

# Sustainability in F/OSS: developers as a non-renewable resource

Graham Percival

<http://percival-music.ca>

Rencontres Mondiales du Logiciel Libre 2010  
Bordeaux, France

Friday, 9 July, 2010



This work is licensed under the *Creative Commons Attribution-ShareAlike 3.0 License*.

## 1 Current development is not sustainable

- Core developers do most of the work
- Losing core developers is bad
- Projects *will* lose core developers

## 2 Keeping developers

- Incentives
- Disincentives

## 3 Preparing for developer loss

- Survival of a species
- Training the next generation: harder than it sounds
- Successes and failures from GNU LilyPond
- Filtering out offers of help
- Dealing with new developers

## 1 Current development is not sustainable

- Core developers do most of the work
- Losing core developers is bad
- Projects *will* lose core developers

## 2 Keeping developers

- Incentives
- Disincentives

## 3 Preparing for developer loss

- Survival of a species
- Training the next generation: harder than it sounds
- Successes and failures from GNU LilyPond
- Filtering out offers of help
- Dealing with new developers

## Do F/OSS projects share the workload?

- Popular view is that F/OSS has lots of developers.
  - e.g., “Given enough eyeballs, all bugs are shallow”
- Actually, workload generally follows Zipf’s law.  
(frequency is inversely proportional to rank)
  - Healy and Schussman, 2003. “The Ecology of Open-Source Software Development”
    - ▶ Data from over 45,000 sourceforge projects.
    - ▶ # of developers, commits / developer, # of emails, etc.
    - ▶ *“The distribution of projects on a range of activity measures is spectacularly skewed, with only a relatively tiny number of projects showing evidence of the strong collaborative activity which is supposed to characterize oss.”* [from paper abstract]
  - Similar results from other studies.

## Is “Number of Commits” a good metric?


- Number of commits is a *vague* measure of project work.
- Problems:
  - Not all commits are equal (new feature vs. 1-line typo fix).
  - Code vs. documentation vs. build vs. translations?
  - Some people break work into more pieces than others.
- Why use them?
  - Easy to measure.
  - Easy to understand.
  - The exact workload distribution doesn't matter for this talk!
- Not a *good* metric, but it's an *acceptable* metric.

## Case study: GNU LilyPond (sheet music typesetter)

- Compiles text files into beautiful printable scores.
- Simple example:

```
{  
  \time 2/4  
  \clef bass  
  c4 c g g a a g2  
}
```

Commands start with \  
Letters are notes  
Numbers are durations



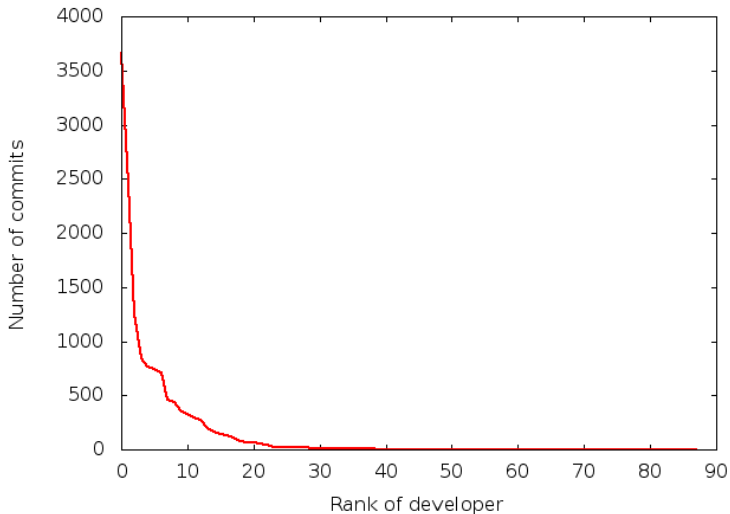
- Computational aesthetics is hard.  
(details not important – this is not a talk about music)

## LilyPond Development

- Code size:
  - $\approx 100,000$  lines of C
  - $\approx 30,000$  lines of scheme (a dialect of lisp)
  - $\approx 25,000$  lines of python
  - $\approx 18,000$  lines of metafont
  - $\approx 450,000$  lines of documentation source files (including translations)
- Began in 1996 by 2 Dutch undergraduates.
- 92 authors in 14 years, 46 in the past 6 months.

## Commits vs. developer rank, last 5 years. (almost Zipf's law)

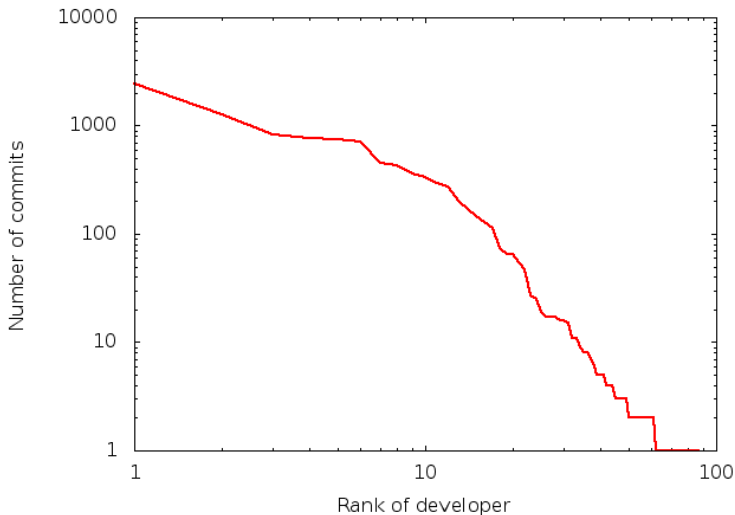
Git commits to LilyPond, 2005 - 2010





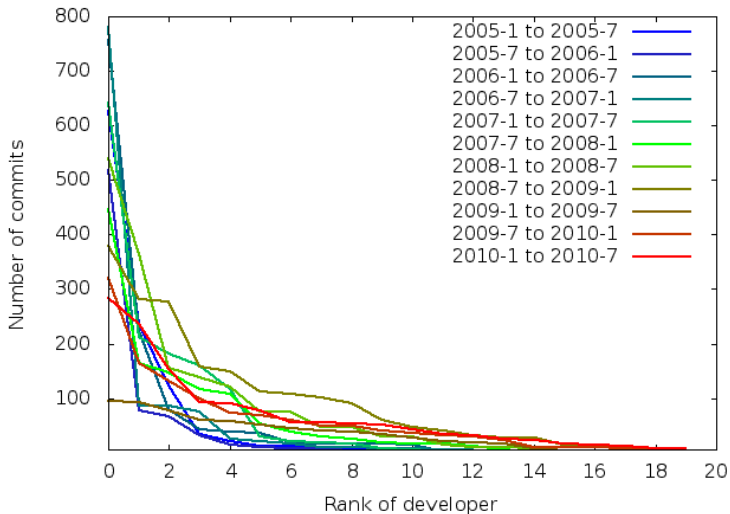
Same graph, log-log scale. (Zipf's law would be a straight line)

Git commits to LilyPond, 2005 - 2010



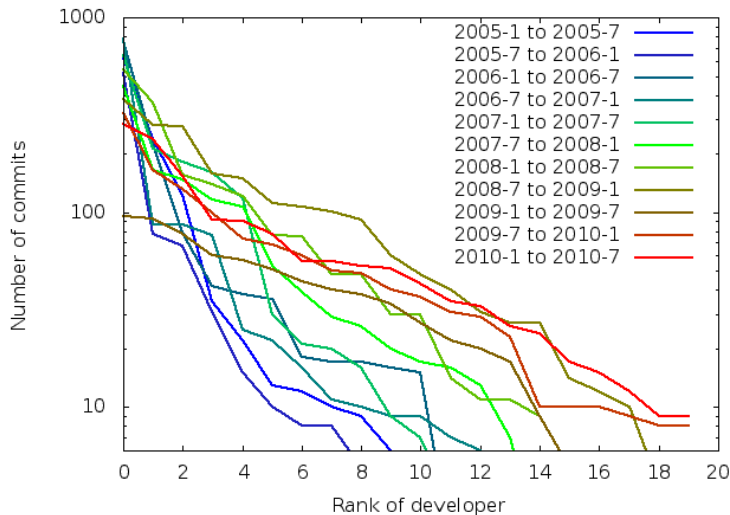
## Split into 6-month intervals.

Git commits to LilyPond, 2005 - 2010 in 6 month intervals



## Split into 6-month intervals, log y.

Git commits to LilyPond, 2005 - 2010 in 6 month intervals



## Effect of losing core developers (selected data)

Top 4 developers, selected 6-month periods:

Date	Commits (name)			
05-1	<b>626</b> (Han-Wen)	<b>237</b> (Jan)	<b>123</b> (Graham)	<b>35</b> (Werner)
06-7	<b>780</b> (Han-Wen)	<b>87</b> (Jan)	<b>87</b> (Joe)	<b>76</b> (Graham)
07-7	<b>446</b> (Graham)	<b>164</b> (John)	<b>148</b> (Joe)	<b>116</b> (Reinhold)
08-7	<b>379</b> (Reinhold)	<b>281</b> (John)	<b>278</b> (Paco)	<b>158</b> (Neil)
09-1	<b>95</b> (John)	<b>93</b> (Paco)	<b>78</b> (Carl)	<b>60</b> (Joe)
09-7	<b>321</b> (Graham)	<b>165</b> (Patrick)	<b>132</b> (John)	<b>99</b> (Neil)
10-1	<b>284</b> (Graham)	<b>236</b> (Paco)	<b>153</b> (Jan)	<b>92</b> (Patrick)

- 2009-1 to 2009-7, the top three overall developers were away.
  - Core developers can motivate others.
- The drop-off in commits is less abrupt in recent years.
  - Less disruption if somebody leaves.

## Developer Loss – it will happen

- Developers can leave due to project problems...
  - Not enough incentives
  - Too many disincentives
- ... but also for non-project reasons.
  - Graduating from high school / university
  - Career change
  - Getting married or having a baby
  - Passing away
    - ▶ Hopefully after a long life, but sometimes earlier.
- Fix project problems, but we'll all die eventually.
  - Developer loss is unavoidable!

## 1 Current development is not sustainable

- Core developers do most of the work
- Losing core developers is bad
- Projects *will* lose core developers

## 2 Keeping developers

- Incentives
- Disincentives

## 3 Preparing for developer loss

- Survival of a species
- Training the next generation: harder than it sounds
- Successes and failures from GNU LilyPond
- Filtering out offers of help
- Dealing with new developers

# Incentives: Financial

## ■ Money:

- Job / full-time contract.
- Cash / short-term contract – might backfire.
  - ▶ Offer a professor \$25 for 10 hours of work?
  - ▶ Users value new features more than bugfixes.
  - ▶ Why work on bugfixes for free vs. new features for cash?

## ■ Invite them to conferences.

## ■ Send them stuff:

- “Swag”: company-branded t-shirts, USB drives, etc.
- Postcards, special beer from your country, buy them dinner if they visit your city, etc.

## Incentives: (almost) Free

- Send them artistic or “end-user” stuff:
  - Beautiful printed sheet music.
  - Professionally-recorded performance.
  - Printed artwork.
  - Game that uses your library / compiler / etc.
- Give praise / credit / feed ego.
- Make development entertaining:
  - Create friendships.
  - Write funny emails on mailing lists.
  - Make them feel like part of a team.
- Ask them!



## Incentives: Risky

### ■ Guilt trip

- Bad: “You do so much work around here... you have to keep on working or else everything will fall apart!”
- Slightly better: “I can’t handle everything at once, and I really need a break. Patrick, Trevor: could one of you handle bug reports for the next two months?”  
(temporary, end in sight, but still pressures individuals)

### ■ Bargain

- “I’d like to release binaries for Windows, but I can’t do that if I need to keep on writing documentation.”

### ■ Both strategies can backfire.

- Use infrequently.
- Gambling about how much people trust you.

## Getting Rid of Developers

- Insult developers (especially from users).
  - Insults to other developers made me shelve some doc work.
- Demand that a particular bug be fixed.
  - Users saying “you must...” prompted me to leave for 4 months.
- Ignore requests for feedback (from users).
  - Our new website was delayed for about 8 months due to this.
- Ignore requests for freeback (from developers).
  - Code style, patch review, architecture changes, etc.
  - We recently lost one of our top 20 developers due to this, and it's a constant disincentive for other developers.

## 1 Current development is not sustainable

- Core developers do most of the work
- Losing core developers is bad
- Projects *will* lose core developers

## 2 Keeping developers

- Incentives
- Disincentives

## 3 Preparing for developer loss

- Survival of a species
- Training the next generation: harder than it sounds
- Successes and failures from GNU LilyPond
- Filtering out offers of help
- Dealing with new developers

# Survival of a species

- Developers can leave with or without prior notice:
  - Graduation will be known in advance.
  - Career change might be unexpected.
  - Accidental death will never give advanced notice.
- Don't rely on advance warning – prepare now!
- How to prepare for loss of developers?
  - Biological analogy: survival of a species.
  - Train new developers to replace those who will leave.

## What needs to be taught?

- Consider each developer – how can they be replaced?
  - Unique knowledge or access?
    - ▶ Build process, login to web server, specialized code, etc.
  - Unwritten policies?
  - Time-saving tips + experience.
- “Apprentices” are vital.
  - Try to do each task by yourself.
  - Discover what you don’t know and document it.
    - ▶ Oral tradition is not reliable!
  - “Apprentice” could even be another core developer.
    - ▶ Documenting unwritten knowledge is the primary goal.

# When should you have apprentices?

- Definitely too late:
  - Dead developer.
  - Developer who left due to a huge argument.
- Maybe too late:
  - Developer leaves due to career change, baby duties, graduation.
- Too early:
  - Developer is currently an apprentice.
  - Policies / code / procedures are changing drastically.
- Start as soon as possible:
  - Training an apprentice takes a lot of time+effort.
  - Biological analogy: don't wait until old age for a baby!

# Training the next generation: harder than it sounds

- Need the right kind of person to train people – technical knowledge, good at explaining, available time, etc.
- Stages of a new developer:
  - 1 Recruitment.
  - 2 Initial training, explain task(s).
  - 3 Patch review and critique.
  - 4 Independent: produces good patches without help.
- How much mentoring to become independent?
  - Some people send perfect patches without any mentoring.
  - Usually new developers need hours of mentoring.
    - ▶ Some of our most active developers started this way.
    - ▶ Sometimes all this mentoring effort is worthwhile.

## Evaluating offers of help (in retrospect) (1)

- *Net gain to the project* =  $T_{work} - T_{mentoring}$ 
  - $T_{work}$  is the amount of time it would take an existing developer to do the work.
  - $T_{mentoring}$  is the time that developer spent helping a new developer learn how to do that task.
- Example 1: Mike (the mentor) asks for doc-writing help.
  - Avery says he can help. Mike assigns him a 10-minute task.
  - Avery needs to be taught how to use svn and diff, makes typos, etc. Avery spends 2 hours working.
  - Mike spends a total of 60 minutes teaching + correcting.
  - Avery is demoralized and leaves the project.
  - Net gain of  $10 - 60 = -50$  minutes. (omit Avery's time)
  - Project would be better off if Avery had not offered to help. :(



## Evaluating offers of help (in retrospect) (2)

- Example 2: Mike (the mentor) asks for doc-writing help.
  - Billy says he can help. Mike assigns him a 10-minute task.
  - Billy is completely unfamiliar with open-source development, and requires 2 hours of mentoring before finishing the patch.
  - At this point, net gain of  $10 - 120 = -110$  minutes.
  - However, Billy is stubborn, and keeps on working in the project. He finishes another nineteen 10-minute tasks.
  - At this point, net gain of  $20 * 10 - 120 = 80$  minutes.
  - Project benefitted from mentoring Billy.
- Example 3: Carlos offers to help.
  - Would the project benefit if Mike mentored him?
  - Probability of Carlos being a net gain?
  - Any ways of minimizing the risk?

## Data from GNU LilyPond

### ■ LilyPond GDP (Grand Documentation Project):

- 1<sup>st</sup> goal – 12-month project to train new doc editors.
- 2<sup>nd</sup> goal – give unlimited mentoring; is this effective?
- 20 volunteers ( $\approx 5$  were already involved in LilyPond).
- I spent  $\approx 700 - 800$  hours mentoring volunteers, up to 4 hours a day.

### ■ Results:

- Only **1 in 4** volunteers were definitely a net gain.
- Another **1 in 4** were not a significant net gain or loss.
- Overall, GDP was not a significant net gain or loss.
- 6 months later, we had **0 people** working on documentation.
  - ▶ (3-4 people who began as doc editors became strong programming developers – GDP was not a *complete* failure!)

### ■ Conclusion:

- Unlimited mentoring is *not* effective.

## Filtering out offers of help

- Not a nice thought, but important to consider.
- Balance mentoring potential developers (risky) and improving the project yourself (no risk).
- A few techniques for finding this balance:
  - “Read the source and submit well-formed patches.”
    - ▶ No risk to existing developers, but far fewer new recruits.
    - ▶ Might turn away some potentially fantastic developers.
  - Write documentation about how to work on your project.
    - ▶ LilyPond Contributor’s Guide is 120 pages!
    - ▶ Answer all questions by referring to that guide.
  - Test tasks: keep a few simple tasks for new developers.
    - ▶ Insist that new developers finish those tasks before asking for help with the work they *want* to do.
    - ▶ Only the really motivated new developers will do them.

## Tips for documentation for new developers

- Difficult to formalize all policies, architecture, tricks.
- Can become another time sink:
  - LilyPond Contributor's Guide: at least 200 hours, mostly from our most skilled developers.
  - We could have fixed a lot of bugs with that time!
- Ask the new developers to add to your guide.
  - These could be used as additional “test tasks.”
- New developers gradually do less “guide writing.”
  - Time to start recruiting another generation of developers.

## Keeping New Developers Happy

- Generally the same things that keep developers happy!
- Fast response time.
  - I try to keep my response within 24 hours.
- Private emails; “newbie developer” mailing list?
  - Many new developers are shy about emailing `lilypond-devel`.
- Praise them, prominently give them credit, don't insult or ignore them.
  - This is harder than it sounds – new developers will make stupid mistakes, but make sure you correct them gently.
  - How many senior developers are available to review patches? 24 hours might not be possible... but try to give an accurate estimate of when the review might happen.

## 1 Current development is not sustainable

- Core developers do most of the work
- Losing core developers is bad
- Projects *will* lose core developers

## 2 Keeping developers

- Incentives
- Disincentives

## 3 Preparing for developer loss

- Survival of a species
- Training the next generation: harder than it sounds
- Successes and failures from GNU LilyPond
- Filtering out offers of help
- Dealing with new developers