Logiciel Libre
TP 1 — Project Presentation

Stefano Zacchiroli
zack@pps.univ-paris-diderot.fr

Laboratoire IRIF, Université Paris Diderot

2016–2017

Evaluation — reminder

1. exam
2. TD: +0/+1/+2 bonus on the exam grade
   - exercises
   - short essays on selected topics, including external speaker interventions
   - to be submitted via DidEL
3. project (mandatory, not CC)

Final note

- 1e session: 50% exam + 50% project
- 2e session: 50% exam + 50% project

Project — principle

Relevant code contribution
to a relevant, existing, Free Software project.

Project — principle

Relevant code contribution
to a relevant, existing, Free Software project.

- . . . code contribution

Project — principle

Relevant code contribution
to a relevant, existing, Free Software project.

- relevant. . .
- . . . code contribution

Project — principle

Relevant code contribution
to a relevant, existing, Free Software project.

- relevant. . .
- . . . code contribution

- . . . existing. . .

Project — principle

Relevant code contribution
to a relevant, existing, Free Software project.

- relevant...
- ...code contribution

- to a relevant...
- ...existing...

Project — principle

Relevant code contribution
to a relevant, existing, Free Software project.

- relevant. . .
- . . . code contribution

- to a relevant. . .
- . . . existing. . .
- . . . Free Software project

Project — meta

- work individually or in pairs (*binômes*)
- no exceptions granted for groups > 2

Project — step-by-step guide

1. find a project you're excited about
2. learn the basics of the project
3. find an outstanding issues you'd like to fix
4. agree with me on a task
5. get the fix done locally
6. push your changes upstream
7. write a report

Step 1 — find a project you're excited about

- *any* Free Software project you like
  - must be licensed under a FOSS license, i.e., a license that is both OSI[1] and FSF[2] approved

1. http://opensource.org/licenses
2. http://www.gnu.org/licenses/license-list.en.html

Step 1 — find a project you're excited about

- *any* Free Software project you like
  - must be licensed under a FOSS license, i.e., a license that is both OSI[1] and FSF[2] approved
- try to find a match between your skills and the technologies used by the project (prog. language, framework, tools, etc.)

1. `http://opensource.org/licenses`
2. `http://www.gnu.org/licenses/license-list.en.html`

Step 1 — find a project you're excited about

- *any* Free Software project you like
  - must be licensed under a FOSS license, i.e., a license that is both OSI[1] and FSF[2] approved
- try to find a match between your skills and the technologies used by the project (prog. language, framework, tools, etc.)
- tip (*non* mandatory): choose a project that maintains a list of "easy hacks", "bugs for newcomers", etc.
- I've collected a list at (link also on the course page) `https://openhatch.org/wiki/Easy_bugs_for_newcomers`

Example

Debian, Django, Fedora, GNOME, KDE, LibreOffice, Linux Kernel, Mediawiki, Mozilla, Mozilla, OpenMRS, OpenOffice, OpenStack, Python, Snowdrift, Ubuntu, VLC, ...

1. `http://opensource.org/licenses`
2. `http://www.gnu.org/licenses/license-list.en.html`

Step 2 — learn the basics of the project

- how to obtain the code (e.g., VCS) of the current development version of the project
- how to build (dependencies, build instruction, README, etc.)
- how to install on your machine
- how to run on your machine

Step 2 — learn the basics of the project

- how to obtain the code (e.g., VCS) of the current development version of the project
- how to build (dependencies, build instruction, README, etc.)
- how to install on your machine
- how to run on your machine

- look for introductory development documentation

Step 2 — learn the basics of the project

- how to obtain the code (e.g., VCS) of the current development version of the project
- how to build (dependencies, build instruction, README, etc.)
- how to install on your machine
- how to run on your machine

- look for introductory development documentation

- try to make a tiny teeny modification
  - e.g., change a label in a UI element, print a debug message at some point, . . .
- compile, install, run ← verify that your change is in

Step 3 — find an outstanding issues you'd like to fix

- Bug Tracking System of the project
- list of "easy hacks" maintained by the project (see before)
- feature requests by users (tip: only consider those that have been acknowledged by developers)
- bug you find (and report) yourself
- "roadmap" documents (more difficult)

- try to reproduce bugs: do they affect your build of the development version too?

- choose a task

Step 4 — agree with me on a task

- without preliminary agreement with me, the chosen task does not constitute a valid project

Step 5 — get the fix done locally

- fix the bug / implement the missing feature

this is the easiest part :-)

(but really, it is)

Step 6 — push your changes upstream

- produce a patch / pull request / whatever is customary for the project you’ve chosen
- inform the developers of your work, attaching your changes
- convince them that your code is good and worth being integrated (“upstream”) as part of the project code
- this might require several iterations during which you improve your code
- remember: upstream developers have the last word, it’s your responsibility to convince them

Step 6 — push your changes upstream

- produce a patch / pull request / whatever is customary for the project you've chosen
- inform the developers of your work, attaching your changes
- convince them that your code is good and worth being integrated ("upstream") as part of the project code
- this might require several iterations during which you improve your code
- remember: upstream developers have the last word, it's your responsibility to convince them
- if you hit a wall, go back to steps 1 or 3 and pick another project/task
  - hence, better start well in advance!

Interlude — exclusion criteria

- do not use the fact this is a university assignment as leverage to convince upstream developers
- it will not work
  - in fact: it will likely work *against* you
- and if I find evidence of it, e.g.:

  ~~*"pour le cours de Logiciel Libre de M. Zacchiroli je dois contribuer un patch à votre projet, pourriez vous m'aider s'il vous plaît?"*~~

  your project grade will be 0

- your *code* should be convincing, not your status as a student

Step 7 — write a report

Length: 10 pages max.

The structure of the report is up to you, but it must contain at least the following information:

- background: chosen project, reasons for that choice
- technicalities: short technical description of the chosen project (programming lang., frameworks, tools, etc.)
- background: chosen task, reasons for that choice
- technical description of your work: what have you done? how?
- references to *public* evidence of your work: commits, patches sent to Bug Tracking Systems, mailing list threads, etc.
- evidence of the fact your work has been integrated upstream
- discussion: difficulties encountered, feedback, etc.

Supervision

- there will be dedicated TP sessions to work on the project
- I'm available throughout the full process (both during dedicated TPs and elsewhere) to give you feedback, hints, and advice for your specific project

Q&A

?