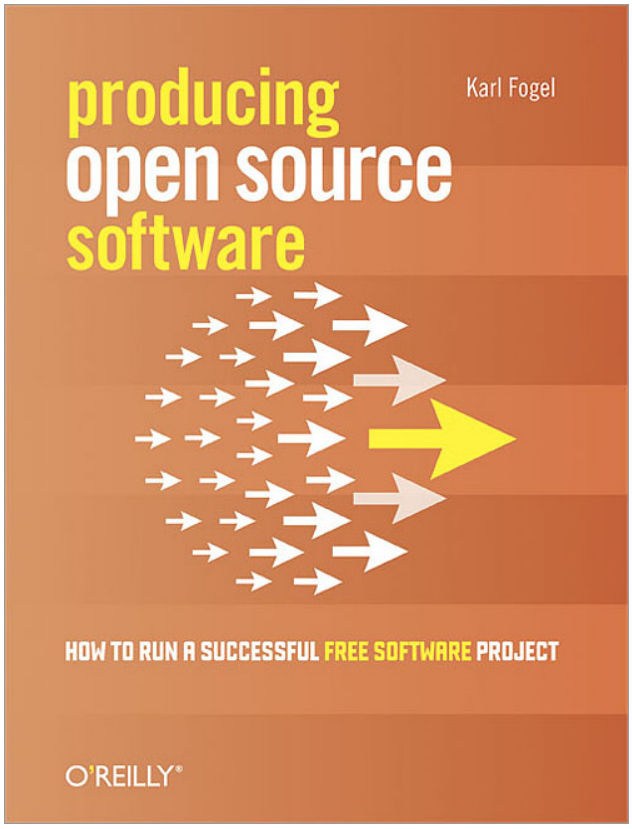
Software Project Management

Syllabus

- Instructor: Hui Lan
- Email: lanhui AT zjnu.edu.cn
- Course home page: http://lanlab.org/course/2020s/spm/

- Recommended textbook:
  Karl Fogel. Producing Open Source Software - How to Run a Successful Free Software Project. 2017. Available online at `https://producingoss.com`. Most of the course materials are from the above textbook.



- Grading:
  - Quizzes - 15%
  - Project - 25%
  - Final Examination - 60%

Quizzes

Number of quizzes.

I will post 7-10 quizzes throughout this semester on our course home page.

Each quiz will be posted one week before its due date.

You should check our course home page at least once a week to not miss any quiz.

Submission instructions.

Class 02395 (consisting of students from Software Engineering (Chuyang), Computer Science and Technology, and Computer Networking) - submit through Duifene. Class code: **AKVCJ**.

Course Project

For people from Software Engineering (Chuyang), you have already spent a lot of time on implementing a software for analyzing condition-dependent gene expression scatter plots. So you will continue to work on (i.e., manage) that.

For other people, you will manage LRR.

You must form project groups by March 2. The group size can be 1, 2, 3, or 4, depending on your situation.

Final Exam

Something I would talk later in this semester.

For now, I think that it will be a closed-book, 90-minute test.

Software Project Management Concepts

Application of management activities - planning, coordinating, measuring, monitoring, controlling, and reporting - to ensure that the development and maintenance of software is systematic, disciplined, and quantified.

dis·ci·plined: showing a controlled form of behavior or way of working.

Textbook management looks so easy and real-world project management is in fact so hard. – Barry W. Boehm

Capable people are no longer available.

For example, the LRR's original author is no longer available to work on the project. He has other more important commitments.

Some people (e.g., Jin Mingyi), maybe also including the original author, think LRR is not maintainable.

Old plan becomes outdated very soon.

You have a plan but you continuously get distracted by additional "requests". A flood of them is a phenomenon called *request creep*.

- A request to add "a few small capabilities" to the product without changing the budget or schedule.

- A request to take on some personnel who are between projects and find something useful for them to do.

- A request to reduce the scope of design review (in order to make up some schedule).

Should you simply be a nice guy and accommodate any request? No. You should think carefully about the impact of the request. You should be able to negotiate a corresponding change in plans, schedules and budgets.

Steps I suggest you to follow.

1. Politely say "No".

2. Tell that I am willing to help if I am not that busy.

3. Analyze the impact of the additional requests on my current schedule.

4. Ask for more money or time, or both. (The management board would usually agree as they have already invested a lot, as long as your counter-requests are reasonable.)

Other difficulties.

Programmers with various background and various personal goals.

Programming languages.

Licenses.

Development processes.

Infrastructures.

Solutions.

- Organizational.
  - Get a lot of money. Get a lot of resources. Get a lot of support from government and funding bodies.
  - Make sure employees write clear documentation and self-documenting code before they leave the project.
  - Have a requirements management procedure.
  - Incentives. Make sure hard and productive workers are rewarded and project parasites are not rewarded (if not downgraded).
  - Make sure useful developers don't get marginalized due to personal idiosyncrasy.

- Motivational. Understanding developers' motivations is the best way — in some sense, the only way — to manage a project.
  - Establish a shared belief that people can do better together. The goal of management is to make people continue to believe so.
  - Get deep personal satisfaction from doing one's best work, and from appreciating and being

appreciated by one's peers.
  - Coercive techniques don't work. People should feel that their connection to a project, and influence over it, is directly proportional to their contributions.

- Set standards for communications.

Proprietary Software

proprietary: relating to an owner or ownership.

The software is owned by a company, which sets out many restrictions on use, modification and distribution. We need to pay (explicitly or implicitly) before we can use it.

Understandable because the company has spent a lot of money and made a lot efforts to develop and update the software.

Understandable because the company does not want its competitors to take advantages of its source code.

Mini assignment (no submission required): read a proprietary software license and figure out the described restrictions.

Must be protected by law.

Good for society as the company makes profits and creates jobs.

For example, the Berkeley Software Distribution (BSD) group did not consider proprietary software a social evil.

But proprietary software can be bad for society if it goes too far.

"The rule made by the owners of proprietary software was, *If you share with your neighbor, you are a pirate. If you want any changes, beg us to make them*." – Richard Stallman.

Free and Open-source Software

These software are usually free of charge.

Many have very high quality.

You can read, modify or redistribute the source code, if the software is free or open source. For example, you can browse the source code for Ubuntu, Lubuntu and Xubuntu. Can you browse the source code for Windows 10?

Free and open-source software are quite prevalent today. Ubuntu, Firefox, Emacs, Nano, Apache Server, etc.

Free software emphasizes ideology.

A noble cause.

For example, Why Schools Should Exclusively Use Free Software?

Representative: Richard Stallman.

Four Essential Freedoms (run, study, modify, redistribute).

License: GPLv3 (approved by both FSF and OSI).

Open-source software emphasizes practicability.

More practical.

Great convenience.

Community driven development model.

Various open source licenses

"Given enough eyeballs, all bugs are shallow."

The difference.

The difference between free and open source is largely on philosophical matters.

The two terms describe almost the same category of software, but they stand for views based on fundamentally different values. Open source is a development methodology; free software is a social movement. – Richard Stallman

Overhead at the Beginning

Opening up a project can add whole new sets of complexities, and cost more in the short term than simply keeping it in-house.

Extra work (or pure overhead at first) for the people least familiar with the project to get started:

- Arrange the code to be comprehensible to complete strangers.
- Write development documentation.
- Set discussion forums.
- Answer questions for potential developers.
- Set other collaboration tools.
- Set a project home page.
- Automate compilation, packaging and installation.

Benefits in the Long Run

Significant organizational benefits - let the outside world know what you are doing.

A low-cost form of advertisement.

Global influence.

Benefits greatly outweigh the costs. For example, Microsoft Loves Open Source (treat open source a development methodology and business strategy).

Reap the benefits.

The sponsor only pays a small number of expert programmers to devote themselves to the project full time, but reaps the benefits of everyone's contributions, including work from programmers being paid by other corporations and from volunteers who have their own disparate motivations (for moral reasons, their employer paid them to, or because they're building up their résumé).

Example: Huawei on Hadoop.

