

---

# Problems and Strategies in Financing Voluntary Free Software Projects

Benjamin Mako Hill <mako@atdot.cc>

Copyright © 2005 Benjamin Mako Hill

This material is licensed under the Creative Commons Attribution-Sharealike 2.0 License [<http://creativecommons.org/licenses/by-sa/2.0/>].

The canonical location for the most recent version of this document is at the author's website [[http://mako.cc/writing/funding\\_volunteers/](http://mako.cc/writing/funding_volunteers/)].

	Revision History
Revision 0.2	June 10, 2005
	Minor corrections and improvements.
Revision 0.1	May 15, 2005

The first version of this paper was written to an accepted talk given at Linuxtag 2005 given in Karlsruhe, Germany.

Revision 0	May, 2004
------------	-----------

This paper is based in part of the research and work done for a presentation on the subject given at the International Free Software Forum (FISL) given in Porto Alegre, Brazil in May 2004.

## Table of Contents

Introduction .....	1
Volunteerism in Free Software Projects .....	2
Benefits of Volunteerism .....	3
Problems with Funding Volunteerism .....	4
Creative Solutions .....	5
Low Risk Solutions .....	5
Funding Labor Strategically .....	6
Outside Organizations .....	7
Transparency .....	8
Conclusion .....	8

It's easier for a successful volunteer Free Software project to get money than it is to decide how to spend it. While paying developers is easy, it can carry unintended negative consequences. This essay explores problems and benefits of paying developers in volunteer free and open source projects and surveys strategies that projects have used to successfully finance development while maintaining their volunteer nature.

## Introduction

The love of money is the root of all evil.

— *New Testament St. Paul, in 1 Timothy, 6:10*

Using voluntary labor, free and open source software<sup>1</sup> projects have become massive commercial successes. This success has been awkward and dangerous for some projects. Today, many commercial and non-commercial entities intent on building on the success of the free software movement find investing and funding these projects difficult. Voluntary projects with an influx of resources are sometimes unsure of the best ways to use these resources without sacrificing the model upon which their success is built. Other projects find their development environment radically changed, sometimes for the worse, by the entrance of paid labor into their communities that brings with it a new style of working and a new type of inter-developer relationships. In these ways and others, many voluntary free software projects find themselves face to face with the reality that it is often easier for a successful volunteer free software project to get money than it is to spend it.

This paper aims to provide a guide for voluntary projects who want to fund development as well as a guide for funders wishing to work with and support these projects. It begins by looking at the role of volunteerism in many free software projects and at the benefits that voluntary labor brings. It turns to real examples from the free software world to demonstrate the varied and dynamic role played by voluntary work. It continues and explores the problems and benefits introduced by paying developers. Finally, it concludes with an annotated survey of strategies that free software projects have successfully used to finance development while maintaining the benefits of their volunteer nature.

## Volunteerism in Free Software Projects

In his famous announcement, Linus Torvalds referred to the Linux kernel project as "just for fun." This phrase is a reminder that, at the beginning, even Linux was built entirely through voluntary labor. It is a reminder that voluntary labor has played a complex and dynamic role in the life of Linux and in many free software projects. To this day, the Linux kernel has maintained its institutionally independent nature and voluntary labor still plays an important role in Linux development. However, the vast majority of contributions to Linux are now the direct result of paid labor. Other projects, like KDE, GNOME, Debian, and Gentoo also incorporate and are driven by the work of paid laborers. Each of these latter examples is driven and led by people working in their spare time, after hours, and outside of companies. In none of these examples are developers paid by the free software project itself.

As free software has been commercialized, paid labor has taken up an increasingly important role in the free software world and many free software projects today are built as works for hire and are released by corporations. An old and well-known example of a primarily non-voluntary project is the postscript system Ghostscript. Although there has been some substantial contributions from the community, Ghostscript is almost wholly written by a single developer. The JBOSS application server is a second example. While JBOSS releases all of their code freely and the JBOSS team also receives patches and contributions, their development is wholly based on and driven by the labor of JBOSS employees. The models pursued by Ghostscript and JBOSS are neither entirely good nor entirely bad; there are benefits and drawbacks of both voluntary and paid labor.

It is also worth noting that, over time, volunteer projects can become non-volunteer projects — and vice versa. Projects based around paid labor — even proprietary development — have been "freed" or "open sourced" successfully and have built thriving volunteer fueled and mixed volunteer and paid labor communities. The Mozilla project and the 3D rendering software Blender are two such examples. Others have floundered. Projects like the X Windowing System have begun as volunteer-driven projects, become projects heavily dependent on paid labor, and then returned to a volunteer-driven development model. The Wine project was initially a volunteer based project that is now largely driven by paid labor

---

<sup>1</sup>The Free and Open Source Software movements are parallel movements with a complex relationship and history outside the scope of this essay. When discussing software, licenses, and development communities, the terms are usually synonymous. When discussing the motivation, philosophy, and politics behind the production of this software, the terms vary wildly. As to the nature of the distinction, an inadequate but useful distinction can be drawn: Free Software is a social movement; Open Source is a development methodology. For the purpose of this essay, I use the term free software to refer to both free and open source software.

within a single company.

## Benefits of Volunteerism

While not every free software project can be or should be driven by voluntary labor, many projects can gain from the work of volunteers and from a primarily voluntary structure. Much of the free software movement's success in creating products that are commercially interesting is directly due to benefits inherent in — and some cases only easily accomplished through — voluntary work.

The metaphor of the bazaar, popularized by Eric Raymond in *The Cathedral and the Bazaar*, describes a inherently superior software development paradigm based around a model of decentralized control and ad-hoc collaboration. The bazaar model is only easily and cheaply realizable through volunteerism. Many people misinterpreted Raymond in his description of the free or open source development process and looked to free software licensing alone as the key to superior software and to eventual commercial success. However, neither licensing nor the availability of source code gives free software its competitive edge. Licensing and source are merely means toward an end: the participation of many individuals and groups in the development, testing, quality assurance and advocacy of software.

Proprietary software companies with many resources can bring in the work of many paid laborers to achieve these ends. The free software development model is distinguished from and has an advantage over proprietary models because it can cheaply and easily bring in the work of volunteers. By harnessing volunteers and fostering a testing and development community that is actively invested in the project, voluntary labor renders free and open source development more effective, equally reliable, extremely inexpensive, and massively cost-efficient.

Perhaps the most important benefit of volunteerism in free software development is institutional independence. Institutional independence in a free software project means that no company or organization has a monopoly on the ability to define specifications or to direct the project. To users and developers, institutional independence means that *they* get to define the specifications; it is a broad perceived autonomy. Projects that are driven or directed by volunteers are more easily able to appear institutionally independent than corporate or organizationally directed projects or any projects that incorporate paid labor.

Sun, IBM, HP, and others can cooperate and compete around the Linux kernel because Linux is seen as being beyond grasp or control of any one of them. Because the Debian project is viewed as institutionally independent, there are multiple companies — a short list includes Canonical, Progeny, Credativ, and Linspire — who support, build on, and contribute to the work of the Debian project. This is comparatively less collaboration between Red Hat-based derived distributions that work with technology that does not share the same level of perceived institutional independence and have diverged over time as a result.

For most free software practitioners, institutional independence also implies that there is less potential for one company or group to "corner" or enclose a market or to create a dependence on their company: a result that that users find attractive. Institutionally independent projects can act as spaces where competing groups work together on a single project and a set of shared goals. Because they are not in the service of anyone, they can provide "common ground" for "coopertition" — a mix of cooperation and competition that is largely foreign to the traditional software development world but is increasingly common in free software. Institutional independence translates into the ability of a software and the development community to, through the work and collaboration of individuals and groups, become stronger and more robust than they would otherwise.

Finally, volunteer projects have the proven benefit of being able to build great products with little — sometimes *no* — financing. It's easy to fund a project that does not need funding! Working from donated bandwidth, hardware and labor, Debian has built the single largest free software development project with total expenditures in the tens of thousands of US dollars. Even most of these expenditures could probably have been avoided without sacrificing the project's achievements. For projects like Debian with limited financial resources, volunteerism can be a great way to get things done.

## Problems with Funding Volunteerism

While there are benefits to volunteerism, the potential productive benefits of paid labor can not be dismissed. Paying a laborer to do a task is a great way to ensure that the task is done and that it is accomplished on time. Some voluntary free software organizations have a history of difficulty in meeting deadlines or in acting predictably. Debian's long-delayed and most recent release, **sarge** has become infamous as an example of this limitation. However, while the benefits of paid labor in solving these problems are obvious, the less obvious drawbacks can be just as serious. These include the tendency of paid labor to crowd out volunteers, the negative impact of paid labor on the transparency of free software projects, and a general degradation in the quality of the software produced.

There is much anecdotal evidence on the negative impact that money can have in voluntary organizations. Recently, research in Scandinavia has put brought to light empirical evidence to back up these claims. Research by Bernard Enjolra at the Institute for Social Research in Oslo, Norway monitored the role and nature of voluntary labor in Norwegian sports organizations. He then kept track of the role and extent of volunteer labor as paid labor connected to commercial income as "commercial resources" was introduced to handle certain elements of book keeping and organizing. The result was that less people volunteered -- for these tasks in particular but also overall -- and that those that volunteered volunteered less. His summary states:

Empirical results using cross-sectional data on voluntary sport organizations in Norway and on their members show a decrease in voluntary work from an increase in commercial income. Voluntary work and commercial income appear as substitutable resources.<sup>2</sup>

In short, when it comes to voluntary work and paid labor, you can have one or the other but not both. Enjolra calls this process "crowding out." While it is unclear why paid labor crowds out the work of volunteers, Enjolra hypothesizes that volunteers are less motivated to work for free when they know that others are being paid to do the same work or will be paid to do the work if they do not. Faced with paid workers in their organization, volunteers wonder why they are not getting paid for *their* work and feel more motivated to volunteer somewhere else. In this way, a small amount of paid labor in an organization or project highly dependent on the work of volunteers can do more harm than good.

While there has been not empirical evidence from within the free software community confirming the applicability of Enjolra's research, I see no reason why Enjolra's conclusions would be limited in applicability to sports organization or to Norway. It's worth noting that Enjolra does not state that the use of paid labor was a "bad" decision in his examples. He only points out that it has a real and often inescapable negative impact on the amount of voluntary labor employed in the organization and that voluntary organizations introducing paid labor should take this into account.

Another drawback of paid labor is a reduction in transparency. The example of the X11 project and the X Consortium can help demonstrate the role that this can play in organizations.<sup>3</sup> The X11 project began as a community driven project with developers from around the world connected to each other via the Internet. The project released regularly twice a year and generated a good deal of serious commercial interest. However, the project's distributed mode of work had serious limitations. Serious network problem early on made coordinated work extremely difficult for much of a year. To solve these and other problems, the *X Consortium* was created and funding and paid labor was introduced. The result of the this was, in the words of developer Jim Gettys, to "make some people 'more equal' than others," and to "disfranchise 'outsiders.'" Quite simply, volunteer labor was crowded out. Among volunteers who stopped contributing regularly to X was Python author Guido von Rossum.

---

<sup>2</sup>Enjolra's finding is published in an article called "Does The Commercialization Of Voluntary Organizations 'Crowd Out' Voluntary Work?" published in the *Annals of Public and Cooperative Economics* (Number 73:3, 2002, pp. 375-398).

<sup>3</sup>In 2000, Jim Gettys told the story of the X Consortium and the X11 project at a talk at USENIX and much of this history in this essay is based on that talk and from personal discussions with him. The talk is available online here: [http://www.usenix.org/publications/library/proceedings/usenix2000/invitedtalks/gettys\\_html/text13.htm](http://www.usenix.org/publications/library/proceedings/usenix2000/invitedtalks/gettys_html/text13.htm)

The more meaningful lesson to be learned from the X Consortium's mistakes lies in a substantial drop in transparency in X to most of its developers. When the consortium was created, paid programmers were moved into a single office. It was now quicker and more efficient to discuss a new feature or a design decision at a whiteboard or blackboard or over the top of the cubicle. Development became quicker and more efficient for people who worked in the office and more difficult to track, follow, and participate in for anyone working from remote or who had less time to devote.

Part of being "more equal than others" was having easier access to information and status and to the other people working on the project. While the project was still "open" to contributions from its community, volunteers had a more difficult time following the project's development. X showed how, by creating an in-group and, by extension, an out-group, the introduction of paid-laborers into volunteer projects can introduce the potential for this effect.

The third problem introduced by paid labor in volunteer projects can also be learned from the X Consortium's experience. In X and in many other projects, the introduction of paid labor can change the direction of the project in ways that do not always serve the interests of users. It seems only natural that paid labor is directed toward funders' needs or desires and influence the "strings" that are attached to money. In the case of X, "strings" attached to the money put certain aspects of the desktop that were controversial off-limits to consortium staff. Because toolkits, desktop environments, and 3D were each controversial or proprietary areas among the funders, staff was not paid to work on these areas. The result was several major gaps in the X11 platform that restricted the ultimate usefulness of the software. Had the workers been working as volunteers, these were problems that would have been addressed early on — there was high demand from users. However, the introduction of paid labor directed the project in other directions.

The impact of these problem will differ in degree and depend on the nature and make-up of the project. Problems will often be minor and ignorable; they will sometimes be catastrophic. The sad end of the X Consortium story is that, not long after it was created, the consortium collapsed. The collapse was due directly and indirectly to the problems attached to funding described above. By the time that the consortium collapsed, the volunteer X project had completely disappeared. X lay dormant for some time before being resurrected later for GNU/Linux by the XFree86 project and now again by a new X.org group using the defunct X Consortium's domain.

## Creative Solutions

While it's important to be critical and realistic about the negative impact that money can have in voluntary free software organizations, it's not always reasonable to suggest that volunteers refuse all money or that corporations refrain from funding voluntary software projects altogether. Money, even paid labor, can be an effective tool in free voluntary software projects when used carefully, strategically, and intelligently.

In terms of the dangers of funding voluntary projects, refusing all funding is a no-risk solution. However, the lack of risk is balanced by a dramatic set of limitation to a project's potential achievements. For some projects, particularly ones where institutional independence and political neutrality are central, this may be the best course of action. For many others, there are better options.

## Low Risk Solutions

The safest way for voluntary free software organizations to spend money without compromising voluntary labor is to not pay for labor -- and the production of code in particular -- or to fund labor indirectly.

One great way to do this, employed frequently by the Debian project, is to use donations to purchase hardware to help facilitate development. The most frequent and largest types of expenditure by the Debian project and its supporters is hardware for servers, other forms of infrastructure, and bandwidth. Each of these are viewed as shared resources within the project which avoids the appearance of paying an individual and valuing some contributions more than others — even if an individual makes disproportion-

ately large use of the machine or infrastructure in question. When a developer steps down, the project infrastructure they use is passed on to another volunteer.

Another good way to fund projects is to build "capacity." Capacity describes work or resources of any organization devoted to something other than the groups' core goal but that makes the project more effective or efficient. Developing or building an accounting system is a good example of capacity. Because it is outside of the core goals and work of the project, funding capacity is less likely to crowd voluntary workers out. Since infrastructure and increased or improved capacity can help an entire project run more smoothly, funding capacity in this way can help a voluntary organization become more streamlined and effective with only very minimal risks of crowding out volunteers, affecting transparency, or resulting in a lower quality product.

Another good low-risk solution is to use funds to organize or sponsor conferences, seminars, meetings or workshops. For groups with less funding, money can be spent on securing a venue, bandwidth, food for attendees, or other conference expenses. For projects with more funding, need-based or merit-based travel aid can be offered to help attendees reach the conference. Conferences provide a venue for sharing information between members of a volunteer team, allow for in-person brainstorming and design sessions, help increase the quality of relationships between members of the project, and can act as a reward to "sponsored" developers.

DebConf, the annual Debian conference, is a good example of strategic conference funding and is how the Debian project spends the vast majority of funding each year. Lodging, bandwidth, the conference venue, and food are normally paid for all attendees based on a first-come-first-served basis until funds are depleted. Additionally, travel aid is offered on a combination of need and merit based systems going first to speakers with accepted proposals, then to official Debian developers, and finally to qualified contributors to the project over the previous year. Funded conference attendance acts both as a reward for active volunteers and a step toward creating a positive, fun, and highly educational event for all attendees. Each year, there are participants who are less involved in the Debian project before the conference but who increase their involvement after. Additionally, DebConf acts as a venue for airing ideas and proposals between developers and for gathering feedback on controversial issues.

A related, and sometimes overlapping model of funding involves funding code sprints. Popular in the Python and Zope communities and with increasingly popularity elsewhere, sprints are intense sessions of development — usually around one week long. They are used as catalysts for development and have seen major leaps forward in the development of features and code within very small amount of time. Sprints are like conferences in most aspects except that the emphasis is more on the production of code and sustained hack-sessions than on presentations and discussions. They also usually involve less people than a conference and attendees are usually limited to the most actively involved volunteers on a project. The Plone Foundation and SLX Debian Labs in Norway has effectively used the funding of sprints to accomplish time-based development goals.

While conferences and sprints can be a clever way to spend money without sacrificing the voluntary nature of development projects, it is worth keeping one important caveat in mind. When a project selects people for funded attendance at the expense of others, it demonstrates favoritism that can be divisive. In the process of organizing these events, it is important to maintain a high degree of transparency and fairness. Organizers should use published and fair criteria to determine conference funded attendance and leave attendance open to all. Good criteria for fair selection includes "first come, first served," constructive activity on mailing lists, the number of important commits to a source code or documentation repository, or a good reputation among fellow developers (e.g., as determined by a fair and representative committee).

## Funding Labor Strategically

Because the creation of software is the central and sole goal of many free software projects, the desire to fund development is often unavoidable. When voluntary projects choose to pay people to develop code, there are certain techniques and recommendations that can minimize the potential negative impacts on voluntary labor.

One important question to keep in mind is: *when* is a project ripe for funding? Paid labor within a young project without a viable and sustained level of contributions from volunteers will often translate into a systemic lack of volunteer contributions going forward. Paid labor early in the life of a voluntary project is more risk-prone than paid labor later. The Debian-Nonprofit turned down funding for paid labor in order to help build a sustainable and viable volunteer project first. This is especially important in the case of limited and time-limited funding. If a project is dependent on funding and the funding runs out, the lack of a volunteer-base to carry the project forward can spell death. Many developers have a difficult time transitioning from paid labor to volunteerism.

When paid labor is introduced into a free software project, is it important to maintain a high level of both criticism and strategy. It is essential to carefully select the type of work that will be funded. There are a number of techniques and guidelines that can be helpful in doing this.

One useful model that shares common ground with the idea of funding capacity described above suggests funding primarily non-technical work. Paying people to do the equivalent of project management, book-keeping, or writing funding applications for a conference can be less controversial and less divisive than paying someone to write code; it can avoid crowding out coders while filling unfilled or neglected roles.

Another related model is to pay programmers to do work that participants recognize that no volunteer can do. Jobs that require an excessive amount of research or that are extremely tedious might be good candidates. Explicitly limited jobs, in terms of the scope and duration, may also make good choices. Time limited efforts, including projects under very tight deadlines, are often appropriate. Features that have remained on a todo-list and that are widely recognized as important are good candidates for strategic funding. Widely advertising feature requests and then only funding unimplemented requests can also be effective. However, each these suggestions can run the negative risk that people not fix a problem or address an important issue if they think that, eventually, they or someone else might get paid to do it.

The most important concept to remember in determining what to fund is the *appearance* of the funding to the volunteers. It's not enough for the funder to think that problem cannot be solved by voluntary labor alone; the *volunteers* must think that the problem cannot be solved by voluntary labor alone or the funding will run the risk of turning off and crowding out those volunteers.

As mentioned in the discussion of choosing conference attendees, choosing paid laborers must be fair. Projects raise funding through the goodwill generated by the work of the entire project and using that funding to pay a small subset of volunteers can be extremely divisive. Fairness and transparency must play a central roles in hiring anybody.

## Outside Organizations

A final, more risky, strategy is to involve the work of outside organizations in the work of the voluntary project. Alluded to in the introduction, Debian provides an example of this strategy. While the Debian project does not fund voluntary work, many outside organizations fund work in Debian toward their own goals and beyond the direction or control of the project. Instances of this range from simple cases of an employer paying an employee to maintain tools used in the employer's business to more complex situations involving companies whose core business is based around selling and supporting Debian or their own distributions based on Debian. Credativ, Progeny, HP, SLX Debian labs, and Canonical are examples of companies and groups who do targeted development within the Debian project from outside of the project.

The key to success in these situations is to maintain the institutional independence of the larger project in the minds of the volunteers. If a voluntary projects looks like an arm of a corporately controlled project, volunteer work will be crowded out. The Fedora project, with its very strong relationship to Red Hat, is an example of a failed attempt as this model; many volunteers or potential volunteers have felt that they do not have as much of a voice in the direction of Fedora as Red Hat's developers and managers. In part because the Debian project is so large and in part because there are so many companies

and organization actively involved in funding Debian work, the situation in Debian is different. Progeny and Canonical (to name just two important examples) have done a much better job than Red Hat in sustaining voluntary work in their volunteer "upstream" project. Their models can provide inspiration to others following a similar course.

## Transparency

Regardless of the steps that a project chooses, transparency will act in a central role in maintaining volunteerism. Had the X Consortium been better about maintaining transparency, their story might not be as dismal.

There are a few concrete steps that can be advocated in terms of maintaining transparency; several have introduced already. Another simple solution is to employ open mailing lists: allow anyone to join and anyone to participate in any list; only create closed or private lists after strongly considering the negative impact on volunteerism and the openness of the project.

It's also important to ensure that the public forums are used by the "internal" developers. Even if developers or programmers work in the same building or in cubicles next to each other, they should resist the temptation of having conversations in person because the information from these conversations will never reach the volunteer community. Intentionally organizing a distributed company is one way that this can be approached by companies working on projects with volunteers. Implementing such a structure was an intentional decision made by Canonical Ltd. in hiring developers and structuring development. Ubuntu maintains no closed lists and makes all development, design, and government decisions in open forums and, due to its distributed nature, is exceptionally transparent for a project incorporating a large body of paid labor from a single company. <sup>4</sup>

Additionally, developers should stay conscious of the openness and transparency of their communication mediums. For the most part, the following rule of thumb can be useful: mailing lists are better than IRC which is better than the phone which is better than chatting in person. Developers should remember that when there are in-person meetings or phone meetings — especially ones where decisions are made — these must be written up and reported back to the community in an open forum.

## Conclusion

Funding voluntary free software projects is not easy if one wants to maintain the project's voluntary nature. For those who go ahead and introduce paid labor into a primarily voluntary free software project, it's important that they keep in mind the difference between free and proprietary development models. Especially for those with experience in proprietary software development, the most natural way of spending money sacrifices many of the benefits of open source and free software development that are tied intrinsically to volunteer labor.

Those spending money should consider pursuing funding models that avoid paying individuals and should strongly consider the benefits and drawbacks described here when doing so. They should only do so strategically and while maintaining a high degree of transparency. Above all, both funders and members of projects should strive to stay critical, stay creative, and stay transparent throughout the entire funding process. Done critically, creatively, and transparently, voluntary free software projects can use money and paid labor to a tremendous benefit that only magnifies their accomplishments.

---

<sup>4</sup>I have written a essay paper detailing the Ubuntu development model pursued in the first year of the Ubuntu project. That paper can be found at [http://mako.cc/writing/to\\_fork\\_or\\_not\\_to\\_fork.html](http://mako.cc/writing/to_fork_or_not_to_fork.html).