Homesteading the Noosphere

by Eric S. Raymond April 1998

After observing a contradiction between the 'official' ideology defined by open-source licenses and the actual behavior of hackers, we examine the actual customs which regulate the ownership and control of open-source software. We discover that they imply an underlying theory of property rights homologous to the Lockean theory of land tenure. We relate that to an analysis of the hacker culture as a 'gift culture' in which participants compete for prestige by giving time, energy, and creativity away. We then examine the implications of this analysis for conflict resolution in the culture, and develop some prescriptive implications.

Contents

1 An Introductory Contradiction

Anyone who watches the busy, tremendously productive world of Internet open-source software for a while is bound to notice an interesting contradiction between what open-source hackers say they believe and the way they actually behave – between the official ideology of the open-source culture and its actual practice.

Cultures are adaptive machines. The open-source culture is a response to an identifiable set of drives and pressures. As usual, the culture's adaptation to its circumstances manifests both as conscious ideology and as implicit, unconscious or semi-conscious knowledge. And, as is not uncommon, the unconscious adaptations are partly at odds with the conscious ideology.

In this paper, we will dig around the roots of that contradiction, and use it to discover those drives and pressures. We will deduce some interesting things about the hacker culture and its customs. We will conclude by suggesting ways in which the culture's implicit knowledge can be leveraged better.

Labels in square brackets refer to the bibliography/endnotes.

2 The Varieties of Hacker Ideology

The ideology of the Internet open-source culture (what hackers say they believe) is a fairly complex topic in itself. All members agree that open source (that is, software which is freely redistributable and can readily be evolved and modified to fit changing needs) is a good thing and worthy of significant and collective effort. This agreement effectively defines membership in the culture. However, the reasons individuals and various subcultures give for this belief vary considerably.

One degree of variation is zealotry; whether open source development is regarded merely as a convenient means to an end (good tools and fun toys and an interesting game to play) or as an end in itself.

A person of great zeal might say "Free software is my life! I exist to create useful, beautiful programs and information resources, and then give them away." A person of moderate zeal might say "Open source is a good thing which I am willing to spend significant time helping happen". A person of little zeal might say "Yes, open source is OK sometimes. I play with it and respect people who build it".

Another degree of variation is in hostility to commercial software and/or the companies perceived to dominate the commercial software market.

A very anticommercial person might say "Commercial software is theft and hoarding. I write free software to end this evil." A moderately anticommercial person might say "Commercial software in general is OK because programmers deserve to get paid, but companies that coast on shoddy products and throw their weight around are evil." An un-anticommercial person might say "Commercial software is OK, I just use and/or write open-source software because I like it better".

All nine of the attitudes implied by the cross-product of the above categories are represented in the open-source culture. The reason it is worthwhile to point out the distinctions is because they imply different agendas, and different adaptive and cooperative behaviors.

Historically, the most visible and best-organized part of the hacker culture has been both very zealous and very anticommercial. The Free Software Foundation founded by Richard M. Stallman (RMS) supported a great deal of open-source development from the early 1980s on, including tools like Emacs and GCC which are still basic to the Internet open-source world, and seem likely to remain so for the forseeable future.

For many years the FSF was the single most important focus of open-source hacking, producing a huge number of tools still critical to the culture. The FSF was also long the only sponsor of open source with an institutional identity visible to outside observers of the hacker culture. They effectively defined the term 'free software', deliberately giving it a confrontational weight (which the newer label '*open source* `<http://www.opensource.org>`' just as deliberately avoids).

Thus, perceptions of the hacker culture from both within and outside it tended to identify the culture with the FSF's zealous attitude and perceived anticommercial aims (RMS himself denies he is anticommercial, but his program has been so read by most people, including many of his most vocal partisans). The FSF's vigorous and explicit drive to "Stamp Out Software Hoarding!" became the closest thing to a hacker ideology, and RMS the closest thing to a leader of the hacker culture.

The FSF's license terms, the "General Public Licence" (GPL), expresses the FSF's zealous and anticommercial attitudes. It is very widely used in the open-source world. North Carolina's Sunsite is the largest and most popular software archive in the Linux world. In July 1997 about half the Sunsite software packages with explicit license terms used GPL.

But the FSF was never the only game in town. There was always a quieter, less confrontational and more market-friendly strain in the hacker culture. The pragmatists were loyal not so much to an ideology as to a group of engineering traditions founded on early open-source efforts which predated the FSF. These traditions included, most importantly, the intertwined technical cultures of Unix and the pre-commercial Internet.

The typical pragmatist attitude is only moderately anticommercial, and its major grievance against the corporate world is not 'hoarding' per se. Rather it is that world's perverse refusal to adopt superior approaches incorporating Unix and open standards and open-source software. If the pragmatist hates anything, it is less likely to be 'hoarders' in general than the current King Log of the software establishment; formerly IBM, now Microsoft.

To pragmatists, the GPL is important as a tool rather than an end in itself. Its main value is not as a weapon against 'hoarding', but as a tool for encouraging software sharing and the growth of *bazaar-mode* `<http://www.tuxedo.org/~esr/writings/cathedral-bazaar>` development communities. The pragmatist values having good tools and toys more than he dislikes commercialism, and may use high-quality commercial software without ideological discomfort. At the same time, his open-source experience has taught him standards of technical quality that very little closed software can meet.

For many years, the pragmatist point of view expressed itself within the hacker culture mainly as a stubborn current of refusal to completely buy into the GPL in particular or the FSF's agenda in general. Through the 1980s and early 1990s, this attitude tended to be associated with fans of Berkeley Unix, users of the BSD license, and the early efforts to build open-source Unixes from the BSD source base. These efforts, however, failed to build bazaar communities of significant size, and became seriously fragmented and ineffective.

Not until the Linux explosion of early 1993-1994 did pragmatism find a real power base. Although Linus Torvalds never made a point of opposing RMS, he set an example by looking benignly on the growth of a commercial Linux industry, by publicly endorsing the use of high-quality commercial software for specific tasks, and by gently deriding the more purist and fanatical elements in the culture.

A side effect of the rapid growth of Linux was the induction of a large number of new hackers for which Linux was their primary loyalty and the FSF's agenda primarily of historical interest. Though the newer wave of Linux hackers might describe the system as "the choice of a GNU generation", most tended to emulate Torvalds more than Stallman.

Increasingly it was the anticommercial purists who found themselves in a minority. How much things had changed would not become apparent until the Netscape announcement in February 1998 that it would distribute Navigator 5.0 in source.

This excited more interest in 'free software' within the corporate world. The subsequent call to the hacker culture to exploit this unprecedented opportunity and to re-label its product from 'free software' to 'open source' was met with a level of instant approval that surprised everybody involved.

In a reinforcing development, the pragmatist part of the culture was itself becoming polycentric by the mid-1990s. Other semi-independent communities with their own self-consciousness and charismatic leaders began to bud from the Unix/Internet root stock. Of these, the most important after Linux was the Perl culture under Larry Wall. Smaller, but still significant, were the traditions building up around John Osterhout's Tcl and Guido Van Rossum's Python languages. All three of these communities expressed their ideological independence by devising their own, non-GPL licensing schemes.

3 Promiscuous Theory, Puritan Practice

Through all these changes, nevertheless, there remained a broad consensus theory of what 'free software' or 'open source' is. The clearest expression of this common theory can be found in the various open-source licenses, all of which have crucial common elements.

In 1997 these common elements were distilled into the Debian Free Software Guidelines, which became the *Open Source Definition* `<http://www.opensource.org>`. Under the guidelines defined by the OSD, an open-source license must protect an unconditional right of any party to modify (and redistribute modified versions of) open-source software.

Thus, the implicit theory of the OSD (and OSD-conformant licenses such as the GPL, the BSD license, and Perl's Artistic License) is that anyone can hack anything. Nothing prevents half a dozen different people from taking any given open-source product (such as, say the Free Software Foundations's gcc C compiler), duplicating the sources, running off with them in different evolutionary directions, but all claiming to be *the* product.

In practice, however, such 'forking' almost never happens. Splits in major projects have been rare, and always accompanied by re-labeling and a large volume of public self-justification. It is clear that, in such cases as the GNU Emacs/XEmacs split, or the gcc/egcs split, or the various fissionings of the BSD splinter groups, that the splitters felt they were going against a fairly powerful community norm.

In fact (and in contradiction to the anyone-can-hack-anything consensus theory) the open-source culture has an elaborate but largely unadmitted set of ownership customs. These customs regulate who can modify software, the circumstances under which it can be modified, and (especially) who has the right to redistribute modified versions back to the community.

The taboos of a culture throw its norms into sharp relief. Therefore, it will be useful later on if we summarize some important ones here.

- There is strong social pressure against forking projects. It does not happen except under plea of dire necessity, with much public self-justification, and with a renaming.
- Distributing changes to a project without the cooperation of the moderators is frowned upon, except in special cases like essentially trivial porting fixes.
- Removing a person's name from a project history, credits or maintainer list is absolutely *not done* without the person's explicit consent.

In the remainder of this paper, we shall examine these taboos and ownership customs in detail. We shall inquire not only into how they function but what they reveal about the underlying social dynamics and incentive structures of the open-source community.

4 Ownership and Open Source

What does 'ownership' mean when property is infinitely reduplicable, highly malleable, and the surrounding culture has neither coercive power relationships nor material scarcity economics?

Actually, in the case of the open-source culture this is an easy question to answer. The owner(s) of a software project are those who have the exclusive right, recognized by the community at large, to *re-distribute modified versions*.

(In discussing ‘ownership’ in this section I will use the singular, as though all projects are owned by some one person. It should be understood, however, that projects may be owned by groups. We shall examine the internal dynamics of such groups later in this paper.)

According to the standard open-source licenses, all parties are equals in the evolutionary game. But in practice there is a very well-recognized distinction between ‘official’ patches, approved and integrated into the evolving software by the publicly recognized maintainers, and ‘rogue’ patches by third parties. Rogue patches are unusual, and generally not trusted 20 ([RP]).

That *public* redistribution is the fundamental issue is easy to establish. Custom encourages people to patch software for personal use when necessary. Custom is indifferent to people who redistribute modified versions within a closed user or development group. It is only when modifications are posted to the open-source community in general, to compete with the original, that ownership becomes an issue.

There are, in general, three ways to acquire ownership of an open-source project. One, the most obvious, is to found the project. When a project has only one maintainer since its inception and the maintainer is still active, custom does not even permit a *question* as to who owns the project.

The second way is to have ownership of the project handed to you by the previous owner (this is sometimes known as ‘passing the baton’). It is well understood in the community that project owners have a duty to pass projects to competent successors when they are no longer willing or able to invest needed time in development or maintenance work.

It is significant that in the case of major projects, such transfers of control are generally announced with some fanfare. While it is unheard of for the open-source community at large to actually interfere in the owner’s choice of succession, customary practice clearly incorporates a premise that public legitimacy is important.

For minor projects, it is generally sufficient for a change history included with the project distribution to note the change of ownership. The clear presumption is that if the former owner has not in fact voluntarily transferred control, he or she may reassert control with community backing by objecting publicly within a reasonable period of time.

The third way to acquire ownership of a project is to observe that it needs work and the owner has disappeared or lost interest. If you want to do this, it is your responsibility to make the effort to find the owner. If you don’t succeed, then you may announce in a relevant place (such as a Usenet newsgroup dedicated to the application area) that the project appears to be orphaned, and that you are considering taking responsibility for it.

Custom demands that you allow some time to pass before following up with an announcement that you have declared yourself the new owner. In this interval, if someone else announces that they have been actually working on the project, their claim trumps yours. It is considered good form to give public notice of your intentions more than once. More points for good form if you announce in many relevant forums (related newsgroups, mailing lists); and still more if you show patience in waiting for replies. In general, the more visible effort you make to allow the previous owner or other claimants to respond, the better your claim if no response is forthcoming.

If you have gone through this process in sight of the project’s user community, and there are no objections, then you may claim ownership of the orphaned project and so note in its history file. This, however, is less secure than being passed the baton, and you cannot expect to be considered fully legitimate until you have made substantial improvements in the sight of the user community.

I have observed these customs in action for twenty years, going back to the pre-FSF ancient history of open-source software. They have several very interesting features. One of the most interesting is that most hackers have followed them without being fully aware of doing so. Indeed, the above may be the first conscious and reasonably complete summary ever to have been written down.

Another is that, for unconscious customs, they have been followed with remarkable (even astonishing) consistency. I have observed the evolution of literally hundreds of open-source projects, and I can still count the number of significant violations I have observed or heard about on my fingers.

Yet a third interesting feature is that as these customs have evolved over time, they have done so in a consistent direction.

That direction has been to encourage more public accountability, more public notice, and more care about preserving the credits and change histories of projects in ways which (among other things) establish the legitimacy of the present owners.

These features suggest that the customs are not accidental, but are products of some kind of implicit agenda or generative pattern in the open-source culture that is utterly fundamental to the way it operates.

An early respondent pointed out that contrasting the Internet hacker culture with the cracker/pirate culture (the "warez d00dz" centered around game-cracking and pirate bulletin-board systems) illuminates the generative patterns of both rather well. We'll return to the d00dz for contrast later in the paper.

5 Locke and Land Title

To understand this generative pattern, it helps to notice a historical analogy for these customs that is far outside the domain of hackers' usual concerns. As students of legal history and political philosophy may recognize, the theory of property they imply is virtually identical to the Anglo-American common-law theory of land tenure!

In this theory, there are three ways to acquire ownership of land.

On a frontier, where land exists that has never had an owner, one can acquire ownership by *homesteading*, mixing one's labor with the unowned land, fencing it, and defending one's title.

The usual means of transfer in settled areas is *transfer of title*, that is receiving the deed from the previous owner. In this theory, the concept of 'chain of title' is important. The ideal proof of ownership is a chain of deeds and transfers extending back to when the land was originally homesteaded.

Finally, the common-law theory recognizes that land title may be lost or abandoned (for example, if the owner dies without heirs, or the records needed to establish chain of title to vacant land are gone). A piece of land that has become derelict in this way may be claimed by *adverse possession* – one moves in, improves it, and defends title as if homesteading.

This theory, like hacker customs, evolved organically in a context where central authority was weak or nonexistent. It developed over a period of a thousand years from Norse and Germanic tribal law. Because it was systematized and rationalized in the early modern era by the English political philosopher John Locke, it is sometimes referred to as the 'Lockean' theory of property.

Logically similar theories have tended to evolve wherever property has high economic or survival value and no single authority is powerful enough to force central allocation of scarce goods. This is true even in the hunter-gatherer cultures that are sometimes romantically thought to have no concept of 'property'. For example, in the traditions of the !Kung San bushmen of the Kalahari Desert, there is no ownership of hunting grounds. But there *is* ownership of water-holes and springs under a theory recognizably akin to Locke's.

The !Kung San example is instructive, because it shows that Lockean property customs arise only where the expected return from the resource exceeds the expected cost of defending it. Hunting grounds are not property because the return from hunting is highly unpredictable and variable, and (although highly prized) not a necessity for day-to-day survival. Waterholes, on the other hand, are vital to survival and small enough to defend.

The 'noosphere' of this paper's title is the territory of ideas, the space of all possible thoughts 20 ([N]). What we see implied in hacker ownership customs is a Lockean theory of property rights in one subset of the noosphere, the space of all programs. Hence 'homesteading the noosphere', which is what every founder of a new open-source project does.

Fare Rideau <rideau@ens.fr> correctly points out that hackers do not exactly operate in the territory of pure ideas. He asserts that what hackers own is *programming projects* – intensional focus points of material labour (development, service, etc), to which are associated things like reputation, trustworthiness, etc. He therefore asserts that the space spanned by hacker projects, is *not* the noosphere but a sort of dual of it, the space of noosphere-exploring program projects. (With a nod to the astrophysicists out there, it would be etymologically correct to call this dual space the 'ergosphere' or 'sphere of work'.)

In practice, the distinction between noosphere and ergosphere is not important for the purposes of this paper. It is dubious whether the 'noosphere' in the pure sense Fare insists on can be said to exist in any meaningful way; one would almost have to be a Platonist philosopher to believe in it. And the distinction between noosphere and ergosphere is only of *practical* importance if one wishes to assert that ideas (the elements of the noosphere) cannot be owned, but their instantiations as projects can. This question leads to issues in the theory of intellectual property which are beyond the scope of this paper.

To avoid confusion, however, it is important to note that neither the noosphere nor the ergosphere is the same as the totality of virtual locations in electronic media that is sometimes (to the disgust of most hackers) called 'cyberspace'. Property there is regulated by completely different rules that are closer to those of the material substratum – essentially, he who owns the media and machines on which a part of 'cyberspace' is hosted owns that piece of cyberspace as a result.

The Lockean structure suggests strongly that open-source hackers observe the customs they do in order to defend some kind of expected return from their effort. The return must be more significant than the effort of homesteading projects, the cost of maintaining version histories that document 'chain of title', and the time cost of doing public notifications and a waiting period before taking adverse possession of an orphaned project.

Furthermore, the 'yield' from open source must be something more than simply the use of the software, something else that would be compromised or diluted by forking. If use were the only issue, there would be no taboo against forking, and open-source ownership would not resemble land tenure at all. In fact, this alternate world (where use is the only yield) is the one implied by existing open-source licenses.

We can eliminate some candidate kinds of yield right away. Because you can't coerce effectively over a network connection, seeking power is right out. Likewise, the open-source culture doesn't have anything much resembling money or an internal scarcity economy, so hackers cannot be pursuing anything very closely analogous to material wealth.

There is one way that open-source activity can help people become wealthier, however – a way that provides a valuable clue to what actually motivates it. Occasionally, the reputation one gains in the hacker culture can spill over into the real world in economically significant ways. It can get you a better job offer, or a consulting contract, or a book deal.

This kind of side effect, however, is at best rare and marginal for most hackers; far too much so to make it convincing as a sole explanation, even if we ignore the repeated protestations by hackers that they're doing what they do not for money but out of idealism or love.

However, the way such economic side-effects are mediated is worth examination. Below we'll see that an understanding of the dynamics of reputation within the open-source culture *itself* has considerable explanatory power.

6 The Hacker Culture as Gift Economy

To understand the role of reputation in the open-source culture, it is helpful to move from history further into anthropology and economics, and examine the difference between *exchange cultures* and *gift cultures*.

Human beings have an innate drive to compete for social status; it's wired in by our evolutionary history. For the 90% of that history that ran before the invention of agriculture, our ancestors lived in small nomadic hunting-gathering bands. High-status individuals got the healthiest mates and access to the best food. This drive for status expresses itself in different ways, depending largely on the degree of scarcity of survival goods.

Most ways humans have of organizing are adaptations to scarcity and want. Each way carries with it different ways of gaining social status.

The simplest way is the *command hierarchy*. In command hierarchies, allocation of scarce goods is done by one central authority and backed up by force. Command hierarchies scale very poorly 20 ([Mal]); they become increasingly brutal and inefficient as they get larger. For this reason, command hierarchies above the size of an extended family are almost always parasites on a larger economy of a different type. In command hierarchies, social status is primarily determined by access to coercive power.

Our society is predominantly an *exchange economy*. This is a sophisticated adaptation to scarcity that, unlike the command model, scales quite well. Allocation of scarce goods is done in a decentralized way through trade and voluntary cooperation (and in fact, the dominating effect of competitive desire is to produce cooperative behavior). In an exchange economy, social status is primarily determined by having control of things (not necessarily material things) to use or trade.

Most people have implicit mental models for both of the above, and how they interact with each other. Government, the military, and organized crime (for example) are command hierarchies parasitic on the broader exchange economy we call ‘the free market’. There’s a third model, however, that is radically different from either and not generally recognized except by anthropologists; the *gift culture*.

Gift cultures are adaptations not to scarcity but to abundance. They arise in populations that do not have significant material-scarcity problems with survival goods. We can observe gift cultures in action among aboriginal cultures living in ecozones with mild climates and abundant food. We can also observe them in certain strata of our own society, especially in show business and among the very wealthy.

Abundance makes command relationships difficult to sustain and exchange relationships an almost pointless game. In gift cultures, social status is determined not by what you control but by *what you give away*.

Thus the Kwakiutl chieftain’s potlach party. Thus the multi-millionaire’s elaborate and usually public acts of philanthropy. And thus the hacker’s long hours of effort to produce high-quality open source.

For examined in this way, it is quite clear that the society of open-source hackers is in fact a gift culture. Within it, there is no serious shortage of the ‘survival necessities’ – disk space, network bandwidth, computing power. Software is freely shared. This abundance creates a situation in which the only available measure of competitive success is reputation among one’s peers.

This observation is not in itself entirely sufficient to explain the observed features of hacker culture, however. The cracker d00dz have a gift culture which thrives in the same (electronic) media as that of the hackers, but their behavior is very different. The group mentality in their culture is much stronger and more exclusive than among hackers. They hoard secrets rather than sharing them; one is much more likely to find cracker groups distributing sourceless executables that crack software than tips that give away how they did it.

What this shows, in case it wasn’t obvious, is that there is more than one way to run a gift culture. History and values matter. I have summarized the history of the hacker culture elsewhere in 20 ([HH]); the ways in which it shaped present behavior are not mysterious. Hackers have defined their culture by set of choices about the *form* which their competition will take. It is that form which we will examine in the remainder of this paper.

7 The Joy of Hacking

In making this ‘reputation game’ analysis, by the way, I do not mean to devalue or ignore the pure artistic satisfaction of designing beautiful software and making it work. We all experience this kind of satisfaction and thrive on it. People for whom it is not a significant motivation never become hackers in the first place, just as people who don’t love music never become composers.

So perhaps we should consider another model of hacker behavior in which the pure joy of craftsmanship is the primary motivation. This ‘craftsmanship’ model would have to explain hacker custom as a way of maximizing both the opportunities for craftsmanship and the quality of the results. Does this conflict with or suggest different results than the ‘reputation game’ model?

Not really. In examining the ‘craftsmanship’ model, we come back to the same problems that constrain hackerdom to operate like a gift culture. How can one maximize quality if there is no metric for quality? If scarcity economics doesn’t operate, what metrics are available besides peer evaluation? It appears that any craftsmanship culture ultimately must structure itself through a reputation game – and, in fact, we can observe exactly this dynamic in many historical craftsmanship cultures from the medieval guilds onwards.

In one important respect, the ‘craftsmanship’ model is weaker than the ‘gift culture’ model; by itself, it doesn’t help explain the contradiction we began this paper with.

Finally, the ‘craftsmanship’ motivation itself may not be psychologically as far removed from the reputation game as we might like to assume. Imagine your beautiful program locked up in a drawer and never used again. Now imagine it being used effectively and with pleasure by many people. Which dream gives you satisfaction?

Nevertheless, we’ll keep an eye on the craftsmanship model. It is intuitively appealing to many hackers, and explains some aspects of individual behavior well enough.

After I published the first version of this paper, an anonymous respondent commented: “You may not work to get reputation, but the reputation is a real payment with consequences if you do the job well.” This is a subtle and important point. The reputation incentives continue to operate whether or not a craftsman is aware of them; thus, ultimately, whether or not a hacker understands his own behavior as part of the reputation game, his behavior will be shaped by that game.

8 The Many Faces of Reputation

There are reasons general to every gift culture why peer repute (prestige) is worth playing for:

First and most obviously, good reputation among one’s peers is a primary reward. We’re wired to experience it that way for evolutionary reasons touched on earlier. (Many people learn to redirect their drive for prestige into various sublimations that have no obvious connection to a visible peer group, such as “honor”, “ethical integrity”, “piety” etc.; this does not change the underlying mechanism.)

Secondly, prestige is a good way (and in a pure gift economy, the *only* way) to attract attention and cooperation from others. If one is well known for generosity, intelligence, fair dealing, leadership ability, or other good qualities, it becomes much easier to persuade other people that they will gain by association with you.

Thirdly, if your gift economy is in contact with or intertwined with an exchange economy or a command hierarchy, your reputation may spill over and earn you higher status there.

Beyond these general reasons, the peculiar conditions of the hacker culture make prestige even more valuable than it would be in a ‘real world’ gift culture.

The main ‘peculiar condition’ is that the artifacts one gives away (or, interpreted another way, are the visible sign of one’s gift of energy and time) are very complex. Their value is nowhere near as obvious as that of material gifts or exchange-economy money. It is much harder to objectively distinguish a fine gift from a poor one. Accordingly, the success of a giver’s bid for status is delicately dependent on the critical judgement of peers.

Another peculiarity is the relative purity of the open-source culture. Most gift cultures are compromised – either by exchange-economy relationships such as trade in luxury goods, or by command-economy relationships such as family or clan groupings. No significant analogues of these exist in the open-source culture; thus, ways of gaining status other than by peer repute are virtually absent.

9 Ownership Rights and Reputation Incentives

We are now in a position to pull together the previous analyses into a coherent account of hacker ownership customs. We understand the yield from homesteading the noosphere now; it is peer repute in the gift culture of hackers, with all the secondary gains and side-effects that implies.

From this understanding, we can analyze the Lockean property customs of hackerdom as a means of *maximizing reputation incentives*; of ensuring that peer credit goes where it is due and does not go where it is not due.

The three taboos we observed above make perfect sense under this analysis. One’s reputation can suffer unfairly if someone else misappropriates or mangles one’s work; these taboos (and related customs) attempt to prevent this from happening.

- Forking projects is bad because it exposes pre-fork contributors to a reputation risk they can only control by being active in both child projects simultaneously after the fork. (This would generally be too confusing or difficult to be practical.)
- Distributing rogue patches (or, much worse, rogue binaries) exposes the owners to an unfair reputation risk. Even if the official code is perfect, the owners will catch flak from bugs in the patches (but see 20 ([RP])).
- Surreptitiously filing someone's name off a project is, in cultural context, one of the ultimate crimes. It steals the victim's gift to be presented as the thief's own.

All three of these taboo behaviors inflict global harm on the open-source community as well as local harm on the victim(s). Implicitly they damage the entire community by decreasing each potential contributor's perceived likelihood that gift/productive behavior will be rewarded.

It's important to note that there are alternate candidate explanations for two of these three taboos.

First, hackers often explain their antipathy to forking projects by bemoaning the wasteful duplication of work it would imply as the child products evolved in more-or-less parallel into the future. They may also observe that forking tends to split the co-developer community, leaving both child projects with fewer brains to work with than the parent.

A respondent has pointed out that it is unusual for more than one offspring of a fork to survive with significant 'market share' into the long term. This strengthens the incentives for all parties to cooperate and avoid forking, because it's hard to know in advance who will be on the losing side and see a lot of their work either disappear entirely or languish in obscurity.

Dislike of rogue patches is often explained by observing that they can complicate bug-tracking enormously, and inflict work on maintainers who have quite enough to do catching their *own* mistakes.

There is considerable truth to these explanations, and they certainly do their bit to reinforce the Lockean logic of ownership. But while intellectually attractive, they fail to explain why so much emotion and territoriality gets displayed on the infrequent occasions that the taboos get bent or broken – not just by the injured parties, but by bystanders and observers who often react quite harshly. Cold-blooded concerns about duplication of work and maintainance hassles simply do not sufficiently explain the observed behavior.

Then, too, there is the third taboo. It's hard to see how anything but the reputation-game analysis can explain this. The fact that this taboo is seldom analyzed much more deeply than "It wouldn't be fair" is revealing in its own way, as we shall see in the next section.

10 The Problem of Ego

At the beginning of the paper I mentioned that the unconscious adaptive knowledge of a culture is often at odds with its conscious ideology. We've seen one major example of this already in the fact that Lockean ownership customs have been widely followed despite the fact that they violate the stated intent of the standard licenses.

I have observed another interesting example of this phenomenon when discussing the reputation-game analysis with hackers. This is that many hackers resisted the analysis and showed a strong reluctance to admit that their behavior was motivated by a desire for peer repute or, as I incautiously labeled it at the time, 'ego satisfaction'.

This illustrates an interesting point about the hacker culture. It consciously distrusts and despises egotism and ego-based motivations; Self-promotion tends to be mercilessly criticized, even when the community might appear to have something to gain from it. So much so, in fact, that the culture's 'big men' and tribal elders are required to talk softly and humorously deprecate themselves at every turn in order to maintain their status. How this attitude meshes with an incentive structure that apparently runs almost entirely on ego cries out for explanation.

A large part of it, certainly, stems from the generally negative Europo-American attitude towards 'ego'. The cultural matrix of most hackers teaches them that desiring ego satisfaction is a bad (or at least immature) motivation; that ego is at best an eccentricity tolerable only in prima-donnas and often an actual sign of mental pathology. Only sublimated and disguised forms like “peer repute”, “self-esteem”, “professionalism” or “pride of accomplishment” are generally acceptable.

I could write an entire other essay on the unhealthy roots of this part of our cultural inheritance, and the astonishing amount of self-deceptive harm we do by believing (against all the evidence of psychology and behavior) that we ever have truly ‘selfless’ motives. Perhaps I would, if Friedrich Wilhelm Nietzsche and Ayn Rand had not already done an entirely competent job (whatever their other failings) of deconstructing ‘altruism’ into unacknowledged kinds of self-interest.

But I am not doing moral philosophy or psychology here, so I will simply observe one minor kind of harm done by the belief that ego is evil, which is this: it has made it emotionally difficult for many hackers to consciously understand the social dynamics of their own culture!

But we are not quite done with this line of investigation. The surrounding culture’s taboo against visibly ego-driven behavior is so much intensified in the hacker (sub)culture that one must suspect it of having some sort of special adaptive function for hackers. Certainly the taboo is weaker among many other gift cultures, such as the peer cultures of theater people or the very wealthy!

11 The Value of Humility

Having established that prestige is central to the hacker culture’s reward mechanisms, we now need to understand why it has seemed so important that this fact remain semi-covert and largely unadmitted.

The contrast with the pirate culture is instructive. In that culture, status-seeking behavior is overt and even blatant. These crackers seek acclaim for releasing ”zero-day warez” (cracked software redistributed on the day of the original uncracked version’s release) but are closemouthed about how they do it. These magicians don’t like to give away their tricks. And, as a result, the knowledge base of the cracker culture as a whole increases only slowly.

In the hacker community, by contrast, one’s work is one’s statement. There’s a very strict meritocracy (the best craftsmanship wins) and there’s a strong ethos that quality should (indeed *must*) be left to speak for itself. The best brag is code that “just works”, and that any competent programmer can see is good stuff. Thus, the hacker culture’s knowledge base increases rapidly.

A taboo against ego-driven posturing therefore increases productivity. But that’s a second-order effect; what is being directly protected here is the quality of the information in the community’s peer-evaluation system. That is, boasting or self-importance is suppressed because it behaves like noise tending to corrupt the vital signals from experiments in creative and cooperative behavior.

The hacker culture’s medium of gifting is intangible, its communications channels are poor at expressing emotional nuance, and face-to-face contact among its members is the exception rather than the rule. This gives it a lower tolerance of noise than most other gift cultures, and goes a long way to explain the example in public humility required of its tribal elders.

Talking softly is also functional if one aspires to be a maintainer of a successful project; one must convince the community that one has good judgement, because most of the maintainer’s job is going to be judging other people’s code. Who would be inclined to contribute work to someone who clearly can’t judge the quality of their own code, or whose behavior suggests they will attempt to unfairly hog the reputation return from the project? Potential contributors want project leaders with enough humility and class be able to to say, when objectively appropriate, “Yes, that does work better than my version, I’ll use it” – and to give credit where credit is due.

Yet another reason for humble behavior is that in the open source world, you seldom want to give the impression that a project is ‘done’. This might lead a potential contributor not to feel needed. The way to maximize your leverage is to be humble about the state of the program. If one does one’s bragging through the code, and then says “Well shucks, it doesn’t do x, y, and z, so it can’t be that good”, patches for x, y, and z will often swiftly follow.

Finally, I have personally observed that the self-deprecating behavior of some leading hackers reflects a real (and not unjustified) fear of becoming the object of a personality cult. Linus Torvalds and Larry Wall both provide clear and numerous examples of such avoidance behavior. Once on a dinner expedition with Larry Wall I joked "You're the alpha hacker here – you get to pick the restaurant". He flinched audibly. And rightly so; failing to distinguish their shared values from their leaders has ruined a good many communities, a pattern of which he and Linus cannot fail to be fully aware. On the other hand, most hackers would love to have Larry's problem, if they could but bring themselves to admit it.

12 Global Implications of the Reputation-Game Model

The reputation-game analysis has some more implications that may not be immediately obvious. Many of these derive from the fact that one gains more prestige from founding a successful project than from cooperating in an existing one. One also gains more from projects which are strikingly innovative, as opposed to being 'me, too' incremental improvements on software that already exists. On the other hand, software that nobody but the author understands or has a need for is a non-starter in the reputation game, and it's often easier to attract good notice by contributing to an existing project than it is to get people to notice a new one. Finally, it's much harder to compete with an already successful project than it is to fill an empty niche.

Thus, there's an optimum distance from one's neighbors (the most similar competing projects). Too close and one's product will be a "me, too!" of limited value, a poor gift (one would be better off contributing to an existing project). Too far away, and nobody will be able to use, understand, or perceive the relevance of one's effort (again, a poor gift). This creates a pattern of homesteading in the noosphere that rather resembles that of settlers spreading into a physical frontier – not random, but like a diffusion-limited fractal wave. Projects tend to get started to fill functional gaps near the frontier.

Some very successful projects become 'category killers'; nobody wants to homestead anywhere near them because competing against the established base for the attention of hackers would be too hard. People who might otherwise found their own distinct efforts end up, instead, adding extensions for these big, successful projects. The classic 'category killer' example is GNU Emacs; its variants fill the ecological niche for a fully-programmable editor so completely that nobody has even attempted a truly different design since the early 1980s. Instead, people write Emacs modes.

Globally, these two tendencies (gap-filling and category-killers) have driven a broadly predictable trend in project starts over time. In the 1970s most of the open source that existed was toys and demos. In the 1980s the push was in development and Internet tools. In the 1990s the action shifted to operating systems. In each case, a new and more difficult level of problems was attacked when the possibilities of the previous one had been nearly exhausted.

This trend has interesting implications for the near future. In early 1998, Linux looks very much like a category-killer for the niche 'free operating systems' – people who might otherwise write competing OSs are now writing Linux device drivers and extensions instead. And most of the lower-level tools the culture ever imagined having as open-source already exist. What's left?

Applications. As the year 2000 approaches, it seems safe to predict that open-source development effort will increasingly shift towards the last virgin territory – programs for non-techies. A clear early indicator is the development of *GIMP* `<http://www.gimp.org>`, the Photoshop-like image workshop that is open source's first major application with the kind of end-user-friendly GUI interface considered de rigeur in commercial applications for the last decade. Another is the amount of buzz surrounding application-toolkit projects like *KDE* `<http://www.kde.org>` and *GNOME* `<http://www.gnome.org>`.

Finally, the reputation-game analysis explains the oft-cited dictum that you do not become a hacker by calling yourself a hacker – you become a hacker when *other hackers* call you a hacker. A 'hacker', considered in this light, is somebody who has shown (by contributing gifts) that he or she both has technical ability and understands how the reputation game works. This judgement is mostly one of awareness and acculturation, and can only be delivered by those already well inside the culture.

13 Noospheric Property and the Ethology of Territory

To understand the consequences of property customs, it will help us to look at them from yet another angle; that of animal ethology, specifically the ethology of territory.

Property is an abstraction of animal territoriality, which evolved as a way of reducing intra-species violence. By marking his bounds, and respecting the bounds of others, a wolf diminishes his chances of being in a fight which could weaken or kill him and make him less reproductively successful.

Similarly, the function of property in human societies is to prevent inter-human conflict by setting bounds that clearly separate peaceful behavior from aggression. It is sometimes fashionable to describe human property as an arbitrary social convention, but this is dead wrong. Anybody who has ever owned a dog who barked when strangers came near its owner's property has experienced the essential continuity between animal territoriality and human property. Our domesticated cousins of the wolf are instinctively smarter about this than a good many human political theorists.

Claiming property (like marking territory) is a performative act, a way of declaring what boundaries will be defended. Community support of property claims is a way to minimize friction and maximize cooperative behavior. These things remain true even when the "property claim" is much more abstract than a fence or a dog's bark, even when it's just the statement of the project maintainer's name in a README file. It's still an abstraction of territoriality, and (like other forms of property) our instinct-founded models of property are territorial ones evolved to assist conflict resolution.

This ethological analysis at first seems very abstract and difficult to relate to actual hacker behavior. But it has some important consequences. One is in explaining the popularity of World Wide Web sites, and especially why open-source projects with websites seem so much more 'real' and substantial than those without them.

Considered objectively, this seems hard to explain. Compared to the effort involved in originating and maintaining even a small program, a web page is easy, so it's hard to consider a web page evidence of substance or unusual effort.

Nor are the functional characteristics of the Web itself sufficient explanation. The communication functions of a web page can be as well or better served by a combination of an FTP site, a mailing list, and Usenet postings. In fact it's quite unusual for a project's routine communications to be done over the Web rather than via a mailing list or newsgroup. Why, then, the popularity of Web sites as project homes?

The metaphor implicit in the term 'home page' provides an important clue. While founding an open-source project is a territorial claim in the noosphere (and customarily recognized as such) it is not a terribly compelling one on the psychological level. Software, after all, has no natural location and is instantly reduplicable. It's assimilable to our instinctive notions of 'territory' and 'property', but only after some effort.

A project home page concretizes an abstract homesteading in the space of possible programs by expressing it as 'home' territory in the more spatially-organized realm of the World Wide Web. Descending from the noosphere to 'cyberspace' doesn't get us all the way to the real world of fences and barking dogs yet, but it does hook the abstract property claim more securely to our instinctive wiring about territory. And this is why projects with web pages seem more 'real'.

This ethological analysis also encourages us to look more closely at mechanisms for handling conflict in the open-source culture. It leads us to expect that, in addition to maximizing reputation incentives, ownership customs should also have a role in preventing and resolving conflicts.

14 Causes of Conflict

In conflicts over open-source software we can identify four major issues:

- Who gets to make binding decisions about a project?
- Who gets credit or blame for what?
- How to reduce duplication of effort and prevent rogue versions from complicating bug tracking?
- What is the Right Thing, technically speaking?

If we take a second look at the "What is the Right Thing" issue, however, it tends to vanish. For any such question, either there is an objective way to decide it accepted by all parties or there isn't. If there is, game over and everybody wins. If there isn't, it reduces to "who decides?".

Accordingly, the three problems a conflict-resolution theory has to resolve about a project are (A) where the buck stops on design decisions, (B) how to decide which contributors are credited and how, and (C) how to keep a project group and product from fissioning into multiple branches.

The role of ownership customs in resolving issues (A) and (C) is clear. Custom affirms that the owners of the project make the binding decisions. We have previously observed that custom also exerts heavy pressure against dilution of ownership by forking.

It's instructive to notice that these customs make sense even if one forgets the reputation game and examines them from within a pure 'craftmanship' model of the hacker culture. In this view these customs have less to do with the dilution of reputation incentives than with protecting a craftsman's right to execute his vision in his chosen way.

The craftsmanship model is not, however, sufficient to explain hacker customs about issue (B), who gets credit for what (because a pure craftsman, one unconcerned with the reputation game, would have no motive to care). To analyze these, we need to take the Lockean theory one step further and examine conflicts and the operation of property rights *within* projects as well as *between* them.

15 Project Structures and Ownership

The trivial case is that in which the project has a single owner/maintainer. In that case there is no possible conflict. The owner makes all decisions and collects all credit and blame. The only possible conflicts are over succession issues – who gets to be the new owner if the old one disappears or loses interest. The community also has an interest, under issue (C), in preventing forking. These interests are expressed by a cultural norm that an owner/maintainer should publicly hand title to someone if he or she can no longer maintain the project.

The simplest non-trivial case is when a project has multiple co-maintainers working under a single 'benevolent dictator' who owns the project. Custom favors this mode for group projects; it has been shown to work on projects as large as the Linux kernel or Emacs, and solves the "who decides" problem in a way that is not obviously worse than any of the alternatives.

Typically, a benevolent-dictator organization evolves from an owner-maintainer organization as the founder attracts contributors. Even if the owner stays dictator, it introduces a new level of possible disputes over who gets credited for what parts of the project.

In this situation, custom places an obligation on the owner/dictator to credit contributors fairly (through, for example, appropriate mentions in README or history files). In terms of the Lockean property model, this means that by contributing to a project you earn part of its reputation return (positive or negative).

Pursuing this logic, we see that a 'benevolent dictator' does not in fact own his entire project unqualifiedly. Though he has the right to make binding decisions, he in effect trades away shares of the total reputation return in exchange for others' work. The analogy with sharecropping on a farm is almost irresistible, except that a contributor's name stays in the credits and continues to 'earn' to some degree even after that contributor is no longer active.

As benevolent-dictator projects add more participants, they tend to develop two tiers of contributors; ordinary contributors and co-developers. A typical path to becoming a co-developer is taking responsibility for a major subsystem of the project. Another is to take the role of 'lord high fixer', characterizing and fixing many bugs. In this way or others, co-developers are the contributors who make a substantial and continuing investment of time in the project.

The subsystem-owner role is particularly important for our analysis and deserves further examination. Hackers like to say that 'authority follows responsibility'. A co-developer who accepts maintainance responsibility for a given subsystem generally gets to control both the implementation of that subsystem and its interfaces with the rest of the project, subject only to correction by the project leader (acting as architect). We observe that this rule effectively creates enclosed properties on the Lockean model within a project, and has exactly the same conflict-prevention role as other property boundaries.

By custom, the ‘dictator’ or project leader in a project with co-developers is expected to consult with those co-developers on key decisions. This is especially so if the decision concerns a subsystem which a co-developer ‘owns’ (that is, has invested time in and taken responsibility for). A wise leader, recognizing the function of the project’s internal property boundaries, will not lightly interfere with or reverse decisions made by subsystem owners.

Some very large projects discard the ‘benevolent dictator’ model entirely. One way to do this is turn the co-developers into a voting committee (as with Apache). Another is rotating dictatorship, in which control is occasionally passed from one member to another within a circle of senior co-developers (the Perl developers organize themselves this way).

Such complicated arrangements are widely considered unstable and difficult. Clearly this perceived difficulty is largely a function of the known hazards of design-by-committee, and of committees themselves; these are problems the hacker culture consciously understands. However, I think some of the visceral discomfort hackers feel about committee or rotating-chair organizations is because they’re hard to fit into the unconscious Lockean model hackers use for reasoning about the simpler cases. It’s problematic, in these complex organizations, to do an accounting of either ownership in the sense of control or ownership of reputation returns. It’s hard to see where the internal boundaries are, and thus hard to avoid conflict unless the group enjoys an exceptionally high level of harmony and trust.

16 Conflict and Conflict Resolution

We’ve seen that within projects, an increasing complexity of roles is expressed by a distribution of design authority and partial property rights. While this is an efficient way to distribute incentives, it also dilutes the authority of the project leader – most importantly, it dilutes the leader’s authority to squash potential conflicts.

While technical arguments over design might seem the most obvious risk for internecine conflict, they are seldom a serious cause of strife. These are usually relatively easily resolved by the territorial rule that authority follows responsibility.

Another way of resolving conflicts is by seniority – if two contributors or groups of contributors have a dispute, and the dispute cannot be resolved objectively, and neither owns the territory of the dispute, the side that has put the most work into the project as a whole (that is, the side with the most property rights in the whole project) wins.

These rules generally suffice to resolve most project disputes. When they do not, fiat of the project leader usually suffices. Disputes that survive both these filters are rare.

Conflicts do not as a rule become serious unless these two criteria (”authority follows responsibility” and ”seniority wins”) point in different directions, *and* the authority of the project leader is weak or absent. The most obvious case in which this may occur is a succession dispute following the disappearance of the project lead. I have been in one fight of this kind. It was ugly, painful, protracted, only resolved when all parties became exhausted enough to hand control to an outside person, and I devoutly hope I am never anywhere near anything of the kind again.

Ultimately, all of these conflict-resolution mechanisms rest on the wider hacker community’s willingness to enforce them. The only available enforcement mechanisms are flaming and shunning – public condemnation of those who break custom, and refusal to cooperate with them after they have done so.

17 Acculturation Mechanisms and the Link to Academia

An early version of this paper posed the following research question: How does the community inform and instruct its members as to its customs? Are the customs self-evident or self-organising at a semi-conscious level, are they taught by example, are they taught by explicit instruction?

Teaching by explicit instruction is clearly rare, if only because few explicit descriptions of the culture’s norms have existed to be used up to now.

Many norms are taught by example. To cite one very simple case, there is a norm that every software distribution should have a file called README or READ.ME that contains first-look instructions for browsing the distribution. This convention has been well established since at least the early 1980s, but up to now it has never been written down. One derives it from looking at many distributions.

On the other hand, some hacker customs are self-organizing once one has acquired a basic (perhaps unconscious) understanding of the reputation game. Most hackers never have to be taught the three taboos I listed in Section Three, or at least would claim if asked that they are self-evident rather than transmitted. This phenomenon invites closer analysis – and perhaps we can find its explanation in the process by which hackers acquire knowledge about the culture.

Many cultures use hidden clues (more precisely 'mysteries' in the religio/mystical sense) as an acculturation mechanism. These are secrets which are not revealed to outsiders, but are expected to be discovered or deduced by the aspiring newbie. To be accepted inside, one must demonstrate that one both understands the mystery and has learned it in a culturally approved way.

The hacker culture makes unusually conscious and extensive use of such clues or tests. We can see this process operating at at least three levels:

- Password-like specific mysteries. As one example, there is a USENET newsgroup called alt.sysadmin.recovery that has a very explicit such secret; you cannot post without knowing it, and knowing it is considered evidence you are fit to post. The regulars have a strong taboo against revealing this secret.
- The requirement of initiation into certain technical mysteries. One must absorb a good deal of technical knowledge before one can give valued gifts (e.g. one must know at least one of the major computer languages). This requirement functions in the large in the way hidden clues do in the small, as a filter for qualities (such as capability for abstract thinking, persistence, and mental flexibility) which are necessary to function in the culture.
- Social-context mysteries. One becomes involved in the culture through attaching oneself to specific projects. Each project is a live social context of hackers which the would-be contributor has to investigate and understand socially as well as technically in order to function. (Concretely, a common way one does this is by reading the project's Web pages and/or email archives) It is through these project groups that newbies experience the behavioral example of experienced hackers.

In the process of acquiring these mysteries, the would-be hacker picks up contextual knowledge which (after a while) does make the three taboos and other customs seem 'self-evident'.

One might, incidentally, argue that the structure of the hacker gift culture itself is its own central mystery. One is not considered acculturated (concretely: no one will call you a hacker) until one demonstrates a gut-level understanding of the reputation game and its implied customs, taboos, and usages. But this is trivial; all cultures demand such understanding from would-be joiners. Furthermore the hacker culture evinces no desire to have its internal logic and folkways kept secret – or, at least, nobody has ever flamed me for revealing them!

Respondents to this paper too numerous to list have pointed out that hacker ownership customs seem intimately related to (and may derive directly from) the practices of the academic world, especially the scientific research commmunity. This research community has similar problems in mining a territory of potentially productive ideas, and exhibits very similar adaptive solutions to those problems in the ways it uses peer review and reputation.

Since many hackers have had formative exposure to academia (it's common to learn how to hack while in college) the extent to which academia shares adaptive patterns with the hacker culture is of more than casual interest in understanding how these customs are applied.

Obvious parallels with the hacker 'gift culture' as I have characterized it abound in academia. Once a researcher achieves tenure, there is no need to worry about survival issues (Indeed, the concept of tenure can probably be traced back to an earlier gift culture in which "natural philosophers" were primarily wealthy gentlemen with time on their hands to devote to research.) In the absence of survival issues reputation enhancement becomes the driving goal, which encourages sharing of new ideas and research through journals and other media. This makes objective functional sense because scientific research, like the hacker culture, relies heavily on the idea of 'standing upon the shoulders of giants', and not having to rediscover basic principles over and over again.

Some have gone so far as to suggest that hacker customs are merely a reflection of the research community's folkways and are actually (for most) acquired there. This probably overstates the case, if only because hacker custom seems to be readily aquired by intelligent high-schoolers!

There is a more interesting possibility here. I suspect academia and the hacker culture share adaptive patterns not because they're genetically related, but because they've both evolved the one most optimal social organization for what they're trying to do, given the laws of nature and and the instinctive wiring of human beings. The verdict of history seems to be that free-market capitalism is the globally optimal way to cooperate for economic efficiency; perhaps, in a similar way, the reputation-game gift culture is the globally optimal way to cooperate for generating (and checking!) high-quality creative work.

This point, if true, is of more than (excuse me) academic interest. It suggests from a slightly different angle one of the speculations in *The Cathedral And The Bazaar* <`http:/~esr/writings/cathedral-bazaar/`>; that, ultimately, the industrial-capitalist mode of software production was doomed to be outcompeted from the moment capitalism began to create enough of a wealth surplus for many programmers to live in a post-scarcity gift culture.

18 Conclusion: From Custom to Customary Law

We have examined the customs which regulate the ownership and control of open-source software. We have seen how they imply an underlying theory of property rights homologous to the Lockean theory of land tenure. We have related that to an analysis of the hacker culture as a 'gift culture' in which participants compete for prestige by giving time, energy, and creativity away. We have examined the implications of this analysis for conflict resolution in the culture.

The next logical question to ask is "Why does this matter?" Hackers developed these customs without conscious analysis and (up to now) have followed them without conscious analysis. It's not immediately clear that conscious analysis has gained us anything practical – unless, perhaps, we can move from description to prescription and deduce ways to improve the functioning of these customs.

We have found a close logical analogy for hacker customs in the theory of land tenure under the Anglo-American common-law tradition. Historically 20 ([Miller]), the European tribal cultures that invented this tradition improved their dispute-resolution systems by moving from a system of unarticulated, semi-conscious custom to a body of explicit customary law memorized by tribal wisemen – and eventually written down.

Perhaps, as our population rises and acculturation of all new members becomes more difficult, it is time for the hacker culture to do something analogous – to develop written codes of good practice for resolving the various sorts of disputes that can arise in connection with open-source projects, and a tradition of arbitration in which senior members of the community may be asked to mediate disputes.

The analysis in this paper suggests the outlines of what such a code might look like, making explicit that which was previously implicit. No such codes could be imposed from above; they would have to be voluntarily adopted by the founders or owners of individual projects. Nor could they be completely rigid, as the pressures on the culture are likely to change over time. Finally, for enforcement of such codes to work, they would have to reflect a broad consensus of the hacker tribe.

I have begun work on such a code, tentatively titled the "Malvern Protocol" after the little town where I live. If the general analysis in this paper becomes sufficiently widely accepted, I will make the Malvern Protocol publicly available as a model code for dispute resolution. Parties interested in critiquing and developing this code, or just offering feedback on whether they think it's a good idea or not, are invited to *contact me by email* <`mailto:esr@thyrsus.com`>.

19 Questions for Further Research

The culture's (and my own) understanding of large projects that don't follow a benevolent-dictator model is weak. Most such projects fail. A few become spectacularly successful and important (Perl, Apache, KDE). Nobody really understands where the difference lies. (There's a vague sense abroad that each such project is *sui generis* and stands or falls on the group dynamic of its particular members, but is this true or are there replicable strategies a group can follow?)

As a matter of observable fact, people who found successful projects gather more prestige than people who do arguably equal amounts of work debugging and assisting with successful projects. Is this a rational valuation of comparative effort, or is it a second-order effect of the unconscious territorial model we have adduced here?

20 Bibliography, Notes, and Acknowledgements

[Miller] Miller, William Ian; *Bloodtaking and Peacemaking: Feud, Law, and Society in Saga Iceland*; University of Chicago Press 1990, ISBN 0-226-52680-1. A fascinating study of Icelandic folkmoot law, which both illuminates the ancestry of the Lockean theory of property and describes the later stages of a historical process by which custom passed into customary law and thence to written law.

[Mal] Malaclypse the Younger; *Principia Discordia, or How I Found Goddess and What I Did To Her When I Found Her*; Loompanics, ISBN 1-55950-040-9. Amidst much enlightening silliness, the 'SNAFU principle' provides a rather trenchant analysis of why command hierarchies don't scale well. There's a browseable *HTML version* `<http://www.cs.cmu.edu/~tilt/principia/>`.

[BCT] J. Barkow, L. Cosmides, and J. Tooby (Eds.); *The adapted mind: Evolutionary psychology and the generation of culture.* New York: Oxford University Press 1992. An excellent introduction to evolutionary psychology. Some of the papers bear directly on the three cultural types I discuss (command/exchange/gift), suggesting that these patterns are wired into the human psyche fairly deep.

[MHG] Goldhaber, Michael K.; *The Attention Economy and the Net* `<http://www.firstmonday.dk/issues/issue2_4/goldhaber>`. I discovered this paper after my version 1.7. It has obvious flaws (Goldhaber's argument for the inapplicability of economic reasoning to attention does not bear close examination), but Goldhaber nevertheless has funny and perceptive things to say about the role of attention-seeking in organizing behavior. The prestige or peer repute I have discussed can fruitfully be viewed as a particular case of attention in his sense.

[HH] I have summarized the history of hackerdom at `<http://earthspace.net/~esr/faqs/hacker-hist.html>`. The book that will explain it really well remains to be written, probably not by me.

[N] The term 'noosphere' is an obscure term of art in philosophy derived from the Greek 'nous' meaning 'mind', 'spirit', or 'breath'. It is pronounced KNOW-uh-sfeer (two o-sounds, one long and stressed, one short and unstressed tending towards schwa). If one is being excruciatingly correct about one's orthography, it is properly spelled with a diaresis over one 'o' – just don't ask me which one.

[RP] There are some subtleties about rogue patches. One can divide them into 'friendly' and 'unfriendly' types. A 'friendly' patch is designed to be merged back into the project's main-line sources under the maintainer's control (whether or not that merge actually happens); an 'unfriendly' one is intended to yank the project in a direction the maintainer doesn't approve. Some projects (notably the Linux kernel itself) are pretty relaxed about friendly patches and even encourage independent distribution of them as part of their beta-test phase. An unfriendly patch, on the other hand, represents a decision to compete with the original and is a serious matter. Maintaining a whole raft of unfriendly patches tends to lead to forking.

I am indebted to Michael Funk <mwfunk@uncc.campus.mci.net> for pointing out how instructive a contrast with hackers the pirate culture are. Robert Lanphier <robla@real.com> contributed much to the discussion of egoless behavior. Eric Kidd <eric.kidd@pobox.com> highlighted the role of valuing humility in preventing cults of personality. The section on global effects was inspired by comments from Daniel Burn <daniel@tsathoggua.lab.usyd.edu.au>. Mike Whitaker <mrw@entropic.co.uk> inspired the main thread in the section on acculturation.

21 Version history

I am solely responsible for what has gone into this paper, and any errors or misconceptions. However, I have welcomed comments and feedback and used them to improve the paper – a process which I do not expect to end at any predefined time.

10 April 1998: Version 1.2 published on the Web.

12 April 1998: Version 1.3. Typo fixes and responses to first round of public comments. First four items in bibliography. An anonymously contributed observation about reputation incentives operating even when the craftsman is unaware of them. Added instructive contrasts with warez d00dz, material on the 'software should speak for itself' premise, and observations on avoiding personality cults. As a result of all these changes, the section on 'The Problem of Ego' grew and fissioned.

16 April 1998: Version 1.7. New section on 'Global implications' discusses historical tends in the colonization of the noosphere, and examines the 'category-killer' phenomenon. Added another research question.

27 April 1998: Version 1.8. Added Goldhaber to the bibliography. This is the version that will go in the Linux Expo proceedings.

26 May 1998: Version 1.9: Incorporated Fare Rideau's noosphere/ergosphere distinction. Incorporated RMS's assertion that he is not anticommercial. New section on acculturation and academia (thanks to Ross J. Reedstrom, Eran Tromer, Allen MacInnes, and others). More about humility, ('egoless behavior') from Jerry Fass and Marsh Ray.

11 July 1998: Version 1.10: Remove Fare Rideau's reference to 'fame' at his suggestion.