

Comparing Jazz to Free Software: Models for Learning, Sharing, and Application

David Baird

May 5, 2004

Contents

1	Introduction	1
1.1	Free Software Definition	2
2	Learning the Basics	2
3	Teamwork	5
3.1	Case Study of Gentoo	5
3.2	Operator Agreements	7
3.3	Tools	8
3.4	Scalability	8
3.5	Jazz	9
4	Conclusion	11

1 Introduction

In this paper, I hope to propose some possible models for how Jazz is created and relate these models to free software development. Figure 1 shows the basic model for computation, and figure 2 shows my variation of this model for humans. Humans get information from the world through hearing, vision, and touch. Information that humans might store in memory include muscle memory (how to physically place fingers on an instrument to achieve a desired

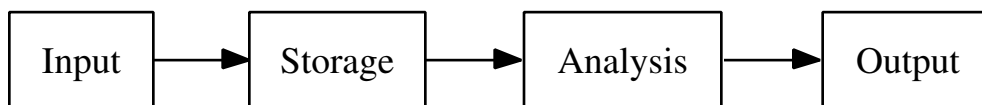


Figure 1: Basic Computational Model

sound), higher level instrument techniques, techniques for faster learning, or how to work with other people.

1.1 Free Software Definition

There are four freedoms granted by free software, the first freedom being numbered zero: freedom to run the software for any purpose, freedom to study and adapt the software which implies access to the source code, freedom to share and redistribute copies of the software, freedom to improve the software and release improvements to the public¹.

2 Learning the Basics

Borrowing and sharing is critical component of Jazz and free software; This is how artists and hackers get information to learn from, adapt to their preferences, or use to help accomplish some goal.

Beginners in Jazz often practice until they can sound exactly like their idols². These musicians believe that mastering solos of the greatest musicians will lead them to create great ideas of their own³. The more they practice, the more their perception and comprehension evolves. Their abilities to listen, analyze, and play improves.

Some things are fairly important to learn in Jazz. John McNeil was shocked to learn that a song can be played in multiple keys⁴. He hid for months in order to re-learn songs in all twelve scales. Some people feel that any musician should be able to play any song they have heard before⁵. Singing tunes is also an important skill to learn for creating “essential linkages

¹(Free Software Foundation, Inc. 2004)

²(Berliner 1994, 124)

³(Berliner 1994, 97)

⁴(Berliner 1994, 66)

⁵(Berliner 1994, 93)

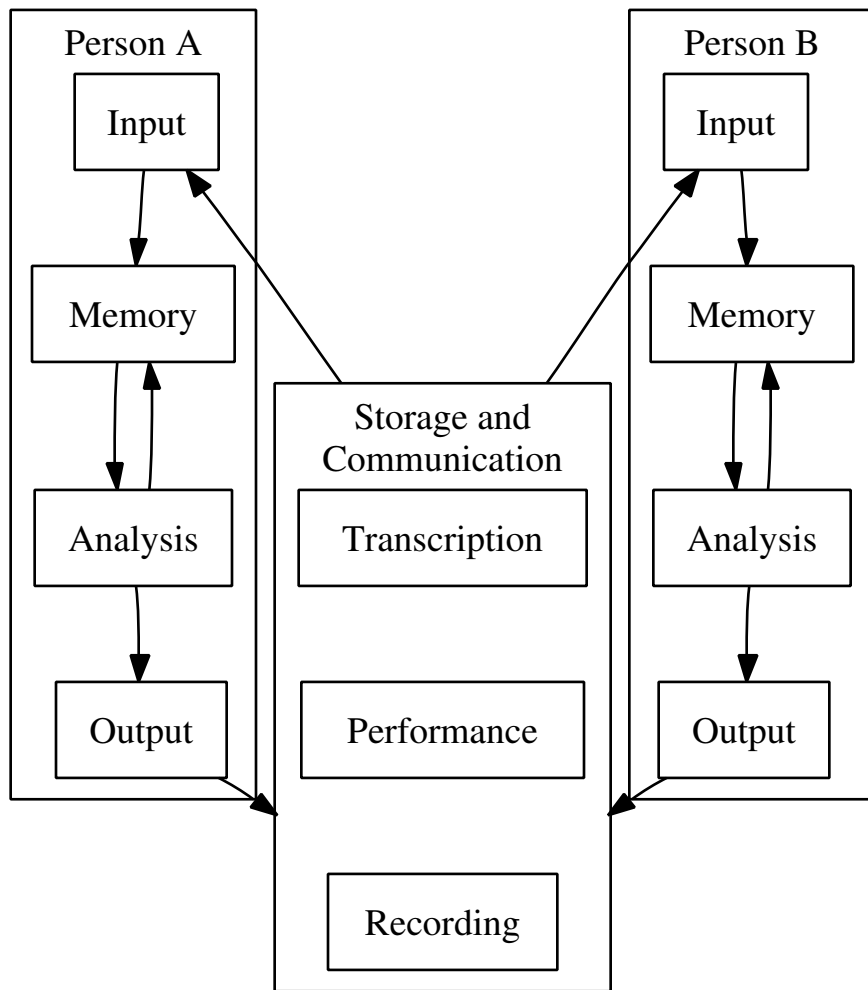


Figure 2: More Sophisticated Model for Human Computing. The edges point in the direction that information flows.

among voice, ear, and instrument.”⁶ Pianists and guitarists learn how to play chords at a young age and excel at harmonic abilities compared to players with that can only play one pitch on their instrument at a time⁷.

Simply listening to music and trying to recreate the same sounds is not always enough. Keith Copeland explains, “it’s hard for drummers to learn from records because you can’t see what the drummer’s doing with his hands and his feet, what sticks he’s using and exactly what part of the drum he’s hitting.” Seeing other people perform, or playing around with peers is also very beneficial to learning.

Likewise in free software, knowing C, a ubiquitous programming language, is generally a prerequisite to earning respect from peers. Knowing how to use the vim or Emacs text editors helps to seriously increase productivity, and most hackers learn at least one of these two editors. Eventually however, free software hackers should be able to pull together a variety of tools and systems to dive into large projects and create solutions. I’d like to say that creating easy-to-understand solutions was a standard, but some hackers create messes and others don’t. Knowing how to deal with other peoples’ messes is a valuable skill too.

Young hackers may be seeking to program any way they can, perhaps starting out with BASIC, finding a copy of Borland C++, or being drawn to the flagrant Microsoft utilities. When they find peers that also like programming, then they can exchange more ideas and hopefully discover Linux at some time. Microsoft is kind of hard to get around since Microsoft software is advertised and pushed hard at many public schools; This is very unfortunate too, as many good souls are drawn into the depths of Microsoft’s ego-boosting/brain-washing software⁸:

... Powerpoint is to business communication what commercial entertainment is to art: a way to sell the viewer, by giving him an effortless experience that never challenges him (“Keep it short!!”) and lulls him into thinking he’s understood something without requiring any effort on his part.

... This year, [my son’s] entire sixth-grade library course is devoted to learning computer tools, including powerpoint. And no, he can’t write an essay properly yet.

⁶(Berliner 1994, 96)

⁷(Berliner 1994, 72)

⁸(Tufte)

- http://www.7nights.com/asterisk/archives/tufte_powerpoint_and_web_content.php

3 Teamwork

Defining good teamwork is a hard problem. In class, we defined teamwork as getting along, getting stuff done on time, and no one straggling behind, but perhaps the most important defining aspects of a good team are knowledge of why the team exists and why each member is a part of the team.

Individual projects in the free software community start with a goal of some sort. There are problems however, such as goals that are too general, people starting projects who aren't willing to follow through and work on the projects, not having any standard to determine when goals have been achieved, or not knowing what next step to take when goals have been achieved.

I apologize for not having the scoop on XFree86.org because they would have made a good case study for projects that hit road blocks. XFree86.org developers had some disagreements and development became halted forcing the free software community to fork the popular and powerful XFree86 X server and create X.org to take over development.

3.1 Case Study of Gentoo

Gentoo, a distribution of Linux, started with a clear goal to ease the user's job of getting and custom configuring the software they wanted, but no there was no breakdown of this goal to measure how well they were achieving this. This could be understandable since Gentoo was highly experimental and no one may have known the best ways to break down the goal. The experimentation and research that went into achieving this goal is of enormous benefit to the free software community. The following quote from Gentoo's philosophy sums up the area where Gentoo succeeded:

I created Gentoo because I couldn't find a Linux distribution that I liked. The one predominant thing that I experienced with Linux distributions is that the "distro tools" that managed the entire system – the tools that were supposed to make everything easier to use – really seemed to want a lot of attention and really

got in the way of what I wanted to do. I wanted to tell them what I wanted to do, but they seemed more interested in telling me what they wanted me to do.

So, I created Gentoo Linux, and designed Portage to be a more perfect tool than what had existed before it. To do this, I made it very flexible in allowing me to do what I wanted to do, and also tried to make it flexible to allow others to do what I thought they might want to do.

If others wanted to see how a package got built, they could look at a relatively easy-to-understand ebuild file and learn from it. If they wanted to tweak how it got built, they took advantage of USE variables. If they wanted to add a package, they created a new ebuild for the tree. If they wanted to use a package, they simply emerged it and dependencies were automatically resolved.

- Daniel Robbins,

<http://www.gentoo.org/main/en/philosophy.xml>

The Portage system of Gentoo truly resulted in giving the user a high degree of concise, expressive control over how software would be configured and what software should be installed. Portage is capable of automatically calculating dependencies, downloading software from a variety of options, and upgrading, downgrading, recompiling, or removing any software component at the user's will.

There are some critical problems that prevent Gentoo from advancing however. At this moment, the main task performed by Gentoo is maintaining Portage and keeping it up to date with the latest software. This is likely a result of a poor operator's agreement (see section 3.2), and a poor definition of future goals as summed up in this quote from Gentoo's philosophy:

The future goal of Gentoo is to continue to strive to create near-ideal tools. Tools that can accommodate the needs of many different users (all with divergent goals) with ease are extremely powerful. Don't you love it when you find a tool that does exactly what you want to do? Doesn't it feel great? Our mission is to give that sensation to as many people as possible.

- Daniel Robbins,

<http://www.gentoo.org/main/en/philosophy.xml>

There were several specific directions that Gentoo could have taken. The weaknesses of Portage really should be addressed, and Portage could have benefited from a complete rewrite with more specific goals to accomplish. Gentoo could also move in a direction to be enterprise friendly: make it easy for large organizations to switch to Gentoo without having to put a lot of money into tech support or IT personnel. Installing various gcc's to target many platforms could be made easier. Peer to peer sharing of ebuilds and precompiled packages would allow the quality and quantity of Gentoo packages to scale higher.

Gentoo achieved their goal of creating a powerful packaging system, Portage, but does not put emphasis in other areas such as ensuring the quality of packages as other more well-rounded distributions like Debian GNU/Linux do.

3.2 Operator Agreements

Medium/large free software projects have an operators' agreement, similar to small businesses. The purpose of the agreement is to specify how developers are chosen, how conflicts will be resolved, and to describe solutions for other operational problems.

Apache's guidelines define a management committee, committers, and developers, a system for voting, and types of items that are voted on⁹. Anyone can be a developer. Interested people just need to get on the mailing list, and start submitting patches and files. If a person is judged to be a good developer for a sustained period of time, they are usually asked to become a committer. Committers have direct access to the code repositories, and integrate the patches of other developers into the code.

Not all systems are as well specified as Apache however. Gentoo Linux has a social contract¹⁰ which states that Gentoo will never depend software that is not published under an open source license (which is not the same thing as free software¹¹). Unofficially (as far as I can tell), Gentoo also has developers and committers, although they might be called different names. This lack of specification has its pros and cons. Developers have less accountability, but are more free to change things at will. A drastic example of the benefits to having a clearer specification was demonstrated June 22, 2003 when a

⁹(The Apache Software Foundation 2004)

¹⁰(Gentoo Technologies, Inc. 2003)

¹¹(Wikipedia 2004)

mistaken commit took down the portage tree, and soon affected individual end users' systems ¹².

3.3 Tools

Creating tools for collaboration and management is not a trivial task. At this time, there are a few categories of collaborative tools: version control systems, bug tracking systems, and build tools.

Some popular version control systems are RCS, CVS, and Subversion. These tools all have the ability to incrementally record changes you make to a project and provide a scheme for multiple access. It is possible to see at what time what changes were made to a project, and possibly even who made those changes. It is also possible to look at a project as it existed at any time in the past. There are some tradeoffs/caveats to providing these features, which gives rise to various version control systems.

3.4 Scalability

There is an economic concept called economies of scale. The basic question is, if you provide a system with a little bit more input, how much more output will you get out of it? It is possible that a little more input would actually result in less output. For any given method of production, there is theoretically some optimum point of production. For example, under poor management, small teams of people might succeed but large teams might fail when given similar tasks to complete (check out EE Junior Design projects for examples of this).

The beautiful thing about free software is how it is more and more successfully leveraging the abilities of many people across the world. When I tell people about the software available for the popular GNU/Linux system, they ask me when people find the time to write all this amazing software. I can only barely describe the mechanisms that make this possible, but there are many people and many systems involved in the process.

Figure 3 shows a very general overview of how software flows from developers to end users. Groups of developers, large and small, may work on a project. There are many, many, many such projects. These projects are

¹²(Welch 2003)

classified in databases and made accessible through search engines. Many of these projects are collected together and organized to form a distribution.

3.5 Jazz

When a band gets together, group dynamics can take a variety of twists. Music could be arranged before a performance, or could be invented on the fly during the performance. Mistakes made during a performance can be catastrophic if handled poorly, could pass without anyone noticing, or could even make the music more interesting.

People think of Jazz as having no rules, which has some truth. Rather, Jazz is a postmodern way of choosing whatever rules you might be interested in, while at the same time always trying “to think of something that the other guy hasn’t done”¹³. Different rules can provide different types of freedom.

Consider arranging a song before a performance. Chuck Israel makes a bold assertion that arrangement actually makes you more free than pure, unplanned improvisation. “...the reason we could be so free is that we already knew the beginning, the middle, and the ending.”¹⁴ This is really a compromise though: the freedom to change the environment of the music has been traded for the freedom associated with understanding the environment ahead of time.

On the other hand, sometimes musicians deliberately challenge each other without any warning. Charles Mingus, described as having a “healthy sense of competition”, would “bring stuff out in you that you didn’t even know was there.”¹⁵ Whole songs could be formed mostly by musicians making up something and seeing how their peers respond.

Jazz teams aren’t without their failures though. Insecure or self-interested musicians may insult other members or pretend they don’t know people they had once played with¹⁶, or people that cannot compromise their ideas may get into fights. Sometimes players aren’t as capable as their reputations made them out to be, and they might blame their mistakes on other members of the band¹⁷.

¹³(Berliner 1994, 122)

¹⁴(Berliner 1994, 289)

¹⁵(Berliner 1994, 377)

¹⁶(Berliner 1994, 52)

¹⁷(Berliner 1994, 54)

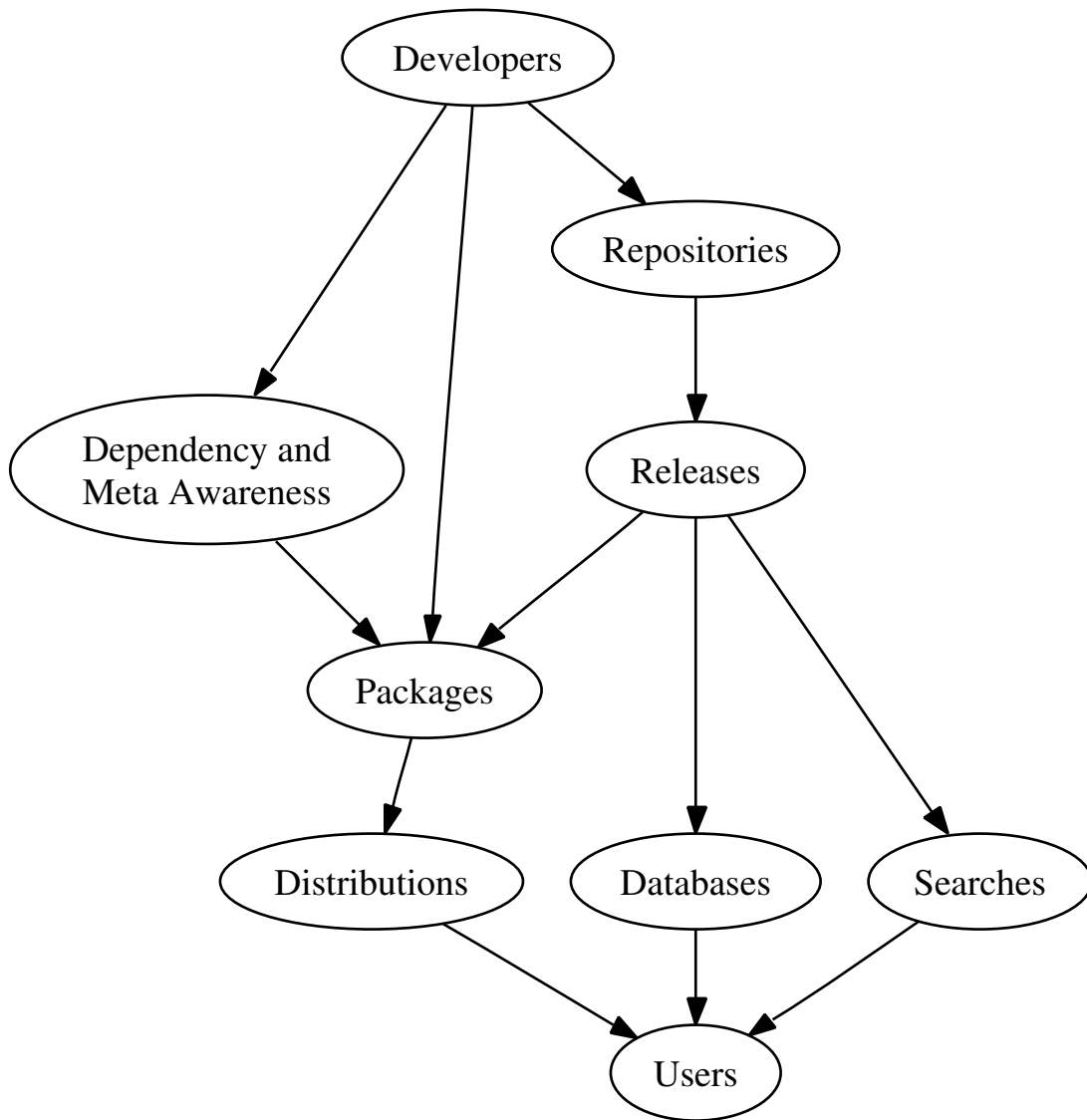


Figure 3: Simplified View of How Free Software Flows From Developers to End Users

4 Conclusion

The jazz and free software communities are similar. They both form meritocracies with interesting members. Members may start young with an interest in music or programming and play around with it at an early age. There is a minimal skill required to become a member of one of these communities, but they are otherwise not very exclusive. There are strategies to work with people, and there are cases when team dynamics can break down.

One way in which jazz and free software seem different is on the issue of scalability. I am not sure how scalability would apply to jazz, but in the case of software, it simply means taking advantage of more people to increase the rate at which new technology is produced.

References

- Berliner, P. F. (1994). *Thinking In Jazz*. The University of Chicago Press.
- Free Software Foundation, Inc. (2004). *The Free Software Definition*. web page <http://www.gnu.org/philosophy/free-sw.html>: Free Software Foundation, Inc.
- Gentoo Technologies, Inc. (2003, July). *Gentoo Linux Social Contract*. web page <http://www.gentoo.org/main/en/contract.xml>: Gentoo Technologies, Inc.
- The Apache Software Foundation (2004). *Apache HTTP Server Project Guidelines and Voting Rules*. web page <http://httpd.apache.org/dev/guidelines.html>: The Apache Software Foundation.
- Tufte, E. *PowerPoint Is Evil*. web page <http://www.wired.com/wired/archive/11.09/ppt2.html>.
- Welch, Z. T. (2003). *Reasons for Forking A Linux Distribution*. web page <http://www.zynot.org/info/fork.html>: The Zynot Foundation.
- Wikipedia (2004, April). *Free software*. web page http://en.wikipedia.org/wiki/Free_software: Wikipedia.