Software Engineering

Lecture 13 – Open-Source Software (OSS)

© 2015-19 Dr. Florian Echtler
Bauhaus-Universität Weimar
<florian.echtler@uni-weimar.de>

Open Source Software: Topics

- Philosophy: The Cathedral and the Bazaar
- Case study: Linux kernel
- Licensing issues

Cathedral

Image source (PD): https://upload.wikimedia.org/wikipedia/commons/7/70/Rheinpanorama_1856_detail_Dom.jpg

Bazaar

Image source (CC): https://www.flickr.com/photos/pamnani/14711280076/

The Cathedral and the Bazaar

- Two fundamentally different philosophies
- Cathedral:
  - Overarching “masterplan”, spanning generations
  - Few brilliant architects/master builders
- Bazaar:
  - Mostly unstructured, little central authority
  - Everyone does what they want
- Both strategies “work” → create results

Software Analogies

- Cathedral “traditional” software processes
  - Plan-driven, highly structured
  - Hierarchy: (senior) developer, (senior) architect, …
  - Perhaps even open-source, but only for releases
- Bazaar open-source development
  - Multiple (incompatible) forks in parallel
  - Source code *always* available, even broken versions
- Somewhere in between: agile methods

Software Analogies (2)

- Analogy originally by Eric S. Raymond
- Until mid-1990s: “Cathedral-style” develop-ment considered a requirement for large projects (even by open-source developers)
- Overturned by Linux (first release 1991)
- “free-for-all” development model

Lessons for OSS: general

Source: http://www.catb.org/esr/writings/cathedral-bazaar/

1. Every good work of software starts by scratching a developer's personal itch.
2. Good programmers know what to write. Great ones know what to rewrite (and reuse).
3. “Plan to throw one away; you will, anyhow.” (F. Brooks, *The Mythical Man-Month*, ch. 11)
5. When you lose interest in your project, your last duty to it is to hand it off to a competent successor.
7. Release early. Release often. And listen to your customers.

Lessons for OSS: users & developers

Source: http://www.catb.org/esr/writings/cathedral-bazaar/

6. Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging.

8. Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.

Lessons for OSS: users & developers

Source: http://www.catb.org/esr/writings/cathedral-bazaar/

10. If you treat your beta-testers as if they're your most valuable resource, they will respond by becoming your most valuable resource.

19. Provided the development coordinator has a communications medium at least as good as the Internet, and knows how to lead without coercion, many heads are inevitably better than one.

Lessons for OSS: ideas & solutions

Source: http://www.catb.org/esr/writings/cathedral-bazaar/

11. The next best thing to having good ideas is recognizing good ideas from your users. Sometimes the latter is better.
12. Often, the most striking and innovative solutions come from realizing that your concept of the problem was wrong.
13. Perfection (in design) is achieved not when there is nothing more to add, but rather when there is nothing more to take away. (Attributed to *Antoine de Saint-Exupéry*)

Lessons for OSS: miscellaneous

Source: http://www.catb.org/esr/writings/cathedral-bazaar/

14. Any tool should be useful in the expected way, but a truly great tool lends itself to uses you never expected.
15. When writing gateway software of any kind, take pains to disturb the data stream as little as possible – and never throw away information unless the recipient forces you to!
17. A security system is only as secure as its secret. Beware of pseudo-secrets.

Open-Source Project Management

- Most OSS projects have 1 – 3 core developers, often also called *maintainers*
  - Manage website/mailinglist/forum etc.
  - Have commit access to main repository
  - Decide which *patches* and *pull requests* to accept
  - Can be volunteers, but also paid for by company
- Other developers (also called *contributors*)
  - Create personal *forks*
  - Submit *patches* or *pull requests*

The Free Software Definition

Source: https://www.gnu.org/philosophy/free-sw.html

A program is free software if the program's users have the four essential freedoms:

- To run the program as you wish, for any purpose.
- To study how the program works, and change it so it does your computing as you wish. (*)
- To redistribute copies so you can help your neighbor.
- To distribute copies of your modified versions to others. By doing this you can give the whole community a chance to benefit from your changes. (*)

(*) Access to the source code is a precondition for this.

Copyright & licenses

- Disclaimer: when in doubt, ask a lawyer.
- Copyright:
  - held by the author of any creative work (including source code)
  - expires ~ 70 years after death of author (depending on country)
- Gives exclusive rights to copy, sell, modify, display, *license* the creative work
- Author gives licenses to others to grant them part of the exclusive rights

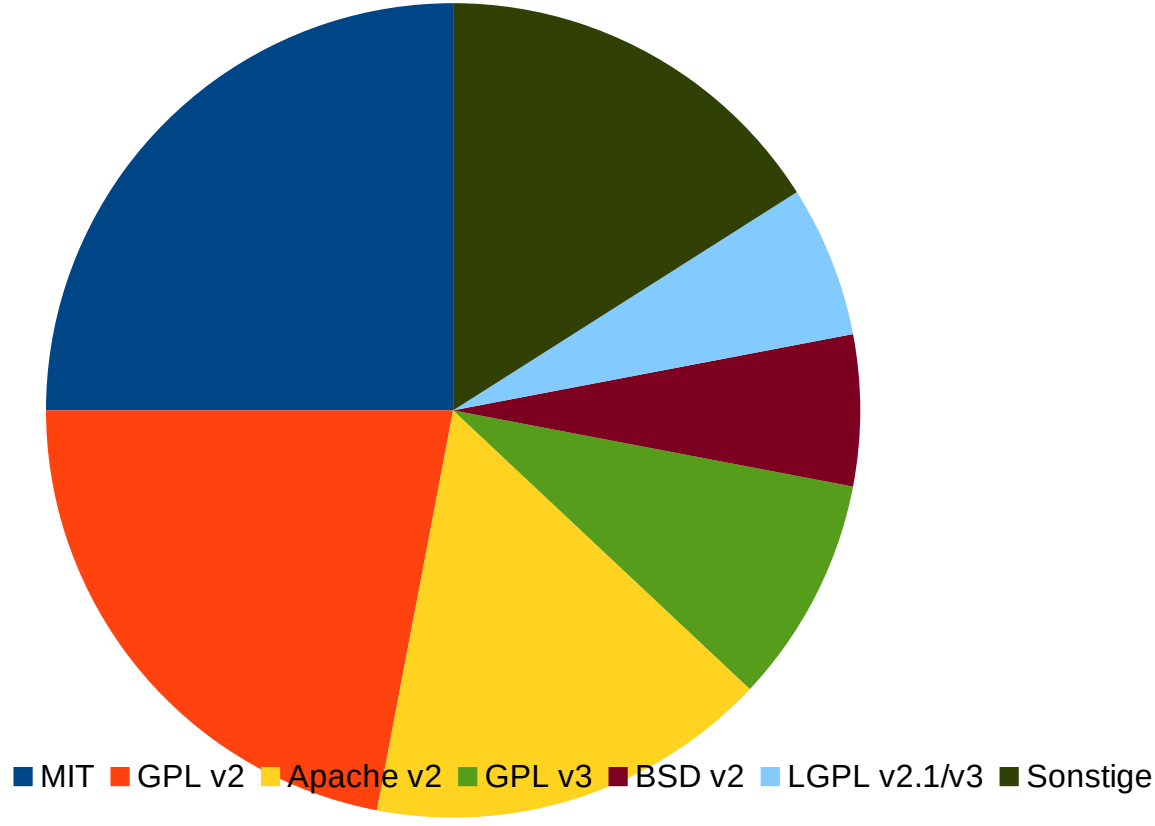
Open source licenses (1)

- ***Open source software != do what you like***
  - Not “free-as-in-beer” …
  - … but “free-as-in-speech”.
- Almost all OSS has specific *licensing terms*
- No monetary cost, but conditions on …
  - Modification
  - Redistribution
  - Access to source code
  - Usage (sometimes)

Open source licenses (2)

Source (FU): https://www.blackducksoftware.com/resources/data/top-20-open-source-licenses

- Most popular licenses
  - MIT/Apache/BSD (~47%)
  - GPL v2/v3 (~31%)
  - LGPL v2.1/v3 (~6%)



Open source licenses: comparison

Source (CC): https://en.wikipedia.org/wiki/Comparison_of_free_and_open-source_software_licenses

License	Author	Link with code using different license	Release changes with different license
GPL	Richard M. Stallman, Free Software Foundation	no	no
MIT (X11)	MIT	yes	yes
Apache	Apache Software Foundation	yes	yes
BSD	University of California	yes	yes
LGPL	Richard M. Stallman, Free Software Foundation	yes	no

Open source licenses: comparison

- GPL/LGPL (GNU General Public License)
  - Modifications must be released under same license
  - “Copyleft” licenses (also called “viral”)
- MIT/Apache/BSD
  - Derivatives can change the license
  - “Permissive” licenses, more commercial-friendly
- Public domain
  - “Revocation” of copyright → Do what you want.
  - Often used for small code snippets etc.

GPL, LGPL, v2, v3, …?

Source (CC): https://www.gnu.org/licenses/quick-guide-gplv3.html

- GPL v2 vs. v3
  - Adds patent-licensing clause
  - Adds provisions against Digital Restrictions Management (DRM)
  - No requirement to disclose source code running only on server
    → compare to Affero GPL (AGPL)
- GPL vs. LGPL (*Lesser* GPL)
  - LGPL allows linking with code using other licenses
  - Often used for libraries to enable widespread use

Open source: earning money?

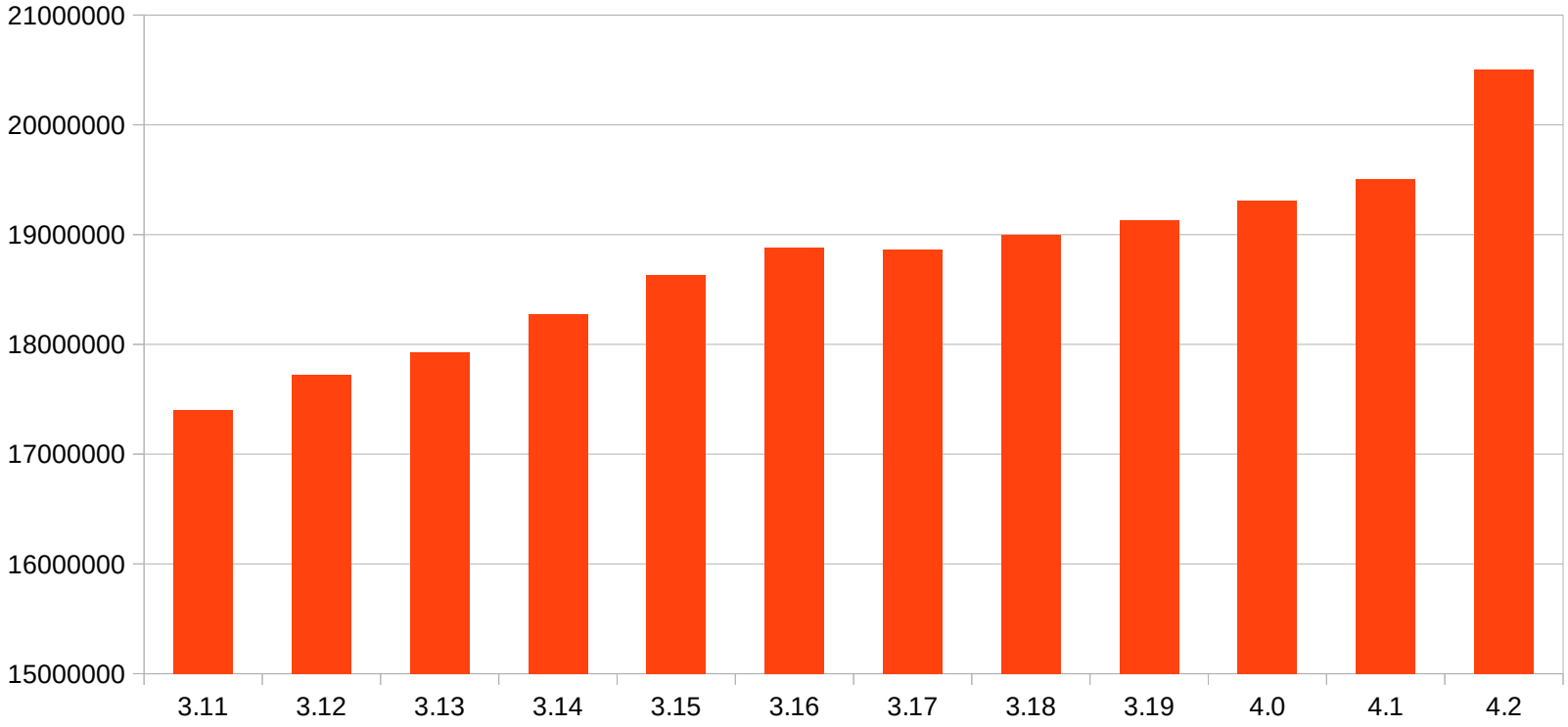
- Possible: see RedHat, Ubuntu, MySQL, Qt, …
- Usually done through *dual-licensing*
  - Public license: often GPL, to prevent competition
  - Paid-for license: e.g. extra features, support, …
- Potential issues:
  - Re-licensing of public contributions?
  - Relations to external OSS developers?

Case study: Linux

- Operating system kernel + hardware drivers
- Developed since 1991 by Linus Torvalds
- Originally written at University of Helsinki
- Replacement for MINIX (based on Unix)
  → long history going back to 1969 at AT&T
- Probably the largest open-source project: estimate for v4.3 = ~20 million lines of C code
- Modular architecture → only small part of codebase running on one given system

Linux: development statistics

Source (CC): https://github.com/gregkh/kernel-history/blob/master/kernel_stats.ods



Linux: development process (1)

- Mostly via mailing lists on www.kernel.org
- Roughly one per subsystem: input, network, video, graphics, audio, USB, … (~ 100 in total)
- 1-3 maintainers per subsystem
- Patches are sent to mailing list, discussed, re-submitted, accepted/rejected
- Strict formatting requirements
- Patch processing automated by scripts

Linux: development process (2)

```
#!/bin/bash
# script for patch submission
git format-patch -s origin/master
vi *.patch # add notes here
scripts/checkpatch.pl *.patch
git send-email --to=fill@me.in --cc=fill@me.in
*.patch
```

Linux: development process (3)

- Accepted patches merged into subsystem-specific development repository
- Subsystem maintainers generate pull requests for master repository
- Master repo: still managed (mostly) by Linus Torvalds (employed by Linux Foundation)
- New releases every ~ 2 months
- ~ 5 -rc (release candidate) versions in between

Questions/Comments?