Schoolchildren are taught these ideas through Horatio Alger-like, rags-to-riches stories in which individuals overcome great odds to achieve the American dream. Never mind that this history lesson leaves many pages unturned—most Americans believe that work provides the basic structure around which successful lives are built. The tenacious grip of work on the American mind makes it difficult for welfare advocates—even in times of economic crisis—to build a case for government assistance; large numbers of Americans believe that any such provision must be earned. During the Great Depression, when a third of Americans had lost their livelihoods through no fault of their own, most recipients of assistance felt demeaned by their situation.

TANF highlights both the emphasis on work in American political culture and the limited nature of the American commitment to providing jobs for those in need. We want people to work, but we don’t want to guarantee them a job. What the country needs now is a paradigm shift toward a new understanding of self-sufficiency. People who can work must be given access to education and training and then must be guaranteed a job, along with the support they need to keep it. When the economy slows, community service jobs should be provided as part of a broader antipoverty policy. Declining poverty rather than falling welfare caseloads should be our measure of success. Franklin Delano Roosevelt failed to make permanent our nation’s boldest experiment with a public jobs program—the Works Progress Administration. It may be time for those who envision a comprehensive antipoverty system to revisit it. As FDR knew, people aren’t truly free when they are captive to the vicissitudes of the market. Among the essential elements of a “healthy and strong democracy,” he told the American people on January 6, 1941, are “jobs for those who can work and security for those who need it.”

Amy D. Burke is a freelance writer and doctoral candidate in politics at Brandeis University.

Sources for this article include the following:
The Politics of Software
The Case of Open Source

Manuel DeLanda

As the United States v. Microsoft antitrust case drew to a close last year, the tone of Judge Thomas P. Jackson’s findings of fact left little doubt in most observers’ minds that the software corporation would be found guilty as charged. The most recent court ruling (June 2001) seems to indicate that the new Republican administration will let the giant corporation off the hook. But Microsoft still faces another challenge to its monopoly, a challenge that may prove more difficult to tackle than the American courts. An internal memorandum written by a Microsoft employee defined the new threat and was leaked via the Internet as the case was reaching its climax. A movement of hackers is attempting to create a new paradigm of software production: collectively created programs in which consumers are also (to different degrees) producers.

This hacker movement, known variously as “Free Software” or “Open Source,” burst into public consciousness a few years ago as a result of its success in the production of reliable and robust software. Some of its products (the server software called Apache) have already acquired a larger market share than Microsoft’s, and the share of some others (the operating system known as Linux) is growing at a faster rate than the corporation’s product. In either case, the success of the movement has gone beyond the expectations of most analysts and has taken most corporate managers by surprise; many of them (at corporations like IBM and SUN) are switching from proprietary to “open” systems.

Although Open Source software is typically free, or sold at relatively low prices, its impact is only indirectly related to monetary questions; it is more intimately linked to the kind of software it makes available. Rather than producing end-user application software (word processors, browsers, spreadsheets, and so on) the members of this movement, who number in the hundreds if not thousands, have concentrated their efforts on the very tools needed to create those applications—what we may refer to without much exaggeration as the means of production of the “information economy” (compilers, debuggers, program text-editors, and so on). In addition, they have created several operating systems, that is, the software platforms on top of which both applications and the tools to produce them run. The crucial importance of the operating system may be grasped from the fact that the Justice Department investigated Microsoft not so much because of its large size or dominant market share, but because it produces both an operating system (Windows) and a series of applications. Controlling both the platform on which programs run as well as what runs on top of this platform gives Microsoft an unfair advantage over competitors. It may, if it wants, delay the release of technical details on a new operating system, forcing its rivals to rewrite applications while Microsoft takes as much time as it wants.

Challenging Microsoft’s dominant operating system with a rival product is not, however, what characterizes the Open Source movement, but rather the specific way in which this challenge is mounted. There are two main forms in which a computer program may exist. When a user buys a particular application from a commercial vendor—a word processor, for example—the program typically has a form that is next to unintelligible to human beings (but perfect for a computer). It consists of long series of ones and zeros that code for specific machine instructions. When that same program is being developed, on the other hand,
machine language is not used as frequently as some other high-level language that is not only readable by humans but is also accompanied by comments that explain (to other humans) what each part of the program is intended to do. This human-oriented version, referred to by the term “source code,” is used to change or further develop a particular program. Whereas Microsoft would never allow the source code of its products to be freely distributed, given that this is its main form of intellectual property, within the Open Source movement a program is distributed exclusively in this form. So it is open to further improvement and development by its users. In short, the movement has defined an alternative conception of how software should be produced, an alternative paradigm that is at once evolutionary and collective.

The freedom to change and adapt a given piece of software allows the formation of development communities within which many of the inevitable errors (or “bugs”) that are part and parcel of any complex program can be discovered and fixed. This community-based debugging results in software that is more resilient against malfunction than commercially available programs. The downside is the inevitable presence of free-raiders: what can stop a particular user (an individual or an institution) from benefiting from this shared source code, altering it a bit, then closing it, and selling it as proprietary software? The Open Source solution to this dilemma is ingenious. It consists of a license agreement known as GPL (GNU Public License) with terms designed to force anyone who uses previously Open Source code to open whatever contributions he or she makes.

It is important to emphasize that this effect is not achieved by abolishing intellectual property—each contributor in fact owns the copyright of whatever piece of code he or she has developed—but by altering the way in which the rights of exclusion that are part and parcel of property rights are deployed. This is the originality of the GPL: rather than actively exploiting the right to exclude, it holds the right in reserve as a threat to potential free-raiders. In addition, the license has become a legal instrument for community-building in two different ways. By enforcing the norm of “sharing code,” it preserves and propagates the values of a once small community, allowing it to grow in proportion to the degree to which the software propagates. It also serves as a means to allocate credit for particular contributions: the license requires that the names of the creators of specific innovations not be removed from any future release, thereby preserving the motivation that replaces monetary incentives in what is, in effect, an “economy of reputation.”

The clever design of the GPL bears witness to what we may call the institutional creativity of some Open Source hackers—an example of their creative achievements beyond the writing of code. Another display of institutional creativity is the kind of governance structures that some of the hackers have devised for the collective production of software. To an outside observer, the idea of hundreds of people scattered around the world working incrementally and improving a program may seem like anarchy, but the actual development of specific projects is anything but anarchic. Each project has a leader (or committee of leaders) who has the final say on what improvements get included in the “official” version of the program.

The unquestioned authority of project leaders is sometimes expressed by saying that they are “benevolent dictators,” but a more appropriate description of their role is that of creators and maintainers of the community supporting a particular project. The leaders must keep all potential developers motivated by immediately releasing any new piece of code so that interested users can get to work on it. They must learn to delegate responsibility for specific areas to certain users (in effect, making them co-developers); to promote cooperation; and to be as self-effacing as possible so as to block any suspicion that credit for the work will not be shared equally or that decisions about the quality of a given piece of code will not be made objectively. Although each of the different Open Source projects has a different governance structure (a single leader surrounded by an inner circle, committees with rotating memberships, and so on) the hackers involved have been enormously creative in solving the
problems of coordinating a highly decentralized workforce and in leveraging the considerable potential of the Internet to lower coordination costs.

Let’s assume for a moment that the combination of the GPL contract and an open, meritocratic governance structure—as well as the programming skills of the hundreds of people that this combination can harness—does indeed represent a novel approach to the production of software. We will still want to know if this new approach can challenge the corporate approach, and in which areas of software production this challenge can most successfully be mounted. Although the Open Source movement has excelled in the areas of production tools and operating systems, its record in the areas of end-user applications and user interfaces is not as impressive. Doubts remain as to its viability in the development of new software products, as opposed the improvement of existing ones. But even if we granted these drawbacks, its limited success would remain politically important, given the centrality of operating systems in the day-to-day running of an information economy.

But it can also be argued that the explosive growth of the Internet was made possible to a large extent by the lack of corporate ownership of its infrastructural software and by its common standards. Much of the software underlying the Internet’s early growth was intellectual property that skilled programmers from around the world gave for free to the Internet community. IRC, the first program that allowed real-time chat, began as an experiment by a programmer in Finland, then was distributed free on the Internet. The software that formed the basis for the first virtual worlds, the Multi-User Dungeons, developed at an English university, also acquired its full potential by being unleashed without charge in the Internet. The same is true for the software at the base of the Internet, the thousands of electronic bulletin board systems connected in networks like Fido. More recently, the very language of the World Wide Web (HTML); the software used to manage the names of domains like .com or .org (BIND); the most ubiquitous e-mail management software (SENDMAIL); and a variety of languages used in commercial Web sites (PERL) are all open standards created outside corporations and owned by no one.

Although all the software just mentioned is hardly visible to casual computer users, it is nevertheless crucial to the working of the Internet as we know it. It is this centrality that makes it politically important. Is there in addition to this political dimension a specifically moral dimension? A subgroup of this movement (going by the name of “Free Software”) believes that there is. More specifically, it believes that the moral legitimacy of intellectual property rights is at stake here.

The economics textbook definition of the moral problem is well known: one begins by distinguishing goods that can be consumed only by a given person (or persons)—that is, goods whose very consumption excludes others from consuming them—and goods that do not possess this property. Food is an example of the first type, a good that is “rivalrous in consumption,” while ideas are an example of the second type: if someone consumes a song or a book, this act by itself does not exclude others from consuming the same song or book, particularly if technologies of duplication and distribution have made the costs of reproduction minimal. The economic problem of intellectual property is that when goods that are not rivalrous in consumption are made subject to property rights, the exclusion aspect of these rights generates social waste. Given that additional copies of a given good may be generated and distributed at virtually no cost (this is particularly true of goods in digital form), excluding people from using them means that wants will go unsatisfied that could easily have been satisfied. On the other hand, not making these goods subject to property rights means that those producing them will have no incentive to do so, particularly if the costs of production are high. Thus the problem of intellectual property needs to be solved by a careful balancing of social costs and producer benefits, a balance that is best achieved case by case.

Given that exclusion seems to be the source of the moral problem, it may be argued
that the GPL, with its creative use of exclusion, constitutes a solution to it. Yet this is not how its creator, Richard Stallman, views the question. He acknowledges the traditional way of posing the problem but adds that in addition to the material social waste there is a “psychosocial damage” associated with intellectual property, an erosion of the community value of helping one’s neighbor. The position taken by his Free Software Foundation is that intellectual property is literally evil, and that the GPL is justified as a necessary lesser evil: if copyrights, patents, and so on disappeared altogether, it would gladly get rid of the GPL. The fact that the creators of this remarkable piece of legal machinery do not have a clear idea of its function leads me to think that the success of the hacker movement is largely a matter of “unintended consequences.” This impression is further enhanced by the fact that the governance structures of many Open Source projects (such as the one for the operating system Linux) seem also to have evolved without any planning at all.

Perhaps we should not look for the abstract principles (such as Stallman’s “freedom”) that justify Open Source; perhaps we should not be in a hurry to issue programmatic statements about the morality of intellectual property in general. Better to learn the lessons of unintended consequences. Institutional reality is much too complex to be dealt with in any other terms than learning by doing. The hackers of this movement have exhibited great institutional creativity in the design of contracts and governance structures, but their contributions have not been around long enough to be more than experiments. The role of philosophy here is to be as experimental as these contributions have been, viewing the movement as a long-term process in the course of which its own moral foundations will be built, one piece at a time. We move to each successive experiment only after learning the unintended consequences of the previous one.

Manuel Delanda is a philosopher writing on science and technology issues.

---

Call for Conference Submissions

“Activism, Ideology, and Radical Philosophy”

sponsored by the Radical Philosophy Association

November 7-10, 2002
Brown University, Providence, RI

The RPA Conference Program Committee invites proposals for talks, papers, workshops, roundtables discussions and posters.

For more information, consult the conference website at http://www.uvm.edu/~radphil/rpa2002call.htm