Collaborative Development in HEP

GridKa School 2017
Karlsruhe

28.Aug. 2017

Benedikt Hegner
EP-SFT - CERN

Outline

- Collaborative Software Development at what scale?
- The different roles in a project
- One example of a ~~bad~~ historical approach
- The pull-request workflow
- Sharing and organizing the work

What is collaborative development?

Essentially every software project having more than one developer is a collaborative project

Complexity increases with increasing number of developers

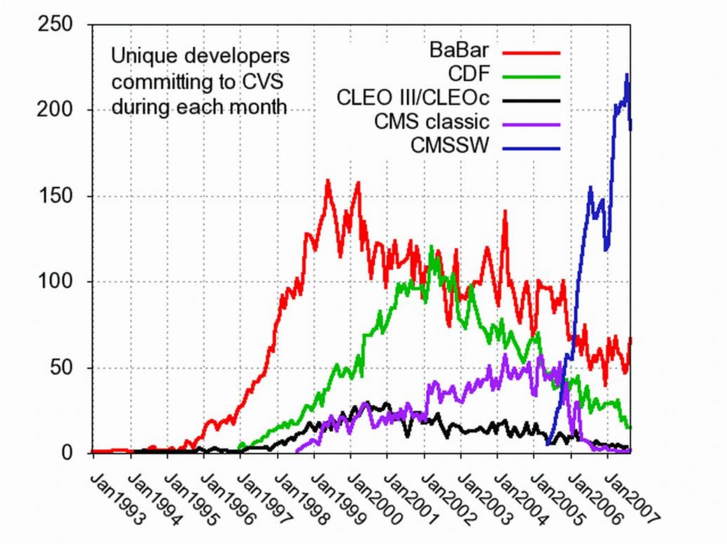
Complexity increases with increasing geographical distance between developers

Collaborative Development in HEP

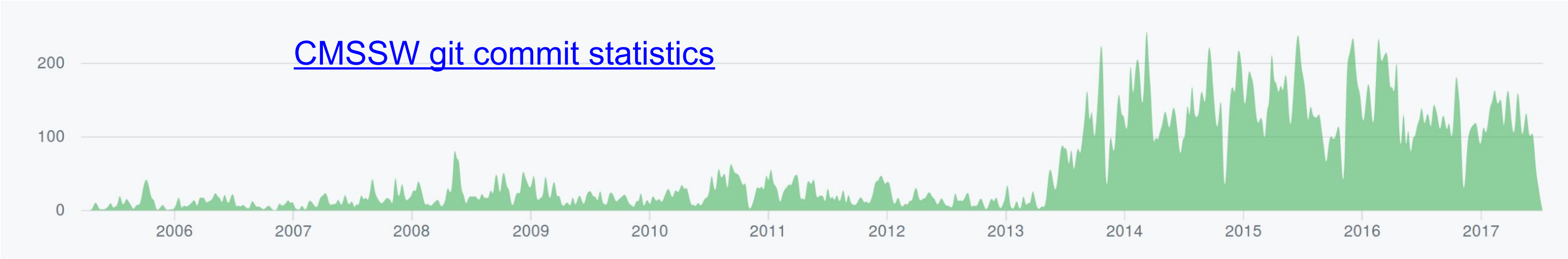
Software in HEP experiments is large

- O(M) lines of code
- O(100) developers

Distributed across the globe



Very extreme environment for scientific software


CMSSW git commit statistics

Benedikt Hegner

Different stakeholders in a project

There are different project roles. The most important ones are

Developer

- Well… developing the software

User

- No project without a user...

Release manager

- Organizing the crowd

Let’s have a closer look at those roles...

The stereotypical developer

- There is this cool feature to add
- Do not want to waste time with
  - Infrastructure
  - Other people’s problems/stupidity
- Want to develop on my own laptop!
- Docs? Nah! Let me do the cool stuff!

The stereotypical external contributor / power user

- Great SW, but there is this important thing missing
  - Hacked something together that certainly goes into the next release!
- Where the heck are the docs?
- Can you make it easier to build?

The stereotypical release manager

- Things should be stable
- No, please not another last minute change!
- Please stick to the rules, guys!

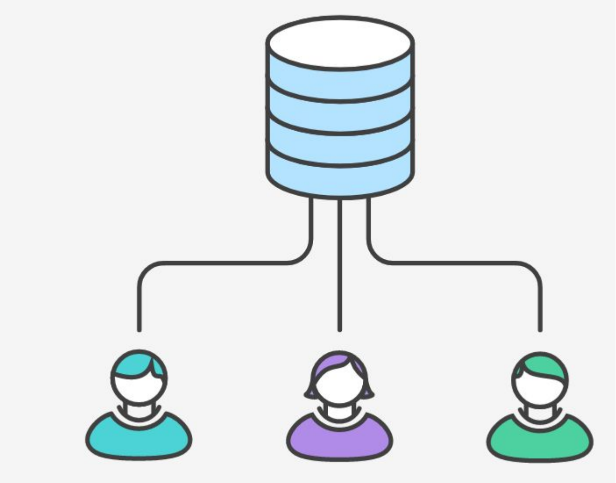
How to work on a common code base?

- No way around having code versioning
  - Some people in research however try!
- Plenty of tools around. These days the two most favourite ones are:
  - git
  - mercurial

But these are just tools. Important is how they are being used...

Being naive - commits to a common repository

- Every developer pushes into the same repository
- The current state of the project is the *HEAD*
- That rarely works out
  - Even with one developer ;-)
  - ...and we are talking about hundreds



Only possible with a very small and disciplined team of developers

In that setup the HEAD is almost always broken

People step onto each others toes

Tag Collector - a past workaround in HEP

Only a limited number of developers is allowed to touch a specific subdirectory

- Prevent “less experienced” developers to destroy code by accident

The current state of the project is not the HEAD, but a combination of O(100) **directory specific tags**

- Developers register their new **tag sets** into a **tag collector**
- Software managers review the tags and decide what enters the next nightly build

Code review and testing done ad-hoc - if done at all!

- Nightlies were almost always broken
- But nightlies were the only resource to build against

This approach totally destroyed the code layout

Interlude - Code Layout I

Functional code layout

- What features are provided to users?
- How do components interact with each other?
- A good functional layout increases codeshare

Physical code layout

- Where do these components live?
- Which directories exist?
- What makes up a single linker library in the end?

Interlude - Code Layout II

The physical layout should try to reflect the functional layout

- Should be intuitive for developers where to find what
- Should be intuitive for users which header to include and what to link to

```
...
+ Reconstruction
     + Ecal
     + Muon
...
```

Instead one ends up with

- One or more developer specific directories
  - Grouping by de-facto responsibility
- At first sight much easier as it avoids conflicts between developers

```
...
+ Reconstruction
     + Dev1MuonAlgs
     + Dev2MuonAndECalAlgs
...
```

Interlude - Code Layout III

Having a developer-centric code layout causes a few problems

- Sharing code / putting code into common packages becomes hard and slow
- Too strong of a signal for explicit code ownership
- Knowledge about code gets too fragmented
- If a change involves multiple packages, it delays things significantly

The LHC approach of tag collectors and directory permissions made it even worse.

Every project risks to end up with a code layout

driven by developer responsibility!

How to come up with a better solution?

There are many problems to solve

1. How can one do development without interference with others?
2. How can other people co-develop the same piece of code?
3. How can one build a coherent release out of things
4. How can one (quality) control what enters a release?

The tag collector workflow focussed on point 3 - thus was very developer and quality unfriendly

Distributed code repositories such as git and mercurial help with points 1. and 2.

How to come up with a better solution?

Every developer has a copy (fork) of the repository

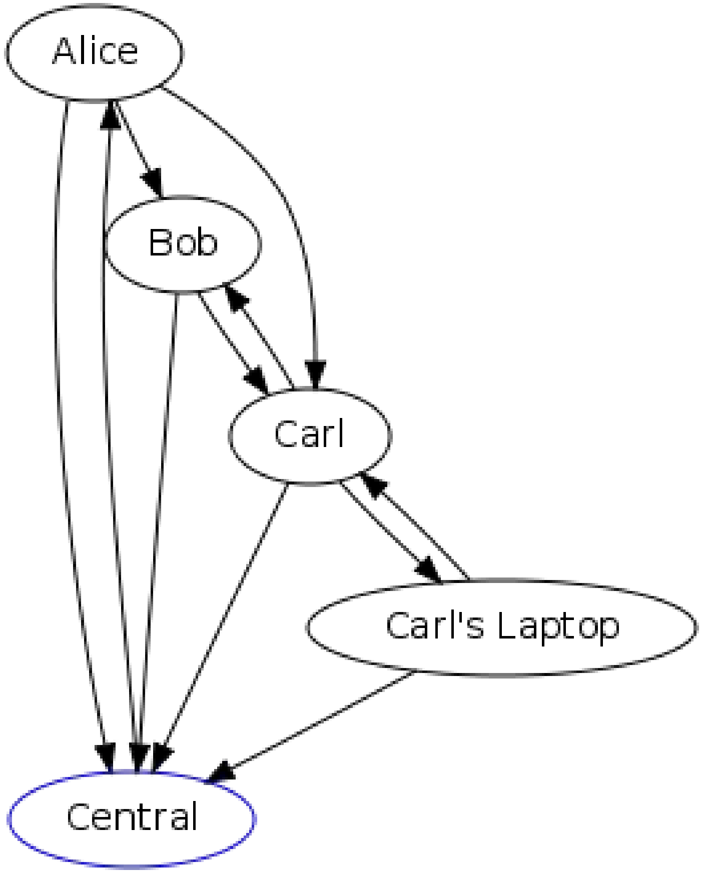
- In a consistent state!

Code can be exchanged without interfering with the official repo

- Bob can test changes of Alice in an early stage

However, it may become very chaotic rather soon…

…and it does not solve the problem of (quality) control

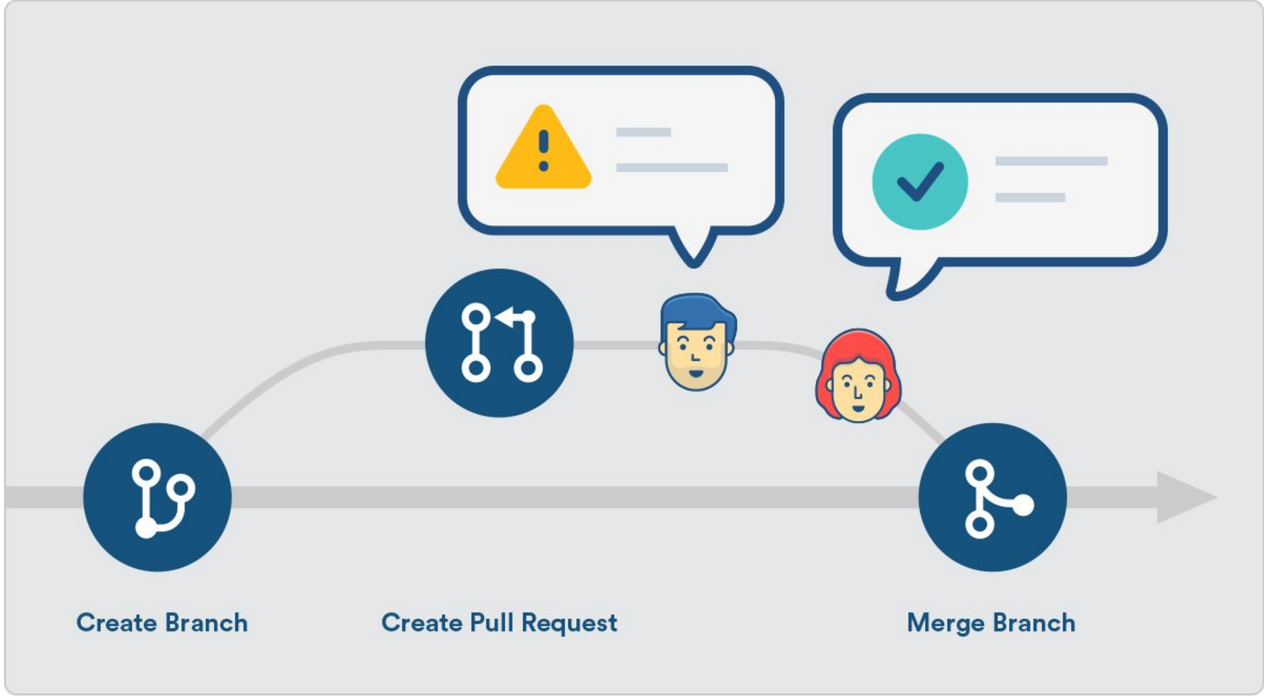


A Mercurial Network

One needs procedures on top of git and mercurial

The concept of pull requests

Instead of pushing to the central repo directly, a developer does changes to the private repo and asks for the inclusion of the changes into the official repo



Ensures that the repository is always in a releasable state!

The stages of a pull request

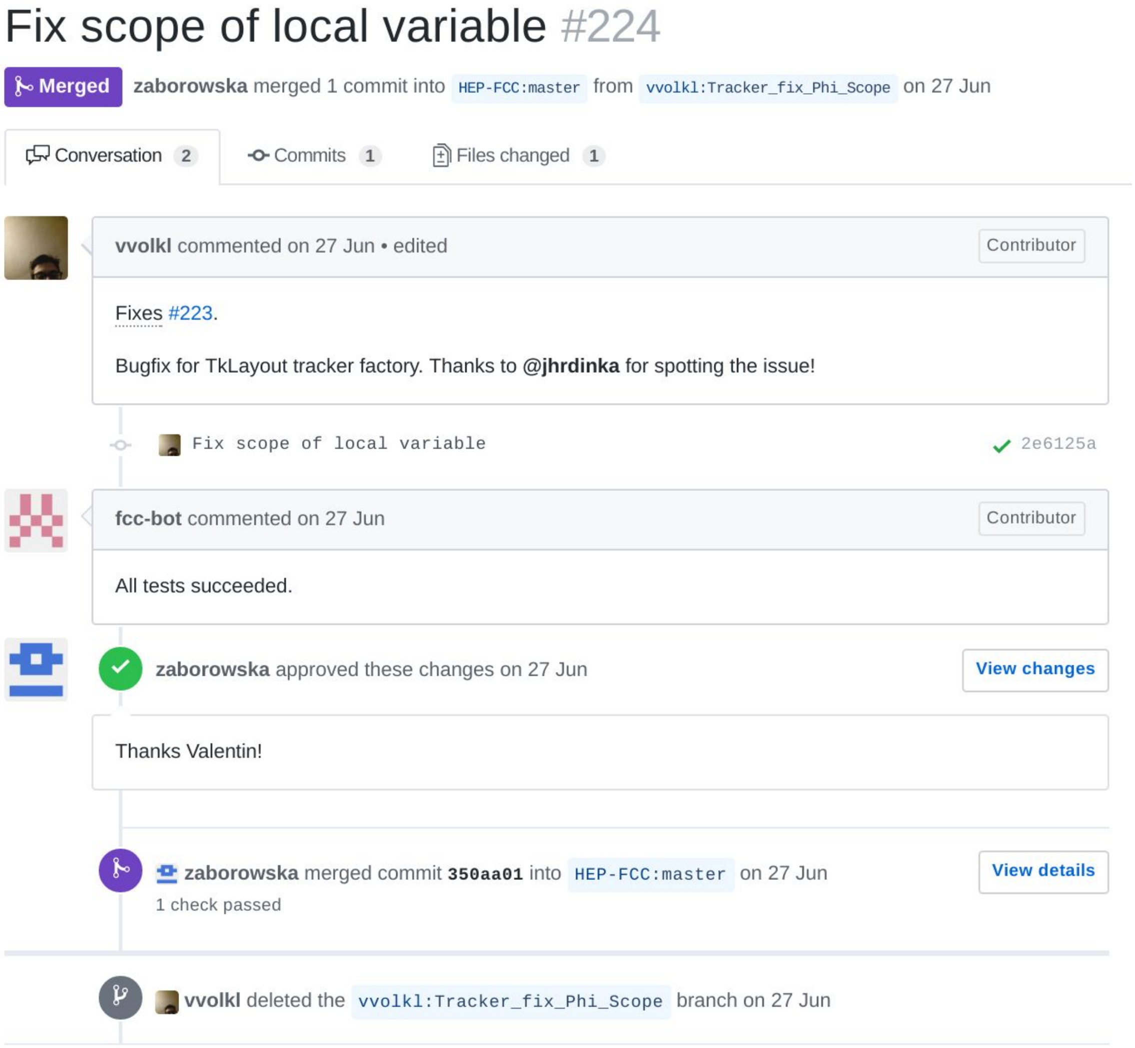
1. Develop a new feature in your own fork and branch
2. Test it ;-)
3. Open a pull-request in the web-interface of your choice
4. Let the project’s infrastructure test it automatically (*continuous integration*)
5. Let another developer review the code (**including documentation!**)
6. Wait for inclusion into the project
7. Be happy :-)

There are plenty of tools to support you with that (gitlab, github, …)

At the beginning not everybody will like their code being inspected

and work being ‘controlled’

A pull-request example



Fix scope of local variable #224

Merged zaborowska merged 1 commit into `HEP-FCC:master` from `vvolkl:Tracker_fix_Phi_Scope` on 27 Jun

vvolkl commented on 27 Jun • edited Contributor

Fixes #223.

Bugfix for TkLayout tracker factory. Thanks to **@jhrdinka** for spotting the issue!

`Fix scope of local variable` ✓ `2e6125a`

fcc-bot commented on 27 Jun Contributor

All tests succeeded.

zaborowska approved these changes on 27 Jun

Thanks Valentin!

zaborowska merged commit **`350aa01`** into `HEP-FCC:master` on 27 Jun

1 check passed

vvolkl deleted the `vvolkl:Tracker_fix_Phi_Scope` branch on 27 Jun

Pull-requests - not only a technical procedure

At first sight, a pull request is about getting a certain piece of code integrated into the code base

- In a well organized and well documented way!

The much more valuable side of this ***constant code review*** is the training value

- The original author gets immediate feedback on code quality
- The reviewers learn about other parts of the code base

  (a.k.a. ***knowledge preservation****)*

A good combination is having two reviewers

- An expert to ensure quality
- A beginner to learn from other code and ask the ‘stupid’ questions

Interlude - Testing I

We’ve heard “testing” a few times now, but what does it actually involve?

It makes sure that your code does what it should

- And that it ***continues*** doing so!

It documents what the code is supposed to do

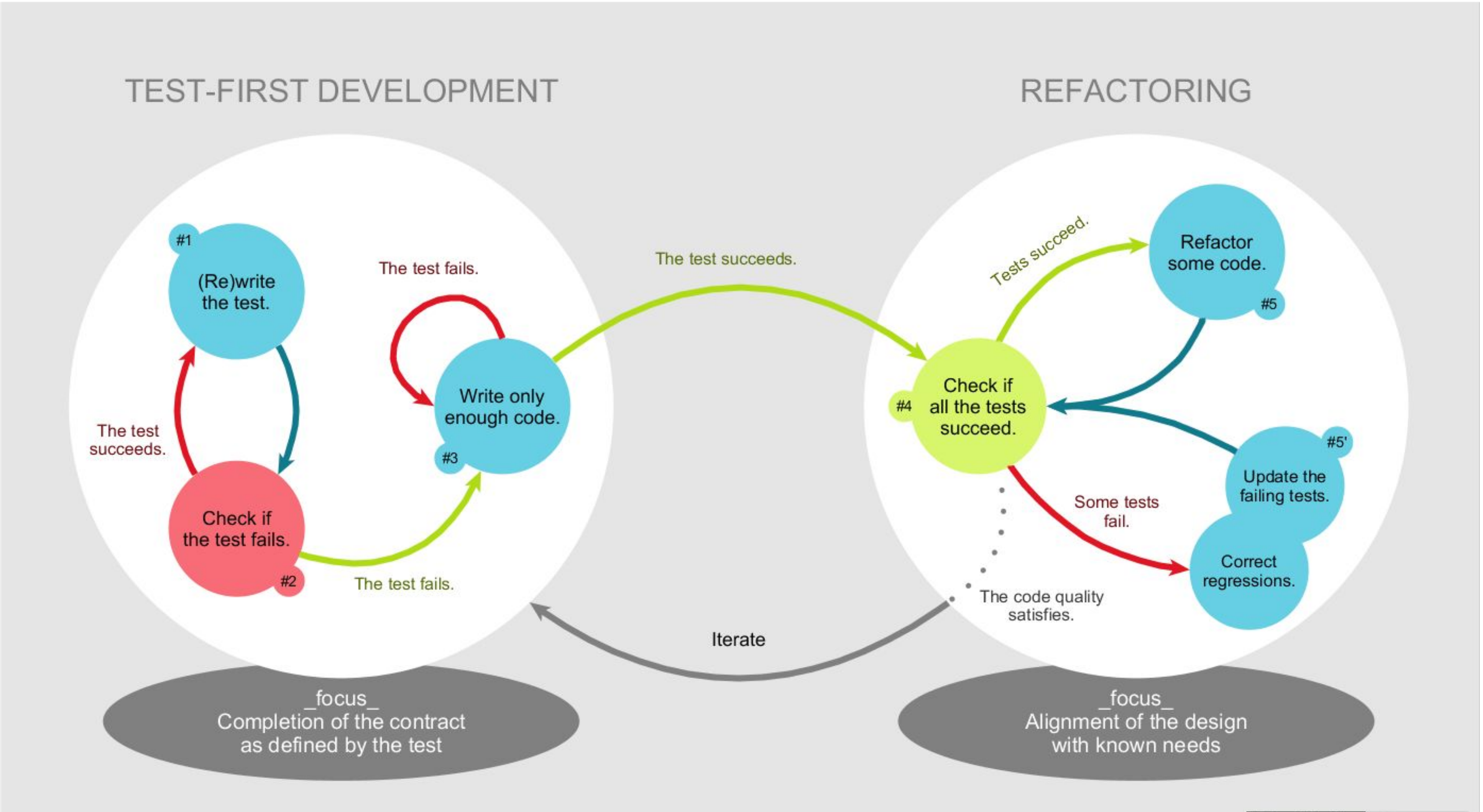
- For your co-workers and your future self

It may be even the starting point for developing your code

- First the test, then the implementation!

  (a.k.a. *Test Driven Development*)

Interlude - Testing II



https://en.wikipedia.org/wiki/Test-driven_development

Interlude - Testing III

Testing can happen at various levels

- Component-wise
- Full integration tests
- By a single developer or centrally

It can test various things

- The behaviour of components (the interface)
- The behaviour of code snippets (implementation specific)

A good test is easy to execute, not written by the actual developer, and run ‘continuously’

Organizing and sharing the work

We’ve been talking about how to integrate your work

But how do you coordinate what to actually work on?

Issue trackers are essential for distributed development to track

- Bugs
- Feature requests
- Todo items

Issue trackers however only record discussions and developments!

Decisions have to be made nevertheless!

Who’s working on what?

Assign tasks upfront

- It is clear who works on what
- No task will be forgotten or ignored

Prioritize tasks and do ***‘task stealing***’

- Pick whatever important task is next in the queue - based on your expertise
- Fixes for imbalanced workloads in volunteer projects
- Requires to ‘give up’ code ownership

In reality always a combination of the two

WIP - work in progress

What if there is a very big chunk of work in the pipeline?

Pull-requests can be marked as WIP (work in progress)

- Not yet ready/finished code
- Tests are expected to fail still

Very good approach to share design ideas and prototypes

share early, share often !

Dealing with power users

Sometimes power users come up with great new ideas and even implementations

The pull-request workflow allows them to easily contribute their developments

- Technical integration process and code review are identical for developers and power users

However, whether a proposed feature is desired or not, is an additional part of the review process

Make sure the requirements for the review are well documented

Everything else leads to frustration!

Licensing

Every software has to go along with a license

- Once multiple people are involved there is almost zero chance in picking a license afterwards
- With power-users/external contributors entering, things are even worse!

So, pick a license early on. And be sure you understand what you want:

- Bring the idea of open-source forward and ‘force’ everybody to do the same (GPL)
- Provide your software to others and let them decide what they want to do with it (MIT, Apache)

https://choosealicense.com/

The HEP community largely ignored that topic and has trouble fixings things now!

Summary

- Collaborative Software Development requires some infrastructure and procedures in place
- As other fields, HEP was only partially successful with custom tools and procedures
- The current approach of pull-requests is a good fit for most of the projects
  - If combined with proper testing
- It helps
  - Developers by giving them a free, yet stable environment
  - Power users by treating them as regular developers
  - Release managers by keeping the repo clean and functional