Capital & Class

http://cnc.sagepub.com

The hacker movement as a continuation of labour struggle

George Dafermos and Johan Söderberg

Capital & Class 2009; 33; 53
DOI: 10.1177/030981680909700104

The online version of this article can be found at:
http://cnc.sagepub.com/cgi/content/abstract/33/1/53

Published by:
SAGE
http://www.sagepublications.com

On behalf of:
Conference of Socialist Economics Ltd

The hacker movement as a continuation of labour struggle

George Dafermos and Johan Söderberg

Abstract

Examining the way in which capital exploits the volunteer labour of free software developers, this article argues that there is a historical continuity between hackers and labour struggle. The common denominator is their rejection of alienated work practices, which suggests that corporate involvement in the computer underground, far from inhibiting further struggles by hackers, may function as a catalyst for them.

Introduction

In this article, we propose that the history of labour struggle is continued in the hacker movement — for example, in the case of employees who crash their employers' computer equipment. Framing the discussion in this way highlights the kinship between the tradition of machine breaking in labour conflicts and hackers who break into corporate servers or write viruses.[1] But this connection is also valid for the hackers engaged in developing free software and open source software (FOSS) — and indeed, we are mainly concerned with the activity of the latter group.

Advocates of 'open source' tend to portray its development model as a neutral advancement of the method for developing software that leads to better technologies.[2] In support of their claim, they can point to the wide adoption of FOSS applications by the computer industry. For instance, the dominant server software and scripting language on the worldwide web are the Apache

HTTP server (Netcraft, 2008) and the PHP programming language (Hughes, 2002). Linux runs on more architectures and devices than any other operating system today (Kroah-Hartman, 2006); Sendmail is responsible for routing the majority of email messages; BIND is indisputably the most widely used DNS server; and even the world wide web itself[3] is free software. Due to the terms under which these products are distributed, they are available for everyone to use, modify, redistribute and sell — that is, redistribute for a price.

The motivation of hackers for writing software and giving it away for free is one of the most widely debated topics among academics studying the hacker movement. Economists try to square this behaviour of hackers with the assumption of the rational economic man. They assume that hackers hand out software for free in order to improve their reputation and thus employability in the future: that the monetary reward has just been postponed (Lerner & Tirole, 2002). But while this statement may describe a current trend in the computer underground, it fails to explain the motivation of hackers prior to the establishment of a market in FOSS products. Neither does the opportunity–cost model take into account hackers who spend their time on illegal activities such as writing viruses and cracking encryptions. When hackers are asked about their motives, they play down the economic incentives and point to the fun of writing software, often comparing the joy of writing free software with the toil of waged labour (Shah, 2006).[4]

In our view, the joy of participating in FOSS projects should be seen against the backdrop of alienated work relations. Hackers gear their labour power towards the use value of the software as opposed to its exchange value: free software is produced to be used, not to be sold. In FOSS projects, work is an end in itself rather than a means for something else. That is the deeper meaning of the common expression among hackers that they write code just to 'scratch their itch' (Raymond, 1999). In attempting to escape from alienated existence, the hacker movement has invented an alternative model for organising labour founded on the common ownership of the means of production, on volunteer participation and the principle of self-expression in work. It is this promise that lies at the heart of the politics of the hacker movement. The practice of 'hacking' indicates the distance between *doing* and wage labour — a claim that can be substantiated using concrete political gains. One example is that of strong encryption programmes like Pretty Good Privacy, which are made publicly available to prevent governments from eavesdropping on citizens. Another case is the surge of anonymous file-sharing networks that have encouraged

mass defection from the intellectual property regime.[5] These systems would not have been possible had decisions over technology still been confined to market incentives, corporate hierarchies and government regulation.

It is true that from the perspective of capital, the hacker community presents an opportunity to tap into a well of gratis labour. Enterprises take FOSS, customise it for their clients, package it under a brand and sell services on top of it, thus lowering the cost of in-house product development and putting a downward pressure on wages and working conditions in the computer sector. In the second half of this article, we elaborate on Karl Marx's theory about 'surplus profit' and the 'equalisation of social surplus value' in order to conceptualise the way business models based on FOSS operate. Even so, we do not conclude that the hacker movement has ceased to be a potential source of resistance against capital. Whether hackers pose a challenge to capital or if they will be more of a threat to organised labour is a question that has to be decided in struggle.

Programming and labour struggle

In our use of the term *hacking*, we mean the act of taking a pre-existing system and bending it to serve a different end from that for which it was originally intended, and hence it is implicit in hacking that its significance cannot be known simply from knowing its point of departure. We emphasise this because the political pretensions of hackers can easily be dismissed. The majority of hackers are white, male and belong to the western middle class. Likewise, the profession of programming is descended from white-collar engineers. Historically speaking, those engineers and the mainframe computers over which they presided were instrumental in imposing management control over factory workers. As for the internet, most people are familiar with its origin in the military-industrial complex. Without doubt, something of that heritage is reflected in the worldview of the hacker movement. A common point of departure when hackers position themselves in the world is the notion of the information age. The concept was cooked up during the Cold War by US social scientists wishing to replace Marxism with a less subversive master narrative. The ideological load of the 'end-to-ideology' argument that underpins notions about the information age has been vividly demonstrated before (Webster, 2002; Barbrook, 2007). Thus it is understandable that a chorus of left-leaning scholars over the years has decried 'cyber-politics' for being individualistic, consumerist and entrepreneurial

(Siegel & Markoff, 1985; Kroker, 1994; Liu, 2004). Many hackers are aware of the paradoxical blend between libertarianism and socialism that underpins their philosophy and indeed, have often pointed it out to scholars.

The politics of hacking is hard to pin down because it is a synthesis of many irreconcilable things. To use a fashionable term, it is a hybrid. On the one hand, there is the line running from the white-collar engineers of the 1950s to present-day hackers; and on the other, another line connects hacking to the resistance of the machine operators working under those engineers. In order to prevent accidents and malfunctions, machine operators have often by themselves and against the wish of managers made efforts to become familiar with the instructions relating to their machinery. In addition, once operators understood how the technology worked, they knew how to reconfigure the apparatus and lower its work pace, which had been previously set at a higher speed by managers and engineers. Managers responded to this practice by hiding the mechanics from the operators (Noble, 1986), and thus the conflict of interest between labour and capital over the exploitation of surplus value was now played out in a struggle over who had access to the technology. It is the same concern that informs hackers' demand for free access to information and free software tools. What workers and hackers have in common is their rejection of Taylorism.[6] This bond is made evident the more routinised the programming profession becomes (Kraft, 1977).

Routinisation was given a strong impetus in the 1950s, when computers began to be used by businesses. A labour market for programmers was created together with educational organisations that trained and certified programmers. The goal of ensuring a smooth supply of computer professionals, however, turned out to be elusive. For as long as a market for programmers has existed, it seems to have been plagued by a supposed labour shortage (Chabrow, 2008). It is not a shortage of trained programmers in absolute numbers that has troubled corporate recruiters, though. Rather, once managers discovered that some programmers were several times more productive than others, they recognised a problem in identifying the right programmer for the job.[7] Given that living labour accounts for two-thirds of total costs in software development projects (Lakha, 1994), and that 'the cost of software has always been development cost, not replication cost' (Brooks, 1987), the question of how to increase the productivity of labour has been a looming issue (Kim, 2006). Over the years, managers grew increasingly disconcerted by the absence of 'a universally accepted classification scheme for programmers' based on

'accepted norms with regard to biographical, educational and job experience data' (Sackman et al., 1969).

The difficulty of employers in deciding whether a potential programmer will perform poorly or exceptionally can be attributed to a failure of capital in measuring this kind of labour. Such a failure is also suggested by the diversity of backgrounds within the software community, as has been often commented by insiders: 'In what other field are you likely to find a Ph.D and a person whose education stopped at the high school level working as equals on the same difficult technical problem, e.g., the development of a compiler?' (Orden, 1967: 147). The inability of capital to measure the labour of programmers is a result of their resistance against Taylorism. In the words of a manager, 'the technologists more closely identified with the digital computer have been the most arrogant in their wilful disregard of the nature of the manager's job. These technicians have clothed themselves in the garb of the arcane wherever they could do so, thus alienating those whom they would serve' (*Datamation*, 1966). The nascent discipline of software engineering grew, to a certain extent, out of managers' compulsion to rationalise the work process of programmers.[8] In contrast to the 'black art' of code writing, software engineering was heralded in trade magazines as 'the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software'.[9] This is not to say that the academic field of software engineering was a managerial plot; but merely that the innovations that sprang from its bosom were systematically deployed by managers in order to extend their control over the labour process. A case in point is that of the assemblers, compilers and technologies that come under the rubric of 'automatic programming', designed to perform a function formerly carried out by a human programmer (Kraft, 1979; Parnas, 1985). Another set of innovations that descends from software engineering is the use of methodologies. Methodologies (i.e. structured approaches or object orientation) provide frameworks within which programmers are constrained from doing things incorrectly. In the hands of managers, however, these techniques have often turned into instruments of control (Glass, 2005; Keggler, 1989; Kraft, 1979), since they increase managers' ability to break the programming process down into functions consisting of a definite number of tasks. Thus specific tasks can be assigned to programmers while managers reserve for themselves the decision-making authority. There is nothing inherently stifling in these methodologies, styles, and techniques — were it not the

case that they have been used to turn programmers into fragment labourers and deny them knowledge of the whole of the labour process (Hannemyr, 1999).

Karl Marx's analysis of how capital incorporates the labour process provides a lens through which the historical transformation of the programming profession from the 1950s till today can be viewed. The subsumption of labour under capital unfolds in two stages: in the first stage, due to the concentration of ownership over the means of production, formerly independent producers and artisans become wage earners. Thus they pass into the *formal* control of capital. 'From the technological point of view', however, 'the labour process continues exactly as it did before, except that now it is a labour process subordinated to capital'. The subsumption of labour under capital is consummated, becomes *real*, only when the labour process itself is transformed in accordance with capital's needs. The second stage is marked by the standardisation of work procedures, the parcelling-out and deskilling of labour, and the absorption of human skills into fixed capital (Marx, 1990 (1864)). These observations of Marx's were expanded upon by Harry Braverman, who foresaw that the factory despotism of his day would soon metamorphose into office despotism. He rightly pinpointed the computer as playing a crucial role in this transition. In hindsight, computerisation has confirmed many of Braverman's suspicions, but it has also made apparent a countervailing tendency. Although capital wrests control over the labour process through the mediation of technology, it has to concede to the workers some leeway in operating this technology.[10] Critics of Braverman quickly responded that he had over-stressed the punitive side of capital, forgetting that capital also extends its influence over workers by giving them a certain degree of 'responsible autonomy' (Friedman, 1977). A case in point are the much-discussed table-tennis facilities put at the disposal of programmers at Googleplex and other high-tech firms. If we choose to see computer firms investing in FOSS development as cases in which research and development costs have been outsourced to volunteer communities, then it becomes clear that 'responsible autonomy' enjoys pride of place among capital's strategies for managing labour. The same thing, however, could also be understood as being the subsumption not only of work but of the whole of society under monopoly capital. In describing the unfolding of such a trend, Harry Braverman made a comment that squarely places FOSS development in the context of labour theory: 'So enterprising is capital that even where the effort is made by one or another section of the population to find a way to nature,

sport, or art through personal activity and amateur or "underground" innovation, these activities are rapidly incorporated into the market so far as is possible' (Braverman, 1974: 279).

The practice of hacking: Free software and open source development

The rise of a consumer market in personal computers in the 1970s coincided with a hacker community spawned outside elite academic institutions. Though mutually dependent, the two spheres soon clashed, since the hacker practice of freely sharing software code obstructed the establishment of a market in software (Gates, 1976). This conflict has intensified over time, as the industry has shifted its focus from selling hardware to selling information and software products. The hacker community was politicised in response to the attempts by capital to enclose computer programmes under intellectual property law.

The struggle of hackers against proprietary software has been championed by the Free Software Foundation (FSF). The organisation was founded in 1985 by Richard Stallman, and its goal is the development of a computer standard consisting entirely of free software (Stallman, 1999). In order to ensure that software made available for free could not be expropriated by individual rights-holders, Stallman devised the GNU General Public Licence (GPL), commonly referred to as 'copyleft'. The free licence makes use of the privilege that copyright law gives authors to specify the conditions for using their creations. With the GPL, conditions are added that increase rather than restrict the rights of the user to run, modify, and re-distribute software. Ironically, it is copyright law that gives the free licence teeth, rendering it possible to enforce violations. But instead of backing individual rights-holders, copyleft installs a regime of common ownership. Proof of the continued relevance of the licence is that tens of thousands of software programmes have been released under the GPL, and the common pool of free software grows bigger by the day.[11]

The existence of this free software suggests that the production of computer applications can be organised without intellectual property relations and, by extension, without the mediation of capital.[12] The making of the Linux operating system kernel gives some hints. Linus Torvalds initiated the project in October 1991. He described it as a tool 'for hackers by a hacker', to which each contributed on a purely voluntary basis (Torvalds, 1991). The development of an operating system is a huge undertaking involving a gigantic programming effort and the expenditure of

thousands of man-hours. Coinciding with the diffusion of the internet in the early 1990s, Linux was the first project that leveraged the computer network for large-scale, geographically distributed collaboration, and the size of the joint effort behind it is positively correlated to its rapid pace of releases. In the first month after Linus Torvalds's announcement, there were three releases. Prior to the release of version 1.0 in March 1994, there were ten releases in December 1993, fourteen releases in January 1994 and eleven in February 1994. This practice of releasing 'early and often' runs counter to typical commercial software development, where users come into contact with the product only in its final stages. Early versions are 'buggy' or flawed, and companies do not want to wear out the patience of their customers. In the Linux project, by contrast, this practice proved decisive in motivating participants, giving them credit for their recent contributions and spurring them to new efforts. Community etiquette prescribes that all contributions are mentioned in the credits file included in each version. In this way, contributors more readily recognise themselves in the product of their collective labour and recognise it as their own.

In the early years, the process by which Linux was developed was rather straightforward: diagrammatically, it followed a straight line from Torvalds who distributed the official version to the individual programmers who downloaded the software, made changes to fix bugs or augment its functionality, and then fed them back to Torvalds to review and decide whether or not to integrate them in the next official version. Traditional yardsticks of software engineering were shunned: there were no deadlines, release dates or system-level design. In the absence of a centrally planned division of labour by which programmers might be assigned specific tasks, developers took on tasks as their own interests best dictated. However, in order to alleviate the strain forced upon the project's coordination by the expansion of the contributing group, an organisational structure gradually took shape: about a dozen hackers who had done extensive work on a domain of the system took on the task of reviewing patches[13] submitted by the wider bug-fixing group. These 'trusted lieutenants' are each responsible for maintaining a part of the kernel, and contributors send their patches directly to them.

Only a handful of FOSS projects have adopted a formal voting procedure for electing project leaders and settling disputes. In most cases, the administration of projects appears as informal, opaque and hierarchical. Upon closer examination, however, there turns out to be a different kind of check against power asymmetries. The

position of lieutenants is granted by means of recognition by the community, and this authority is constantly subject to withdrawal (Moody, 2001: 81, 84). Hence, the role of the lieutenant is not that of a leader in the customary sense of the word. Even Linus Torvalds, in spite of his high prestige, has been forced to back down from decisions when developers threatened to sideline him. Essentially, the direction of the Linux project derives from the cumulative synthesis of modifications contributed by individual programmers (van Wendel de Joode, 2005; Ingo, 2006). When two different solutions compete for the same problem, both are tried out (Torvalds, 2004). Thus conflicts over technical issues are 'resolved' in parallel development lines. In a community that 'rejects kings, presidents and voting, but believes in rough consensus and running code' (Clark, 1992), decisions are made by those who do the work (McCormick, 2003), and the freedom of developers to vote with their feet is key. Basically, it is this right to 'fork'[14] a development project that is protected by the General Public Licence. Project leaders are thus kept on their toes, because the relevance of a fork depends on the commitment of its developers and users.

The same philosophy can also be read out from the modular architecture of the Linux kernel. Described in technical terms, modularity is a form of task decomposition. It is used to separate the work of different groups of developers, creating, in effect, related yet separate sub-projects. Because a modular system 'can be built piecemeal, and others can help by working independently on some of the various components' (Moody, 2001), a modular design decreases the total need for coordination and enables parallel development. Torvalds explains, 'With the Linux kernel it became clear very quickly that we want to have a system which is as modular as possible. The open-source development model really requires this, because otherwise you can't easily have people working in parallel. It's too painful when you have people working on the same part of the kernel and they clash' (Torvalds, 1999). The parallel development structure of the Linux kernel is consummated in *parallel releases* of the product. The parallel release structure for Linux was initiated with version 1.1 in April 1994, when Linux was split into two trees: the stable and the development branch.[15] In retrospect, the phenomenal growth of Linux can be traced back to this decision. Contrary to the expectation that as Linux grew in size and complexity its rate of development would inevitably slow down, an analysis of Linux for the years 1994 to 2004 shows that the development branch keeps growing at a super-linear rate (Robles, 2005; Godfrey & Tu, 2000). The important point to note here is that modularity is not just a choice of design that

has proven technically superior in managing a decentralised and collaborative software project.[16] This particular design choice reflects the development process in which it was made, and as such, modularity helps to reinforce the social relations and the values of the hacker community.

A labour theory approach to FOSS development raises the question of how work is distributed in the hacker community. In the fourteen versions of Linux (from version 2.6.11 to 2.6.24) released in the space of nearly three years (from March 2, 2005 to January 24, 2008), 83,432 changes were contributed by 3,678 distinct individuals. Averaging 5,000 changes per (stable) release and 2.7 months between (stable) releases, Linux grows by a phenomenal 10 per cent per year.[17] Volunteers account for approximately 27% of changes, followed by Red Hat (11.2%), Novell (8.9%), IBM (8.3%), Intel (4.1%), Linux Foundation (2.6%), Consultant (2.5%), SGI (2.0%), MIPS Technologies (1.6%), Oracle (1.3%), MontaVista (1.2%), Google (1.1%), Linutronix (1.0%), HP (0.9%), NetApp (0.9%), SWsoft (0.9%), Renesas Technology (0.9%), Freescale (0.9%), Astaro (0.9%), Academia (0.8%), Cisco (0.5%), Simtec (0.5%), Linux Networx (0.5%), Q Logic (0.5%), Fujitsu (0.5%), Broadcom (0.5%) and others. The overall participation by firms has been steadily

Table 1: Number of individual developers and employers

Kernel version	Number of developers	Number of companies
2.6.11	483	71
2.6.12	701	90
2.6.13	637	91
2.6.14	625	89
2.6.15	679	96
2.6.16	775	100
2.6.17	784	106
2.6.18	897	121
2.6.19	878	126
2.6.20	728	130
2.6.21	834	132
2.6.22	957	176
2.6.23	991	178
2.6.24	1,057	186
All	3,678	271

increasing, as has the number of contributing developers (*see Table 1 on page 62*).[18]

Despite the large number of contributors, however, the majority of work is still done by a relatively small group of core developers. The top ten contributors account for 15 per cent of changes, and the top thirty for 30 per cent (Kroah-Hartman, Corbet & McPherson, 2008). Similar distributions of work across the development community have been observed in other big free software projects such as Apache, Mozilla and FreeBSD (Mockus et al., 2002; Dinh-Trong & Bieman, 2005). The numbers suggest an asymmetry in workload — but that does not necessarily translate into a concealed, centrally planned division of labour. Rather, the division of labour in FOSS development is the immediate result of the usual procedure by which one joins a project and advances from peripheral (yet necessary) activities such as problem-reporting and problem-fixing to the development of new functionality. Since the right to commit changes to a project's central repository (i.e. version-control system)[19] is conferred only to those contributors with a long history of accepted patches, one typically joins a project by reporting problems and submitting fixes to problems already reported. In this way, FOSS projects have found a mechanism for the selection of programmers to the core development group that resonates with the strong meritocratic ethos in the hacker community.[20]

How capital is enrolling user communities in the valorisation process

The development model of hackers has won widespread acceptance in the business community in the last ten years. An open invitation to big business was sent in 1998 after the Freeware Summit in Palo Alto, at which many of the movers and shakers in the hacker subculture had gathered with the goal of getting corporations involved. Crucial to their plan was the choosing of a label that sounded less threatening to the computer programming status quo than did the term 'free software'. The meeting decided to use the label 'open source' instead. It quickly spread, and has become the term by which most outsiders know hacker software today. Shortly after the summit, IBM announced its commitment to open source. The company has since invested heavily in Apache and GNU/Linux, parading its support for open source with the old hippie slogan 'Peace, Love, Linux'.[21] Oracle, Compaq, Dell, Hewlett Packard, Intel and many others quickly followed suit, while the influx of multinationals spurred the growth of medium-sized companies such as Red Hat, Novell and MySQL, which specialised

in FOSS products and services. Nowadays, almost every major computer company, with the notable exception of Microsoft, is going out of its way to befriend the hacker community (Weber, 2004). It is reasonable to assume that the composition of the hacker community has been transformed by the recent inflow of capital. A study of 287 FOSS projects came up with the estimate that approximately 40 per cent of all contributors were either directly paid to perform the task in question, or encouraged by their employers to partake in free software projects during office time. Further, roughly 58 per cent of the survey respondents had day-jobs in the IT sector, while another 20 per cent were computer science students. In consideration of this, it is not unjustified to look upon the FOSS community as a basin for the provision of 'lifelong learning' to employees. While these observations are consistent with other studies that report a sizeable involvement of hired employees in volunteer development communities, this survey is additionally interesting because it registers traces of workers' discontent. About 17 per cent of the respondents said they were working on FOSS projects without their supervisors being aware of it. This finding should caution us from jumping to the conclusion that the extensive representation of firms in the hacker community translates into corporate control over those activities (Lakhani & Wolf, 2005).

The corporate embrace of the FOSS development model should be seen against the background of a restructured labour market. A feature of this restructuring is the effacement of the border between consumers and producers. Such a trend was already being proposed by futurists in the 1980s under the label 'the rise of the prosumer' (Toffler, 1981). Nowadays, it is part of the canon of intellectual property critique to situate the conflicts surrounding copyright law in the context of active consumers and fandom. Both futurists and critics expect that fans will bring about a more democratic, participatory form of media consumption. They cherish the altruism of communities working together. Barely masked under the fanfare is the bottom line of profit. Business gurus are more upfront: with them, the opportunity to get rich quickly by enrolling the community is directly spelled out (Hagel & Armstrong, 1997; Libert, Spector & Tapscott, 2007; Silver, 2007). It thus becomes evident that this trend borders on more familiar experiments in laissez-faire capitalism and old-hat marketing scams such as viral marketing, Tupperware parties and pyramid schemes. What they have in common is the involvement of the customer as the chief promoter and developer of the product. As critical voices have pointed out before, work assignments are

being self-sourced and crowd-sourced in an economy that increasingly depends on the unpaid labour of volunteers and users (Terranova, 2004; Gimenez, 2007).

FOSS firms provide a good point of departure for theorising about a situation that has variously been described as 'the real subsumption of society' and 'the social factory'. We do not believe that as a result the economy has been rendered immeasurable.[22] On the contrary, the profitability of FOSS firms can be harboured within Marx's theory of value. Red Hat is an example of what might, borrowing from Marxist terminology, be called a 'surplus profit' business model. A cornerstone in Karl Marx's economic theory is that labour is the source of surplus value. Furthermore, the amount of surplus value that a capitalist can accumulate depends on the number of labourers he sets in motion. Marx acknowledged a possibility, however, for the individual capitalist to acquire more surplus value: sometimes the capitalist manages to position his venture so favourably that the surplus value of labourers hired by competitors flows into his pockets instead. The textbook example is the capitalist who invents a superior technique for producing goods. The cost of producing an item falls below the social average, i.e. the average cost competitors pay when they produce the item. The units are produced at different costs, but since they are identical, all the items are sold on the same market for the same price. Hence the most cost-efficient capitalist — the one who produces the unit at the lowest cost — earns his efficiency gain as a bonus from the other capitalists. This boon is known as 'surplus profit'. The advantage is ephemeral, since every other capitalist will try to catch up with the inventor. When the majority has adopted the superior way of doing things, the average production cost will even out at the new equilibrium. The surplus profit vanishes for the individual capitalist. It is not efficiency gains in 'absolute terms' that provide the sought-for benchmark. It is efficiency gains vis-à-vis other comparable producers. The crucial point here is that surplus profit exists per definition as a deviation from the norm.

The existence of the FOSS business models can be understood as a variation on this theme. Companies such as Red Hat hire labourers to customise free software and provide support services in addition to it. These activities generate a modest amount of surplus value. The input of waged labour is marginal in comparison to the vast amount of volunteer labour involved in writing the main body of code. Gratis labour is not, though, automatically voided of value. It has value if it duplicates waged labour performed elsewhere in the economy. In other words, the value of unwaged labour by FOSS developers stands in relation to the waged labour of in-house

programmers. Both are working towards equivalent code solutions. For as long as the social average cost of solving a computer problem is determined by waged labour and intellectual property relations, volunteer labour (hackers) and free licences cut costs below this social average. In this case, surplus profit does not emanate from the reduction of staff due to a technological innovation, but is created when work migrates from paid labourers to unpaid users due to an organisational innovation, i.e. *crowdsourcing*.

It remains an open question as to whether the copyright-dependent fraction of the capitalist class (Microsoft, Hollywood, record companies) can follow suit and close the gap in production costs. Microsoft's 'shared source' policy, where selected customers are given restricted access to Microsoft's source code, could be seen as an attempt to close in on the distance between proprietary software and FOSS. However, going by what historical experience has taught us, the managerial preoccupation with control will probably spoil the efforts. It might be that these companies are unable to imitate the FOSS model and still sustain their high profitability. If our statement is correct, the surplus-profit business model of Red Hat will continue to prosper in the margins of society, leeching off the differential level in the cost of production.

From this we can draw two important conclusions. First, that hackers and campaigners against intellectual property law are wrong in thinking that FOSS enterprises, powered with free markets and free technology, are destined to supersede and replace intellectual property monopolies. Red Hat can only be profitable in relation to the inflated social average production cost of Microsoft. Both depend in different ways on the existence of intellectual property rights. Hence, abolishing intellectual property is incompatible with capitalism, and this statement is not falsified by the existence of enterprises that profit from FOSS products and services. Second, if we are to follow our reasoning to its logical conclusion, Red Hat's shareholders are not freeriding on the community of volunteer developers but, through the 'equalisation of social surplus value', they are intensifying the exploitation of programmers employed by Microsoft. That claim is counter-intuitive and should perhaps not be pushed too far. It is worth mentioning, nonetheless, since it highlights what corporate enthusiasm over open source boils down to. Namely, the expectation of managers that free and open-source licences will impose an overall downward pressure on the wages and working conditions of in-house computer programmers.

Having said this, we are still a long way from delivering a final verdict on FOSS. While companies certainly hope to pitch user

communities against waged workers, crowdsourcing is but one possible outcome from the current situation. When confronted with such concerns, hackers usually point to the growing employment opportunities within FOSS businesses. It could well be that professional FOSS developers end up in a stronger position compared to programmers working with proprietary software, since free and open-source licences remove the edge that firms otherwise have over employees due to their ownership of the means of coding. When software tools are made publicly available under a free licence, the main scarcity left on the market consists of the skill to write software code, which gives the advantage to living labour. In order to determine which is the most plausible scenario, more research into the emerging labour market of FOSS developers is required. Additionally, we have to take into consideration the unique position of the computer sector within capitalism today. This creates a 'chicken-in-every-pot' situation for programmers, irrespective of whether they are working with FOSS solutions or with proprietary software. That favourable position is at the expense of every other worker, since computer technology is pivotal in the neoliberal reformation of capitalism that most people have encountered as weaker unions, flexible labour markets and deskilling. Thus, while some computer programmers are confident that they will ride out the crowdsourced mode of capitalism, other workers may not be so fortunate (Ross, 2006).

An assessment of the surplus-profit model of FOSS firms must also be weighed against the subjective side of this story, and the political claims made by hackers. It is worth bearing in mind that the computer underground was forked out of the New Left and the movement surrounding 'appropriate technology' (Markoff, 2005). Strong voices within the computer underground continue to stress social and ethical concerns about free software, and many hackers choose free licences for political reasons. Hence, capital was not the mastermind behind FOSS development, though the computer industry is a fast learner and tries hard to recuperate the disturbance. A sign of this is that companies are making millions out of the volunteer efforts of hackers. Simply to state this fact closes the matter for sceptical commentators. They believe that the subversive potential of FOSS development, if there ever was one, has by now been exhausted. But by analogy, the same critics should also say that there is nothing subversive in workers' struggle, since companies profit from them. In our opinion, the situation is exactly the opposite. It is the fact that FOSS communities have been made sources of surplus value for capital that provides the spark that might radicalise the hacker movement even further, throwing parts

of it into direct struggle; and it is for the same reason that their challenge to capital's domination is congruent with the resistance of waged workers.

References

Barbrook, R. (2007) *Imaginary Futures: From Thinking Machines to the Global Village* (Pluto Press).

Berners-Lee, T. (2000) *Weaving the Web: The Past, Present and Future of the World Wide Web* (Texere).

Brooks, F. P. (1987) 'No silver bullet: Essence and accidents of software engineering', *Computer,* vol. 20, no. 4, April.

Castoriadis, C. (1976) *The Revolutionary Problem Today.*

Chabrow, E. (2008) 'The new IT worker shortage', *CIO Insight, 15* January, available online at <www.cioinsight.com>.

Ciarrocchi, P. (2005) 'Introduction to Linux kernel development process', at <http://linux.tar.bz/articles/2.6-development_process>

Clark, D. D. (1992) 'A cloudy crystal ball: Visions of the future', plenary presentation at 24th meeting of the Internet Engineering Task Force, Cambridge, Mass., 13–17 July.

DiBona, C., S. Ockman & M. Stone (eds.) (1999) *Open Sources: Voices from the Open Source Revolution* (O'Reilly).

Dinh-Trong, T. T. & J. M. Bieman (2005) 'The FreeBSD project: A replication case study of open source development', *IEEE Transactions on Software Engineering,* vol. 31, no. 6, June, pp. 481–494.

Editorial (1966) 'The thoughtless information technologist', *Datamation*, vol. 12, no. 8, quoted in N. Ensemenger & W. Aspray (2000) 'Software as labor process', *Proceedings of the International Conference On History of Computing: Software Issues*, Paderborn, Germany, 5–7 April.

Friedman, A. (1977) *Industry and Labour: Class Struggle at Work and Monopoly Capitalism* (Macmillan).

Gates, B. (1976) 'Open letter to hobbyists', *Homebrew Computer Club Newsletter*, vol. 2, no. 1, p. 2.

Gimenez, M. (2007) 'Self-sourcing: How corporations get us to work without pay!' *Monthly Review*, vol. 59, no. 7.

Glass, R. L. (2005) 'The plot to deskill software engineering', *Communications of the ACM*, vol. 48, no. 11, November, p. 22.

Godfrey, M. W. & Q. Tu (2000) 'Evolution in open source software: A case study', *Proceedings of the International Conference on Software Maintenance*, San Jose, California, October, pp. 131–142.

Hagel, J. & A. Armstrong (1997) *Net Gain: Expanding Markers Through Virtual Communities* (Harvard Business School Press).

Hannemyr, G. (1999) 'Technology and pleasure: Considering hacking constructive', *First Monday,* vol. 4, no. 2; online at <www.firstmonday.dk>.

Hardt, M. & A. Negri (2000) *Empire* (Harvard University Press).
Hardt, M. & A. Negri (2004) *Multitude: War and Democracy in the Age of Empire* (Penguin).
Hughes, F. (2002) 'PHP: Most popular server-side web scripting technology', *Linux Weekly News*, 3 June, at <http://lwn.net>.
Ingo, H. (2006) *Open Life: The Philosophy of Open Source*, trans. Sara Torvalds, self-published.
Keggler, J. (1989) 'Structured programming', message posted to <comp.unix.wizards> newsgroup, 3 February.
Kim, W. (2006). 'On assuring software quality and curbing software development cost', *Journal of Object Technology*, vol. 5, no. 6, July–August, pp. 35–42.
Kraft, P. (1977) *Programmers and Managers: The Routinization of Computer Programming in the United States* (Springer-Verlag).
Kraft, P. (1979) 'The routinizing of computer programming', *Sociology of Work and Occupations*, vol. 6, no. 2, May, pp. 139–155.
Kroker, A. (1994) *Data Trash: The Theory of the Virtual Class* (St. Martin's Press).
Kroah-Hartman, G. (2005) 'How to do Linux kernel development', at <http://lxr.linux.no/source/Documentation/HOWTO>
Kroah-Hartman, G. (2006) 'Myths, lies, and truths about the Linux kernel', keynote presentation at Linux Symposium, Ottawa, 19–22 July, online at <www.kroah.com>.
Kroah-Hartman, G. (2007) 'Linux kernel development: How fast it is going, who is doing it, what they are doing, and who is sponsoring it', Proceedings of the Linux Symposium, Ottawa, Canada, 27–30 June.
Kroah-Hartman, G., J. Corbet & A. McPherson (2008) 'Linux kernel development: How fast it is going, who is doing it, what they are doing, and who is sponsoring it', *Linux Foundation White Paper*, April, at <www.linux-foundation.org>.
Lakha, S. (1994) 'The new international division of labour and the Indian software industry', *Modern Asian Studies,* vol. 28, no. 2.
Lakhani, K. & R. Wolf (2005) 'Why hackers do what they do: Understanding motivation and effort in free/open source software projects', in J. Feller, B. Fritzgerald, S. Hissam & K. Lakhani (eds.) *Perspectives on Free and Open Source Software* (MIT Press).
Lerner, J. & J. Tirole (2002) 'Some simple economics of open source', *Journal of Industrial Economics*, vol. 50, no. 2, June.
Levy, S. (1994) *Hackers: Heroes of the Computer Revolution* (Penguin).
Levy, S. (2001) *Crypto: How the Code Rebels Beat the Government Saving Privacy in the Digital Age* (Viking).
Libert, B., J. Spector & T. Dapscott (2007) *We Are Smarter Than Me: How to Unleash the Power of Crowds in Your Business* (Wharton School Publishing).

Liu, A. (2004) *The Laws of Cool: Knowledge Work and the Culture of Information* (University of Chicago Press).
MacCormack, A., J. Rusnak & C. Baldwin (2006) 'Exploring the structure of complex software designs: An empirical study of open source and proprietary code', *Management Science*, vol. 52, no. 7, July.
Markoff, J. (2005) *What the Doormouse Said: How the 60's Counter-Culture Shaped the Personal Computer* (Viking).
Marx, K. (1990 (1864)) 'Results of the immediate process of production', appendix to Capital, Vol. 1, trans. B. Fowkes (Penguin).
McCormick, C. (2003) 'The big project that never ends: Role and task negotiation within an emerging occupational community', Ph.D dissertation, University of Albany, NY.
A. Mockus, R. T. Fielding & J. D. Herbsleb (2002) 'Two case studies of open source software development: Apache and Mozilla', *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 3, July, pp. 309–346.
Moody, G. (2001) *Rebel Code: Linux and the Open Source Revolution* (Penguin).
Naur, P. & B. Randell (eds.) (1969) *Software Engineering: Report of a Conference Sponsored by the NATO Science Committee*, Garmisch, Germany, 7–11 October 1968, Scientific Affairs Division (NATO).
Netcraft (2008) *Web Server Survey*, May, at <http://news.netcraft.com>.
Noble, D. (1986) *Forces of Production: A Social History of Industrial Automation* (Oxford University Press).
Oram, A. (ed.) (2001) P*eer-to-Peer: Harnessing the Power of Disruptive Technologies* (O'Reilly).
Orden, A. (1967) 'The emergence of a profession', *Communications of the ACM*, vol. 10, no. 3, March, pp. 145–147.
O'Reilly, T. (2001) 'Remaking the P2P meme map' in A. Oram (ed.) *Peer-to-Peer: Harnessing the Power of Disruptive Technologies* (O'Reilly).
Parnas, D. L. (1985) 'Software aspects of strategic defense systems', *Communications of the ACM*, vol. 28, no. 12, December.
Raymond, E. S. (1999) The Cathedral & The Bazaar (O'Reilly).
Robles, G. (2005) 'Empirical software engineering research on libre software: Data sources, methodologies and results', Ph.D thesis, Universidad Rey Juan Carlos, Madrid.
Ross, A. (1991) *Strange Weather: Culture, Science, and Technology in the Age of Limits* (Verso).
Ross, A. (2006) 'Technology and below-the-line labor in the copyfight over intellectual property', *American Quarterly*, vol. 58, no. 3, pp. 743–766.
Sackman, H., W. Erikson & E. Grant (1968) 'Exploratory experimental studies comparing online and offline programming performance', *Communications of the ACM*, vol. 11, no. 1, January, pp. 3–11.

Shah, S. (2006) 'Motivation, governance, and the viability of hybrid forms in open source software development', *Management Science*, vol. 52, no. 7, July.
Siegel, L. & J. Markoff (1985) *The High Cost of High Tech: The Dark Side of the Chip* (Harper & Row).
Silver, D. (2007) *Smart Start-Ups: How Entrepreneurs and Corporations Can Profit by Starting Online Communities* (John Wiley & Sons).
Stallman, R. M. (1999) 'The GNU operating system and the free software movement', in C. DiBona, S. Ockman & M. Stone (eds.) *Open Sources: Voices from the Open Source Revolution* (O'Reilly).
Taylor, F. W. (1911) *The Principles of Scientific Management* (Harper).
Terranova, T. (2004) *Network Culture: Politics for the Information Age* (Pluto Press).
Toffler, A. (1981) *The Third Wave* (Bantam).
Torvalds, L. (1991) 'Free minix-like kernel sources for 386-AT', message posted to <comp.os.minix> newsgroup, 5 October, at <http://groups.google.com/group/comp.os.minix>.
Torvalds, L. (1999) 'The Linux edge' in C. DiBona, S. Ockman & M. Stone (eds.) *Open Sources: Voices from the Open Source Revolution*, pp. 101–119 (O'Reilly).
Torvalds, L. (2004) *Linux kernel management style*, Linux kernel file added 10 October (Documentation/ManagementStyle), also at <http://lxr.linux.no/linux/Documentation/ManagementStyle>.
Virno, P. (2004) *A Grammar of the Multitude*, trans. I. Bertoletti, J. Cascaito & A. Casson [Semiotext(e)].
Weber, S. (2004) *The Success of Open Source* (Harvard University Press).
Webster, F. (2002) *Theories of the Information Society*, 2nd edition (Routledge).
van Wendel de Joode, R. (2005) 'Understanding open source communities: An organizational perspective', Ph.D dissertation, Delft University of Technology.

Notes

1 Ross (1991) made the parallel between labour conflicts and hacking in order to disprove the public image of hackers as merely apolitical, juvenile pranksters.
2 For example, according to Tim O'Reilly (2001), open source is 'about making better software through source sharing and network-enabled collaboration'. See also <http://www.openp2p.com/p2p/2000/12/05/images/800-opensource.jpg>.
3 We are referring to the core components of the software, i.e. HTTP, HTML and URI. See the website of the World Wide Web Consortium at <http://w3c.org>, or see T. Berners-Lee (1999).

4 In a similar vein, Hannemyr (1999) maintains that hackers become freelance consultants (and, one might add, FOSS entrepreneurs) not in order to make a profit, but to avoid having to work as waged labour inside a firm.

5 These and other examples are described in the anthology by DiBona, Ockman & Stone (1999). For a detailed account of how hackers made encryption schemes available to the public, see Levy (2001).

6 Also referred to as scientific management, Taylorism is a managerial credo rooted in the time-and-motion studies conducted by Frederick Taylor (1911). Its practice is synonymous with the fragmentation of the labour process that ensues from the parceling-out and deskilling of the labour of execution.

7 An influential early study by the System Development Corporation for the Advanced Research Projects Agency of the Department of Defence of the United States showed great individual differences in programmer performance. The report underlined the significance of finding a mechanism 'to detect and weed out these poor performers [as this] could result in vast savings in time, effort, and cost'. See Sackman, Erikson & Grant (1968).

8 The field of software engineering was defined in 1968 at the NATO Software Engineering Conference, which stressed that 'backward techniques' were at the heart of the problem facing software as a professional field (Naur & Randell, 1969: 10), thus reflecting the concerns about the management of programmers that weighed heavily on managers' minds.

9 This is the definition of software engineering given in the 'IEEE Standard Glossary of Software Engineering Terminology', *IEEE* std 610.12-1990 (1990).

10 Cornelius Castoriadis (1976: 75) has followed this thought to its logical conclusion: 'What shows the critical importance of the distance between the official organisation of production and the reality of the labour process is the effectiveness of the form of struggle called "working to rule" … no sooner do the workers start to apply with the utmost precision and to excruciating detail the rules and the instructions they are supposed to apply than the factory is thrown into full chaos'.

11 As of June 16, 2008, Sourceforge.net (the largest application service provider for free software projects) hosts 83,731 projects licensed under the GNU GPL, at <http://sourceforge.net>.

12 On a more general note, this presupposition is consistent with what some theorists have referred to as *the communism of capital*. See, for example, Virno (2004).

13 A *patch* is a small piece of software designed to fix problems or update a computer programme.

14 A project *forks* when, as a result of their discontent with the official branch, (a subset of its) developers make a copy of the code base and start independent development, thus creating an alternative project. On account of the rights granted to its users, FOSS can be forked without the permission of its original creator(s).

15 It should be noted that the number of branches comprising the development process has increased over time. In parallel with version 2.6, the convention used for numbering releases was changed in order to accelerate the rate of stable releases, so that a new version is released every two to three months. As a result, excluding subsystem-specific branches, the development process now consists of the main 2.6.x kernel tree (which is where code from the experimental branch is merged prior to being released as a new major stable release), the 2.6.x.y-stable kernel tree (from which minor stable releases are made), and the 2.6.x-mm kernel tree (experimental). For a detailed discussion of the different branches, see Ciarrocchi (2005) or Kroah-Hartman (2005, 2007).>

16 Recent empirical research shows that modular design is characteristic not just of Linux but of the FOSS development model in general. Indicatively, MacComack, Rusnak & Baldwin (2006) demonstrate that the code structure of Mozilla became increasingly modular after its release under an open-source license.

17 The size of the Linux kernel code-base is 8.8 million lines in version 2.6.24.

18 These statistics are taken from Kroah-Hartman, Corbet & McPherson (2008).

19 A *version-control system* is a software tool commonly used in large-scale software development to track and provide control over changes to the software product under development.

20 On the hacker ethic, see Levy (1994).

21 See <http://www-03.ibm.com/servers/eserver/linux/passport.swf>.

22 We are here adressing one of the claims made by Hardt and Negri (2000, 2004).