How the free software development model can improve productivity

Anders Rune Jensen

May 2004

Abstract

Free software [3] has been around for more than 20 years and has had a profound impact on how software is developed. Everything from small scripts to very large projects, like the Linux kernel and Ximian Evolution consisting of several million lines of code, have been developed as free software.

This essay will examine how crafting software as free software improves productivity compared to proprietary software development methods.

1 Introduction

To firmly understand how free software works lets us turn to its definition. For a piece of software to be licensed under a free software license the following freedoms must be upheld:

- The freedom to run the program, for any purpose (freedom 0).
- The freedom to study how the program works, and adapt it to your needs (freedom 1).
- The freedom to redistribute copies so you can help your neighbor (freedom 2).
- The freedom to improve the program, and release your improvements to the public, so that the whole community benefits (freedom 3).

These freedoms allows a radical different development model compared to proprietary software, for one because the source code is available and secondly because you are free change and redistribute the source code.

This article will in Section 3 use the term open source [5]. Free software is a subset of open source, but the arguments presented by Eric Raymond in The Cathedral and the Bazaar [1] applies also to free software, since the arguments were gathered by studying free software projects.

2 The effect of available source code

Having the source code available allows the users of a particular piece of software to do a lot more useful things with it. If the software has defects, the user has

the potential to create better bug reports by inspecting the source code. The user might even provide a patch that fixes the problem or he/she might hire another developer or the author to fix the problem for him/her. These facts makes bug reports more valuable to the developer, allowing him to fix bugs faster.

The fact that everyone can view the source code gives the developer an incentive to write good code, because this will help him get better bug reports and will help him attract new developers, as new developers are more reluctant to join a project if code is easier to understand. A developer needing a particular feature in his software is more likely to find an existing project, modify that to his needs, than he is to start his own from scratch. This is a win-win situation. A recent example is Apple using the HTML renderer from KDE, expanding it to their own browser, safari, and releasing the source back [4]. It is apparent from the email in [4], that the choice of KHTML and KJS was based on the code size and quality of the code.

Someone might argue that the source code being available for anyone to change, increases the chances of forking a project. A new developer might have ideas that conflict with the current maintainers ideas and is forced to do a fork to keep developing his ideas. Some users will join the fork and in turn this will decrease the help the current maintainer can get from users. If the project only has a few users this might even completely fragment and destroy the project.

A fork will decrease the number of users a project will have, but it will increases the development effort since the original maintainer has to convince his users that his original idea is better than the new forked project, while the new developer has to really deliver something worthwhile for users to change project. So while this might seem as a negative effect it actually has the potential of increasing productivity.

3 The user

Extreme Programming [6] values user feedback very highly and involves the user through user stories and quick releases. This is very similar to open source development as Eric Raymond points out [1]: “Release early. Release often. And listen to your customers.”.

A major difference between open source development and Extreme Programming is the amount of users available and the role which these play. In Extreme Programming the user is mainly used to ensure that the final product satisfies the customers demands. Where as in open source the user can contribute to the project on many different levels. The user can help other users, write documentation, translate the program or documentation to other languages, write bug reports or fix bugs. Eric Raymond characterizes the effect of having a large user base as: “Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.”.

Given that anyone can contribute to a project, how do free software projects ensure quality?

How a free software projects responds to outside work is highly dependent on the source of the work. The communication medium most often used is a mailing list or a bug repository. These allows peer review of incoming work making it easier to spot errors. Some kinds of work, eg. a translation or a

new driver, expand the functionality of the software and the developers might not be able to test it. The way this has been dealt with in the kernel is that these drivers live in separate trees, as patches to the official kernel, until they have proved themself stable. Translations is often peer reviewed by translation teams. The effect of external contribution was summarized by Eric Raymond as: "If you treat your beta-testers as if they're your most valuable resource, they will respond by becoming your most valuable resource.".

Free software is a highly evolutionary approach to developing software. One might even call it empiricism applied to software: "Empiricism, on the contrary leads to a liberal or anarchistic view of society, a democracy in which individuals contribute "data" to the government of a loosely organized, changing network of people" [2].

4 Conclusion

In this essay I have argued that free software improves productivity. The productivity is gained by harnessing the effect of external contributions. To harness this the developer has to have a high communications skill to be able to motivate and coordinate external contributions, often with people he has never met. Futhermore the ability to write clear and clean code can be significant to the choice of which project an external contributor will focus his energy on. This was seen in the example where Apple chose the KDE HTML engine instead of the Mozilla engine, which was at the time was more complete, but the code for the KDE engine was cleaner and better structured. It is worth also worth mentioning that the Mozilla project started its life as the proprietary browser Netscape, but was later Open Sourced after Microsoft has won the browser war. On the otherhand the KDE html engine has all its life been Free Software.

References

[1] The cathedral and the bazaar. `http://www.catb.org/~esr/writings/homesteading/cathedral-bazaar/`.

[2] Bo Dahlbom and Lars Mathiassen. *Computers in Context: The Philosophy and Practice of Systems Design*. Blackwell Publishers, Inc., 1993.

[3] The free software definition. `http://www.gnu.org/philosophy/free-sw.html`.

[4] Greetings from the safari team at apple computer. `http://lists.kde.org/?l=kfm-devel&m=104197092318639&w=2`.

[5] Open source initiative osi. `http://www.opensource.org`.

[6] Extreme programming: A gentle introduction. `http://www.extremeprogramming.org`.