Fabricating Freedom: Free Software Developers at Work and Play

Jonah Bossewitch
GSOC 6023: The Political Economy of Media
Prof. Paolo Carpignano
December 22, 2007

> “Concentrate on the effect of the telegraph on ordinary ideas: the coordinates of thought, the natural attitude, practical consciousness… not through frontal assault but, rather, through the detailed investigation of a couple of sites where those effects can be most clearly observed.”
>
> James Carey[1]

Gigabytes of digital ink have been spilled analyzing the political, cultural, social, and economic forces which describe free and open source software (FOSS) projects. In this essay I will share and reflect on some of my personal experiences working on one particular free software project, and speculate on ways in which these observations may be extrapolated beyond my experiences. How are free software projects organized? What are some of the motivations that factor into participation and contribution? What kinds of activities are involved in participating in a free software project? How does producing free software differ from producing other information goods? How does the experience of working on a free software project inform and transform the participant?

Towering Changes

The events of September 11th 2001 had a dramatic impact on the world, with effects spanning the geopolitical to the personal. My life was no exception, and as a New Yorker I was deeply traumatized by the attack and its aftermath. My career also took a surprisingly sharp turn as the attack suddenly altered the local economic landscape and the job market. At the time I was looking for a new job, since the bubble that my preceding employer was riding had finally burst. The city's economy was devastated and some incredibly promising leads vaporized along with the towers. I faced some difficult choices and decided to pursue a role that I intuitively felt was a questionable fit – consulting. I did not relish the prospect of becoming a mercenary coder, as I preferred to choose where to devote my labor, ideally to organizations whose mission I respected. Furthermore, I identified myself as an artisan/craftsman and enjoyed working on sophisticated, long term projects that I could devote myself to, learning and improving my knowledge and skills along the way. I had never tried consulting and endevoured to keep an open mind about the new experience. I consoled myself with the knowledge that while I would be trading off breadth for depth, and encountering a wide variety of situations and problems, even if the encounters were shallow. While I might not get to know any project or system intimately well, the varieties of languages and constraints might make the experience worthwhile. And besides, I might just like it.

In 2001 I was hired as a software developer at a small interactive marketing company called Abstract Edge[2]. The company was founded during the Internet gold rush of the late nineties when three of the founders developed the website which helped organize and manage the *Million Mom March*[3]. Abstract Edge aspired towards advertising, marketing and strategy consulting, but at the time were essentially a web agency that built dynamic web sites for their clients. Their clients included some non-profits such as *The Gift of New York* and *Give Kids the World* as well as corporate clients including *Clairol Professional* and the New York City's *Marriott Hotels*. Abstract Edge had survived the dot-com collapse by cautiously managing their expansion and not getting drawn into the greedy mania of venture capitalism. They operated on a small profit margin, maintained a small staff with minimal overhead, and frequently lowballed their bids in response to requests for proposals.

In the summer of '02 we pitched a project with The American Legacy Foundation[4], a multi-million dollar non-profit organization founded as a part of the settlement in the class-action litigation against the tobacco industry. The American Legacy Foundation developed the successful "truth" campaign, a youth-focused anti-tobacco education campaign focusing on smoker cessation. They were planning a sister site to the truth.org, this one designed to support anti-smoking activists, to teach them how to more effectively organize and how to promote an anti-smoking message. They wanted this site to function as a community environment, enabling activism through viral social networking. While sites like these are commonplace in 2007, in 2002 the kinds of functionality they envisioned was not commonplace. This was the Web 1.0 era, in the days before the popularity of Wikipedia, MySpace, and Facebook, and sites with strong support for participatory social media were just beginning to emerge.

The American Legacy Foundation had already spent a year developing content for this environment when they sought out a technical partner to implement their concept. The had already developed a vast amount of resources for the initial site launch, and had compiled over a dozen 3-inch binders full of material, including articles, activities, news, facts, timelines, surveys, biographies. This content was dense and richly connected and they needed a system for editors to organize and administer it. Due to the controversial nature of their content, and the litigious nature of their adversaries, everything published on the site needed scientific *and* legal sign-off prior to publishing, and they wanted a system to manage this workflow. The environment also needed to support membership, profiles, discussion boards, commenting, email, synchronous chat, and workflows for submission and moderation.

Abstract Edge had initially contracted to deliver this entire environment on an incredibly compressed timeline, and as the lead technical architect on the project I knew that our previous development models were not up to the task. In the past, Abstract Edge had usually delivered custom solutions, developed in house from scratch. Software is composed of many layers, so it is a slight misnomer to talk about developing anything from scratch , but we were not leveraging platforms or frameworks that were becoming necessary to keep up with the rising bar of user expectations. The requirements for systems like these were growing in complexity as the media itself grew in popularity and importance.

Contentment Management

In the early days of the web, organizations were content with one-way mass publishing managed by a technical support staff. The importance of managing an organization's presence and message on the web continued to grow and marketing and communications personnel became very concerned that they had lost precise control over publishing. Publishing control now resided in the hands of webmasters, or worse, third party consultants who needed to be contacted and paid for every small adjustment and modification. Management began to prioritize the development of systems which allowed for non-technical administrators to organize and manage publication to the web. These systems tended to be inherently complex, since they strove to embody the hierarchy and bureaucracy of the organization they served. They were designed to capture the organization's processes and needed to model its roles and structures accordingly. These processes can be captured informally – through the cultural of use within the system, or formally – rigidly enforced by the system's explicitly defined rules.

Many organizations instinctively gravitated towards rigid enforcement and modeling, often underestimating the difficulty involved in explicitly defining these processes, and neglecting the importance of engineering flexibility into these systems so that these processes could evolve and change over time, in response to shifting needs and conditions.

As organizations desired increasingly complex publishing systems, modeled specifically on their workflows and processes, and intended to be administered and used by non-technical staff, software solutions emerged which reflected these demands. This class of systems is known as “Content Management Systems”, and high-end enterprise systems like this can be immediately traced back to the early nineties. At the turn of the millennium, enterprise content management systems were expensive, and cost upwards of millions of dollars to provision, deploy, and support. Classic users of content management systems are news sites like CNN and the New York Times, which have strict editorial workflows and a vast depth of content. But a variety of different domains can be modeled using content management tools, including brochure-ware, e-commerce, corporate intranets, educational course management, and community organizing.

As organizations began to outgrow their first-generation web sites, they sought replacements which enabled them to administer the sites themselves, without extensive technical expertise. Many homegrown content management systems were developed, but especially as blogs and wikis gained popularity, richer and more powerful interfaces were in high demand. At the same time, FOSS software aimed at this problem space were maturing and consolidating around a few prominent platforms.

Progressive Progress

The American Legacy Foundation's project was enormously complex, and I believed that approaching it using our traditional development processes carried a great risk of failure. I began to research other alternatives, examining hosted solutions, proprietary toolkits, and open source solutions. The risk of using a sophisticated framework involved the time required to become proficient with the complex concepts and constructs . As with any specialized tool, learning to use it proficiently will likely increase efficiency and productivity, but will require education and time to master.

My supervisor hesitantly agreed with my reasoning, and after a very rapid evaluation process, we selected a young and unproven open source CMS called Plone for the project[5]. We invested part of the development budget to bring in an Plone expert to jumpstart our training with the software, and hired a second freelance developer proficient in the Python programming language to to assist with the development. The fact that Plone was free software was a relevant factor in our decision, but mostly for financial reasons, not political or strategic– we did not have a good understanding of the community we were about to join and the transformative experiences would accompany them.

At the time I used open source software daily, but had not participated closely in any projects beyond asking a few questions on mailing lists. I was largely uneducated about the differences between open source licenses, and was only peripherally aware of the politics around intellectual property, free culture, and free software. I had experimented with GNU/Linux in college, and although I thought it was great that the source code was available, but was not a free software evangelist. I had been studying development methodologies and project management literature, and was very interested in alternative processes for creating better software, but from a practical perspective, not an ideological one.

As we began to settle into the rhythm of development, something was distinctly different about this effort. In the past, the open source technologies which I had used were separated by layers of abstraction below the actual problem I was trying to solve. So, I might be developing on an open source operating system (e.g. GNU/Linux), in an open source language (e.g. Perl or Python), against an open source database (e.g. PostgreSQL or MySQL) and running an open source web server (e.g. Apache), but I wasn't developing operating systems, programming languages, databases or web servers – I was developing web applications. In this case, the application we were developing was very similar to the tool we were using to develop it with – the specifics of our problem could be construed as a very thin layer on top of the underlying Plone platform. Unsurprisingly, many of the issues and challenges we faced were very similar to the issues that other members of the software community were facing, and closely resembled the problems that this project was attempting to generalize and address. Their aptitudes, interests, and goals were often very similar to our own and we found that we had a great deal in common with other members of this development community.

We began to work quite closely with the community, regularly communicating over email in IRC chats about the kinds of issues we were dealing with and how others had attempted to solve them. The immensity of our challenge suggested automating certain mundane operations that we would likely have tackled manually with brute force had our project been a little smaller. When we described the infrastructure we were developing to automate certain aspects of our development, there was a very keen interest in our solution. The freelancer we had hired turned out to be quite brilliant and had also had personal experience participating in open source development. We shared some of the software we were working on, and other developers began to work with it, and found it clever and useful. We were invited to a small workshop, also known as a “sprint”, that the developers of this project were organizing in Rotterdam to share our progress and participate in the ongoing development of the next iteration of the platform.

Our company was not in the habit of sending the developers on business trips, and it was hard for management to understand how they would directly benefit from sending us to this event. We explained to them the advantages of participating in a peer production effort, recapitulating many of the (now) standard arguments explaining the self-interested benefits of participating in open source development. In addition to the efficiency of having multiple developers test and debug our software, we also made the case that capturing developer mindshare was a valuable long-term investment for the agency, as we could steer the community towards solving problems where we held stakes. Management grudgingly gave way and agreed to sponsor our participation. I distinctly recall my co-workers determination to attend the event, mentioning that if our employers decided not to send us, he would use his vacation days and attend at his own expense. I remember thinking how strange it sounded to consider using vacation days for a work-related event, but then I had never attended a sprint or a developers conference and did not fully appreciate their appeal.

A “sprint” (a.k.a. “hackathon”) is a multi-day focused development session, inspired by a practice described by the agile development methodology “extreme programming.”[6] Unlike a traditional conference there are no planned speakers or formal talks, although there are plenty of impromptu training sessions and short presentations. Instead, participants self-organize and form a consensus around a few small projects they can accomplish in a the alloted time. They work intensely in pairs or small teams using the pair programming approach, also advocated by the extreme programming development philosophy. Experienced developers often pair up with less experienced developers, and there is usually a coach leading the session, and helping to set the agenda and track activities on a whiteboard. These in-person meetings are important occasions, where development on the project is advanced, leadership in the community is established, development approaches are shared and exchanged, and architectural and strategic planning that is sometimes difficult to work out asynchronously can discussed. Sprints are also very social, as the intense work ethic gives way to an intense play ethic. Sprinters will often code late into the evening, and then stay out socializing late into the morning.

Roderdam was an incredible adventure, and I met many independent developers who had been involved with the project for years. The Plone community attracts many fiercely independent entrepreneurs, and represents a particular blend of hybrid economies. The developers and companies participating in the community rarely compete directly against each other. Rather, they compete against other platforms that occupy the same solution space, and they regularly compete against these formidable platforms in pitches. By targeting larger and more complex engagements, the platform continues to grow, rather than cannibalizing itself with squabbles, forks, and rebranding.

The Plone community is also very reflective and self-conscious about its image and processes, and constantly seeking to refine the way it grows. New features are added to the project through a mixture of consensual prioritizing and client needs. Clients sometimes unwittingly underwrite the creation of a generalized new feature, which they happen to need first, but this is still economically fair and rational, since they are leveraging dozens of other features that were, in turn, developed and shared under the same arrangement. This arrangement is made formal through legal and cultural structures that enforce and encourage this style of cooperation. Community members even devote themselves to branding and marketing the project, as greater recognition of the software is better for all of the participants in the Plone ecology. The experience expanded my thinking about the possibilities of a software career, and taught me a great deal about the appeal of working long-term on a FOSS project. In today's fragile and uncertain employment market, the identity and cohesion that a projects like Plone offer to developers is one way to stitch together a post-modern career in a post-geographic, global economy.

The Ouroboros of Freedom

Software projects like Plone are often misunderstood by focusing exclusively on their underlying technological platform. A better representation comes from considering the larger ecology in which the platform is embedded. As Paul Everitt, founder of ZEA Partners and executive director of the Plone Foundation phrases it “The software is an artifact of the community.” A fuller elaboration of a technology project’s ecology takes into account the platform, the community, and the processes that bind them together. This ecological model of software projects incorporates the dynamic lifecycle of the project over time, and provides insight into how a project might behave under complex, unanticipated circumstances.

Plone environments typically present specific affordances to people using the software, many in the form of innovations and decisions that the Plone community decided to incorporate into it. The community includes vendors (the various individuals, organizations, corporations and universities participating in the development of the platform), clients (individuals, corporations, non-profits contracting the vendors to create Plone-based solutions) and users (the administrators, editors, members and visitors of the client's environment's). These diverse communities of participants are connected to each other through formal and informal structures and processes ranging from legal entities and contracts to technically mediated mailing lists and collaborative cyberspaces. The Plone project did not attempt to single-handedly create the rules and processes which constitute this ecological model. Instead, they built upon the edifice of the free software movement, using many of the tools and practices found within those projects, infusing their project with flavors of the culture they built upon.

FOSS ecologies have been a breeding ground for experimenting with various models of structure and governance which promote constructionist learning within the community. Since writing software is an act of creative expression, it is predictable that the artifacts created by a software community capture the values of that community through the inclusion (and omission) of features and the metaphors used in the software they create. The recursive questioning of meta-structures is a habitual pattern of programmer's thinking, and it is no surprise to see this analytical gaze turned back on itself. The community's proximity to the architecture of their own communication channels encourages a reflexive attitude towards their own communicative superstructure.

“Eating your own dogfood” is a popular saying in the technology sector that is used to describe a project that consumes its own product. FOSS projects regularly consume other FOSS products, creating a feedback loop that reinforces the processes and values understood by those communities. The software that manages code repositories and bug-tracking systems incorporates styles of collaboration, consensus building, decision making, and conflict resolution which percolate throughout the community as a whole. Cultural practices around mailing lists and group chats bring together the people with ideas and the people who can act on them. Wikis and self-organized conferences encourage autonomously motivated action and peer-production, not usually in response to authority or hierarchy. FOSS projects are organized for community and access to knowledge, not market-driven production and selling. Their tools embody and reflect these priorities. You could say that they eat their own dogfood on process, and that they are what they eat.

Smoke Break

Back in New York City, our anti-tobacco project was suffering from delays and setbacks all too common in the software world. But, as Abstract Edge began to pitch new projects, it became apparent that Plone was a powerful tool for approaching a wide range of efforts.

As a developer, working as a mercenary can be a devastating for morale. In most situations, your work is by definition a one-off or throw-away, meant to be written once never to be returned to. The pressure to deliver a solution quickly often comes at the expense of careful design, and the details under the hood are nothing to be proud of. There is very little continuity between engagements, and individual projects rarely provide opportunities to develop a sophisticated infrastructure, or to attend to the aesthetic sensibilities of creating beautiful software or writing elegant code. Software development is a form of production whose characteristics bear a strong resemblance to traditional craftsmanship. The metaphors which dominate the activity revolve around objects, constructions, and fabrication and experts are regarded as virtuosos.

Like other forms of material fabrication, writing software requires imagination, invention, and ingenuity. While there are ongoing attempts to automate and commodify aspects of software production, it is still fundamentally a creative act which relies on human judgment and intuition, and increasingly, teamwork and collaboration. In an era when software environments mediate the dynamics of human interaction, the designation of software architecture is more than just an analogy. Software has come to resemble traditional architecture as a leading art, and many programmers venerate their profession, guarding their integrity and regarding the products of their labor as important and influential. For individuals who care about their products of their labor, shoddy craftsmanship is disappointing and demoralizing. There are some styles of consulting where artistry and attention to detail is valued, but in many situations cutting corners in the standard. With excellent project management, trust can be established between the client and the agency, but it is difficult to foster a dynamic which embodies good faith.

Participating in an open source project allowed us to transcend the limitations and constraints of the oppressive conditions of mercenary work, and escape into a world of self-improvement, creative expression, and reflexive design. The software functioned as an intermediate object of collaboration, cutting across the boundaries of space and time. In introduced continuity between our projects, and allowed us to collaborate with other developers and companies without incurring burdensome bureaucratic overhead.

As a independent freelancer, or an employee of a small company it is difficult to find peers or colleges to learn from. Even accolades from a superior are hollow compared to the honest criticism and respect from experienced professionals. There is little an active learner appreciates more than having their work vetted by someone whose skills and experience they honor and respect. Open source communities offer precisely this sort of learning environment, with motivational dynamics that are similar to the academic world. Publishing a piece of code, having a patch accepted, and gaining commit privileges to project's repository are forms of peer review that recognize and validate a programmer's skills and accomplishments.

Blurred Borders

Beyond the social and reputation capital earned and exchanged on FOSS projects, programmers enjoy a great deal of personal satisfaction and fun in their work. Many FOSS developers are technical hobbyists, and their involvement resides at the juncture between work and play. Many of the mundane and annoying demands of client is the kind of work that only the perverse would voluntarily subject themselves to. However, the typical work on a free software project is composed of professional socializing and communication, as well as the construction of elegant and harmonious environments, and these activities can be incredibly rewarding. Contrary to popular myths, many free software developers earn a respectable wage working on these projects, either through direct services, or through an employer who sponsors their work. They are also building stable careers, and sometimes businesses, around their respective watering holes. Even most ideologically committed free software projects, such as Debian and Wikipdia, pay members wages to carry out key functions that have been empirically determined to suffer when left to voluntary contributions.

After Rotterdamn, many more sprints and conferences followed. Plone's international emphasis was somewhat unique, and its strong multilingual support helped create a positive feedback loop. European governments and municipalities frequently selected Plone since funding Plone development meant that their investment in projects remained local, as opposed to lining the coffers of companies like Microsoft, IBM, or Sun. This, in turn, helped improve Plone's multilingual support, which led to its widespread international adoption and uptake. We traveled to locations like Berne, Vienna, and Naples, and even participated in a sprint held in an Austrian Castle. I never would have guessed that becoming a software developer would take me backpacking across Europe in my late twenties, or that I could convince my employer to finance these excursions. Many of my fellow attendees were independent freelancers, funding themselves to attend these gatherings.

The gatherings were incredible networking opportunities, and occupied a novel place between work and play. They were part work, part vacation. Often participants would attach traditional vacation time before or after the event, and event organizers often incorporate local sightseeing and festivities. The community functioned as the hub of a diverse network of sectors. Corporations, non-profits, governments, and educational sectors all encountered each other through the intermediary of this project's ecology. We were gallivanting around exotic locations, but the value proposition of these meetings was real and meaningful. Production was was happening, but it was self-organized, self-determined, and self-satisfying.

While trade and academic conferences are traditional activities for some professionals, FOSS programmers had inflected their own personality and style on the events. The variety of formats at free software conferences usually conforms to their non-hierarchical, do-it-yourself, ethic. Nobody reads papers to each other, there are typically ad-hoc “birds of a feather” sessions organized on the fly, plenty of unstructured time for conversations, and everybody's favorite, the “lighting talks” – where each presenter has 5 minutes to present an idea or a work in progress. This creates many different kinds of spaces and dynamics for interactions between participants. In contrast to stereotypically stuffy professional conference, free software conferences have reputations as fun events, not to be missed.

The social dynamic established within the Plone community reminded me of a medieval guild or a modern American college fraternity. In addition to the incredibly unbalanced gender ratio, participants often adopted nicknames, went through initiatory hazing rituals, and were motivated by peer pressure to contribute to the progress of the imagined community. Especially since developers regularly transmit communications using electronic media, these punctuated opportunities to interact in-person facilitated ritual communication, strongly enforcing cultural practices and social bonds.

The sprints were also very productive, as an opportunity to focus on a self-selected challenge, without the typical interruptions of an office setting. The conditions were conducive to the state of “flow”, allowing developers to get into “the zone”, an essential precondition for developer productivity[7]. This psychological state is characterized by energy, concentration, timelessness, and immersion and is thought to emerge with appropriate balance between challenge and skill. Programming is the kind of activity which is extremely well suited for eliciting flow, especially under the right environmental conditions.

My co-worker became more and more involved in the project, to the point where his work for Abstract Edge became a transient activity which filled the time between conferences and FOSS work. Instead of splitting our development responsibilities down the middle, I began to run interference for him with management, in order to allow him to to work on abstract infrastructural problems. I was responsible for the specific client requests, and he was working on generalized and abstract tools that I would use to solve the specifics. The arrangement resembled research and development at a large corporation, but we were at a very small agency, and our labor was all supposed to be billable.

When I finally left Abstract Edge, I was able to take my software and community of colleges and friends along with me to my new job. There is distinct irony in the logic that the best way to insure continuity across jobs is to make sure that nobody owns the intellectual products of your labor. Under most employment arrangements, an employee does not own the software written while working. But, if the software is released under free license, it can continue to be used and developed even after leaving. From this perspective, writing open software on employer's payroll is a powerful act of resistance. But as more companies are reorienting their software businesses as services, and the entire industry transitions from selling software artifact to selling software services, the liberation of software is economically rational, and resistance is becoming the norm. Companies are beginning to recognize that owning software is a liability not an asset, but for individuals, it is part of their personal portfolio and a record of their past accomplishments.

The Specter of dotCommunism

Some members of the Plone community recently organized a campaign to publicly pledge some of their disposable labor to the project. The “10% Plone Manifesto[8]” is an explicit attempt to exert control over the direction that the platform grows, and the kinds of problems that developers devote their time to. The introduction to the manifesto explains:

> “In the last years many of us have started Plone consulting or development companies with roots in the Plone community. To most of us it was a dream come true — being able to make a living from writing free software and working with this great community at the same time.
>
> But... time takes its toll, and everyday life, mortgages, money, administration and having to earn to feed the kids, all push us further into the tunnel of dull office-people rather than cool free software developers.

> Sure, we publish free software, more than just occasionally, but most development is steered by what our clients need, not what we think Plone needs or what we want to build.
>
> **Take control!**
>
> We want to stay ahead of the curve. We want to stay updated. We want to keep contributing, even in a world of building software for a living. Inspired by the practices of the amazing Google, we came up with our own preemptive strike; The 10% Plone Manifesto."

This declaration captures the sentiment that this community is trying to preserve. Independence and self-determination are core values for these developers and they are attempting to reclaim control over the type of software they develop. As the language in this Manifesto alludes, many of them consider themselves to be part of a peaceful revolution. They are struggling to overthrow and disrupt the dominant powers and hierarchies, carving out a benevolent and enlightened alternative, all while making mortgage payments and raising a family. .

The Programmers' Condition

As this essay suggests, much of the participation in an FOSS project goes beyond the work of Hannah Arendt's *homo faber*, and clearly encompasses her formulation of action.[9] Action is manifest in the capacity of participants to initiate beginnings, and to publicly assert their identity in communicative acts towards a public society. For Arendt, these actions are intrinsically political since they allow to participants exercise their agency through speech and persuasion. Much of the educational value, and perhaps some of the individual appeal of participating in FOSS projects comes from realizing these higher forms of *viva activa*.

The experience of sharing and participating in a vital community have the power to mobilize individuals to action by exposing them to real politics and civic engagement, in Arendt's sense of the terms. FOSS projects can help foster self-determination, and allow individuals to explore variations on democratic forms of governance that are vital to our emerging societies. These experiences translate easily and directly into traditional forms of grasssroots activism and it is quite commonplace for FOSS participation to function as an indoctrination (usually with the "gateway" movements of free software and free culture), teaching participants to practically engage directly in politics and personally taste the fruit of political action.

For me, the experience of working closely on a free software project was highly transformative – cognitively, personally, emotionally, and socially. Not only was participation in this project technically instructive, but it educated me on issues of ethics and governance, and raised my political consciousness. I made lifelong friends, became concerned and engaged with pressing issues of social justice, and felt empowered to improve my life's conditions. While my experiences are distinct and unique, this pattern of awakening is not entirely uncommon. My experiences have demonstrated to me that these projects are about much more than just technology, or even social engineering and architecture – they are also petri dishes of civic activation, engagement and maintenance whose form should be closely studied.

The American educational system stopped trying to teach people to share in kindergarten, whether or not everybody learned the lesson. The direct experience of participating in common-based peer-production validates the counter-intuitive proposition that sharing is not necessarily altruistic, as it can support self-interests and result in greater value and wealth. With cooperation and trust, everybody can gain a smaller piece of a much larger pie, netting an absolutely greater value than selfish competition. Yochai Benkler's treatise, *The Wealth of Networks*[10] makes a strong analytic argument for this economic reality, but many people learn best through direct experience. Some will continue to suspect his arguments until their own experiences lend supporting evidence to his theory.

Learning how to share is much more difficult lesson than most people acknowledge, and free software developers sometimes demonstrate this ethic in humorous ways. They share beer, cigarettes, and even joke about sharing sexual partners, all with the knowing “open source” refrain. They believe that they are struggling to construct a common good, one that will transcend the particular project or client that consumes their day-to-day subsistence. They are redefining the boundaries between work and play, work and action, and fabrication and communication. They are a part of a movement, but they are also continually starting movements. One character at a time.

1 Carey, James W. “Technology and Ideology: The Case of the Telegraph” in his *Communications as Culture* (Winchester: Unwin-Hyman, 1989), p. 210.
2 <http://abstractedge.com>
3 <http://www.millionmommarch.org/>
4 <http://www.americanlegacy.org>
5 <http://plone.org>
6 See <http://www.extremeprogramming.org/> for an introduction.
7 Csikszentmihalyi, Mihaly (1990). *Flow: The Psychology of Optimal Experience*. (New York: Harper and Row).
8 Geir Bækholt, The 10% Plone Manifesto <http://www.jarn.com/blog/the-10-plone-manifesto> (December 16, 2007)
9 Arendt, Hannah. (1958). *The Human Condition.* (Chicago: University of Chicago Press).
10 Yochai Benkler, (2006). *The Wealth of Networks: How SocialProduction Transforms Markets and Freedom.* (New Haven, CT: Yale University Press.