Benefits of Open Source Practices

Michael K. Johnson
Developer
Red Hat, Inc.

http://people.redhat.com/johnsonm/

Benefits of Open Source Practices

I am a member of the Open Source community, and not the HEP community. My knowledge of challenges facing the HEP community are therefore second-hand.

I discovered a great deal of disagreement about precisely which challenges the HEP software community is facing.

Not all of this talk will apply to everyone here.
Use what is useful to you, ignore what is not.

Benefits of Open Source Practices

Overview

Current Problems/Challenges

Suggested Improvements

Observations

Costs/Benefits Summary

Current Problems/Challenges

Permanent forking (divergent development) is common

- Multiple maintainers duplicate work
- Improvements to one stream do not benefit others
- Frustrating

Current Problems/Challenges

Limited "productization"

- Wasted time maintaining system code
  - That could benefit many
  - That others would help maintain if they had access
- Wasted effort
  - Trying to help others install non-productized software

Current Problems/Challenges

Design problems: Not designed to facilitate:

- Outside contributions
  - Example: Mozilla when first released
- Transfer of maintenance
  - New graduate students have long learning curve

Current Problems/Challenges

Limited communication

- Developers are in closely-knit groups that are hard to join
- Competition inhibits communication and collaboration

Current Problems/Challenges

Perception of false implications:
- Open Software
  - Implies bazaar development model
    - Implies lack of design and thus limited maintainability

Suggested Improvements

Overview

Societal

Technical

Trust is key

- These suggestions imply a community that builds trust
- Knowing who you trust implies also knowing who you do not trust

Suggested Improvements
Societal

Cooperation with competitors is possible

- Consider using or creating outside organization to help
  - Unix vendors
    - Usenix
    - Open Group
    - IETF
  - Linux distribution builders
    - Linux International
    - Linux Standard Base (LSB)
    - XFree86

Suggested Improvements
Societal

Cooperation with competitors is possible

- Example: Red Hat Linux
  - Competitors use it as base of their distributions
    - Red Hat takes advantage of this
      - Red Hat Linux is the "trusted base"
        - Recognition and (potential) market share
      - Allows us to provide more interface stability
      - We can re-include their changes and benefit from their experiments
      - We can take advantage of our competitors’ mistakes
    - Our competitors take advantage of this
      - Try some modifications before we do
      - When they make a good modification, they get reputation and market
    - Users take advantage of this
      - Competition enhances both our products and our competitors’
      - Choice between distribution providers with different priorities

Suggested Improvements

Societal

Cooperation with competitors is possible

- Example: Red Hat Linux
  - We still have proprietary processes
    - We do not publish schedules ahead of time (avoiding vaporware)
    - We do internal development when appropriate
      - We don’t always publish code the instant we write it
      - We don’t distribute binaries without source
        - Except where legally constrained
    - We prefer to develop in the open
      - It is our default policy
      - Otherwise we are just one more competitor for Microsoft to crush

Suggested Improvements

Societal

Cooperation with competitors is possible

- Make a default policy of cooperating
  - Choose secrecy only because of well-developed arguments
  - Ignore vague fears, life is too short...
  - Choose secrecy for modules rather than projects when possible
    - Has technical benefits as well (covered later)
- CERN has an explicit policy allowing GPL distribution
- US labs currently have no explicit policy

Suggested Improvements
Societal

Understand forking's large long-term costs
- The ability to fork gives freedom from fear of coercion
- Taking advantage of that freedom has large costs

Maintainer's judgment is more important than
- Patching skill
- Time available to patch

Maintainer's job is primarily to reject patches
- Applying patches is a smaller secondary function
- Accepting patches does not imply applying them

Maintainer may apply patches
- As-is
- With modifications
- By entirely re-writing

Listen to Jeremy Allison's talk next for more detail

Suggested Improvements
Societal

Maintain forks as patches, not as modified source

- Case study: RPM packages as maintained by Red Hat
  - An RPM source package normally contains
    - The original source package
    - A set of patches to that source
    - Shell script to patch and build
  - RPM is a good tool for maintaining slightly forked versions
- Case study: procps raw forks nearly impossible to merge
  - A maintainer ignored feedback for too long
    - Other developers created several new versions
  - Merging was more difficult than would have been worthwhile
    - Nice features never made it into main version

Suggested Improvements

Societal

Build expectation that changes are sent to maintainer

- As GNU-style unified diffs (diff -u)
  - Easiest diff format to apply by hand
- Using the same coding style as the modified code
- Including changes to documentation if applicable
- Separate functionality should be in separate patches
- Have an environment in which the changes can be discussed

Suggested Improvements

Societal

When you are not working with the "current development version"

- Try to remake your patches against the current development version
  - Increases the probability that your patch will be accepted
  - Large reduction in future upgrade costs for small investment now

Suggested Improvements

Societal

Use IT expertise

- Request software engineers from IT departments as a resource
- Consider these engineers to be collaborators
- Bring them in at the beginning of the process

Suggested Improvements

Societal

Use IT expertise

- Software engineers could assist with
  - Formulating requirements
  - Architecture, particularly modularization
  - Toolmaking
  - Productization and release management
  - General software engineering practices
- Software engineers could reduce other demands on IT
  - Reduced ongoing maintenance costs
  - More efficient software
  - Unified underlying architectures
  - More potential resource sharing
    - Enable reuse of both hardware and software

Suggested Improvements
Societal

Getting started right

- Consider extra startup resources as a bootstrap cost
  - The first "deliverable" provides "plausible promise"
- Make sure everyone knows who the technical leader is
  - Try to know what the non-leaders do
    - Personal web pages can help with this in large projects

Suggested Improvements
Societal

Release early and often

- Clearly separate development and production releases by version number
  - Make sure version numbers are unique
- Usually critical for maintainer’s ability to accept incoming patches
- Public CVS archives
  - Generally are no substitute for frequent releases
    - Except for some very small development/user communities
    - But are better than nothing (and good for other things)

Suggested Improvements
Societal

Getting started right

- Communicate requirements documentation expectations
  - Example: Mozilla rule
    - If the job requires more than a day of work,
      - Describe it to the developer newsgroup before starting
- Express coding standards explicitly for each project
  - But not verbosely
  - Borrow coding standards documents from successful projects
  - Following coding standards
    - Will speed up maintenance and coding
    - Will make it easier for "casual" users to contribute small fixes
      - Small fixes are often the ones that the authors never get around to
- See Bob Jones’ talk later for a good example of the process

Suggested Improvements
Societal

Getting started right

- Use networked CVS or other SHARED version control system
  - Need more than maintainer having private CVS archive
  - Even read-only access helps
    - Third party patches can track maintainer’s version
  - Avoid conflicts without too much overhead
    - Overhead in making changes has an inordinately strong slowing effect
  - Read-write access
    - Requires more trust
    - Explicitly specifies trust relationships
- See Bob Jones’ talk later for a working example

Suggested Improvements

Societal

Getting started right

- Encourage maintainability
  - The more maintainable it is, the better outside contributions will be
  - My latest favorite:
    - Read The Practice of Programming, by Kernighan and Pike
      - See http://cm.bell-labs.com/cm/cs/tpop/
  - This all is important with any development model
    - Just gets more important when you have more contributors

Suggested Improvements
Societal

Have an explicit productization process

- Production releases should be fully productized
- Development releases usually need less productization

Suggested Improvements
Societal

Have an explicit productization process

- Productization is NOT just packaging
- Productization includes
  - Installability and uninstallability
  - System integration
  - Customization potential
  - Testing
    - Build process
    - Built product
    - Integration
    - Consider distributing test cases, not just running them
  - Analyze fix distribution requirements
    - High distribution costs? Large formal testing requirements
    - Low distribution costs? Smaller formal testing requirements
    - Releasing early and often lowers distribution costs

Suggested Improvements
Societal

Have an explicit productization process

- Lacking productization resources?
  - Call every release a development release
  - Productization resources may show up later, perhaps in another group
- Productization has
  - High cost
  - Hidden, hard-to-measure, "negative" benefit:
    - Fewer bugs experienced

Suggested Improvements
Societal

- Encourage lurking, watching each others' projects
  - Learn from each others' successes and failures
  - Encourages reuse
    - Software engineers and physicists will see different reuse potential
- Publish your work
  - After you have something that works -- plausible promise
  - After you no longer have immediate proprietary interest
  - Publish more widely than you think makes sense
- When projects languish, pass the baton
  - Or at least publish the fact that the project is stagnant
  - Someone may pick up the baton later

Suggested Improvements

Societal

Consider publishing products (not programs) as a PUBLICATION

- Peer review still essential
  - Set up organization to provide peer review of software publications?
  - Poorly written code should disqualify equally as badly written language
  - Establish formal conventions for citation of source code projects
- Publish source code with papers
  - In journals
  - In conference proceedings
  - Particularly when reproducability and verifiability relies on the source code

Suggested Improvements

Technical

- Consider common Open Source standards
  - automake/autoconf
  - Existing coding standards (GNU, Linux -- just have one)
- Build on Open Source tools
  - Don’t reinvent the wheel, and use free wheels
  - Lowers the barriers to entry for new collaborators
  - Gtk+, GNOME, Glade, Qt, KDE, libxml, gsl, Mesa, RPM, etc.

Suggested Improvements

Technical

Use modular software techniques

- Not just multiple C++ files...
- Shared libraries, run-time loaded libraries, separate programs
- Strong separation forces better design
- Can help cleanly separate proprietary from public code
  - Some advantages of Open Source without giving secret research away
- Improves generalization to fit more institutional procedures

Suggested Improvements

Technical

Use modular software techniques

- Intrinsic benefits
  - Interface stability
  - Debugability
  - Maintainability

Suggested Improvements

Technical

Use modular software techniques

- Case studies:
  - Unix text filters
    - Extreme modularization
    - Historical success
  - GIMP plugins
    - Very high modularization
    - Contributed strongly to meteoric success
    - Simplicity encouraged third-party participation
  - Linux kernel loadable modules
    - Easy to keep personal work private
    - Harder to distribute binary-only
  - XFree86 4.0
    - Developers prefer new design
    - Much delayed by need for updates of old source base

Suggested Improvements

Technical

Build collaborative structures that encourage outside participation

- Technical structures with a primarily societal purpose
- Mailing lists
- Web: lxr, bonsai, mailing list archives, Zope/Squishdot, mod_virgule, wiki
- Usenet
- CVS
- IRC
- Find a set that matches the participants’ needs
- Archived discussion helps new folks get up to speed

Observations

Open Source is no replacement for

- Maintenance
  - Maintenance changes form, but needs to happen
- Management
  - Strong leadership is essential for Open Source projects
    - Leadership must be based on respect, not seniority
    - Leadership must be technically sound
    - Internal schedules and other needs may influence, but will not control, outside contributors
- Manpower
  - No silver bullet

Open Source gives flexibility to change the maintenance, management, and manpower relationships

- Can sometimes give new life to dead projects

Open Source does NOT imply giving up ownership or control

- That is one more tradeoff

Observations

Projects work well when they have

- Well-defined goals
- Clearly-defined leadership
- Consistent code base
- Participants who respect each other
- Participants with varying talents

Costs/Benefits Summary

Time

- Managing all these added processes takes time
- Maintenance help from more collaborators saves time
- Extra testing help from users finds bugs quicker
  - Less likely later to experience result-invalidating bugs
- More reasonable growth in maintenance burden
- More efficient use of support staff
  - Costs less per task
  - Less frustrating and more satisfying for support staff
- Streamline deployment and acceptance
- Internal expansion eased by external testing and use
- Much more effective peer review from "lots of eyes"
  - Even GEANT 3 has had random code readers fixing bugs

Costs/Benefits Summary

Money

- Real productization requires test hardware
- Resources to support collaborative spaces
- More resources can be shared
  - Internal to your organization
  - With external organizations

Summary

Most projects can benefit from taking advantage of some (more) of these Open Source common practices. Most of those projects can benefit from being entirely or partially Open Source. The costs to take advantage of Open Source practices can be high, but the benefits are also considerable and for many projects outweigh the costs.

Thanks to

Rene Brun, Philippe Defert, Bob Jones, Arash Khodabandeh, Juergen Knobloch, German Melia, Eric McIntosh, Les Robertson, Ben Segal, Jamie Shiers, and Jes Sorenson at CERN for suggesting this talk and helping me understand some of the challenges being experienced by various parts of the HEP community.

Q&A