriers
  - Responsibility for development allocated to the individual or team with best capacity to deliver
- Diverse contributor base
  - Paid and volunteer contributors
  - Many companies working towards goal, including competitors
- Resilient to organizational change
  - Leadership earned with experience
  - If individuals cease to participate, there are others to take the place

Communications

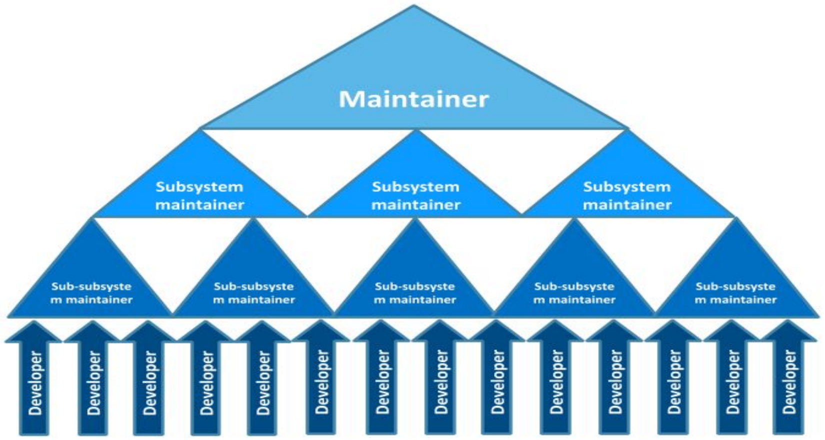
- Open Source developers are spread across the world
  - Rely on mailing lists, IRC/Slack or equivalent
  - No f2f meetings

- Email is very important
  - Development and discussions happen on open mailing lists

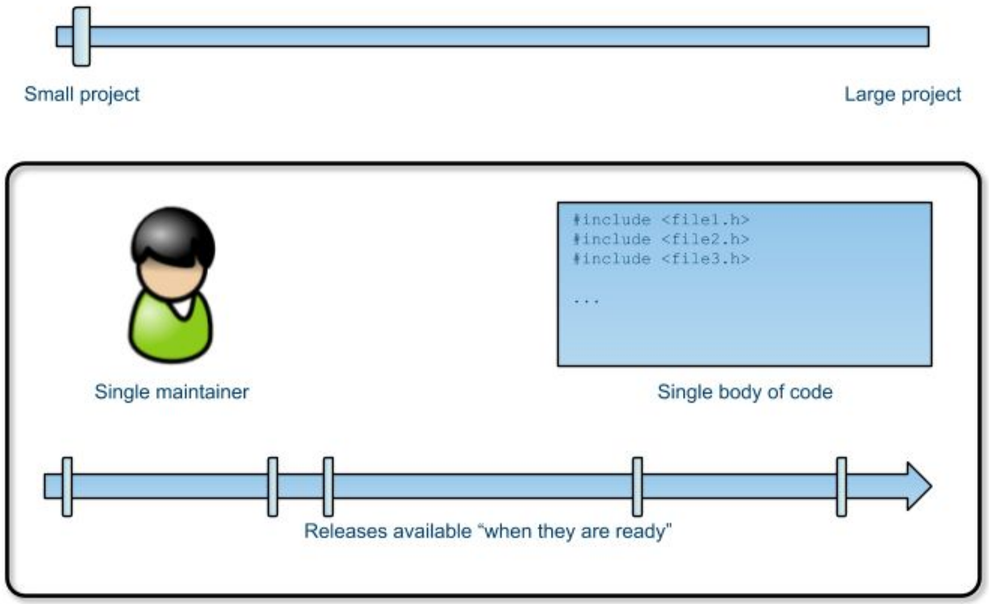
- Chat software
  - Most projects use chat for live discussion (irc, glitter, slack, etc.)

Project hierarchy

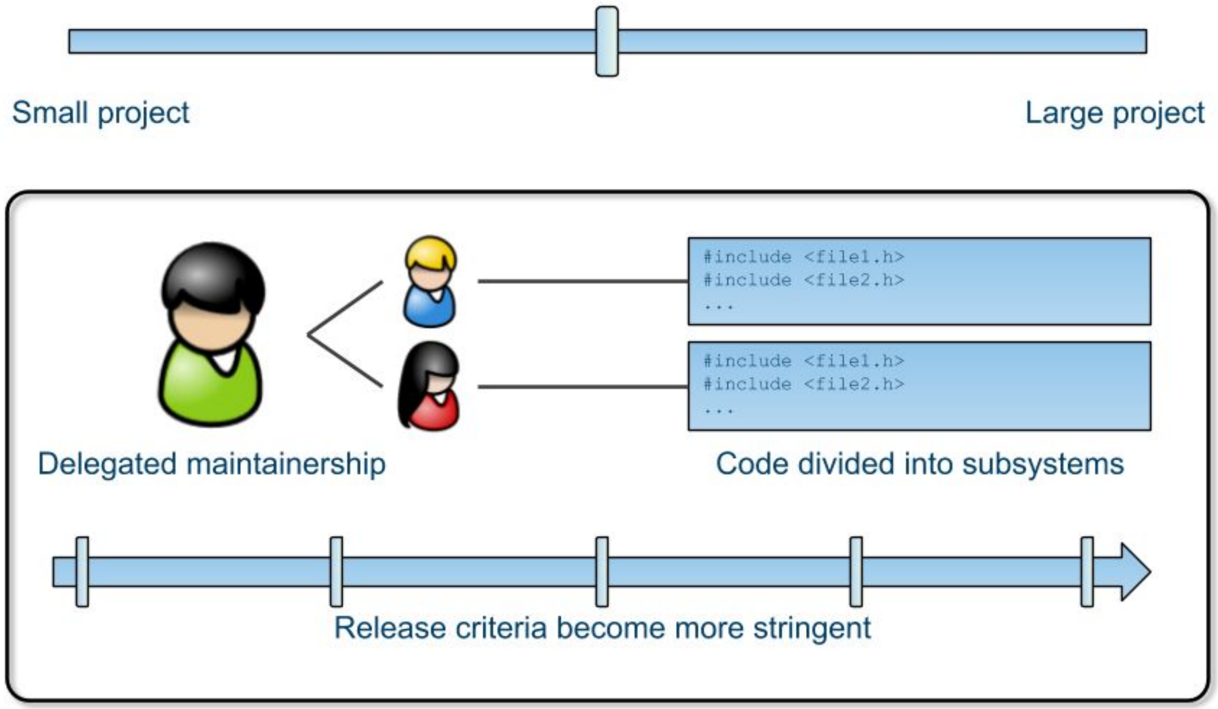
- Tight vertical hierarchy
- Loose horizontal structure
- “Bottom up” development
- Strong in code
- Small incremental changes flow upwards
- Meritocracy drives advancement and acceptance



Project hierarchy - small project



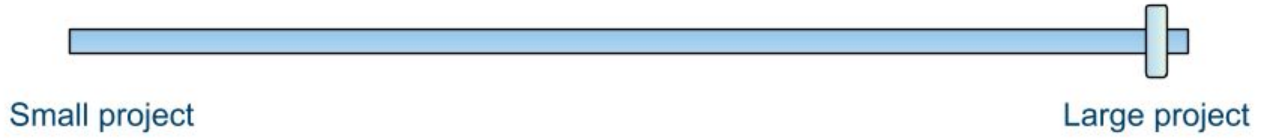
Project hierarchy - medium project

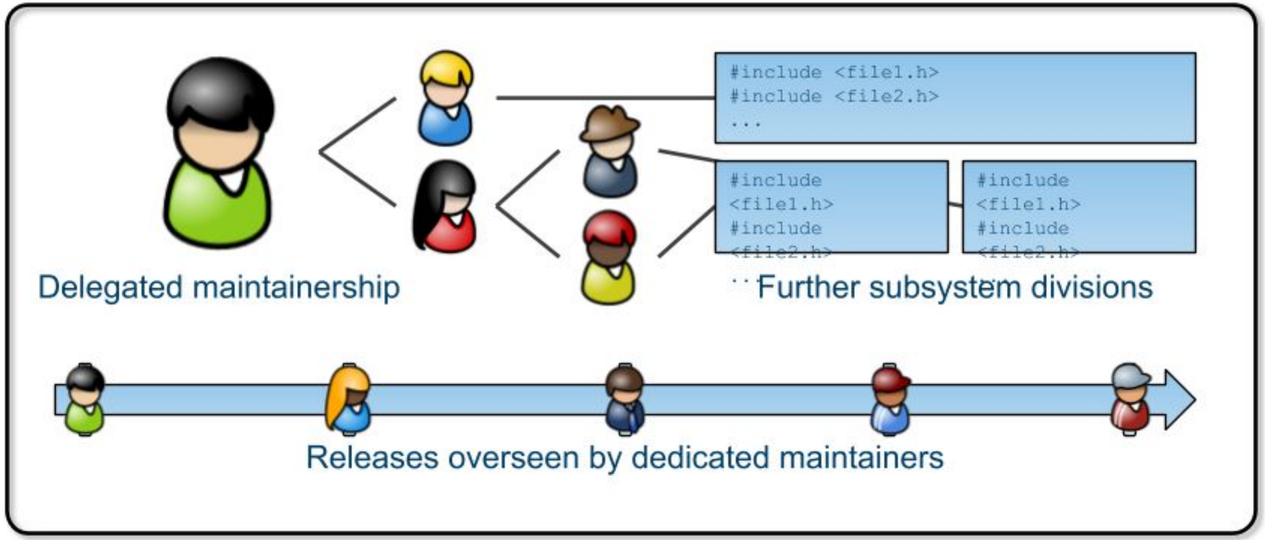

Small project
Large project
Delegated maintainership
Code divided into subsystems
Release criteria become more stringent

```
#include <file1.h>
#include <file2.h>
...
```

```
#include <file1.h>
#include <file2.h>
...
```

Project hierarchy - large project





Decision process

- **Decisions are decentralized by appointing trusted delegates**
  - In Linux, called “subsystem maintainers”
  - Trust built by past record of good participation and wise decisions on smaller issues
- **Not all decisions need pass through delegates**
  - Trivial fixes may go directly to any maintainer
- **Decentralized nature requires extra focus on transparency**
  - Discussions must happen in the open: Mailing lists, IRC, Slack, Glitter, etc.
- **Often the discussion itself is the documented record of the outcome**

Influence

- **Influence in a project is based on reputation in that project/community**
  - Reputation is gained through code contributions and participation
  - Reputation in one community doesn't necessarily carry to others
  - Reputation in own company doesn't carry to open source community
  - Must prove oneself on each project to gain influence
- **The writer of the best code has the most influence**
  - It doesn’t matter who you are, who you work for, how it was done before, or how smart you are
- **Behavior is important**
  - Willingness to take and provide feedback (politely)
  - Ability to explain motivations behind decisions

Taking responsibility

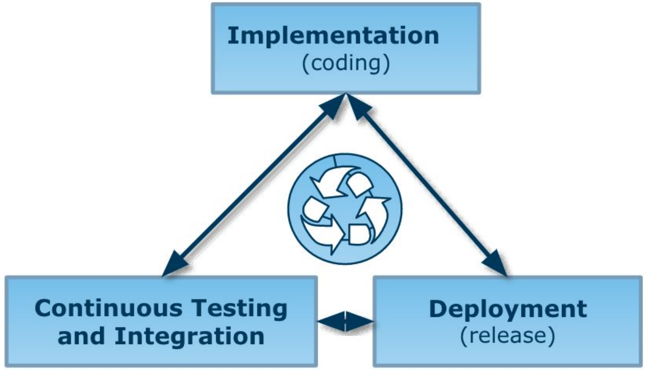
- **OSS development methods favor extra scrutiny on code origins**
  - Less direct oversight on any individual developer
  - Low barriers to entry in distributed development
- **Anonymous contributions are almost universally unwelcome**
  - Who wrote this?
  - Who do we go to if there’s a problem?
  - What if it wasn’t submitted by the owner of the code?
- **Requiring submitters to claim ownership of their code using real names results in higher quality code and better maintenance**
  - Credit can be given where due
  - If there are questions on the code, anyone can easily see who to ask
  - Each developer is personally guaranteeing the quality and the origin of the code

Peer review

- **In open source development peer review is mandatory and public**
  - Peer review helps maintainers process code faster based upon reviews by trusted devs
  - Anyone can comment, particularly those impacted by the change
- **Maintainers process large volumes of similar requests**
  - Be as straightforward as possible
  - Don’t assume familiarity
  - Respond promptly
- **Feedback is typically direct, efficient, and very honest**
- **Private communication is often discouraged**
- **If you have a valid technical reason to dispute feedback:**
  - Respond courteously
  - Highlight points the reviewer may not have considered

Continuous development, testing, integration

- Early discovery means faster triage and fixes
- Smaller changes make troubleshooting easier
- Regressions are noticed earlier
- Helps subsystem maintainers determine which code submissions to accept
- Projects may have multiple build and test cycles
- Typically tightly aligned with feature freezes



Release early and often

Release Early

Others can provide feedback and participate in development

New ideas can be incorporated while code is still flexible

Problems can be flagged by others before getting too far

Release Often

Smaller changes are easier to understand, debug, and drive to maturity

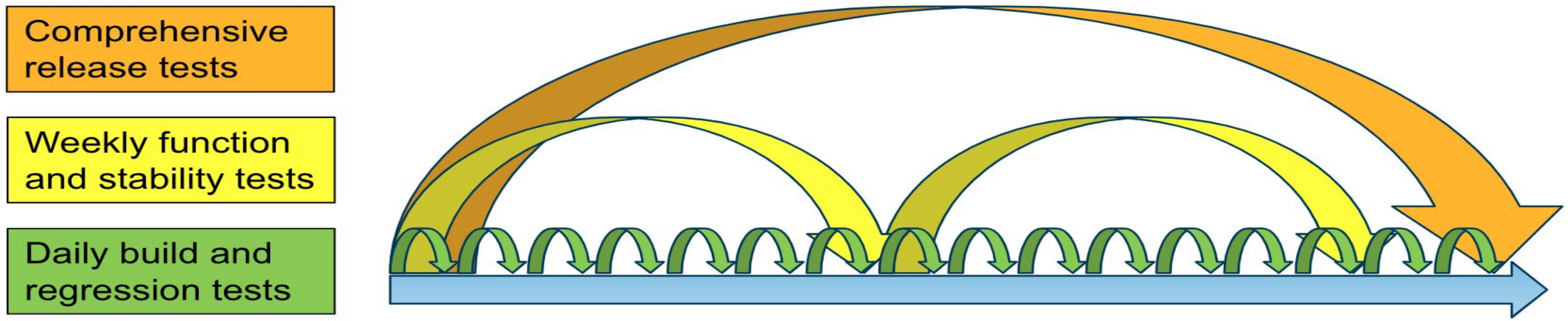
Maintain rapid pace of development and innovation

Overlapping release cycles

Overlapping releases with relevant testing seek to balance developer and user needs

Developers need up-to-date code

Users need a stable base to build products


Comprehensive release tests
Weekly function and stability tests
Daily build and regression tests

Modularity of code

- **Modularity helps projects scale**

- **Less contention over common code**
  - Smaller core with features implemented as plugins reduces collisions
  - Modularity creates a natural separation of tasks and scope

- **Features are available more quickly to developers**
  - Modular designs often have fewer interdependencies
  - Features released into the development tree as they are completed

Takeaway:

Adopt and follow an upstream first approach. The rewards are great and proven.

LF ENERGY

Working with upstream open source projects

Ibrahim Haddad, Ph.D.
VP Strategic Programs, Linux Foundation
Executive Director, LF AI Foundation