Abstract

Based on more than 2 years of daily use of the Ubuntu Linux system and 6 months of on-line and in-person fieldwork among the developers working to develop and maintain it, this thesis examines the individual and collaborative day-to-day practices of these developers as they relate to the computer operating system that is the result of their labour. Despite being spread across the industrialized world, these *Ubuntu hackers* socialise, share their knowledge, and come to depend on each other in their work across the Internet, as well as in their in-person meetings at conferences and summits. I argue that these shared and negotiated on-line and in-person practices constitute a *community of practice* (Wenger 1998) rooted in a more than 40-year old "oral" computing tradition based on the Unix operating system which has spawned a lively interdependent on-line eco-system of *free software* projects built on the reciprocal sharing of knowledge and source code which, guaranteed by cleverly crafted copyright licenses, has resulted in a cumulatively improved system developed openly on-line in a fashion which has made it a viable alternative to the mainstream IT industry.

Taking the Ubuntu system as my point of departure, I examine the network of practices, processes and actors in which it has been constructed. Through a strategically selected constellation of theories, I seek to describe and analyze the three central dimensions of a community of practice: *Joint enterprise*, *mutual engagement* and *shared repertoire*.

Joint enterprise is the Ubuntu hackers' negotiation of their different personal motivations for contributing to this cause in relation to the community's overall, institutionally stated goal of spreading the use and awareness of free software. Following Alfred Gell's work on social relations mediated through technical work (Gell 1999), I argue that these diverse interests are reflected in every little part of the system and results in conflicts and social negotiations centred around the fascination of learning the details of technology, enjoying working together on a technical challenge, and sharing an ethical commitment to write software that can be shared and understood for others to use and learn.

Mutual engagement is the complementary social practices through which the

Ubuntu hackers work to develop and maintain the system. I argue that these practices contain a duality between what Claude Lévi-Strauss has called *engineer work* and *bricolage* (Lévi-Strauss 1994): Between visualising and designing new features as well as redesigning and reimplementing odd bits of left-behind work to fit their needs through a haphazard extending, fixing, fussing and testing. I argue that this mutual engagement enables them to learn from one another by collaborating, discussing, sharing their work, and helping each other both on-line and in-person.

Shared repertoire consists of the experiences, stories, tools, and slang through which the Ubuntu hackers come to share through their daily use and customization of the Ubuntu system – the shared built environment which they adopt as their own. I argue that this taking up residence within the Ubuntu system is similar to Tim Ingold's notion of *dwelling* (Ingold 2000), and that the system affords the hackers' work to such an extent that it becomes an extension of the hacker's mind, as suggested by Clark & Chalmers (1998) – the all-encompassing means for achieving their individually diverse goals.

Furthermore, I claim that this community of practice is built not only through the hackers' *shared history of learning* to dwell and build within Ubuntu system, making it their own, but also through coming to trust and collaborate with the hundreds of developers and millions of users dwelling there in dependence on its continued, guaranteed development and integrity. I argue that this continuous building of the system, in what Matt Elliott calls a *stigmergic* fashion (Elliott 2006), which empowers the hackers to change the system freely, communicating directly through the changes they make to their shared environment. This stigmergic collaboration rests on the cultivation of a *mutual accountability* that is built in part on a reciprocal trust based on the hackers' personal on-line reputations verified through technical and cryptographic means central to membership of the community of practice. As well as in the reciprocal governance of the respected and well-reputed hackers leading the project in a manner similar to the rule of Melanesian *big-men* as described by Marshall Sahlins.

Finally, I suggest that it is the possibility to adopt, learn, configure, and even build the system according to their own needs, which opens the meritocratic community of practice for new contributors to scale the steep curve of

learning necessary to enter and learn among them. In this way, the Ubuntu hackers come to share more than just the same system, they come to share a common history, and, to a certain extent, a shared identity through their practice.

Introduction

Loading Ubuntu...

22:15:51 kernel: [17179588.740000] ts: Compaq touchscreen protocol output
22:15:51 kernel: [17179588.784000] NET: Registered protocol family 23
22:15:51 kernel: [17179588.816000] found SMC SuperIO Chip (devid=0x5a rev=00 base=0x002e): LPC47N227
22:15:51 kernel: [17179588.816000] smsc_superio_flat(): fir: 0x230, sir: 0x2f8, dma: 03, irq: 3, mode: 0x0e
22:15:51 kernel: [17179588.816000] smsc_ircc_present: can't get sir_base of 0x2f8
22:15:51 kernel: [17179588.940000] parport: PnPBIOS parport detected.
22:15:51 kernel: [17179588.940000] parport0: PC-style at 0x378, irq 7 [PCSPP,TRISTATE,EPP]
22:15:51 kernel: [17179589.032000] Linux agpgart interface v0.101 (c) Dave Jones
22:15:51 kernel: [17179589.048000] agpgart: Detected an Intel 855 Chipset.
22:15:51 kernel: [17179588.188000] pci_hotplug: PCI Hot Plug PCI Core version: 0.5
22:15:51 kernel: [17179588.192000] shpchp: Standard Hot Plug PCI Controller Driver version: 0.4
22:15:51 kernel: [17179588.364000] input: PS/2 Mouse as /class/input/input1
22:15:51 kernel: [17179588.392000] input: AlpsPS/2 ALPS GlidePoint as /class/input/input2
22:15:51 kernel: [17179588.572000] input: PC Speaker as /class/input/input3
22:15:51 kernel: [17179588.648000] wbsd: Winbond W83L51xD SD/MMC card interface driver, 1.5
22:15:51 kernel: [17179588.648000] wbsd: Copyright(c) Pierre Ossman
22:15:51 kernel: [17179588.648000] wbsd: probe of 00:0a failed with error -16
22:15:51 kernel: [17179588.704000] hw_random: RNG not detected
22:15:51 kernel: [17179588.740000] ts: Compaq touchscreen protocol output
22:15:51 kernel: [17179588.784000] NET: Registered protocol family 23
22:15:51 kernel: [17179588.816000] found SMC SuperIO Chip (devid=0x5a rev=00 base=0x002e): LPC47N227
22:15:51 kernel: [17179588.816000] smsc_superio_flat(): fir: 0x230, sir: 0x2f8, dma: 03, irq: 3, mode: 0x0e
22:15:51 kernel: [17179588.816000] smsc_ircc_present: can't get sir_base of 0x2f8
22:15:51 kernel: [17179588.940000] parport: PnPBIOS parport detected.
22:15:51 kernel: [17179588.940000] parport0: PC-style at 0x378, irq 7 [PCSPP,TRISTATE,EPP]
22:15:51 kernel: [17179589.032000] Linux agpgart interface v0.101 (c) Dave Jones
22:15:51 kernel: [17179589.048000] agpgart: Detected an Intel 855 Chipset.
22:15:51 kernel: [17179589.048000] agpgart: Detected 16252K stolen memory.
22:15:51 kernel: [17179589.056000] agpgart: AGP aperture is 128M @ 0xb0000000
22:15:51 kernel: [17179589.492000] 8139too Fast Ethernet driver 0.9.27
22:15:51 kernel: [17179589.492000] ACPI: PCI Interrupt Link [LNKF] enabled at IRQ 10
22:15:51 kernel: [17179589.492000] ACPI: PCI Interrupt 0000:01:01.0[A] -> Link [LNKF] -> GSI 10 (level, low)
->
22:15:51 kernel: [17179589.492000] eth0: RealTek RTL8139 at 0xdfb24000, 00:02:3f:1a:5a:a3, IRQ 10
22:15:51 kernel: [17179589.516000] 8139cp: 10/100 PCI Ethernet driver v1.2 (Mar 22, 2004)
22:15:51 kernel: [17179589.556000] ACPI: PCI Interrupt 0000:00:1f.5[B] -> Link [LNKB] -> GSI 10 (level, low)
->
22:15:51 kernel: [17179589.908000] ieee80211: 802.11 data/management/control stack, git-1.1.13
22:15:51 kernel: [17179589.908000] ieee80211: Copyright (C) 2004-2005 Intel Corporation
22:15:51 kernel: [17179589.948000] usbcore: registered new driver hiddev
22:15:51 kernel: [17179589.964000] input: Logitech Optical USB Mouse as /class/input/input4
22:15:51 kernel: [17179589.964000] input: USB HID v1.10 Mouse [Logitech Optical USB Mouse] on usb-0000:00:
22:15:51 kernel: [17179589.964000] usbcore: registered new driver usbhid
22:15:51 kernel: [17179589.964000] drivers/usb/input/hid-core.c: v2.6:USB HID core driver
22:15:51 kernel: [17179589.996000] eth0: link up, 100Mbps, full-duplex, lpa 0x45E1
22:15:51 kernel: [17179590.004000] ipw2200: Intel(R) PRO/Wireless 2200/2915 Network Driver, 1.1.2kmprq

22:15:51 kernel: [17179590.004000] ipw2200: Copyright(c) 2003-2006 Intel Corporation
22:15:51 kernel: [17179590.004000] Driver 'ipw2200' needs updating - please use bus_type methods
22:15:51 kernel: [17179590.380000] intel8x0_measure_ac97_clock: measured 55453 usecs
22:15:51 kernel: [17179590.380000] intel8x0: clocking to 48000
22:15:51 kernel: [17179590.380000] ACPI: PCI Interrupt 0000:01:04.0[A] -> Link [LNKA] -> GSI 10 (level, low) ->
22:15:51 kernel: [17179590.380000] Yenta: CardBus bridge found at 0000:01:04.0 [14c0:0012]
22:15:51 kernel: [17179590.380000] Yenta: Using CSCINT to route CSC interrupts to PCI
22:15:51 kernel: [17179590.380000] Yenta: Routing CardBus interrupts to PCI
22:15:51 kernel: [17179590.380000] Yenta TI: socket 0000:01:04.0, mfunc 0x00111c12, devctl 0x46
22:15:51 kernel: [17179590.612000] Yenta: ISA IRQ mask 0x0860, PCI irq 10
22:15:51 kernel: [17179590.612000] Socket status: 30000006
22:15:51 kernel: [17179590.612000] Yenta: Raising subordinate bus# of parent bus (#01) from #01 to #05
22:15:51 kernel: [17179590.612000] pcmcia: parent PCI bridge I/O window: 0xc000 - 0xdfff
22:15:51 kernel: [17179590.612000] cs: IO port probe 0xc000-0xdfff: clean.
22:15:51 kernel: [17179590.612000] pcmcia: parent PCI bridge Memory window: 0xe0000000 - 0xefffffff
22:15:51 kernel: [17179590.612000] pcmcia: parent PCI bridge Memory window: 0xa0000000 - 0xaffffffff
22:15:51 kernel: [17179590.620000] ACPI: PCI Interrupt Link [LNKG] enabled at IRQ 10
22:15:51 kernel: [17179590.620000] ACPI: PCI Interrupt 0000:01:02.0[A] -> Link [LNKG] -> GSI 10 (level, low) ->
22:15:51 kernel: [17179590.620000] ipw2200: Detected Intel PRO/Wireless 2200BG Network Connection
22:15:51 kernel: [17179590.824000] ipw2200: Detected geography ZZR (14 802.11bg channels, 0 802.11a channels)
22:15:51 kernel: [17179590.948000] cs: IO port probe 0x100-0x3af: excluding 0x230-0x237
22:15:51 kernel: [17179590.948000] cs: IO port probe 0x3e0-0x4ff: excluding 0x4d0-0x4d7
22:15:51 kernel: [17179590.948000] cs: IO port probe 0x820-0x8ff: clean.
22:15:51 kernel: [17179590.952000] cs: IO port probe 0xc00-0xcf7: clean.
22:15:51 kernel: [17179590.952000] cs: IO port probe 0xa00-0xaff: clean.
22:15:51 kernel: [17179591.064000] lp0: using parport0 (interrupt-driven).
22:15:51 kernel: [17179591.076000] NET: Registered protocol family 17
22:15:51 kernel: [17179591.120000] SCSI subsystem initialized
22:15:51 kernel: [17179591.124000] ieee1394: sbp2: Driver forced to serialize I/O (serialize_io=1)
22:15:51 kernel: [17179591.124000] ieee1394: sbp2: Try serialize_io=0 for better performance
22:15:51 kernel: [17179591.144000] Adding 1461872k swap on /dev/disk/by-uuid/48fa4fcc-4186-4baf-90b0-a8e294b21408. Priority:-1 extents:1 across:1461872k 22:15:51 kernel: [17179591.236000] EXT3 FS on hda1,
22:15:51 kernel: [17179591.596000] kjournald starting. Commit interval 5 seconds
22:15:51 kernel: [17179591.604000] EXT3 FS on hda3, internal journal
22:15:51 kernel: [17179591.604000] EXT3-fs: mounted filesystem with ordered data mode.
22:15:51 kernel: [17179591.604000] kjournald starting. Commit interval 5 seconds
22:15:51 kernel: [17179591.612000] EXT3 FS on hda4, internal journal
22:15:51 kernel: [17179591.612000] EXT3-fs: mounted filesystem with ordered data mode.
22:15:51 kernel: [17179592.168000] NET: Registered protocol family 10
22:15:51 kernel: [17179592.168000] lo: Disabled Privacy Extensions
22:15:51 kernel: [17179592.168000] IPv6 over IPv4 tunneling driver
22:15:51 kernel: [17179592.176000] eth1: NETDEV_TX_BUSY returned; driver should report queue full via ieee_device->is_queue_full.
22:15:51 kernel: [17179597.520000] ACPI: AC Adapter [ACAD] (on-line)
22:15:51 kernel: [17179597.596000] ACPI: Battery Slot [BAT1] (battery present)
22:15:51 kernel: [17179597.612000] ACPI: Power Button (FF) [PWRF]
22:15:51 kernel: [17179597.612000] ACPI: Lid Switch [LID]
22:15:51 kernel: [17179597.612000] ACPI: Power Button (CM) [PWRB]
22:15:51 kernel: [17179597.612000] ACPI: Sleep Button (CM) [SLPB]
22:15:51 kernel: [17179597.788000] pcc_acpi: loading...

22:15:51 kernel: [17179597.912000] ACPI: Video Device [GFX0] (multi-head: yes rom: yes post: no)
22:15:52 kernel: [17179599.816000] [drm] Initialized drm 1.0.1 20051102

Something like this ran across my screen the first time I started Ubuntu Linux on my computer in November 2004.

It all started a month previously as I was browsing on the Internet looking for a new laptop computer. I noticed that some shops offered to sell me a computer with no operating system. An operating system is the collection of programs which not only manages the computer's resources but also provides a programming interface making it possible for other applications to make use of the system. Microsoft Windows is one such operating system, the cost of which is included in the price of most computers. But I had heard that there were other, cheaper, operating systems available, so I decided that this was a good opportunity to save some money, and ordered the computer *sans system*. I had heard about the whole Linux phenomenon, of computer geeks sitting in basements around the world, collaborating over the Internet, building a computer operating system in their spare time, which could be downloaded, installed, used and modified completely for free, and though I had a few friends who used it, their computer skills eclipsed mine so thoroughly that I had never considered it something for me to use. Yet, buying a new computer, I felt ready for an experiment. So I decided to install Linux.
But looking on-line, I found a haphazard jungle of different interpretations and ideas of what Linux was. There were literally hundreds of different versions of Linux for download. Being unable to assess these different versions on their technical merits, I eventually decided to go with a version called *Ubuntu* whose developers proclaimed it to be "Linux for Human Beings." They related its open development to Desmond Tutu's South African ideology of Ubuntu, a Zulu word meaning "humanity to others", while promising easy installation and use – even for less technically minded people like me.
When my new computer arrived, I installed the version of Ubuntu I had downloaded from the Internet, happy to find it working, but unnerved at the newness of it all: the file system was full of opaque folder names, changing a setting required typing esoteric commands, and whenever I started up or shut down the computer, it spouted hundreds of lines of technical messages like

those featured above.
If computers have souls, then the operating system is the eye through which the soul of the machine is reflected. And what and how much it reflects, is wholly up to the designers of the operating system. As I got used to the system, I became more and more intrigued with the transparency of it: All the technical details were laid bare before me, revealing the immense complexity of the operating system as it offered itself for interpretation. The complexity hinted at the enormous effort and depth of technical knowledge that had gone into its making and which I had simply taken for granted until now.
I soon learned that the transparency within the Ubuntu stemmed from its being *open source* software. Open source refers to the *source code* – which is what the programmer writes. Source code is simply text which, when conformed to a strict set of guidelines of commands and declarations defined in a programming language, can be used to instruct the computer. But in order for the computer to execute the code, the source code usually has to be translated into binary *object code* consisting of 1s and 0s using another program called a *compiler*. Thus, programming is not a direct interaction with the computer, but rather the writing of code that, when compiled – or *built* – can be run by the computer. The compiled version of the program is no longer readable by the programmer, it has become an opaque software tool which, if properly programmed, can be run, accept an input and produce an output, providing services and presenting its work through messages like those above, but the exact manner in which it works can only be discovered by examining the original source code. The object code can be run, copied onto a disk and sold, but it can no longer be edited in any way. This is the distinction between *closed-source* software such as Microsoft Windows, where Microsoft sells you a copy of the object code, but keeps the source code secret, only to be seen and altered by the programmers employed by Microsoft; and open source software such as Ubuntu where the source code is freely available, thus inspiring the term *free software* with which the Ubuntu hackers prefer to denominate their work.
A group of Spanish computer scientists measured the size of a Linux system similar to Ubuntu, and found that it contained around 230 million lines of source code. When they translated this into the effort spent on writing this code using a standard software industry cost estimate model, they found that

it would correspond to almost 60.000 man-years of work (Amor-Iglesias et. al. 2005). By comparison, it took an estimated 3.500 man-years to build the Empire State Building in New York, and 10.000 man-years to build the Panama Canal. This immense effort makes modern operating systems such as Ubuntu among the biggest and most complex engineering projects ever conceived and built.
But though I had all of the blueprints, all the technical details available to me in the form of these millions of lines of open source code, I did not have the esoteric engineering knowledge necessary to appreciate nor modify it. So instead, I turned my interest to the people who did, curious to learn more about how they collaborate to build such an intricate system, and to learn why they were doing all of this work just to give it away.

The focus of the thesis

In April 2006 I initiated anthropological fieldwork following the development cycle of a new version of Ubuntu, contributing and learning among the Ubuntu developers. These developers are computer enthusiasts who often refer to themselves as *hackers*, though not in the sense used by the press to describe malicious meddlers who break into computer systems. Rather, a hacker, in the original computer jargon, is "A person who enjoys exploring the details of programmable systems and how to stretch their capabilities, as opposed to most users, who prefer to learn only the minimum necessary"[1], thus, the term has become a shibboleth,[2] identifying those who use the broader sense (rather than the narrow intrusion-oriented sense) as members of the hacker community (Levy 1994, Turkle 1984: 207-225).
My initial impression of the Ubuntu project was that it was a club for bright boys with a penchant for computers, an impression which was supported by an on-line web survey I had developed in order to gather quantitative data on

[1] Many of these jargon terms, including this quote, are taken from the *Jargon File* – an on-line dictionary of hacker terms and jargon that has been compiled by various hackers since the early 1980's. It is currently under the editorship of hacker Eric S. Raymond. The Jargon File has been published by MIT Press under the name “The New Hacker's Dictionary” (1996).

[2] A shibboleth is commonly a unique pronunciation, word, behavior, or practice used to distinguish one group of people from another, and to identify individuals as either members of the group or outsiders. For example a saying that is strongly used by members of a group and regarded as meaningless, unimportant or misguided by outsiders.

the basic statistics of the Ubuntu community and to which I had received around 300 responses. The results showed that the Ubuntu community consists predominately of young (average age was 29 years) well-educated (more than 50% had a university degree, half of which majored in computer-related disciplines) men (97,5% male respondents[3]), living in the industrialized world (50% in Europe, 33% in North America, 10% in Australia and New Zealand with the remaining 7% spread over the rest of the world), and who spend much time working on the computer (75% of respondents spent more than 40 hours on the computer in an average week), both as a profession (50% of the respondents working in the IT industry, and another 25% of the respondents being students in related fields) but also as a hobby.[4]

Thus my field site was the Ubuntu community – those users of the Ubuntu system scattered all over the world who actively participate and contribute in the shaping and developing of the system both technically and socially. I define the "Ubuntu system" as the body of software that is actively developed, maintained and distributed by the Ubuntu community – both in the form of a Installation CD and in the installed system itself through which hackers interact with the computer. In April 2007, Mark Shuttleworth, the leader of the Ubuntu project, estimated that around 10 million computers were running the Ubuntu system.[5]

---

[3] The lack of female open source contributors, or perhaps rather, the overwhelming male dominance, is often a topic of discussion, even attracting a report on the matter from "Free/Libre/Open Source Software: Policy Support" (FLOSSPOLS) project (Krieger, Nafus, Leach 2006). The report concludes that rather than the result of a lack of technical interest among women, the lack of women in free software is related to a similar lack of women in the IT industry and related fields of education, rooted in a social environment that is, often unconsciously, unwelcoming to women (Ibid. 17). Though male hackers often deny such social factors, it is difficult to deny that gatherings such as free software conferences are fundamentally gendered spaces – much like boy's summer camps – where the few women present are not expected to be technically capable or inclined to participate. Unfortunately, a further examination of these issues is beyond the scope of this thesis – in part due to the fact that all of the informants I interviewed were male.

[4] The results of my survey generally corresponded with the trends of the recent EU-funded "Free/Libre/Open Source Software: Policy Support" (FLOSSPOLS) survey of more than 5000 free software hackers (Ghosh 2005b, 2006).

[5] Cf. https://wiki.ubuntu.com/MeetingLogs/openweekfeisty/askmark. Though there are millions of users of the Ubuntu system, it is important to note that only a fraction of these contribute actively to the development of the system. As stated, my focus is solely on this group of active, contributing users.

I joined the Ubuntu on-line community on the same terms as the Ubuntu hackers, contributing to and using the same system, sharing their experiences with the system, and meeting them in-person on the same terms as they do at the conferences at which they gather, experiencing the same social and technical means and limitations through which they develop the system.

Illustration 1: An Ubuntu installation CD containing the core Ubuntu system which provides access to the thousands of software packages of the Ubuntu system.

As my fieldwork progressed, I noticed that by far most of the day-to-day practice of the Ubuntu community was tied to the constant maintenance and improvement of the Ubuntu system, while the political or ethical discussions related to the reciprocal sharing of source code, which had been the focus of most of the earlier anthropological work so far (which I discuss below), seemed to be considered a well-defined implicit premise of the community. Intrigued by this new form of social collaborative organization and practice which allowed users and developers to collaborate to build and maintain such an immensely complicated engineering work, and seeking to fill this gap in the literature, I decided to focus on *how the Ubuntu hackers relate to the Ubuntu system in their individual and collaborative day-to-day practices*. I sought to examine the social and cultural relations in which the Ubuntu

system is developed so as to gain a better understanding of how this feat of engineering had come about and continued to improve.

Argument of the thesis

In examining the Ubuntu hackers' day-to-day practices, I argue that the Ubuntu hackers' shared use and development of the Ubuntu system constitutes a *community of practice* around their collaborative work and commitment to the project. By positing the Ubuntu community as a community of practice, I explore how the Ubuntu hackers are using new technical and social means to manage and share knowledge and skills on-line, and how these means of learning and sharing are reflected in the system itself. I argue that though the Ubuntu community offers complete access to every technical detail of the transparent system they develop, the social boundaries of this on-line community are defined through the active use and development of the system itself. Because of this, membership and participation in this community is gained through a *shared history of learning* the specialized knowledge and social norms this use and development requires, making the group of developers a meritocratic group joined only through dedicated collaborative work. Thus, despite the Ubuntu system solely consisting of free software, the freedom it offers can only be fully appreciated by hackers capable of developing it.

For this group of hackers, the Ubuntu system is the all-encompassing means offering them the freedom to fulfil their diverse personal, social motivations for contributing to the system. By building a system that works for each of them individually, the Ubuntu hackers come to construct a system which reflects their practices. But they also seek to ensure that the Ubuntu system, as well as the community of practice through which it is built, is open to all, depending on the users' willingness to invest the time and effort to scale the steep curve of learning necessary to adopt, learn, configure, and even build the system according to their own needs, and master the core practices and social norms required for membership.

I argue that this shared practice and history of learning to collaboratively build and maintain the Ubuntu system results in a careful mutual trust in the hackers' complementary abilities through which the integrity and solidity of the intricately complex Ubuntu system is guaranteed, and which the many

users of the Ubuntu system come to rely on. And similarly, it is through this reciprocal trust that the diversity of motivations and conflicting interests within the community of practice is managed under the reciprocal *big-man* leadership and ethos of a few prominent and respected core Ubuntu developers.

Previous research on free software

It is only in the past few years that comprehensive anthropological field studies of free software communities have begun to appear. Inevitably, these field studies have been multi-sited as the researchers have sought to come to terms with the physically distributed and on-line nature of the hacker communities. Christopher Kelty (2003, 2005) did fieldwork among hackers in Boston, Berlin, Bombay, and Bangalore focusing on the role of reputation and on-line technical infrastructure in shaping free software communities, while Yuwei Lin's (2004) fieldwork alternated between a *Linux User Group*, hacker conferences and on-line collaboration as she focused on how software innovation takes place in free software communities, examining how such innovation may be related to a shared notion of "hacker culture." And Gabriella Coleman's (2004, 2005) fieldwork, initially among San Francisco hackers and later among *Debian Linux* developers both on-line and off-line, focused on the hackers' ethical and political motivations for working with free software and their use of legal tools such as software licenses in maintaining the free availability of their work.

Meanwhile, other anthropologists, including James Leach (2005), Lars Risan (2005), and Gregers Pedersen (2006) have explored how the open sharing of source code within the free software communities relates to anthropological notions of gift culture and ownership as seen in ethnographic examples outside of the industrialized world.

Anthropological interest arrived much at the same time as many other social scientists began to take notice of the free software movement starting with the success of Linux up through the 1990s.[6] Indeed, the influx of scientific

[6] These social scientists began studying free software communities seeking to understand the inner workings of on-line collaboration, the open sharing of source code and the motivations for participating in such projects and their implications in such diverse disciplines as sociology (Kuwabara 2000, Castells 2001), philosophy (Himanen 2001), technology studies (Tuomi 2001), economics (Ghosh 2005a), business (O'Mahony 2002),

attention to the free software communities has been so strong that the editors of the recent anthology "Perspectives on Free and Open Source Software" noted:

> It has been said that the average Navajo Indian family in 1950s America consisted of a father, a mother, two children, and three anthropologists. Many in the F/OSS [Free/Open Source Software] community no doubt are starting to feel the same way, as what began as a software topic has attracted the efforts of so many researchers from sociology, economics, management, psychology, public policy, law, and many others.
> (Feller, Lakhani et. al.2004:xxv)

Most of these studies based their qualitative data on the publicly archived on-line exchanges between hackers and on surveys of both source code and developers, but also, due to the highly technical nature of the field, directly on literature written by hackers.[7] Many of whom act in a dual role as both informants for other academics and as academics themselves, making it difficult to distinguish between "indigenous" hacker interests and academic distance. One hacker, Eric Raymond, even refers to himself as "an observer-participant anthropologist in the Internet hacker culture" (Kelty 2002:note 18). So while it is true that free software communities have attracted attention from many academic disciplines, these communities differ from the case of the Navajo Indians in that much of the direct qualitative data used in these studies have been gathered by the hackers themselves, whose enthusiastic, self-reflective interest in their field has produced some of the best analyses on the topic.

Methodology

The scientific attention and the hackers' own self-reflective interest in their practices make the on-line free software communities a somewhat atypical anthropological field, since many of the experienced and well-travelled hackers have come to take the academic interest completely for granted, as I

communication (Ratto 2003), political science (Weber 2004), and law (Moglen 1999, Lessig 2000, 2001, 2004) as they attempted to re-align their respective disciplines with regards to issues such as economical incentives, community and business collaboration, meritocratic governance and organization, ownership and copyright law.

[7] Such as Raymond (1997, 1998), Riemens (2002), Reagle (2004), Hill (2004), Michlmayr (2004), Krafft (2006), Fogel (2005) and Stallman (2005).

found upon introducing myself at dinner on the first day of my first Ubuntu Developer Summit where the Ubuntu developers meet to plan the development, schedule and goals for the next release of Ubuntu:
- “Another anthropologist! It seems that most free software projects have an anthropologist attached these days,” one of the veteran hackers at the table replied, to the general agreement of the other hackers at the table.[8]
- “Really?” was the best reply I could muster.
- “Yeah, I often find myself talking to people at conferences and they ask a lot of questions and I ask them what they do and they say some social science, and I ask 'is this ending up in your thesis' and they go 'yeah' in a sort of apologetic fashion.”
- “Sorry I couldn't be more creative in my choice of field” I responded, unable to hide my disappointment.
- “Oh, no,” he replied “it is definitely a field that deserves as much study as possible. I'm sure it'll be quite helpful.”

I soon found this to be the general attitude among the Ubuntu developers who for the most part shared a keen awareness that they are breaking new ground socially, economically and politically. As another developer said put it, “We are your fruitflies,” arguing that much like insects bred for accelerated evolution for easy study, the free software communities are moving so quickly that it is a space not only for easy social experimentation but also for studying such radical social development in the making.

Many hackers, especially those with academic backgrounds, felt some kinship with me and sought eagerly to distance themselves temporarily from the community to offer their own observations and analysis of the community, as well as being very interested to hear mine. The narratives and observations they supplied were already scrutinized under their own analytical gaze before being passed on to me, showing how these hackers continually seek to master the dynamics of the communities of which they are part, as if it is a mathematical puzzle to be solved. I shared in these reflections while keeping a continuous anthropological focus which the hackers neither sought to nor

[8] I haven't been able to confirm this notion. I met only one other anthropologist doing fieldwork in a free software community during my fieldwork, and from what I have been able to surmise, most hackers tend to lump the various social sciences mentioned above together as anthropologists or social scientists. In either case, there is still very little anthropological work which has been published on the free software communities.

could maintain outside of the specific discussion.
Christopher Kelty, one of the few academic anthropologists in this field, has sought to operationalize his informants' self-reflection by making a distinction between what he calls *first-order observation* – the traditional qualitative data gathered first-hand by the anthropologist – and *second-order observation* – the descriptions and analyses "made by the actors that we are supposed to be studying." Kelty concludes that "It amounts to a situation where we compete with a lot of other people who are also providing explanations for the situation that we find ourselves amid," as we almost recursively are forced to examine not only the social facts themselves, but also the actors' representations of those facts and other actors' representations and understandings of those representations (Kelty 2003).
I find Kelty's first- and second-order distinctions useful to separate the reflections expressed and disseminated by hackers through the free software communities from my own fieldwork data where I could observe and correlate the actual actions of the Ubuntu hackers directly with the reflections which they expressed in the interviews. This focus on the hackers' day-to-day practices in relation to their own second-order reflections on these is inspired by my reading of the French anthropologist of science Bruno Latour's studies of the work practices of scientists (Latour 1987). Latour argues that in order to understand scientific and technical work, it is necessary to study the process, networks, action, and forms of labour through which that work is created. And, that once a scientific fact has been settled or a machine runs efficiently, the scientists and engineers using it come to take it for granted as they only need to focus on its inputs and outputs rather than its internal complexity, thus turning that body of work into a *black box* (Ibid: 2-3). Latour's studies are based on the *unfolding* of such scientific or technological black boxes, by tracing the network of practices, processes and actors through which they are constructed. I use Latour's approach as a methodological premise to examine how the software of the Ubuntu system is used and negotiated both socially and technically through practice, thus inductively exploring the interrelations between the Ubuntu system and the Ubuntu community. But though Ubuntu and other free software projects may appear to be a Latourian black box to me and other non-hackers, its radical technical openness through its source code and public processes allows hackers to

study, adopt, adapt and rework it continuously, making the system a transparent *white box* to the few capable of appreciating it. Thus, I seek to follow the Ubuntu hackers as they navigate this white box in order to unfold the layers of social and technical complexity of which it consists.

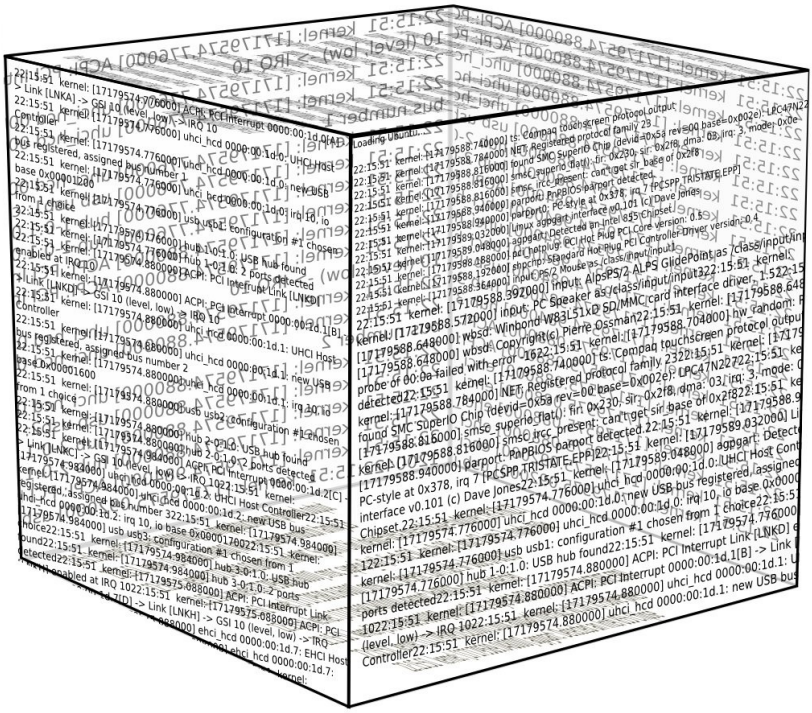

Illustration 2: The Ubuntu system as a transparent white box: Layers upon layers of source code openly available to the knowing user – the hacker – to constantly renegotiate, challenge, and redefine according to his needs.

In order to examine how the Ubuntu system is developed and its role as the centre of gravity of the Ubuntu community, I spent 6 months participating in

the development of Ubuntu, including a complete Ubuntu development cycle from June to November 2006.

Based on the insights of Coleman (2005) and Lin (2004), I sought to integrate the on-line and in-person aspects of my fieldwork as closely as possible, and following my first in-person meeting with the Ubuntu hackers at the Ubuntu Developer Summit in Paris in late June 2006, I sent out a fieldwork visit request to 22 of the Ubuntu developers I had met there. Between August and November, I gradually spent more and more time visiting the hackers in their own working environments, participating in their everyday life and interviewing them, as well as taking part in the interactions of the whole community on-line. I visited a total of 17 developers in 5 different countries in Europe and North America before ending my fieldwork at the following Ubuntu Developer Summit in Mountain View, California in November 2006, an event which gave both coherency and closure to the project.

Naturally, the methods I used and the data I gathered depended on whether I was in an on-line or an in-person context. The on-line context is wholly textual: Not only is all of the technical construction and maintenance of systems and infrastructure built through textual source code, but all of the communal means of communication and socialization also take place through a textual dimension through mailing lists, *blogs*, *web forums*, and *Internet Relay Chat (IRC) channels*[9] through which the Ubuntu developers interact and collaborate.

In order to do participant observation in this on-line space, I began contributing to the system by writing the system help and documentation, rather than the system itself due to my lack of technical understanding. In this way, I could take part in shaping Ubuntu alongside other community members while slowly developing a feel for the everyday exchanges and work in the community. But despite this participation, I was frustrated at my inability to participate fully in the on-line technical exchanges, and talk with the core developers, and it wasn't until I met the hackers in-person that I felt like I had really entered the field and became able to relate fully to the Ubuntu hackers.

[9] Developed in Finland in 1988, IRC made realtime textual interaction possible on a global scale. With IRC, a user can engage in several public discussions – so-called *channels* – as well as initiate multiple private conversations at the same time. Expert users deftly navigate back and forth between conversations, fluidly interacting with their on-line friends and co-developers.

The casual and often asynchronous on-line interaction was offset by the intense in-person interaction when visiting the individual developers or participating at conferences. During the five free software conferences I participated in, I met and talked casually with a lot of different individuals. I introduced myself as an anthropologist studying the Ubuntu community and never once was I asked to explain what an anthropologist is, much in keeping with the self-reflexivity discussed above.

When visiting the individual developers in their homes, I usually stayed with each developer for a few days, working alongside him and participating in his daily routines. Despite some initial concern for their privacy, the developers were generally very hospitable. But I was very conscious of not letting my visit disturb their daily routines too much, since that would to some extent compromise the everyday nature of the setting I sought to study.

Being in their homes and meeting their families and friends brought all of the loosely defined on-line and conference world into a well-defined real-life situation which I fleshed out through hours of interviewing, delving into the informants' background, especially with regards to computers and free software. A central part of the visits was the humanly mediated computer interview (Markussen 2002) – an attempt to explore the digital space of file hierarchies, social contacts and links contained within the informant's computer, with the informant as a guide explaining his configurations of the Ubuntu system. Since the informant's use of the computer was mediated through himself in this way, it is difficult to separate a hacker's actions from his descriptions of his actions, so I used "think-aloud" protocols (Lethbridge 2005), sitting with the developer as he worked through one or two of the tasks on his to-do lists, having him explain step-by-step his concerns and considerations in solving those tasks, and asking for examples of concrete situations and work cases where I could ask the informant to walk me through

the on-line paper trail he had left behind in order to see how he related to his own interactions on-line. This set the individual developer's tasks into context within the greater whole of the continuous development process of Ubuntu and not only helped to compensate for my lack of technical expertise, but also exposed just how much my understanding of these processes was mediated by the developers themselves, who, by continuously adapting, reworking and studying the system, came to appreciate its technical details as a transparent white box. I sought to alleviate this mediation by moving my focus from the specific tasks to the hackers' shared interactions in order to unfold the community of practices that define the white box of the Ubuntu system.

Theoretical approach of the thesis

Through my focus on a single free software project in order to study the developers' concrete collaboration and negotiation of the system which they produce, I came to consider the day-to-day issues and practices taking place in that space as inductively constituting what the educational theorists Etienne Wenger and Jean Lave have called a *community of practice* (Wenger 1998). A community of practice is more than a shared practice associated with a technical skill or specific knowledge, it consists of social relationships developing around shared interests, concerns and goals where members act as resources for one another, sharing activities and socialising through a shared set of routines, tools and norms (Ibid. 45-47).

In this way, I adopted a different approach than the anthropological studies of Coleman, Kelty and Lin mentioned above, which had focused on a wider notion of "hacker culture" by gathering data among hackers in general, following themes such as reputation, reciprocity, innovation and ethics across conferences and projects.[10] Especially Gabriella Coleman's ethnography of the Debian Linux community (2005) has inspired my own, and despite her

[10] It is worth noting that both Yuwei Lin (2004) and Matt Ratto (2003) identify the on-line hacker communities which they examine as communities of practice. While Ratto mentions it only in passing, Lin uses the term to explode the notion of a unified hacker culture, arguing that "the notion of 'hacker' acts as a boundary object that allows individuals to articulate their perceptions of, and perform collective practices in, the hacker social world." (Lin 2004:279). But since she uses Wenger's theory solely in relation to her chosen theme of IT innovation in free software communities, and not to examine the specific practices within a project, I find I can use little of her insight in relation to my examination of the Ubuntu community.

distinct research focus, her conclusions with regards to hacker ethics and the importance of in-person meetings is a reference throughout this thesis. Similarly so with Christopher Kelty's insights regarding the role of hacker informants' self-reflexivity and use of source code as means of argument in shaping and defining on-line communities. In combining these insights from related fieldworks with Wenger's overarching idea of the community of practice, I have sought to define a durable structure to examine the different elements of the Ubuntu community through. Following Wenger's claim that a community of practice is defined through three dimensions of relation, namely *mutual engagement*, *joint enterprise*, and *shared repertoire* (Wenger 1998:72-85), I have built a strategically selected constellation of theories to describe and analyze each of these dimensions in relation to the practices of the Ubuntu hackers.

Mutual engagement refers to the shared practices revolving around complementary contributions, overlapping forms of competence, and everyday sociality, which bind members of a community of practice together into a social entity. I draw upon Claude Lévi-Strauss' dichotomy of the *engineer* and the *bricoleur* (Lévi-Strauss 1994) to distinguish the complementary elements of the Ubuntu hackers' mutual engagement in developing the Ubuntu system. Furthermore, I use the American philosophers Clark & Chalmers' notion of *active externalism* (Clark & Chalmers 1998) to explore how Ubuntu developers extend their mind unto the computer in order to maintain the flow of interaction necessary for their shared development work on-line.

Joint enterprise consists of the diverse interpretations and motivations, which each developer brings to the community to be joined and continually renegotiated as the shared goal of the project, and in turn shaping the practices of the community. In seeking to understand the individual developers' motivations for their work, I draw upon Alfred Gell's theory that the aesthetic values worked into a given artefact also mediates the social relations between the maker and the spectator (Gell 1999). I build on the hypothesis that the way the Ubuntu developers relate to their code and imbue it with different aesthetic values reflects their diverse motivations for writing and contributing that code to the project, and that these contributions come to shape the shared goals of the project.

Following this, I also examine what Wenger calls *mutual accountability*, which springs directly from the negotiation of the joint enterprise as means to manage the individual motivations of the developers. I use the Australian media researcher Matt Elliott's notion of *stigmergic collaboration* and Christopher Kelty's exploration of how free software hackers build trustful reputations through development (Kelty 2005b) to link the Ubuntu hackers' reciprocal trust in each others' abilities with the reciprocal rule of Melanesian *big-men* as elucidated by Marshall Sahlins (Sahlins 1963).

Shared repertoire consists of the capabilities, the communal resources, the routines, sensibilities, artefacts, vocabulary, and styles that members have developed through their mutual engagement over time. In the case of the Ubuntu hackers, a central part of this shared repertoire is the Ubuntu system itself. I use Tim Ingold's notion of *dwelling* to describe how the individual hacker adapts the Ubuntu system for their personal use, both on the local computer and on-line, making the system itself the multiply inhabited, shared, and continually built scene of their shared interaction.

These three dimensions of practice are tied together through what Wenger calls a *shared history of learning* (Wenger 1998: 86ff). As the practices of a community slowly change, adapting to match its continuously changing technical and social circumstances, its members are tied together by the shared history of continuously ongoing learning through which they maintain their membership. It is by exploring the Ubuntu hackers' shared history of mutual engagement, joint enterprise, and shared repertoire, of learning through practice, that I seek to examine the community of practice around the Ubuntu system.

Structure of the thesis

The thesis seeks to explore the web of social, cultural and historical connections of the community of practice in which the Ubuntu system is made. Through five chapters, I will use five different social, technical and historical perspectives to give a nuanced description and analysis of the Ubuntu system. Through these perspectives, I will build on Etienne Wenger's suggested three dimensions of relations that compose a community of practice to examine how these fit with the practices of the Ubuntu hackers.

Starting with **chapter 2**, I will examine the Ubuntu system through a

historical perspective on the cultural and technical values upon which it is built, not only in order to give the background necessary to understand the argument of the following chapters, but also to outline the overall goals of the Ubuntu community through which the *joint enterprise* of the individual hackers are negotiated by answering the questions: Where did the project and the system come from? Which goals does the project strive to achieve? Based on second-order hacker reflections and narratives as well as the insights of other researchers in the field such as Gabriella Coleman and Steven Weber, this chapter will give a short introduction to the history of the personal computer and and the distributed development of the Unix operating system, which afforded the technical platform for customization and development that inspired the social context of the free software eco-system into which the Ubuntu project was born.

The Ubuntu system contains more than 19.000 software packages, each developed with its own technical quirks and social relations, by developers particularly interested in that package. In **chapter 3** I focus on the case study of *Synaptic*, one such software package, through a sociological perspective, based on interviews and think-aloud protocols, in order to describe the personal backgrounds and everyday lives of the two Ubuntu hackers developing it, and the close and personal collaborative practices through which *Synaptic* has been – and continues to be – developed. Through this description, I seek to answer two questions: Who are the people working on the system? How do they work on it? In order to explore the *mutual engagement* shared between the developers by drawing upon Claude Lévi-Strauss and Clark & Chalmers in analysing how the Ubuntu hackers' shared practices are complementary, depending one the knowledge and interaction with one another, as well as the flow afforded by the intimate work with the computer.

In **chapter 4,** I widen my perspective from the one-on-one collaborative practices of the Synaptic hackers to the ways in which the Ubuntu hackers negotiate the *joint enterprise* of the project by bringing their diverse motivations and interests in play in their personal contributions to the Ubuntu system in order to answer the question: Why do they contribute to the Ubuntu system? I draw upon Alfred Gell to analyze how the developers of three different Ubuntu software packages imbue their code with different aesthetic

qualities based on their different motivations, and how these motivations may come to conflict. I compare these motivations with the second-order explanations of hacker aesthetics and ethics commonly given by the hackers themselves to conclude more specifically on the nature of the joint enterprise of Ubuntu as touched upon in chapter 2.

Having examined the everyday hacking practices and individual motivations of the Ubuntu hackers, in **chapter 5**, I turn to a phenomenological perspective in order to examine how each Ubuntu hacker individually adopts and customizes the system as a whole for their personal use and development practices in order to answer the question: How do the hackers themselves use the system? I draw upon a case study from my humanly-mediated computer interviews of one hacker's configuration of the Ubuntu system as well as the Tim Ingold's notion of *dwelling* in order to analyse the Ubuntu system itself as the all-encompassing means that both the Ubuntu hackers and users come to depend upon, and as the *shared repertoire* through which the Ubuntu hackers develop their mutual engagement in the project over time.

This will set the stage for **chapter 6** where I will explore how the individual Ubuntu developers' collaborative practices, motivations and uses of the system are negotiated from a community- and system-wide perspective. I will seek to answer the question: How do the Ubuntu hackers negotiate and take responsibility for their shared work on the Ubuntu system? and I do this by drawing upon Matt Elliott's concept of *stigmergic collaboration* to explore how the individual Ubuntu hackers are empowered to maintain and extend the system. Following this, I will use Christopher Kelty's work on on-line reputation and trust as well as Marshall Sahlins' ethnography of Melanesian big-men to analyze a situation where this open collaboration results in a breakdown of trust, and how mutual accountability is ensured through a reciprocal trust defined through both technical and social means, which, along with a few trusted hackers' reciprocal big-man leadership manages the diversity and conflicting interests of the community to guarantee the stability of the system

Finally, **chapter 7** will sum up my main argument by revisiting how the hackers' mutual engagement, joint enterprise, and shared repertoire are expressed through their motivations, goals, practices, use and discussions of the system. And how the shared history of learning and adapting to these

practices shape the intricately complex white box that is the Ubuntu system.

Chapter 2

Tracing the cultural and technical roots of the Ubuntu system

We build our computers the way we build our cities -- over time, without a plan, on top of ruins.
- Ellen Ullman (1995)

Despite being a young endeavour, Ubuntu, both as a community and an operating system, builds on traditions from IT industry and computer science academia going back to the first digital computers. In this descriptive chapter, I draw upon leading hackers' own second-order narratives and reflections of this history, as well as relevant analytical insights from social scientists in the field to outline the origins of technical and social values and practices carried on in the Ubuntu system and its development community in order to present the free software communities in sufficient detail to understand the background of the Ubuntu project. This includes its overall institutional humorous-but-serious goal of "world domination" based on which the Ubuntu hackers negotiate what Etienne Wenger calls their *joint enterprise* – their individual motivations and interpretations of those overall goals (Wenger 1998:77-82) which in part defines the Ubuntu community of practice.

The personal computer and proprietary software

In 1975, the Altair 8800, the first computer affordable to home users became available, and soon after its appearance, a market for selling commodity software was established. Up until this point, software had been considered by the IT industry as an added benefit bought with a computer, or something that had to be written on the computer for a specific purpose. The Altair was the first commodity computer for which the manufacturer did not provide generalized software, and so software had to be written or bought separately. Seeing a golden opportunity, two young hackers, Bill Gates and Paul Allen, wrote a version of the popular and accessible programming language BASIC for the Altair which greatly increased its programming potential. But not all of the interested users could stand waiting for their copy of the BASIC paper

tape to arrive in the mail, and they began making copies of the versions that had already arrived and shared them among themselves following the rule that if you took a tape, you should bring back two copies to give away at the next meeting of their monthly Silicon Valley “homebrew computer club” meeting (Levy 1994: 227-229). This copying angered 19-year old Gates so much that he wrote an “Open Letter to Hobbyists” which was published in the Homebrew Club Newsletter:

> Most of these "users" never bought BASIC (less than 10% of all Altair owners have bought BASIC) [...] Why is this? As the majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid? [...] Who can afford to do professional work for nothing? What hobbyist can put 3-man years into programming, finding all bugs, documenting his product and distribute for free?
> (Gates 1976)

Gates' idea of selling software met criticism from both the hobbyist hackers, who found it imperative to share the software so that everybody could use it and learn from it, and from business people who objected that it was impossible to sell something which was basically just a string of 1s and 0s which could be copied so easily (Stephenson 1999:33). But the latter soon changed their minds as personal computing began to spread, and the demand for usable applications and games grew among users outside the hacker community.

As political scientist Steven Weber points out, this was the first clash between the nascent ethics of open source and proprietary software – two fundamentally different economic models for the production of software (Weber 2004:37): Proprietary software requires the source code to be closed to the user so that he is unable to copy and modify it as he pleases, both to avoid new versions of the software that the company cannot support, and to avoid unlicensed redistribution from which the company does not profit. In opposition to this is the open source model which requires the free sharing and modification of source code in order for any interested hacker to be able to use and adapt the code to his needs, whether profitable or not.

While many hackers continued to share the source code to their programs, it mostly happened in such a casual manner on tapes and floppy disks at

computer clubs and universities that was no match for the market-driven explosion of commodity software that arrived with the personal computer. By 1981, the new PC market was dominated by Apple Computer, a company started by Steve Jobs and Steve Wozniak, two members of the same computer club that had frustrated Gates so. At this point IBM, the old industry leader, finally introduced a PC of its own, bundled with the MS-DOS operating system developed by Bill Gates' company, Microsoft. Apple integrated its hardware and its operating system Mac OS, which by 1984 included a mouse-driven graphical user interface (GUI), tightly together in order to make computer as appealing as possible to the wider public. Microsoft on the other hand sold MS-DOS, and later their own GUI superstructure Windows, separately to all of the competing hardware producers who had begun making IBM-compatible PCs. With a shared software platform, the price of commodity hardware plummeted, making personal computers even easier to build or buy, and by the mid-1990s, Microsoft dominated the PC operating system market completely (Stephenson 1999:80-90).

Free software and the heritage of Unix

Up through the 1970s, as PCs and commodity software changed the IT industry landscape, another operating system called Unix, first developed in 1969, was becoming very popular on powerful multi-user computers at companies and universities around the world (Salus 1994: 119-136). Its popularity stemmed from an empowering design philosophy "that users know better than operating system designers what their own needs are" (Raymond 2004:6), and the fact that AT&T, whose engineers developed Unix, had been forbidden from entering into the computer business as a result of a settlement in an anti-trust case in 1956, which required the company to license its non-telephone technology to anyone who asked (Ibid:33-34). Since its inception in 1969, Unix has seen such widespread adoption and development that hacker-turned-novelist Neal Stephenson describes it as the hacker "Gilgamesh epic:"

> What made old epics like Gilgamesh so powerful and so long-lived was that they were living bodies of narrative that many people knew by heart, and told over and over again – making their own personal embellishments whenever

> it struck their fancy. The bad embellishments were shouted down, the good ones picked up by others, polished and improved and, over time, incorporated into the story. Likewise, Unix is known, loved, and understood by so many hackers that it can be re-created from scratch whenever someone needs it.
> (Stephenson 1999:88)

Like an oral history, Unix, and most other software with shared open source code, was developed slowly over time, shared and copied on disks at conferences and at user group meetings where users could take the bits they liked and improve on them, and thus slowly improve system, hoping that their changes might be adopted by AT&T for the next official Unix release. Unix did not play much part in the rise of the personal computer since it was designed for expensive high-end equipment that was mostly unavailable for hobbyist hackers, and because another anti-trust case against AT&T in 1983 allowed the company to commercialize Unix and charge exorbitant license fees for all use of the source code (Weber 2004:39). This made the sharing of source code and programming ideas that defined the university hacker communities very difficult, as any hacker would have to sign a non-disclosure agreement in order to access the system.

Richard Stallman, a hacker at the MIT Artificial Intelligence lab, found it morally intolerable for an owner of software to say "'I won't let you understand how this works; I'm going to keep you helplessly dependent on me and if you share with your friends, I'll call you a pirate and put you in jail.'" (quoted in Moody 2001:28), and in 1984, he quit his job at MIT in order to devote himself to what he called *free software* – software that anyone can freely could study, use, modify and redistribute in any manner they please as long as they also freely redistribute their own changes to the source code. To advance his goal, he began writing a completely free version of Unix which he, in a typical case of hacker wit, called *GNU* – a recursive acronym for "GNU's Not Unix." And building on the same "oral tradition" described by Stephenson, interested hackers soon began to send him improvements and suggestions, which he could incorporate in new releases of the GNU programs.

As the project grew, Stallman created the *GNU General Public License* (GPL) to legally ensure that the GNU programs would remain free software. The GPL ensured that users could not modify the rights granted by the GPL to any user, meaning that any GPL software even if improved and combined with non-GPL software could only be redistributed on GPL terms.
Gabriella Coleman points out that with the GPL Stallman cleverly hacked traditional copyright law by inverting it: It was not a question of enforcing an authorial monopoly, but to ensure that his software was freely available and impossible to monopolize, in effect coercing all users of his software to give away their own improvements (Coleman 2005: 85-87). In this way, Stallman managed to use the GPL to formalize a sort of *generalized reciprocity* where "the return favour is not determined by time, quantity or quality: the expectation of reciprocity is indefinite" (Sahlins 1972).
Sahlins' definition depends on social proximity, but as Christopher Kelty argues, with digital technology and its inherent possibility of endless copies at zero-cost, the F/OSS community has succeeded in scaling this system of exchange from a local level to a global one through software licences such as the GPL (Kelty 2002). The reciprocity of *GPL'd* software, implicit in the "oral tradition" of Unix, does not lie in the individual transaction or specific social relationship between developer and user, which is often anonymous or even non-existent, but rather in the general shared interest and use of the code. Economist Rishab Ayer Ghosh compares it to a tribal cooking pot where each hunters contributes a small piece of meat and all can take out a bowl of stew of increased value (Ghosh 2005a). The anthropologist James Leach stresses that though each hacker retains complete copyright and ownership over his own code, it is only when combined with the rest of the GPL'd code that that code becomes coherent and valuable, thus GPL code is *multiply owned* by all contributors (Leach 2005). Thus, as Gregers Petersen concludes, free software is a culture of sharing, rather than one of gift-exchange, where each contributor cedes the right to control his property, *sharing* it freely under the condition that all other contributors will do the same (Petersen 2006).

The Linux open model of development

By 1991, Stallman and other contributors had produced almost all of the components required for a complete free Unix system, which he distributed

over the still-nascent Internet. Up through the 1980s, access to the Internet had, much like Unix, generally been limited to major companies and educational institutions that could afford the expensive Internet infrastructure and Unix licenses. But the hackers there would rather use the GNU utilities than the proprietary Unix code which they could no longer freely hack. One of these hackers was Linus Torvalds, a young Finnish Computer Science student. Frustrated at waiting in line at the university Unix terminals, Torvalds wanted to run Unix on his new PC at home in early 1991, and all he needed to complete his GNU system was a kernel – the core program which allocates resources to the various applications and users present on the system – which he experimentally began to write over the summer holidays that year.[11] Since Finland had invested heavily in telecommunications infrastructure, he was one of the first to have Internet access from home, and on October 5th, he announced his kernel project on the Internet:

> This is a program for hackers by a hacker. I've enjoyed doing it, and somebody might enjoy looking at it and even modifying it for their own needs. It is still small enough to understand, use and modify, and I'm looking forward to any comments you might have.
> (quoted in Moody 2001:45)

Torvalds' project became known as Linux. And though it used an out-dated design and just barely worked, it did make it possible to run a free Unix system on commodity hardware. Hackers who wanted to use Unix at home soon began to adopt Linux, since it was a system with potential for growth, to which they could contribute and which generally seemed like a lot of fun. What was remarkable about Linux was that Torvalds embraced the fact that the Internet allows people from all over the world to collaborate instantly whereas earlier, such collaboration had been limited to the slow and random "oral history" exchange of tapes and disks. He used the Internet to constantly release new, rough versions of the code for others to test simply by making it available for download on-line, incorporating *patches* – code improvements – and suggestions which other hackers sent to him, discussing designs and plans

[11] Torvalds based his design on Minix, a small redesigned version of the Unix kernel made by computer science professor Andrew Tanenbaum for educational purposes. Tanenbaum refused to collaborate with Torvalds, as he considered his design to be old and awkward.

on mailing lists and giving them the opportunity to add their own changes as they proved themselves technically capable. This *open model of development* had Torvalds acting as a focal point for the community due to his intimate technical understanding of the project and diplomatic mailing list leadership (cf. Ratto 2003).

This was completely unlike the *closed model of development* typically used within the IT industry. Considered the only viable model, the closed model focused on a tightly knit group of developers sitting together to collaborate intimately to produce a release according to the tight deadlines and many feature specifications traditionally imposed upon them by the management, the rule being "Adding manpower to a late software project makes it later" as each new team member took up resources for training and integration into the team (cf. Brooks 1995). Even the GNU project, with all its freedom of modification and redistribution, had kept its development process closed to ensure an effective work environment, and only rarely released new versions, incorporating suggested changes based on closed discussion.

The open development model of Linux presupposed that the source code was openly available to anybody interested in participating in the project – a stance that was incompatible with proprietary software development where a closed development model is the only way to ensure that others couldn't steal the work and trade secrets being developed. In a seminal paper first recognising the differences between the two models of developement, hacker Eric Raymond characterized the closed development model as a *cathedral*: A closed structure architected and maintained by a chosen few compared to the open model of development which he remarked often “resemble[s] a great babbling *bazaar* of differing agendas and approaches,” the centre of which is the individual hacker's need to scratch his own itch and make the software work well for his own needs (Raymond 1997).

Raymond's metaphors build a generalized and idealized dichotomy between a monolithic, authoritarian, closed (and thus inherently bad) Cathedral against a democratic, distributed, open (and thus inherently good) Bazaar. Raymond's dichotomy has a clear political agenda as he seeks to position free software at the positive pole of his libertarian worldview by comparing the open development model to an idealized capitalist free market where self-interest drives an open exchange between equals. Yet Raymond's metaphor turns out

to be severely imprecise, as the open development model still tends to work through a centralized meritocracy led by project leaders such as Linus Torvalds, while several observers have noted how closed development companies such as Microsoft or Apple function like idealized open development models internally within their organizations (cf. Bezroukov 1999). Furthermore, as computer scientists Antony Senyard and Martin Michlmayr have shown, most successful free software projects are initially characterised by closed development by a small group of developers similar to traditional software development, and it is only once its code base has matured sufficiently to be somewhat usable that it can bloom into an open model of development (Michlmayr & Senyard 2004). Certainly, the main significance of the open development model lies in the on-line public nature of its collaborative environment and sharing of source code which allows for a much greater degree of interaction and integration between previously unrelated software projects, but to claim that this in itself guarantees an egalitarian and open process would be faulty.

Linux soon matured, and when Torvalds licensed it under the GPL, the free Unix system that Stallman and thousands of hackers around the world had worked and hoped for became a reality.

For a new generation of young hackers accustomed to the binary opacity of MS-DOS or Windows, finding GNU/Linux – a massive feat of engineering built with the architectural transparency of Unix and with all its millions of lines of source code freely available – was nothing short of an epiphany (cf. Coleman 2005: 292), and with the introduction of the World Wide Web and the following commercialization and popularization of the Internet from the mid-1990s, more and more hackers all over the industrialized world began to get involved with Linux and free software on their own PCs at home, User Groups and conferences began to appear for Linux in much the same manner as they had for Unix 15 years earlier.

With all of this success in the face of the massive market dominance of Microsoft, Torvalds was quoted as defining his plan for the future as “World domination. Fast,” which has since become a recurring humorous-but-serious theme as the primary goal of many free software projects.

I contend that this near-epiphanic finding and learning to use GNU/Linux,

along with discovering the possibilities of on-line collaboration constitute a central part of the *shared history of learning* (Wenger 1998: 86ff) of skills and norms that that defines the Ubuntu community of practice, and that it is through the same kind of youthful enthusiasm and ambition displayed by Linus Torvalds that many hackers come to understand their joint enterprise.

Debian and the free software eco-system

As interest grew, so did the collaboration between projects, as more and more applications came to be supported under Linux, and even more began to be written directly under the GNU GPL for use with Linux. Despite its growing popularity, it was still a formidable task to get Linux installed and functional as it involved compiling, configuring and maintaining everything yourself. To make it easier to install and run Linux, *distributions* of Linux began to appear, integrating much of the new software with GNU/Linux for easier installation and maintenance. In this way, a flow of new software packages appeared as the distributions sought to incorporate the latest changes from the Linux and GNU projects into their releases. This flow has made it common to refer to projects like the Linux kernel as the *upstream* for the distributions which in turn are the *downstream* for the original developers. Many hackers call this the *free software eco-system* as it is characterized by heavy interdependence between the projects – not only between the upstream software producers and the downstream integrators and redistributors, but also between the various upstreams who use the tools written and provided by other upstreams (cf. illustration 3).

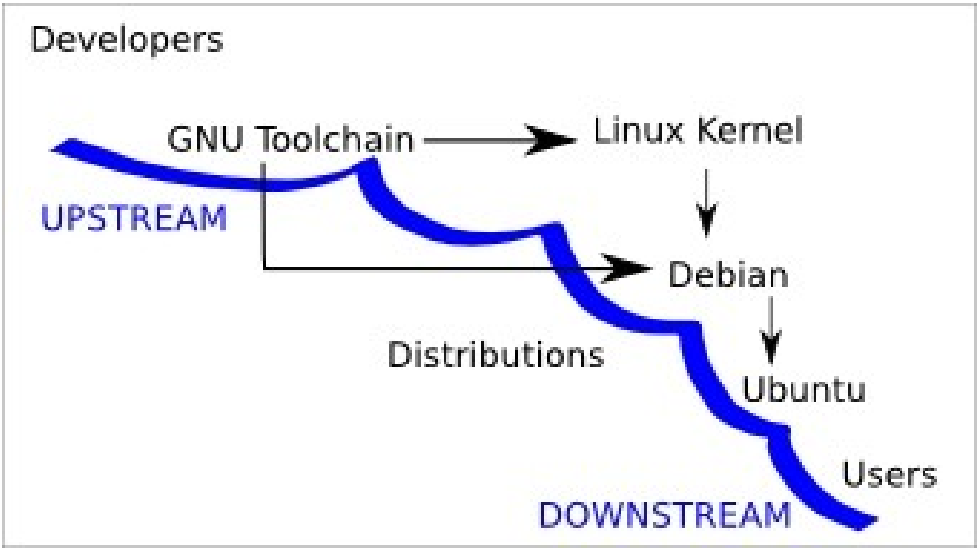

Illustration 3: A simplified diagram of the free software eco-system centered around the flow of software from the upstream developers such as GNU and the Linux kernel downstream through the distributions such as Debian and Ubuntu which integrate and distribute it to the users.

Many of these early distributions were made by software companies such as Red Hat and SuSE that based their business on added value, support and services to the Linux operating system which they integrated in a closed development model. In opposition to this was the non-commercial Debian distribution, founded in 1993 by young hacker Ian Murdock who sought to build an on-line community of volunteers to collaborate in the same open model of development used by the Linux kernel hackers. Taking infrastructural and collaborative pointers from the development of Unix and Linux, Murdock sought to make distribution development as modular as the core operating system itself by managing the upstream software as modular software packages in order to integrate the work of all of the upstreams in one distributive system. Each software package was to be integrated and maintained by a Debian developer in order to maintain a clear line of responsibility between the upstream and the distribution.
For instance, the upstream GNU developers release a new version of the text

editor *Emacs*.[12] The Debian developer responsible for Emacs then acquires the source code for this latest version, tweaking, configuring and compiling it to ensure that it works with the rest of the Debian system – since some packages *depend* on the presence of other packages on the system in order to work, it is necessary to check that these will work with the new version as well. He then builds a new Emacs package consisting of the original source code and his own patches and uploads the new Emacs package to the on-line Debian archives from where it can be downloaded and installed on the individual Debian systems using a *package management tool* – a specialized program for the installation, removal and updating of the modular packages of the system. Since the Debian developers patched the upstream source code to ensure that it worked with their system, they also risked introducing new *bugs* – programming and design errors in the software – into the software. In order to keep track of these, the quickly growing Debian community also developed an on-line *Bug Tracking System* – an on-line database where developers and users could register the bugs they found in the Debian packages so that they could be tracked to see whether they came from upstream or whether they had been introduced downstream. The Debian Bug Tracking System created a public forum centered around bugs within each package and the related discussion of their severity and proposed solutions. Keeping an open and public record of the errors of the software is the essence of the open development model, since it allows any interested hacker to find confirmation of a suspected bug and participate in fixing it.

At first, the Debian Developer community was open to all interested hackers but as the project grew, an application process was instituted to ensure that new Debian Developers were not only technically capable but also able to discern whether a software license (of which there are hundreds) is compatible with Debian's policies of supporting the free software dogmas and refusing to ship software not under open source licenses in order to keep Debian secure against potential lawsuits (cf. Coleman 2005: 394-407). Jordan, an Ubuntu contributor, compares the process of becoming a Debian Developer to getting a PhD as it requires much dedication and skill and can take months or even years to get approved. The Debian Developers have built an egalitarian community with intricate on-line voting schemes and annual

[12] For more on the history and development of Emacs, see Kelty 2006, Lin 2004.

elections for the position of Debian Project Leader (DPL) from which Murdock stepped down in 1996. Yet all the non-technical contributors for such areas as artwork, documentation and distribution who do not have the technical expertise to apply have no vote or influence over the direction of the project, and there are strong groups with much vested power even among the Debian Developers themselves which control vital processes within the community (cf. Coleman 2005).

Ubuntu and the commercialization of Linux

While Debian developed the communal aspects of free software development, the dot-com boom of the late 1990s brought Linux into the broader public eye. Due to the increasing industry interest for Linux webservers, a growing number of developers agreed on rebranding Linux and the open model of development as *open source*, as they were unhappy with the ambiguity of the term free software in a business context. This sparked debate in the developer communities between those who saw the central part of free software to be the pragmatic use of the source code – including use in conjunction with proprietary closed source software, and those who saw free software as the only, dogmatic way to ensure their uncompromised freedoms in the form of the generalized reciprocity formalized in the GPL (cf. Weber 2004: 112-115). This rebranding facilitated the IT industry to embrace the open model of development without having to relate themselves directly to Stallman's software freedoms, enabling them to find viable business opportunities in free software.

One Debian Developer who managed to do this was a young South African entrepreneur named Mark Shuttleworth. In the late 1990s, he built a digital signature business based on Debian technology, eventually selling his company at the height of the dot-com boom and became a multi-millionaire nearly overnight.

After spending a couple of playboy years becoming the first African and the second tourist in space, he began to consider how to invest his money. Thankful for all the free software tools that had made his success possible, he decided to give back to the free software communities, and having seen how

proprietary software like Microsoft Windows was forcing countries like his native South Africa to spend millions on software licenses, and in disappointment at the unwillingness of the commercial Linux distributions to embrace the possibilities of the open development model and challenge Microsoft for more than a minimal market share, Shuttleworth made it his lofty goal to break Microsoft's majority market share of PC operating systems. As he later stated in the first bug report to be entered into the Ubuntu bug tracking system:

> Microsoft has a majority market share in the new desktop PC marketplace. This is a bug, which Ubuntu is designed to fix [...] Non-free software is holding back innovation in the IT industry, restricting access to IT to a small part of the world's population, and limiting the ability of software developers to reach their full potential.

Remarkably, this argument was a complete reversal of the Bill Gates' statement to the hobbyists 30 years previously. By casting Microsoft's monopoly on desktop PC operating systems as a bug to be fixed, Shuttleworth defines proprietary software and the related closed model of development as means of profit from the bygone pre-Internet, pre-commodity-hardware era, which now are active impediments to software innovation. By positioning Ubuntu as the solution to this bug, Shuttleworth put the dogmas of the free sharing of information at the centre of software innovation, reflecting this in choosing the name *Ubuntu* – an ancient Zulu word meaning “humanity towards others” whose associated ideology of shared humanity summed up the open model of development of Debian on which he sought to base his project and his experiment of software innovation.
By defining the main goal of the Ubuntu project as breaking Microsoft's global dominance on PC operating systems, Shuttleworth touched upon the theme of global domination – but unlike Linus Torvalds and the many hobbyist hackers who had considered the notion to be only half serious, Shuttleworth decided to put a lot of capital and effort into the project. In early 2004, he contacted 12 free software developers, among them several of the most active Debian developers, and brought them together for a meeting in London to hire them to work on a completely free (both *gratis* and *libre* in an attempt to sidestep the open source vs. free software debate) Linux

distribution based on Debian – which at this point was highly regarded and widely used among hackers due to its technical quality and openness which to no small degree stemmed from its elaborate technical infrastructure.
Breaking with one community to take a project in a new direction, called *forking*, is one of the most significant political actions that can be taken in a free software community. Forks usually happen when a free software project makes a controversial decision such as making radical design changes or changing its license. The developers who disagree with the decision break out of the main project to form a new project that builds on the same codebase but without the controversial design changes or under the old license.
Well aware of the political significance of a fork, Shuttleworth had realized that he couldn't make these changes against the powerful groupings within the egalitarian community of Debian which had agreed on an inclusive stance to its users, labelling the Debian system the “universal operating system”, thus making it untuned to any single user group. Thus he sought to present Ubuntu as a *derivative* of Debian focused on desktop usability to be shipped for free across the world at his expense in order to win over Windows desktop users.
Knowing that desktop users want to have the latest software available, Shuttleworth decided that Ubuntu should follow a strict time-based release cycle dictating that the Ubuntu community must release a new version of the Ubuntu system every 6 months - unlike Debian's fairly erratic feature-based release cycle of releasing a new version whenever they feel they have made enough progress to warrant one (cf. Illustration 4).

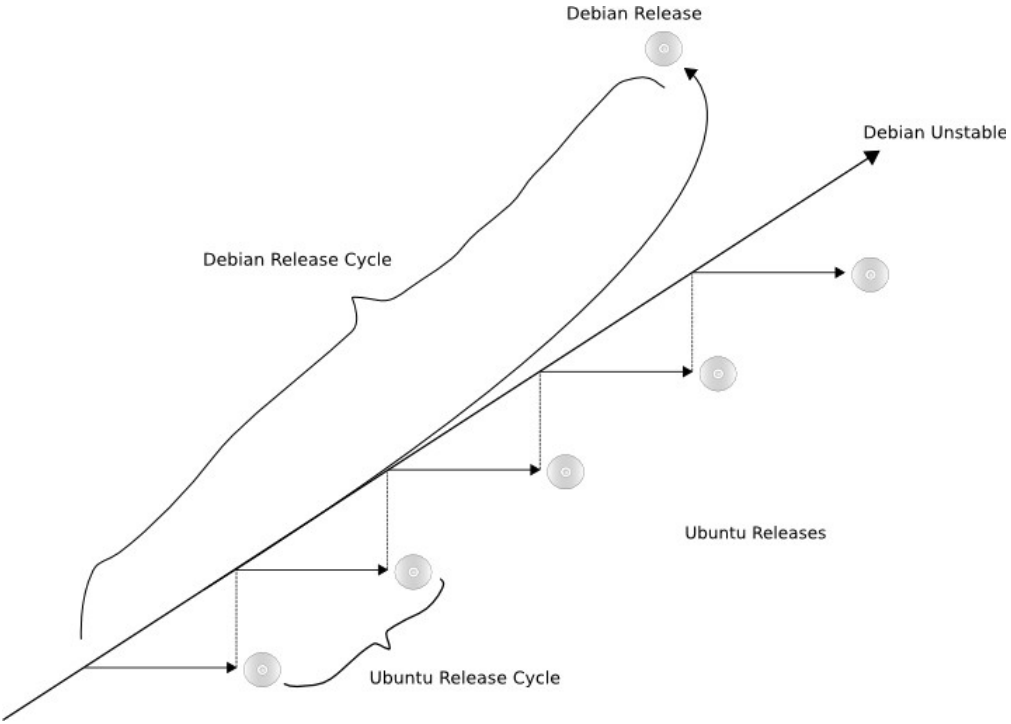

Illustration 4: The differences in Debian and Ubuntu's release cycles. Both branch off from the collection of the latest, volatile upstream free software of Debian Unstable and seek to stabilize that into a releasable state.

Ubuntu and Canonical

At the time of writing, three years since Shuttleworth first began hiring the Debian developers, Ubuntu has seen six releases, and grown to be one of the most popular Linux distributions. Using the catchphrase “Linux for Human Beings”, Ubuntu has come to be seen as the more user-friendly version of Debian, attracting many users and contributors due to its clearly stated meritocracy that acknowledges technical and non-technical contributions on equal terms (this will be explored further in chapter 6). Shuttleworth sits at the centre of this meritocracy in a role similar to that of Linus Torvalds in the Linux Kernel project. But Shuttleworth's role as employer, top authority in the community governance structure, and sole financer of the project has made his leadership much more explicit – a fact that he acknowledges in giving himself the humorous on-line moniker *SABDFL* – an acronym for *Self-*

Appointed Benevolent Dictator-For-Life. By clearly stating this fact, Shuttleworth makes no false pretences and soaks up much of the focus on the success of Ubuntu which in no small part has been achieved by hiring some of the most capable and ambitious Debian developers and leveraging the new breakthroughs in free software which they have been part of. This has created a fair amount of social friction between the Ubuntu and Debian communities, partly because many of the other Debian Developers feel that Ubuntu is stealing their thunder, and partly because Ubuntu is diverting resources away from a genuine communal project to a project controlled, in effect, by one man.

Many of the core Ubuntu developers see Ubuntu as a much needed reconfiguration of the social and technical structures of Debian – a kind of "Debian 2.0": a necessary streamlining in order to make free software more appealing and accessible in order to challenge the dominant paradigm of proprietary, closed-source software.[13] They accept that such change would be impossible within Debian and as a young project, they use this opportunity to continually redefine and refocus the processes of the Ubuntu community to better suit their joint enterprise towards world domination.

Similarly, Canonical, the firm through which Shuttleworth employs the core Ubuntu developers, seeks to test the limits of the open model of development, combining it with traditional closed development model concepts such as feature specifications and set release cycles in an attempt to bring about change in the way the software industry is run, while hoping to become profitable by selling support service contracts on the Ubuntu system – though this has yet to happen. As one Canonical employee says with a smile, referring to the processes adopted by both Canonical and the Ubuntu community, underlining how intertwined the two are: "I don't know if there is a 'usually' in anything we do yet."

The developers employed by Canonical work full-time from their homes, in a way symbolising how their hobby has become their job. They only meet for a few week-long Ubuntu Developer Summits and development sprints sponsored by Canonical which take place at hotels alternately in Europe and North America with the odd meeting in Australia. On-line, the Ubuntu community has copied most of the Debian modular infrastructure, including

[13] Similarly, Michael Banck, a Debian Developer, calls Ubuntu's development process and community structure "the first evolutionary challenge to Debian" (Banck 2005)

the use of mailing lists, wikis and IRC channels as their main means of communication, where Canonical employees and community volunteers work together, since the textual media on-line do not indicate who is paid to work on Ubuntu and who is not (and the core Ubuntu developers have decided not to publish such lists anywhere, fearing that it would swamp them with user requests, undermining the open development model with constant begging for new features).

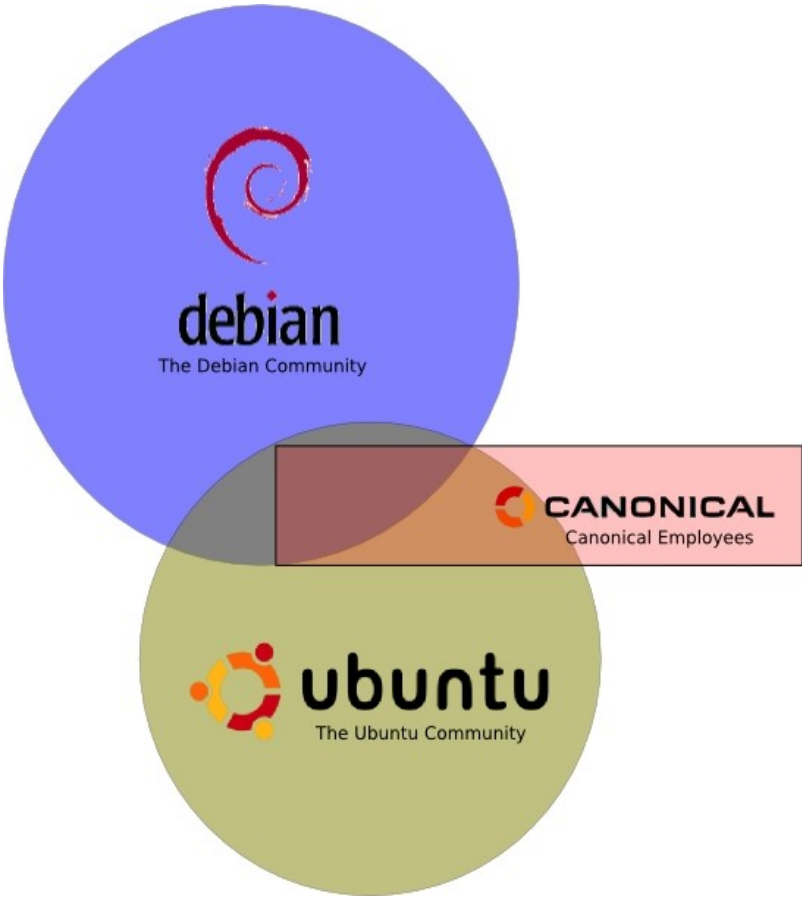

Illustration 5: A simplified diagram of the relationship between the Debian and Ubuntu communities in relation to Canonical. Mark Shuttleworth and the first core Ubuntu developers are located where all three fields overlap. The Canonical developers not involved in either community are working on Launchpad and other closed development projects.

But Shuttleworth is also seeking to extend that infrastructure with the construction of a new web-based collaboration platform for free software projects called *Launchpad* which the Ubuntu community now uses. More than half of Canonical's technical employees work on Launchpad (cf. Illustration 5) which includes a bug tracker, a web-based translation system, and a

revision control system.[14] Through this infrastructure, the core Ubuntu developers work side by side with hobbyist contributors all drawn by their shared interest in building, maintaining and improving a free software operating system for all (especially themselves) to use.

Conclusions

As Ellen Ullman remarked in the quote that opened this chapter, computer systems are built like cities, mixing designs, without an overall plan, on top of ruins, always learning from and reimplementing old mistakes (Ullman 1995). And the Ubuntu system is no exception, being built upon a long, intricate "oral" academic hacker tradition derived from Unix through the GNU project to Linux and Debian, with hackers continually jousting to improve, shape, and keep free their access to the computer both by legal and technical means in an interdependent "eco-system" of shared values and code. Since the inception of Unix, this tradition has refined and encouraged trust in the user's abilities through continuous reinvention, reimplementation and reaffirmation of the shared and open source code.

In opposition to this is the closed model of development – the accepted industry fashion in which to effectively and profitably produce software

[14] A revision control system allows for the management of multiple revisions of the same software so that several people can collaborate and coordinate changes by incrementing each revision with a revision number associated with the developer who made the change. Shuttleworth's overall goal with Launchpad is to create a technical infrastructure for a seamless and automated interaction between upstreams and downstreams, allowing downstreams to notify upstreams of bugs and upstreams to respond and fix these, easing the exchange of patches in both directions, using revision control to allow for easy merging of patches. Such an infrastructure is designed not only to make it very easy to make derivative versions of Ubuntu to be customized as they see fit, but also to help hackers to avoid fixing the same problems twice by ensuring an automated exchange of source code (Hill 2006: 26-27,351-357). As one Ubuntu hacker puts it, "with Launchpad, Ubuntu will be like Debian in revision control."

which companies like Microsoft have used to create and win dominance in a market based on cheap commodity hardware.
It was only with the rise of the Internet and the success of Linux' open development model orchestrated through open on-line collaboration by volunteers working from home, building on the value of instantly shared, freely modifiable, cumulative improvements to the software, that the old "oral tradition" became a viable industry alternative.
Born from the money and ambition of a dot-com IT billionaire and the talent and energy of developers and contributors brought together from Debian and other free software projects, the Ubuntu system is not only the latest instance of a 40-year-old development tradition, it is also the latest convergence of those two opposed trends of software development which have been at odds since the first personal computers were produced.
By hiring and defining the tasks of the core Ubuntu developers working alongside the community contributors in an open model of development, which allows all interested parties to contribute and collaborate, Mark Shuttleworth has managed to create a hybrid model of development in which all development work is openly and freely available, yet the core of the Ubuntu system is in practice developed according to the commercial closed development model, architected by Mark Shuttleworth and Canonical, a company with clear profit motives in selling support for the Ubuntu system. This leaves Mark Shuttleworth balancing between a role as a commercial industry leader similar to Microsoft's Bill Gates, and a role as a community leader such as Richard Stallman. Between the free software dogmatism that uncompromisingly seeks freely accessible knowledge for all and the pragmatic materialism that seeks profits and the vendor support to make Ubuntu and free software sustainable in the commercial market in the long run. But through their dedication and enthusiasm, Shuttleworth and the core

Ubuntu developers have matched the individual hopes and
users and hobbyist developers with their grand vision: Bringing forth free software as an alternative to the dominant proprietary, closed-source, closed-development paradigm of Microsoft. Or, as the free software hackers humorously call their end goal for their joint enterprise: World Domination.

Chapter 3

Hacking Synaptic

– a case study of the Ubuntu hackers' development work

The Ubuntu developers work together on-line, exchanging files in order to build and maintain the complexity of the Ubuntu system. But though they work by themselves, their work is not the clinical work of a lone genius, as is often associated with programming, but rather a messy, social practice based on the uneven collaboration and sharing of knowledge through digital means. In this chapter, I will show how two Ubuntu hackers – one employed by Canonical to work on Ubuntu, and one a hobbyist contributor – have come to participate in the Ubuntu community, and how they collaborate in the open development model of the Ubuntu community on *Synaptic*, just one out of the more than 19.000 software packages within the Ubuntu system.[15] Though I cannot generalize on the basis of a sociological description of just two of the Ubuntu hackers, I seek to use it to offer insight on working practices and stories of finding and using Linux which I have found to be quite common among the Ubuntu hackers, which in turn can help give a nuanced impression of who the Ubuntu developers are. I will draw upon Claude Lévi-Strauss and Clark & Chalmers to argue that their shared day-to-day social and technical practices are complementary, depending on shared interaction and knowledge, as well as the flow afforded by the intimate work with the computer. And that this interplay of diverse, yet complementary competences is what binds the members of the Ubuntu community of practice together into a social entity through what Etienne Wenger calls the *mutual engagement* (Wenger 1998:73-77). Finally, I will explore how the Ubuntu hackers socialise and collaborate

[15] Of the 17 hackers I visited and interviewed in detail during my fieldwork, Michael and Sebastian were the only developers who were working closely together on the same software package, as well as representing different degrees of involvement in the community. The fact that they are both German is not directly relevant in this case, perhaps apart from noting that close collaboration often occurs among developers who share the same native language.

both on-line and in-person, expressing this mutual engagement in chat channels, on mailing lists and at free software conferences.

The Synaptic developers 1: Michael

Michael is one of the core developers of Ubuntu. He is 29 years old and lives in a small village in Western Germany. He grew up on the nearby family vineyard, and he returned to this area with his wife – his high school sweetheart – so that she could work as a physiotherapist in one of the nearby villages after Michael finished his MA in Computer Science and began working fulltime for Canonical in November 2004– his first fulltime job. Michael has been fascinated by computers, experimenting with programming and understanding how the hardware and software interacted to produce results on the screen since the early 1990s:

> I kind of played with it and found it totally interesting to be in complete control of this whole computer thing. To understand how it works and exactly what it's doing .. I mean exactly what it's doing at the software level. [...] I was just eager to try things out and learn. Not afraid of spending complete nights on stuff.

As he became more proficient with the computer, he got an Internet connection which enabled him to participate in worldwide discussion groups, opening up a much wider pool of knowledge on computers and exposing him to Linux:

> What I mostly liked about Linux, again, was that you are in control. That you are able to understand each aspect of the system. There are no secrets. In Windows, it's full of secrets. Even if you got really good documentation and stuff, at least in my view, you don't have a real chance to understand the system. And even if you did, there would be nothing you could do with this knowledge. It's just ... 'Okay, now I know how it works, but there's nothing, you know, to change, to do... to do with it. So what I really, really found interesting with Linux was that it was open and that you had the chance to understand it.

He fswitched to Linux in 1998 when he began university, and there he met other Linux hackers, and he began going to Linux conferences around Germany, becoming immersed in the free software communities which led to his involvement with Debian:

> One of the nice things about Debian was that I was able to contribute back to it in a way that was not too hard to do. I mean for Debian, all you had to do – “all you had to do” in quotes – was to become a package maintainer which just meant downloading new software, which I enjoyed anyway, compiling, testing it and uploading it. [...] It was not very hard. I just decided that I wanted to contribute some applications that I used anyway, so I had to package them anyway...

By 2000, Michael had become a Debian Developer, maintaining several Debian packages, including Synaptic, a package management application which had been “orphaned” when the upstream developer abandoned it in 2002:

> I maintained "Synaptic" as a package at this time and when I noticed that the upstream guy went away [...] I decided that we need such an application to make Debian easier to use for people and that I should maintain it myself, that I should keep working on it. [...] I just decided to start hacking on this thing. And... then I got a working version after a couple of days that was better than the last one and looked good enough for me so I decided to write this mail to debian-devel [the main Debian mailing list] to say 'okay, here's this thing. Please test it if you're interested' and, to be honest, there wasn't a lot of response. People were just like 'Yeah, heh. fine. Do it, go for it. Have fun.'

He initiated a major redesign of original graphical user interface, doing most of the work on his own, but constantly asking and receiving feedback and testing from other developers. It was this work along with a recommendation from then-Debian Project Leader Martin Michlmayr that resulted in Mark

Shuttleworth inviting Michael to London for the first informal Ubuntu meeting for a weekend in early 2004. These 12 hand-picked developers, most of them involved with Debian, came from all around the world and for many of them, this was the first time that they had the opportunity to meet each other in person, at least, under more quiet and relaxed circumstances than among the hundreds of people that attended the annual Debian conferences. During the meeting Shuttleworth outlined his vision for the as yet unnamed project, and offered them all jobs on the project which all except one accepted.

Back home, the developers began working together on-line, and though Michael spent most of his time writing his thesis, he also spent a lot of his spare time working on Synaptic for the first Ubuntu release in October 2004. He graduated shortly after the release, and soon afterwards he signed his first one-year contract with Canonical to work on package management in particular. To Michael, the job is “pretty much what I wanted [...] it's the best of all possible worlds” – it allows him to work on free software fulltime, having fun with the technical challenges while sharing his results and knowledge with his friends and work-mates on-line.

The Synaptic developers 2: Sebastian

Sebastian contributes regularly to Ubuntu. He is 28 years old, and lives in his native Southern Germany where he is now in the final year of school to become a nurse. He already works a lot of shifts as a geriatric nurse when he is not doing his courses, and he enjoys his work greatly.
Sebastian has had access to computers pretty much all his life due to his father's job as a software engineer at a big engineering firm. But though he soon grasped the principles of programming and became proficient with the computer, his interest did not move towards the deeper, technical aspects of the computer, but rather towards the excitement at the possibilities of new technology, and the challenge of learning to use a new system. "It's basically about hearing about new stuff and trying it out" he says.
After experimenting with different versions of Microsoft Windows and various other operating systems – both free and proprietary – he happened upon Debian in 2002 which he soon became quite proficient with, learning by exploring and fixing things on his own system, and helping others with tricky and interesting issues on the German Debian users' web forums. Based on these experiences, he wrote a fair bit of translation and documentation to make Debian more accessible to German users. One of the packages he worked on was Synaptic where he collaborated a bit with Michael. Though he spent much time contributing to the project on-line, Sebastian never met with

anyone in the Debian community. His impression was that it was “a quite closed community – most often, Debian developers only will work with other Debian Developers,” and his contributions were mostly individual efforts with little direct collaboration.

So in May 2005, Sebastian decided to sever his links to the on-line communities he had joined. He closed his web forum accounts and unsubscribed from the mailing-lists that he had joined. He filled the gap with his new job as a nurse, and with lots of sports – running, swimming and bicycling – his favourite hobby, as he humourously notes: “I have more bicycles than I have computers.”

But when he tried out Ubuntu in January 2006, he soon filed a bug regarding some usability issues in another of Michael's package management projects, and the two soon began working together on Synaptic as well. His main reason for contributing to Ubuntu is simple: “It was winter, so I couldn't do sports.”

Sebastian says that contributing to Ubuntu is a good way to balance the different sides of his personality: The empathic and social aspects of working as a nurse on one side, and the cognitive, learning experiences with the computer which lets him “find relaxation through coding,” programming and writing patches to Michael's projects, scratching the itches and annoyances he

finds in the interfaces, seeking to resolve the kind of issues he knows that many users have had in Debian. He considers himself lucky that Michael is so receptive to his ideas since if he wasn’t, there really wasn't much Sebastian could have done since he doesn't have the technical privileges to upload his changes himself (I will discuss these technical privileges further in chapter 6).

Though both Michael and Sebastian are involved in Ubuntu, their level of commitment, programming skills, working conditions, prior experiences with the free software communities and incentives for contributing specifically to Ubuntu differ enormously. Michael is a long-time developer who has become employed to work on what was his hobby, while Sebastian is a long-time user who has found a new, challenging aspect of his interest in computers by learning to program and contribute to his operating system of choice. As I remarked in chapter 2, it is through these individual histories of discovering and learning about Linux that the Ubuntu hackers have come into their shared community of practice. And it is through the continued learning and experience in their shared practices that they maintain their membership. But in order to gain a better understanding of the shared practices of Michael and Sebastian, we need to know a little bit more about what Synaptic is and how it

has been developed.

Collaborating on-line on Synaptic

Synaptic is a graphical *package management tool* used for the upgrading, installation and removal of software packages on Ubuntu and other Debian-based systems. Since all of the free software available in Debian-based systems are packaged in modular Debian software packages, package management tools such as Synaptic are essential to maintain, update and

expand such a system as they provide the user interface to the database of all available software in the on-line archives from where the latest versions can be downloaded for smooth installation or upgradal without the user having to download and compile the source code themselves.
Synaptic was originally developed by the Brazilian developer Alfredo Fujima for the Debian-based Linux distribution Conectiva, based on the *Advanced Packaging Tool* (or APT for short) which is used from the *command line*.
The command line is a legacy from the days when computers didn't write their responses on screens but on *teletypes* – a kind of typewriter unto which you could type commands, and when you hit "enter" the computer would react and the teletype would print the computer's responses on the paper.[16] The command line is a digital representation of the same command-response interface used by the teletype, and until graphical user interfaces were popularized by the Macintosh in 1984, all interaction with computer happened through the command line. Even though graphical user interfaces now are commonplace, many hackers still prefer to use the command line as it offers a more direct and technically transparent interaction with the computer since it uses the same kind of textual input and output as they use to program, enabling them to see the exact messages offered by the object code, whereas the graphical user interfaces require another level of graphical abstraction to be programmed on top of the textual input.
Thus APT and Synaptic offers two completely different modes of interaction to make the computer perform the same actions. Where installing the software package "foo" through APT would require typing the command "apt-get install foo", Synaptic is built on top of this concept to allow installation by pointing

[16] Unix was originally designed for use with teletypes, and all of the Unix tools were commands to be entered on the command line. These commands were necessarily short with two-, three- or four-letter names such as **cp**, **cat** and **grep** as typing them took time on the slow teletypes.

and clicking on the package “foo” with the mouse, making package installation much easier for less experienced computer users (cf. Illustration 6).
When Alfredo Fujima left Conectiva in 2002, he abandoned Synaptic in the process. Soon afterwards, Michael took over the project in a fashion typical of free software projects which Scott, another core Ubuntu developer, describes quite precisely:

> You've picked a package that is, for all intents and purposes, abandoned by anyone who previously cared about it. This happens from time to time, and is a side-effect of the mostly volunteer nature of Open Source development, people move on. However this is also one of its strengths ... the package may not be being maintained by the previous owner, but now the time is ripe for somebody else to come along, pick it up, and take it over themselves! [...] What kind of person would do this? Well, the kind of person who clearly uses it sufficiently to be finding bugs; and with the developer skills to be able to fix both bugs and write patches for them.

Since the source code of Synaptic is licensed under the GPL, Michael was legally allowed to modify and redistribute it as he pleased, and since there was nobody else to collaborate with, Michael found himself in the position Scott describes, having the skills and the motivation to work on it. He took on the double role of being the developer upstream and the packager downstream in Debian, hacking on it in his spare time while at university. But as he improved it and uploaded new versions of it, he also inherited the hackers who used Synaptic in Debian, and who were directly affected by his work when they installed the new version of the package, and who tested it and reported bugs, who complained when his hacking broke old features and who sent in patches, translations and suggestions with improvements. One of these hackers was Sebastian who eventually came to continue their collaboration on the Ubuntu system.

The complementary elements of hacking

Working on Ubuntu, Michael generally spends most of his time working on package management applications such as APT and Synaptic. Most of his work is shaped either by *features* or by *bugs*.
Features are new capabilities to be built into the Ubuntu system and the

applications that are shipped with it. These features are typically discussed and drafted at the biannual Ubuntu Developer Summits before being approved by the Ubuntu Technical Board – the governance entity with the overall responsibility for defining the technical direction of the Ubuntu system, led by Mark Shuttleworth and Canonical CTO Matt Zimmerman, formally the chief engineer of Ubuntu. When the first version of Ubuntu was released, Synaptic was chosen as the easiest interface for new Linux desktop users to get to grips with the Debian package infrastructure, since it allowed them to install additional programs without using the command line, and at Shuttleworth's request, Michael has spent a lot of time developing Synaptic and other graphical package elements to make that infrastructure even easier to grasp.

When developing, Michael discusses each feature as a problem to be solved with the other developers on IRC, getting ideas and hearing opinions, drafting solutions and finding out any structural prerequisites, and coordinating efforts before beginning the actual implementation. Since Sebastian is one of the few others with an intimate knowledge of the Synaptic source code, it is often with him that Michael discusses these matters, and depending on Sebastian's own interests, he may help develop the code, using the revision control system to suggest alternative paths and ideas which Michael can merge with his own code.

Bugs are issues or design errors in the existing code which may produce hiccups or crashes when tested or used. You can only test whether a program works by compiling it, thereby turning it into an opaque blob of binary code, and running it. Thus, a central part of programming is testing the compiled program and noting down any discrepancies between how it is specified to work, and how it actually works. Such discrepancies are usually bugs to be fixed in the source code – which in turn must be compiled again in order to be tested. Therefore, every time they release a new version of Synaptic, usually including a new feature which in turn may introduce new bugs, Michael and Sebastian call for users to test the new version and report any bugs they find in the Ubuntu bug tracker. Reporting a bug involves writing a description of how to reproduce the bug so that Michael and Sebastian can test the bug on their own systems as they work to narrow down which part of the source code might be at fault in order to fix the bug.

Sebastian does a fair bit of *bug triage* on Ubuntu's package management tools: reading and replying to reports, marking bugs describing the same issue as duplicates, reassigning bugs that are related to other software packages, exchanging opinions and solutions based on their knowledge of the structure and style of the source code, and querying the bug reporter for more information on how to reproduce the bug or under which circumstances the bug occurred which can be gathered by looking at the cryptic spatterings of text, called *logs*, *traces* or *dumps*, which the program leaves behind when it dies. Sometimes these queries result in continuing conversations with the bug reporters as he and Michael try out various solutions in their attempt to locate and fix the bug.

This duality within the practice of hacking between building new and maintaining the old has striking similarities to Claude Lévi-Strauss' famous dichotomy of the engineer and the bricoleur (Lévi-Strauss 1994). Where the engineer solves specific problems with specific tools designed for that purpose to build and extend new features, the bricoleur is a tinkerer, using all means at hand to find a solution, but only to maintain and stabilize the inherently unstable collected remnants of already existing human work (which Lévi-Strauss calls “idea treasure”, borrowing a term from Mauss and Hubert) (Ibid. 27-30).
For hackers, the feature-based collaboration is the work of engineers, following plans to implement clear solutions, while the bug-based collaboration is the work of bricoleurs, constantly fussing and fretting, searching out answers based on circumstantial evidence like a hunter tracking an animal.[17]
But these two aspects of hacking transcends Lévi-Strauss' dichotomous definition of the engineer as they are not so much sharply opposed distinctions as they are complementary parts of the same practice – much like how a doctor both manages the idealized healthy human body according to a deductive anatomical model and medicinal theory, and bases his diagnosis on recognizable symptoms and circumstantial evidence gathered according to an

[17] Indeed, computer folklore has it that the very first bug to be found was in fact an actual insect which had lodged itself within the machinery of one of the earliest digital computers.

inductive, semiotic model supported by his personal, empathic experience; treating the patient according to both, whilst also consulting other doctors or experts, only later to find out whether his diagnosis and treatment was correct and not the cause of further malady.[18]

Thus, the daily hacking practices of Ubuntu hackers such as Michael and Sebastian is not the work of lone engineers, but rather a complex interplay between two complementary elements of hacking which are maintained through social interaction both on-line on mailing lists and IRC and in-person at conferences and summits. These complementary practices are what Etienne Wenger's calls the mutual engagement that defines a community of practice (Wenger 1998:73). As this mutual engagement depends on the members' diverse engagements and overlapping competences in a shared practice which I contend is at play here.
In the last part of this chapter, I will show how the Ubuntu hackers' mutual engagement is similarly clearly reflected in the way they use the Ubuntu system to arrange their work environment at home and at conferences and the way they interact with one another both on-line and in-person.

Deep hack mode and asynchronous sociality

As described above, the Ubuntu hackers work at home, and accordingly, they prioritize having a separate work room in order to be able to shut the door to the home around them and focus on their work. As when Ubuntu hacker Colin went house-hunting with his wife and son, having his own study was "non-negotiable." Hackers tend to demand calm around themselves, arguing that their work, especially writing entirely new pieces of code, requires intense

[18] The Italian micro-historian Carlo Ginzburg argues that the deductive and inductive practices of modern science are based in two fundamental semiotic paradigms: The stringent Galilean paradigm focused on universal reproducibility and abstract deduction, and the elastic Evidence paradigm ("paradigmo indizario") focused on the interpretative and inductive, personal gathering of signs (Ginzburg 1986). He argues that the Evidence paradigm, dating back to the practices of hunter gatherers, hasn't been recognized for its role in modern science which is generally attributed to the dominant Galilean paradigm, much like a mule only recognizes the horse as its parent. Similarly, computer programming is often considered to be a completely deductive practice of clean mathematical abstractions, though it is equally a feverish hunt for the remaining bugs, inductively tracking them down and removing them.

concentration made impossible by external disturbance.
Ubuntu hacker Paul describes how under the right circumstances, hacking can put him “in the zone” – a state of uninterrupted concentration where he loses himself from the outside world, focusing solely on the computer to the point where the interaction with the computer happens effortlessly, allowing him to immerse himself in the code to the point where "You don't need to think about the variable names[19], they will just appear in your mind when you need them." Many hackers refer to this state of deep concentration as “Deep Hack Mode” (cf. Coleman 2005:231) – a term defined in the on-line compendium of hacker slang known as the Jargon File as “a Zen-like state of total focus on The Problem that may be achieved when one is hacking,” and it is very similar to what the American psychologist Mihali Csikszentmihalyi calls flow: “the state in which people are so involved in an activity that nothing else seems to matter” (Csikszentmihalyi 2002:4). It is worth noting how such diverse activities as programming, playing computer games or writing essays all depend on the computer's continuous feedback and need for interaction, its minimal requirement of physical exertion, and its huge mindful potential to induce and maintain the flow state for hours on end. The flow state requires complete concentration and control, pooling all your attention in a tunnel-like vision focused on the screen, making you willfully unaware of your surroundings, including the physical actions you perform to interact with the computer, empowering and guiding you only towards the work on the screen. This interaction with the computer is similar to the American philosophers Clark & Chalmers' notion of “active externalism” through which they argue that certain mental tasks depend on physical objects to be solved efficiently. They give the example of the letter tiles in a Scrabble game where we depend on physically moving the tiles in order to win new perspectives on the possible words to be spelled rather than on some internal process on our own. Thus, “the re-arrangement of tiles on the tray is not part of action; it is part of *thought*.” (Clark & Chalmers 1998:7; emphasis original). In this way, the computer and the system through which it works becomes an extension of the hacker's mind, it is only through constant, uninterrupted interaction and focus on the computer that he can solve the task at hand. Such technical intimacy usually affords only one user, and as Paul explains, any outside interruptions

[19] Variables, in software code, are containers, used to temporarily hold some type of data.

will give him a jolt, breaking all of the mental connections and surplus afforded by the flow state. The Jargon File describes how the importance of not being interrupted is deeply engrained in hacker etiquette:

> ... if someone appears at your door, it is perfectly okay to hold up a hand (without turning one's eyes away from the screen) to avoid being interrupted. One may read, type, and interact with the computer for quite some time before further acknowledging the other's presence (of course, he or she is reciprocally free to leave without a word). The understanding is that you might be in *hack mode* with a lot of delicate *state* in your head, and you dare not swap that context out until you have reached a good point to pause.

I experienced this several times in my visits and interviews with Ubuntu hackers, whose partners and friends over time have come to allow room for and be forgiving of these programming-related eccentricities. In turn, the hackers themselves often take care to balance their time on the computer in relation to their family's needs, when they reach one of those “good points to pause.”

I have found that this duality is also very characteristic for the way that hackers interact with each other on-line where you can't assume that people are communicative at any given moment. As Ellen Ullman points out, this inability to be interrupted makes hackers somewhat asynchronous to one another – at least in the short term (Ullman 1995:132). This is reflected clearly by the fact that all of the Ubuntu hackers' preferred on-line communicative means are textual and thus – at least to some extent – asynchronous. Email, newsgroups and web forums postings and bug tracker comments are all based on users reading and replying asynchronously. Even real time communications such as IRC chat channels and Instant Messaging bend to this rule as developers “ping” each other, and if there's no immediate response, they can ask their question and let the other answer when he has time or attention to spare:

...		
09:00	**carlos**	pitti: ping
[...]		
09:07	**pitti**	carlos: pong
09:08	**carlos**	pitti: I did a mistake yesterday night and latest Edgy export has the plural form bug (bug

		#2322)
09:08	**Ubugtu**	Malone bug 2322 in rosetta "Truncated plural forms" [Critical,In progress] http://launchpad.net/bugs/2322
09:09	**carlos**	pitti: I'm exporting a new version with that
fixed,		but it would take around 2-3 hours
09:09	**carlos**	am I late to have it in the prerelease version?
09:09	**pitti**	carlos: ah, then I'll rebuild the edgy packs this afternoon
09:09	**pitti**	carlos: it won't go into RC anyway
09:09	**carlos**	ok
09:09	**pitti**	carlos: the plan is to upload the final packs tomorrow
09:10	**pitti**	carlos: thus I'd like to have today's in perfect shape
09:10	**carlos**	I see
09:10	**carlos**	ok
09:10	**carlos**	pitti: I will ping you when the new version is available

Here, Carlos needs to notify Pitti of a new bug which he needs to take into consideration when building a group of packages for upload. Since Pitti is busy, the conversation doesn't continue until Pitti is able to respond and they can coordinate their work. Most of the Ubuntu hackers' day-to-day interaction takes place on IRC where they can pick up on interesting discussions and be available if someone needs to ask a question. The hackers deftly navigate back and forth between conversations, fluidly participating as the IRC client automatically notifies them when someone “pings” them or even just mentions their on-line moniker. And even if they miss something, they can always go back to check the chat logs or mail archives as all interactions within the community are recorded and publically archived on-line. At times, IRC is such an easy and non-intrusive way of quick communication that it supersedes conversation even when developers are in the same room. Mark Shuttleworth enjoys relating the story of how he went out to buy beer during one of the first gatherings of the core Ubuntu developers at his London flat. When he came back he found 18 hackers sitting in his living room, working in silence, exchanging textual information on IRC.[20] This anecdote illustrates

[20] I saw the same trend again and again at conferences, one time even witnessing two

how the work environment provided by the system takes precedence over face-to-face discussions simply to avoid breaking the flow state afforded by the computer.[21]

Socialising on-line

Although this mode of working seems highly individual, there is a constant flurry of interaction taking place through the computer. Each developer attends IRC channels and subscribes to mailing lists reflecting their different interests around the community. Most of these discussions are essentially long question and answer sessions, as hackers turn to fellow hackers as well as manuals, web searches and documentation to find the answers necessary to retain their mastery of the computer. Each upstream project has its own mailing lists and IRC channels, as well as the Ubuntu community itself which is sub-divided into smaller groups based on interest, centered around a shared IRC channel. Typically, an Ubuntu hacker is subscribed to between 10 and 30 mailing lists, logged on to up to 15 different IRC channels, and also receive mail notifications about the latest changes to the system and new bugs and comments on bugs from Launchpad to the software projects that they're working on, and read blog updates on the Planet Ubuntu webpage.
With all these emails, chat conversations and bug reports constantly requiring

---

hackers quietly sitting in opposite ends of a conference room, having a furious argument on IRC about who should be responsible for fixing a troublesome piece of software, and it didn't end until one of them looked up and saw the other hacker sitting at the far end of room and contentiously shouted: “Stop being such an arsehole!”

[21] This asynchronous sociality is not only a norm well suited to hackers' mode of collaboration, but it is at times also a necessity as the Ubuntu hackers are spread across the multiple time zones, mostly in North America, Europe and Australia, making exchanges such as this common:

15:56	**mdz**	good evening
15:57	**zul**	afternoon
15:58	**ajmitch**	morning
15:58	**mdz**	good UTEvening

It is easy to forget that the Ubuntu community spans the entire globe, since it mostly becomes an issue when it comes to finding IRC meeting hours that fit all members of the community, and meetings typically rotate between being early morning, late afternoon, or late evening to accommodate as many time zones as possible.

attention, the computer easily becomes a source of constant distraction. Digital theorist Linda Stone has uses the term “Continuous Partial Attention” to describe the stance of keeping one item in focus while constantly scanning the periphery for something more interesting or relevant, being ready to synch with that item instead (Stone 2006). This continuous scanning state of constant readiness for interruption appears to be directly opposed to the uninterruptible flow state, yet the Ubuntu hackers are extremely apt at filtering, handling, and prioritising the large amounts of computer-mediated information that pass before them every day.

As media theorists Geert Lovink and Ned Rossiter have noted, on-line life is defined by passive consumption of information, as it would be impossible to relate and reply to all the information that passes by (Lovink & Rossiter 2005). This can literally be seen in the mailing lists and the IRC channels which the individual Ubuntu hackers subscribe to. Most of these they are interested in, but not so deeply so as to participate actively and regularly – unless a topic comes up which they care deeply about. Instead, they spend most of their time and energy on relatively few mailing lists and IRC channels, where they participate much more actively as they are well acquainted with the people there, and where they contribute with most of the discussion, joking and socialising to which others can relate passively. As Chris Kelty remarks, “people are not always in the public, but they are always ready to be” (Kelty 2005a:200).

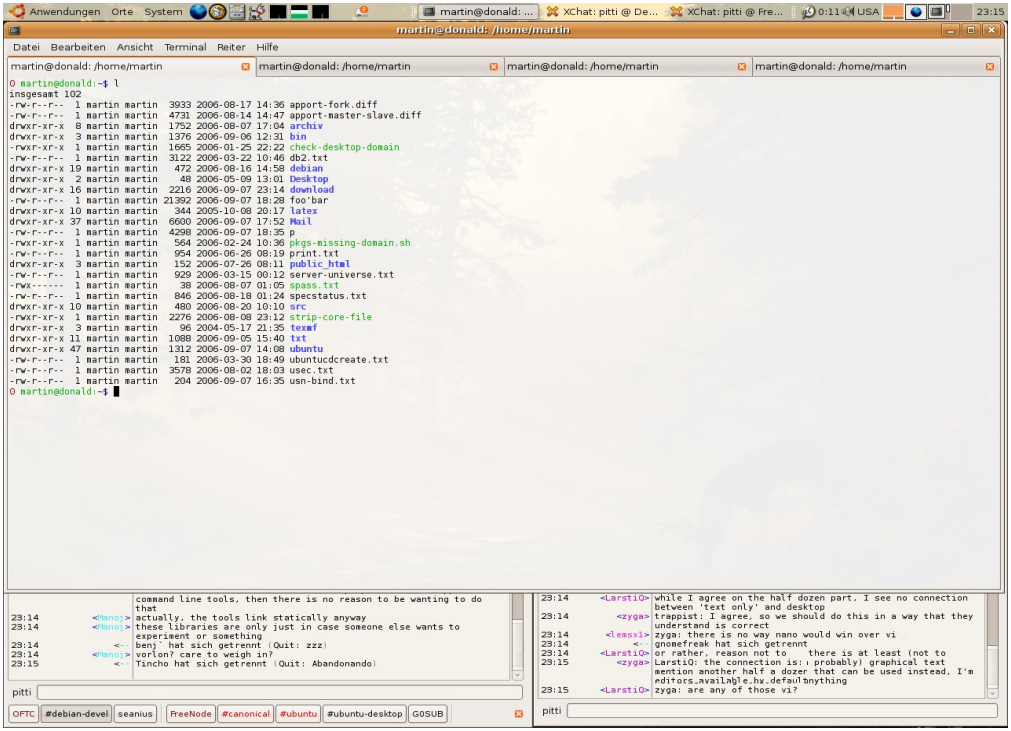

Illustration 7: A typical Ubuntu hacker's main work space. With the command line dominant over the constant textual chatter of the community.

Hackers' on-line sociality is defined by how they use the computer to "effectively filter out the noise from the signal" as a recurring hacker metaphor goes. This duality between focusing intensely on your work and constantly scanning the periphery for new information to relate to, is summed up quite nicely by the way Ubuntu hacker Martin has organized his workspace (cf. illustration 7): He has set up two IRC clients monitoring the channels he is most interested in, and on top of them, he has placed his main tool, the command line terminal (which will be explored further in chapter 5), in such a way that the last 10 lines of conversation are displayed from the IRC clients, allowing him effortless participation in the constant flow of

community chatter as part of his work on the Ubuntu system. In this way, the hacker's continuous flow of interaction with the computer comes to the point where the system becomes an extension of his mind. It is through this "active externalism," as Clark & Chalmers have defined it, that the individual hacker finds the flow afforded by the computer allowing him to incorporate the interaction with the system he continues to build with the effortless on-line sociality with his peers. It is in this complementary flow that the Ubuntu hacker enters the mutual engagement of the shared community of practice.

Free software conferences: Meeting in-person

But even though, or perhaps exactly because, the Ubuntu hackers spend so much time communicating through their computers, they relish the opportunity to meet in-person. The biannual Ubuntu Developer Summits are where the Ubuntu hackers convene to plan out the coming development cycle of Ubuntu, and socialise with the physical rapport not afforded by the on-line textual means. I find, as the American anthropologist Annette Markham has concluded on her experiences on IRC, that on-line and in-person are two perspectives of the same reality that needs to be accepted on equal terms, even if they aren't experienced as such (Markham 1998:211-212). The two perspectives differ solely in the means and tools through which they socialize. Just as an in-person meeting has different strengths and weaknesses in the audial and visual clues which can both guide and distract, on-line communications have similar strengths and weaknesses to which the Ubuntu hackers strain their attention in an attempt to adapt.[22]

The first day of the summit is characterized by general feeling of reunion, not only meeting old friends, but also meeting people that you have only corresponded with on-line, but wouldn't recognize in person and the people who were here for the same reasons as you, but whom you didn't know yet. During the conference the hackers wear themselves down with sleep

[22] Though the hackers thoroughly enjoy the intense, social interaction of the conferences, not all hackers are equally well accustomed to dealing with in-person situations compared to their usual on-line interaction. This is demonstrated quite clearly by Ubuntu hacker Tollef's observation on his personal weblog following the Ubuntu Summit in Mountain View: "Showering is *not* optional when you interact with other people. Seriously, some people didn't just smell a bit, they stunk. Eww." It is worth noting how Tollef himself reserved this observation to be expressed on-line, in a way gently perpetuating this divide.

deprivation and cumulative hangover in order to fit as much socialising and shared hacking as possible into a single week. Much of this takes the form of *Birds of a Feather* discussions where hackers discuss new features upon which the Technical Board plan out the coming releases and feature goals (as discussed above, also cf. illustration 8). All of this intense social interaction reaffirms and re-energizes the hacker's shared interest in the project. For Sebastian, for instance, attending the Ubuntu Summit in Paris not only gave him an opportunity to meet Michael in-person after three years of on-line collaboration, which gave their friendship a new dimension of shared in-person experiences, but it also offered him the opportunity to meet other Ubuntu contributors whom he hadn't interacted with on-line, and with whom he began collaborating and discussing his work following the conference. These in-person meetings gave Sebastian a much wider active and personal exposure to the Ubuntu community than he could have had through the passive consumption on-line. He wasn't just ready to be in the public of the Ubuntu community – he was there, in-person, for the whole week. Thus the in-person meetings helped define new social relations for Sebastian to augment through the daily interaction and collaboration on-line.

Illustration 8: A typical Birds of a Feather session – or BOF for short. A BOF is a staple unit of discussion at hacker conferences, based on the idea that "birds of a feather flock together", these are informal discussions where hackers with shared technical interests get together to share ideas on a certain topic.

In this way, hacker conferences are, as the American anthropologist Gabriella Coleman puts it, "the quintessential hacker vacation" (Coleman 2005:312) – as it combines the shared fun of hacking with the social interaction (enhanced by the social lubricants such as alcohol and shared accomodations) with others who share their interest and passion for free software, thereby offering the hackers new ideas and inspiration and motivation to contribute to the

project.
Mark Shuttleworth and Canonical are very aware of the conferences' stabilising function on the continued work of the Ubuntu community, and they use the biannual conferences as occasions to sponsor both hobbyist contributors (such as Sebastian) and developers from various upstream projects to participate in the conferences so that these developers can offer their technical expertise and build social in-person relationships with the Ubuntu developers in order to help future collaboration.[23]
Thus, on-line and in-person social interaction complement each other much like the Ubuntu hackers' bug fixing and feature development work does. Rather than being completely distinct spheres of reality, the asynchronous on-line interactions which affords the intense flow central to working with computer and the direct, synchronous, in-person interactions at conferences which affords deeper discussion and physical rapport are complementary parts to the mutual engagement of the Ubuntu hackers.

Conclusions

This chapter has asked, ‘who are the Ubuntu hackers and how do they collaborate?’
Through a sociological case study of two Ubuntu hackers, Michael and Sebastian, I have tried to show how differently the Ubuntu hackers live and work away from the computer, and how this affects their presence on-line. One is a full-time developer fully integrated in the free software communities on-line, and the other finds contributing to Ubuntu a relaxing hobby that has given him a group of new friends both on-line and in-person. The case of Synaptic shows how the hacking practices within the open development model of the Ubuntu community are fundamentally social, as the Ubuntu hackers take over and collaborate through a complementary practice between what Lévi-Strauss calls engineer work and bricolage: Both visualising and designing new features as well as redesigning and reimplementing odd bits of left-behind work to fit their needs through a haphazard extending, fixing,

[23] Even so, it is only a fairly small number of Ubuntu hackers who gather at any given Ubuntu Developer Summit. Apart from the around 25 hackers paid by Canonical to work full-time on Ubuntu, Canonical sponsors around 40 community contributors and upstream developers to participate at any given summit, and though they encourage other interested developers to join in, few can find the time and money to do so on their own.

fussing and testing. All of this combined in an openly social practice where they constantly rely on the knowledge of other hackers. Thus, much like the American computer scientist Gerald Weinberg who argued, more than 35 years ago, that most programmers spend more than two thirds of their time working with other people rather than working alone (Weinberg 1971: 35), I claim that the practices of the Ubuntu hackers fit with the Etienne Wengers' notion of mutual engagement in a shared community of practice. Where diverse competences produced through shared histories of learning complement and overlap both through on-line discussions on IRC and mailing lists characterized by a constant asking and answering of technical questions, and passing assignments and suggestions from one to another on how to best negotiate features and bugs in the software incorporated effortlessly into the work with the computer as an extension of the hacker's mind, as suggested by Clark & Chalmers; and through in-person discussions at conferences to decide the overall direction of the next release of the system and build the social relations which tie the community together.

Chapter 4

Learning more than mastery

- the diverse motivations of the Ubuntu hackers

Though the central characteristic of the Ubuntu community is the shared interest in Ubuntu as a free software operating system, with all the technical and social opportunities for mastering and sharing knowledge of the computer that is contained in its use and development, each contributor to Ubuntu comes to the project with their own motivations, interest, and goals. Participation in free software projects has been expounded in what I have previously referred to as the second-order writings of "hacker culture" commonly as motivated mainly by a "hacker ethic" driven by a desire for mastery of the computer which, when artfully exposed through hacking, showcases the hacker's ability for others to judge and appreciate. In this chapter, I will first present the aesthetical values represented by within this notion of a "hacker ethic" in order to nuance it through Alfred Gell's theory that a given artefact mediates social relations between the creator and the spectator based on the spectator's attitude towards the technical process through which the artefact was created (Gell 1999:172). Following this, I will examine how the developers behind three different software packages in Ubuntu relate to their code and imbue it with different aesthetic values which in turn shape their relationship with their co-developers and express their different motivations for writing and contributing that code to the project.
I argue that the three main motivations of the Ubuntu hackers are primarily social and that the learning of mastery of the computer and the development of the Ubuntu system in turn is the all-encompassing means to fulfil three main social motivations: to prove their ability through technical elegance, to socialize by sharing a technical challenge, and to fulfil a personal ethical commitment by writing software to help others. Further, I will claim that

though all of these diverse motivations are centered on the intimate shared history of learning the intricacies of the computer, they do conflict from time to time, and that it is by negotiating these motivations that the Ubuntu hackers continuously define their concrete *joint enterprise* (Wenger 1998:77-82) in the face of their overall goal of world domination.

Hacking as an art

From its inception, the computer was called "the universal machine" due to its ability to dynamically simulate any other medium – including new media such as interactive communications and simulations (Kay 1984) – by programming the rules and algorithms appropriate for that medium into the computer. Because of this, the computer has become a metonym for technology as it has been adapted as a central tool for all manner of sciences and arts. Programming the computer is a practice characterized by the exquisite duality of being an abstract and conceptual exercise while at the same time being capable of producing remarkably concrete results (cf. Coleman 2005: 233). As seminal hacker Fred Brooks describes it:

> The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures. (Brooks 1995)

Brooks presents the practice of programming as something akin to an art, writing, building the "pure thought-stuff" of rules and algorithms to produce far-reaching real life results from missile targeting to money transfers.[24] But to hackers, the computer does not just simulate or improve other media, it is a medium of expression, a craft in its own right that can be used to create something entirely new. As one of Gabriella Coleman's Debian informants describes it (Coleman 2005: 233, footnote 11):

> I think it can be art, but it is not always... If I had to pick a comparison, I

[24] Brooks argues that collaborative programming work should be perfomed much like a surgical team during surgery with one surgeon performing the most critical work himself while directing his team to assist with less critical parts, thus maintaining one coherent vision for the development of the code.

> would pick carpentry because carpentry always has that range. You can start with just making a bookcase or something utilitarian all the way to creating something like creating a piece of art with wood.

In this way, hackers perceive programming as a craft spanning from the trivial to the momentous. Just as humming a melody or scribbling a shopping list are instances of practices that can be elevated into fine arts when performed under the right circumstances, hacking spans from trivial, "dirty" hacks such as changing the colour of text editor background (cf. Coleman 2005: 237-238) to majestic hacks such as writing a whole operating system like Unix, the source code of which not only has been read and appreciated in university class rooms, but also has attracted a volume of annotated comments (cf. Black 2002: 119-131).

What separates these works are not only their scope, but also the *elegance* with which they are implemented. This aesthetic of elegance is often described as a balance between the design of the code and the function it performs similar to the way mathematical proofs express hugely complicated abstract terms in a concise, yet precise representation. In this way, hackers often consider computer programs as applied mathematics: Their elegance comes not only from fulfiling its function as well as possible, it should also be correct and precise to read. This has made hackers draw comparisons between programming and a wide range of the fine arts, including literature (Knuth 1974, Black 2002), painting (Graham 2004), and music (Kay 1984, Hofstadter 1979) where the infusion of complexity, function, and beauty traditionally has been considered art.[25]

But whereas programming is the act of instructing the computer to perform a task, *hacking* is the playfully clever exploration of and experimentation with computers to master every intricate detail of the system, the computing equivalent of musical jamming. Hacking is, as the renowned hacker Richard Stallman puts it, "playfully doing something difficult, whether useful or not"

[25] A good example of how closely mathematics and elegant computer functions are related can be found in anthropologist Christopher Kelty's dissection of the Unix and Linux command line tool, **grep** (Kelty 2006). For a fuller discussion on how programming languages and more traditional notions of artistic beauty can be combined, see Black (2002) or Ratto (2003:38-45). Practical examples can be found in the programming genre called *Perl Poetry*, for more on this, see http://www.perlmonks.org/index.pl?node=Perl%20Poetry.

(Stallman 2002). Similarly, hacker and self-proclaimed “internet activist” Patrice Riemens compares the explorative, creative element of hacking to the formula "l'art pour l'art' (art for art's sake), arguing that "hackers are focused on the pursuit of knowledge and the exercise of curiosity for its own sake" (Riemens 2002).[26]

This is the essence of what journalist Stephen Levy called the “hacker ethic” in his seminal book on early hackerdom[27]: a collection of ethical notions that hackers seem to have in common: Boundless technical curiosity, desire to take apart and master new technology and share your knowledge, belief that it is possible to create works of art and beauty on a computer (and that these can change your life for the better) and a meritocratic notion that hackers should be judged only on the quality of their hacking, rather than any other criteria such as age, academic degree or position (Levy 1994: 39-49). In this way, hacking is seen by hackers reflecting on their craft as a fundamentally individual act of winning mastery of the computer, perceiving it as a playground that can be explored, and through which they can create and express themselves in that abstract space as a carpenter would a piece of wood. It is especially the results of hacking which is to be shared, to show off their ability in the elegance of their code for other hackers to appreciate with the computer as the final and impartial judge,[28] rather than the process itself.

I contend that the above generalized ethos of the practice and motivations of hacking as an art is too simple a model to explain the motivations of the Ubuntu hackers. I argue that the mutual engagement explored in chapter 3 is central to the motivations of the hackers working on Ubuntu, depending on what they want to achieve with through their mastery of the computer.

Social values infused in the code

[26] This notion is typically echoed among hackers as an explanation for “reinventing the wheel” by writing new software which essentially performs the same functions – perhaps because they find that it was implemented in the wrong language, because there are better, more aesthetically pleasing ways of achieving the same results, or simply just for the fun of learning about the machine themselves (cf. Coleman 2005:229, Raymond 2004:376)).

[27] Levy's book, called “Hackers: Heroes of the Computer Revolution” and first published in 1984, played a big part in spreading and glorifying the concept of a shared hacker culture among the hackers themselves.

[28] Eric Raymond also supports this interpretation of hacking in his essay “Homesteading the Noosphere” (1998), as he sees the individual hackers' contributions to free software projects as bids to win a better reputation for their skills.

In order to understand how the joint enterprise behind the Ubuntu hackers' practice is negotiated, it is worth examining how they relate to the end result of their labour: the code itself. I contend that the Ubuntu hackers have clear values with which they seek to imbue their work – depending on their social relations to their perceived audience. These values are placed in what can be described as the spectrum of the "hacker aesthetic" – between the form and the function of the code as presented above.

In order to anthropologically analyze the values hackers attribute to their work, I assume the "methodological philistinism" advocated by Alfred Gell, who contends that we must take a stance of "resolute indifference towards the aesthetic value of works of art" in order to make a coherent anthropological analysis of the specific objective characteristics of art objects without succumbing to the fascination of the process through which they are crafted (Gell 1999:161-162). Gell argues that our appreciation of aesthetic value is closely connected to our understanding of the technical process through which an artefact has come into being. When the difficulty of that technical process transcends our understanding, it forces us to construe it as magical (Gell 1999:169), thus adding to our fascination. Following this, Gell contends that

the technical process through which an artefact has been created is "not only the source of its prestige as an object, but also the source of its efficacy in the domain of social relations." (Ibid. 178). He applies this in his analysis of the delicately carved prows of the Kula-canoes of the Pacific Trobrianders. There he turns away from traditional aesthetic considerations and argues that the prows not only showcase the skill of the carver but also his magical ability, thus leveraging the prestige of the canoe as an art object with great effect in the social relations between the skilled canoe carver and the spectators, who come to consider his skill as magic ability (ibid.:175-178).

Similarly, I turn away from the traditional considerations of the aesthetics of

source code in order to argue that the Ubuntu developers write their source code based upon design values which seek to invoke specific attitudes with their audience, attitudes typical for the social relations that they seek to encourage through their work.

Since the source code can be copied endlessly at no cost, the magic of its technical process does not lie in the aura of individual original representation or the spectator's appreciation of the manual skill that went into producing it, but rather in the mental ability required to write, understand, and master the code. Thus it would seem that the skill of a hacker can be measured by how well he has managed to fulfil the dual requirements of writing code that fulfils its intended function while remaining elegantly concise – often to the point of convolution. The greater the gap between the amount of code written and the result produced, the more magical and artistic it will appear in the eyes of the reader. And though the results produced by the computer reveals the final verdict of hacking ability, understanding of source code – or lack thereof – is often seen as a good indicator of the ability of the individual hacker, making intricately complex code not only playful challenges but also assertions of skill and dominance. Following this, the Norwegian anthropologist Lars Risan argues that the hacker infuses the code with his skill and charisma in a sacramental way: It is not just a symbol of the hacker's skill, it *is* the hacker's charisma – visible in his code (Risan 2005). Thus the hackers can show off their abilities through the elegance of their code, and, by extension, their own minds.[29]

This kind of "artful hacking" is how hackers commonly present their craft as described above, and though a fair number of hackers do write code to assert their skills, it is only a few of the Ubuntu hackers who define hacking as an art in this way. Rather, they define hacking as the act of winning and retaining mastery over the computer, which, when it has been achieved and the program works as planned, is an immensely satisfying experience. As Ubuntu

[29] Hacker folklore is full of stories of elegant hacks that shows off the ability of a programmer and challenges the reader to understand it. One is the story of the Mel, a so-called "real programmer" whose code was so intricately written that it used every single possible function of the hardware to run as fast as possible, but which also made it so difficult to understand that the storyteller spent almost 2 weeks just to understand the code, only to remain unable (or respectfully unwilling) to alter it. The story, like much other hacker folklore and jargon, can be found in Eric Raymond's "The New Hacker's Dictionary" (1996).

and Synaptic hacker Michael describes it:

> It's just like having finished a puzzle... it gives you a sense of accomplishment... a creative outlet, but not really an art. To be an art it has to involve passion and feeling. Programming is not about love and great feeling. [...] It is mostly a mastery of skill. A complex skill... [...] It has to have some meaning. It has to make life easier in some way – solve problems somehow.

If the code *functions* well enough not to warrant closer examination, nobody will care whether the code is elegant or not. It is not until a hacker needs to modify the working code himself that he becomes concerned with its design, the programming language in which it is written, and the shape of its internals – all of which are elements which will task his skill and control of the computer. But in order to improve the source code, you need to understand it, and this easy understanding is often at odds with the “elegance” and artfulness advocated above.

Instead, the Ubuntu developers promote values such as transparency and modularity in order to ensure that others will be able to work on the code once they move on to other projects. In his analysis of the Linux kernel developer community, the American media researcher Matt Ratto calls this continuing coming and going to software projects a *culture of reworking*. Reaching back to the "oral tradition" of Unix, it is a culture of developers creating software "inherently directed towards 're-designers' – other programmers with the skills and knowledge to re-work the programs for their own use" (Ratto 2003: 114). In this way, Ubuntu hackers focus on the lasting value of their work which through the open model of development becomes cumulatively more maintainable and perfect for any conceivable use.

Yet other hackers focus on the results produced by the code rather than how elegant or maintainable the code might be to other developers. Michael sums up this position quite well, as he sees it merely a positive side effect if others can read his code. He argues that generally, you need to understand the problem that the source code seeks to solve in order to understand the source code – you have to be in the same frame of mind as the developer who wrote it, and then it won't matter how complex the code is. What matters most to him is not what you share, but the fact that you are sharing freely on the same terms as everybody else. This notion stems from the ideology of the free

sharing of information driving projects like GNU and Debian built, which Gabriella Coleman argues is built into a “corporeal ethos” through shared technical practices such as those described above. Coleman sees this ideology of sharing as having initiated a new reworking of the concept of intellectual property through other projects such as the Lawrence Lessig's *Creative Commons* which seeks to introduce elements of the free software ideology in other forms of intellectual property such as photography, films and writing (Coleman 2005:187).

Thus, there appears to be three different sets of social relations that can mediated through the source code between a hacker and his intended audience and potential collaborators, depending on what design elements and aesthetic values the individual hacker favours.

One set of social relations follows an ideal of the lone brilliant artist whose work is to be appreciated unaltered and which asserts his seemingly magical ability in relation to other hackers through complex and concise code appearing as puzzles to be solved. Another set elevates an ideal of communal authorship seeking to make the code maintainable and understandable to enable the open collaboration that may result in the continued use and development and eventual perfection of the code. And a third set that puts the free sharing of as its ideal, offering users new capabilities to improve their use of the computer.

Almost all of the Ubuntu hackers will argue that these three ideals are not at odds with one another and that it is possible to provide for all three, since they themselves not only appreciate the challenge of intellect that complex code represents and often use it as a way to show off their abilities, but also enjoy the easy mastery of the computer offered by code written with maintainability in mind and the immediate usefulness of code written with ethical commitment to provide those functions with little worry for elegance or maintainability. In this way, the hackers follow their shared history of learning both skills and their aesthetic notions of hacking according to the “hacker ethic.”

In the final part of this chapter, I will use the examples of three software packages in Ubuntu to show how different sets of social relations and

motivations can be built and mediated through the code – to the point where the above design values may come to conflict with each other, thus breaking with the generalized idealism of the "hacker ethic" and showing how the Ubuntu hackers negotiate their joint enterprise in relation to the overall goals of the Ubuntu project.

Writing beautiful code to assert one's ability

Peter is the developer of *Adept*, a package management tool like Synaptic, which I explored in chapter 3. But whereas Synaptic is designed for use with Ubuntu's standard desktop environment called GNOME, Adept is designed for use with the desktop environment KDE. Young, ambitious and highly talented, Peter began developing Adept, hoping to incorporate some new features that were being developed in the Debian community, but he found it difficult to work with the existing APT code upon which Synaptic was built, as he considered its design flawed as it forced him to write a lot of code just to accommodate it. He expressed his frustration on his blog:[30]

> Existing code just hates me. I always have to work around some brain damage. I had to ditch kmdi. It just didn't work. I spent a lot of time discovering new and interesting ways in which it can break. This is very frustrating experience. And i'm experiencing the same with konsolepart now. Buaaaahaaa. *Rethink your design*. [original emphasis]

In the end, he had rewritten a great part of the core APT functions, and gave up on the original APT code, basing Adept on his new work – which several other Ubuntu developers mention as examples of really well-engineered code, though, by their own admission, it also requires much effort of them to read and modify. As Adept no longer was based on the APT code which had troubled Peter so, he stopped his collaboration with the Synaptic developers. As Michael, the lead developer of Synaptic, hesitantly explains:

> That's a difficult kind of story... [...] I think he wanted to do it differently, and he wanted to do it, like, in his way... and he just started something new [...] at least I felt that initially he have been very challenging: 'I will do

[30] Peter's blog is a public diary, containing poetry, rants about bad code and reflections on his life. It is very honest, at times conveying depressions and remorse, acknowledging that his perfectionism at times makes it difficult for others to work with him.

> better than Synaptic which is kind of not good and I can do better and this sort of thing.' [...] some people are there to prove a point. To show that they can do better...

Though not initially intended, frustration at the lack of mastery that the APT code offered him led Peter to develop Adept by himself to prove the worth of his design and assert his mastery of the computer directly in the code.[31] Michael for his part is distinctly unhappy that Peter decided not to collaborate with them, as he, like most Ubuntu hackers, doesn't share this view of the on-line environment as an arena for contest.

Hacking as a shared technical challenge

Most Ubuntu hackers see their community as a place for collaboration where they can joke and socialize whilst sharing the technical challenges of

[31] Indeed, it appears that many of the biggest forks and rivalries in the free software communities have been sparked by a technical frustration based on the fact that the project's programming language or choice of design did not match the interests and abilities of some hackers, thus making it difficult for them to contribute and show off their abilities. The classic example being the two free software desktop environments, KDE and GNOME, written in the two different programming languages C++ and C, respectively. One of the reasons for the initiation of the latter project allegedly being the founders' lack of skill with C++, which led to some ridicule from the KDE developers.

development, writing code so as to support other hackers' understanding to make it possible for them to extend and adopt it for their own needs in the hope that this open, communal authorship of the code will yield the best results. Ubuntu developer Scott is a keen proponent of this, enjoying the free availability of the source code and likes the challenge of mastering the computer. Nothing frustrates him more than not being directly able to fix a problem either due to the source code being difficult to read, or due to being given inadequate information to reproduce the problem. He loves being able to do “drive-by bug fixing” where he can read just the small part of the code relevant for his fix without having to understand all of its details and the mindset of the programmer who wrote it. Scott likes to move from project to project as his development interests change, and he works hard to make maintainability a key design element in his code. For example for his current work on the new *Upstart* package, he clearly indents and comments it, introducing other developers to it so that they can help maintain it and take it over when he moves on. He appreciates the high ambition of the Ubuntu project that “Things should *just work*” - a phrase that has become something akin to a motto within the community. It is an idea which Ubuntu hacker Matthew expresses thus:

> Computers behave deterministically. Software behaves deterministically. The combination of these two things lets us produce software that works 100% of the time. That's harder than getting to 99% of the time, but even so accepting the "Make it work 99% of the time and add a bunch of preferences to let the 1% make it work themselves" approach is accepting inferior software. We aim high, and we fairly consistently hit. And that's how it should be.

Apart from expressing a deep conviction of the potential for perfection of computing technology typical among hackers, Matthew's statement is also remarkable in that he states that it is something that “we”, meaning the Ubuntu community, can do. In this way he includes the wider community involved in the open development model with their testing and bug reports. Indeed, as Ubuntu hacker Martin explains, it is the appreciative user feedback and the discussions to find solutions that many hackers find to be the most rewarding:

> So one part is of course solving problems with programming and the technical stuff, and the other thing is that I really love to help other people. So if they come to me and have an interesting problem - "hey, I want to achieve this-and-this" - my brain starts engage and propose different solutions and to discuss them and to implement them. Well, I love challenges, basically!

In this way, the challenges of making the software "just work" based on bug reports and of implementing user suggestions both spring from interactions within the community, sharing technical challenges. But as both hardware and software continue to evolve, and new features and new hardware constantly introduce unknown factors and bugs into software which otherwise "just works", often breaking software which worked well before and requiring both engineer redesign work and bricoleur bug fixing work in order to remain stable. The goal of perfect software will continue to offer a never-ending stream of both shared technical challenges and appreciative user feedback on which the Ubuntu developers thrive.[32] As Ubuntu hacker Paul says with a smile, after having spent 4 hours working with Matthew to hack the LED lights on an obscure brand of tablet PC trying to get them to work properly: "I shall be annoyed until I figure it out."

Sharing an ethical commitment

For the Synaptic hackers Michael and Sebastian, the essential matter is not whether the code is easy to maintain or beautiful to read, but that it is useful and free to be shared – a quality which they have come to consider with as something akin to an ethical obligation. As Michael explains:

> I think it's really really important to be able share your work and your knowledge with other people. And especially if you are gifted in a way that you... that you understand more about this than other people. Should, in my

[32] Similarly, Linus Torvalds has explained that it was due to the interest from the Usenet discussion groups that he continued to develop the Linux kernel beyond the first couple of versions: "Without Usenet, I'd probably have stopped doing Linux after a year or something ... it was a fun project, but at some point I'd have felt that hey, it's done, I've proved it, I did this thing, it was fun. What's the next project in life? But because people started using it, motivation went up, there was a sense of responsibility, and it got a lot more motivating to do it. And so thanks to Usenet I just continued to do it." (Moody 2001:69).

> opinion, be a very good reason to help and share more than people who understand not as much about it.

It was this notion of creating something with the computer that is useful and helpful to other people that led both Michael and Sebastian to get involved in making the Debian package management infrastructure understandable and usable for less technically inclined users – first through Debian and Synaptic, and eventually to Ubuntu. Their collaboration and friendship seems to be in no small part based on strong, personal ethical commitment which they have found to extend beyond software: One is a vegetarian, arguing that he cannot eat the meat of any animal which he is not capable of killing himself and the other is a vegan seeking to pay respect to all life.[33] They have found that the ability of software to create endless copies of concrete results through digital means empowers them to follow through on their ethical and political ideals and make free software more accessible to more people, thereby hoping to help create a sustainable alternative to the corporate closed model of development.

For Michael, working on Synaptic has offered the challenge to use his abilities to improve free software, making package management more stable and usable, which has been only reinforced by his employment by Canonical. For Sebastian, though appreciative of the goals and achievements of the Ubuntu community, he is not as closely attached to Ubuntu, and he says that the future of Ubuntu isn't very important to him. As much as he'd like to see Ubuntu made as accessible and usable as possible – especially for people in the third world – he has other goals in his life and does not feel committed to Ubuntu directly but rather to the ideals the project represents. As he concludes: “If people come to me with a problem, I will always help them.” For now, contributing to Ubuntu is a meaningful and intellectually stimulating way of expressing his ethical convictions.

But at times, this ethical commitment is challenged by the need to make Ubuntu “just work” for as many people as possible. As many users, and Canonical, want to make it easier to use Synaptic to install proprietary software, including software that supports proprietary media formats which most home users have come to expect and need if they are to be lured away

[33] Though not uncommon, vegetarianism is far from the norm among hackers, and most happily eat whatever they please.

from Microsoft Windows. Recently, Sebastian made proprietary software available for installation from the Internet on the Ubuntu default system, and though the Ubuntu community officially maintained its pledge to ship a system consisting only of free software on the CD-ROM, Sebastian and Michael's work made it much easier to install proprietary software straight away, treading a careful balance between the ideological hopes of the free software movement and the pragmatic quest to solve Ubuntu's bug number one and win world domination for free software.

Conclusions

In this chapter, I have examined the diverse motivations of the Ubuntu developers for contributing to the Ubuntu system. I have claimed that the Ubuntu hackers' various motivations for participating in such an inherently social community of practice are reflected in how they position their work in relation to their fellow hackers, and drawing upon the work of Alfred Gell, I have sought to provide a more nuanced rendition of the common understanding of hacker motivation as represented in a range of second-order hacker reflections. Based on this analysis, I have found three main motivational factors for the hackers' mutual engagement in Ubuntu: proving their ability through technical elegance, having fun by socialising and working together on a technical challenge, and sharing an ethical commitment to writing software that can be shared freely for the benefit of others.
The centre of gravity from which these three factors extend is the not quite reified goal of continuous learning through practice that is a vital part of any community of practice (Wenger 1998:86) (cf. illustration 9).
Most of the Ubuntu hackers I have interviewed consider it possible to accommodate the goals of all three motivational factors, but as I have sought to show above, the motivational factor prevalent within a given project will shape it on a coding level. By shaping the code for either elegant beauty, maintainable extension, or easy study or pragmatic use, the developers create the potential for conflict between other developers seeking use that code for their own needs. These factors shape not only Michael and Sebastian's Synaptic, Peter's Adept and Scott's Upstart, but also each of the more than

19.000 other upstream projects which are packaged as part of the Ubuntu

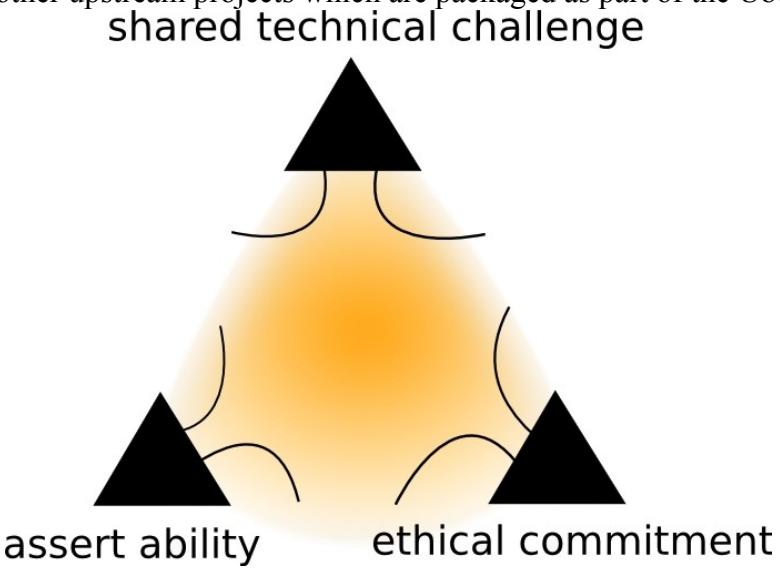

Illustration 9: A depiction of the field of motivational factors within which Ubuntu is developed. At the warm core, it is the continuous learning through practice which engages the Ubuntu developers, yet this learning can be channeled in different direction depending on how the social interests of the individual developer.

system.

Remarkably, the Ubuntu hackers explain their contributions to Ubuntu and to free software with their enjoyment of the socialising and connecting with users, learning from the technical challenges involved in its development, and the ethical commitment to free software rather than any a direct desire to prove their ability. This is partly due to hackers not wanting to present themselves as vain enough to consider this their main motivation, and partly due to Mark Shuttleworth picking and hiring developers with those social and ethical values, offering them further incentive through employment, travels and responsibility. In this way, the joint enterprise of the Ubuntu developers is to some extent socially engineered to build a community of practice founded on the positive development values that Shuttleworth seeks to groom to further his overall goal of world domination: communal perfectionism and an ethical commitment to free software.

Chapter 5

"A system that works for me"

– how Ubuntu hackers use and configure their system

To the Ubuntu hackers, the computer is not merely a tool, it is the environment in which they work, interact and collaborate. Like any other craft, hacking requires specialized tools, but for the hacker, these tools are programs such as text editors, compilers, debuggers, IRC chat clients, and email programs which all constitute functions accessible through the computer. Since the way that the individual hackers perform their work can vary a fair deal, so can the way in which they configure these tools for their own use.

Having examined a few select packages among the thousands of the Ubuntu system to show how the Ubuntu hackers participate in a community of practice based on their diverse motivations and shared work with improving the system, I now turn to the Ubuntu system as a whole to examine how the Ubuntu hackers carefully configure the system as the all-encompassing means to afford their personal mastery of the computer through which they fulfil their social and personal interests.

In this chapter, I describe how one Ubuntu hacker, Martin, uses and customizes the Ubuntu system in order to argue that the Ubuntu system is not simply a tool to be used, but rather the scene of the *shared repertoire* (Wenger 1998:82-85) of tools and routines through which the Ubuntu community of practice is defined – a space at once present both on the local computer and on-line, multiply inhabited, shared, and continually built by the hackers in a way similar to what Tim Ingold describes as *dwelling*. I claim that the continued stable development of the Ubuntu system takes place through a process which Ilkka Tuomi calls *sedimentation*, which allows the Ubuntu developers as well of the millions of people using the system to dwell within the system, adopting it as their own.

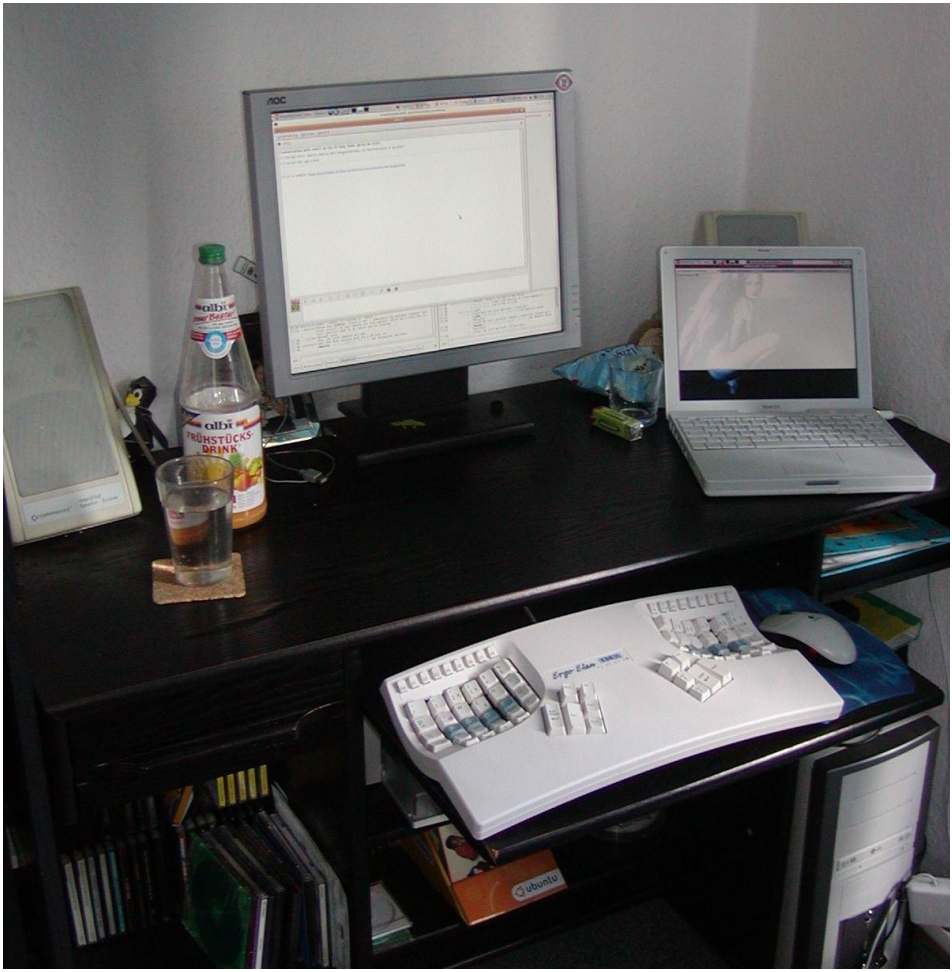

Illustration 10: Martin's work place. The laptop is used for testing and travels. The special keyboard is ergonomically designed to reduce typing-induced stress on the wrists, as Martin has had typing-related injuries.

Configuring and using the Ubuntu system

Like most Ubuntu hackers, Martin works at home, in the small commune apartment which he shares with his girlfriend and a friend. As they all have their separate rooms, it has been possible for him establish his work environment in his bedroom, in the corner furthest away from the entrance. He works with two computers: his main system where he does most of his work, and his laptop, which brings with him for conferences and uses for

testing various volatile programs which might compromise his system (cf. Illustration 10).
Until he began working for Canonical, Martin rarely ever used the system's graphical user interface, but mostly just the command line – the old command-response interface of Unix and teletypes described in chapter 3. The command line gives him a clear and transparent means of interacting with the computer through the concise and extensible commands he inputs, it also allows him to spread his work beyond his own machines, as he can use it to

log on to computers at the Canonical data centre, to computers related to other free software projects around the world or to his own server halfway across the country, with the abstract and transparent textuality of the command line offering him the same level of control of those remote systems as if he were there in person.

But most importantly, the command line allows him to configure his work flow in as much detail as he wishes. He uses the command line for email, for editing text and source code, and for manipulating the software packages that

he builds, tests and uploads to the Ubuntu servers. All of this work involves some repetitiveness, which he constantly seeks to limit by writing new *scripts* – series of commands to be invoked with a single command. Martin gives the example of having to spend 15 minutes calculating how much each member of the commune owes whenever they received a new telephone bill. Eventually, he decided to write a script which can parse the digital copy of the bill which the telephone company sends, and output how much each member of the commune needs to pay. In this way, he has optimized his work flow by

making the script doing the work for him, so he no longer has to remember all the details of the phone bill calculations. As he concludes:

> It always starts out like that. You're doing your task, or a task you've done before, and at some point you wonder "why do I always need to type five different commands to achieve this task which I'm doing over and over again?" [...] And so, some times when you have nothing urgent to do, you just think about the details and say "let's factorize this into a function or [...] let's factorize this into a *shell* script [shell is another computing term

for the layer of interaction between the user and the computer system].

This practice is very common among the Ubuntu hackers who continuously customize their work environment to fit their current work, adapting and configuring the entire system to fit their needs. Martin has been customising his systems for over 8 years, slowly evolving with the each new version of Debian or Ubuntu he installs. As he explains:

> It's [the command line] basically like an organism that over time you can dynamically adapt it to your needs and to your current tasks so that you can work on it quite efficiently. This is what makes it so much fun, because you can never do this with GUI [Graphical User Interface] applications. I mean, it might be easy to learn them, but it always take the same time to do, so it is hard to factorize common tasks to them.

In this way, the Ubuntu system with the command line, its most flexible and transparent representation, allows Martin and other Ubuntu hackers to build a shared repertoire of tools and routines spanning almost all of their mutual engagement in the project, including programming, managing their emails and communications, emulating and testing other systems, and accessing other computers on the Internet, thereby constituting an entire digital landscape which they can adapt for their needs.[34]

Dwelling and building in the Ubuntu system

The way that the Ubuntu hackers arrange the digital space of the Ubuntu system shares more than a few similarities with how human beings arrange, use and build the physical space they inhabit. The British anthropologist Tim Ingold, seeking to understand how we come to be part of the environment we inhabit, breaks with the traditional "building perspective," which posits our cities and buildings as the concrete manifestation of our purely cultural ordering of our environment (Ingold 2000:178-181). Instead he defines a "dwelling perspective" based on Martin Heidegger's term "dwelling" which springs from Heidegger's examination of the German word *bauen*, which today means building in the sense of cultivation, preservation and construction, but the etymological roots of which also means dwelling as in "the whole manner in which one lives one's life on the earth," in short

[34] It is an acknowledged fact that hackers have widely different needs and preferences when it comes to customising their system depending on their personal preferences and the hardware at their disposal. Thus, the Canonical-employed Ubuntu developers have from the beginning sought to encourage and enable community hackers to rebrand and re-package Ubuntu for their specific needs. This has not only resulted in a wide range of derivative distributions, but also actual sister versions of Ubuntu, sharing the same infrastructure but using other central parts of the free software projects available. For instance, the Kubuntu version, which uses the KDE desktop environment rather than the GNOME desktop environment that is the default used by Ubuntu.

meaning that “I dwell” is identical to “I am” (Ibid. 185). Based on this, Heidegger concludes that “To build is in itself already to dwell ... Only if we are capable of dwelling, only then can we build” (quoted in Ingold 2000:186). Heidegger argues that we are always dwelling, whenever we use and inhabit our environment, and it is only by dwelling that our environment can be understood and thus organised in the mind. Following this, Ingold argues that building cannot be understood as a simple process of realising a pre-existing architectural design through raw materials – it is only through dwelling, relating to the world one inhabits, that building is possible.

Thus, Ingold argues, as children grow up in environments furnished by the work of previous generations, they dwell in a built environment that is shaped through human intentionality, and through that environment they adopt “specific skills, sensibilities and dispositions” with which they in turn continue build the environment (Ibid:185-186).

Based on the above examination of the Ubuntu hackers' use of the Ubuntu system, I contend that a similar process of dwelling is at play. As I argued in chapter 2, computer systems are like cities: Layers upon layers of human intentionality that evolve as new inhabitants move in, constantly re-exploring and redefining its structures and applications, adapting and re-designing them for new purposes, maintaining them differently as needs change. And the Ubuntu system, and especially the command line at its heart, stems from the almost 40-year old “oral tradition” of Unix technology being refined and redesigned for varying needs.

Thus the system, like any other built environment, has been shaped by the activities of several generations of predecessors whose work is now reflected in the skills, dispositions, and dwelling of the Ubuntu developers. When they adopt and learn the details of the system, customising it to fit their needs, they begin to dwell within the system, internalising their work flow and social relations to their shared community directly in the system itself, turning it into an extension of their mind, as described in chapter 3.

It is in each hacker’s adoption and customization of the shared repertoire of the tools and routines of the Ubuntu system – and with it the associated community of practice – that their shared history of learning to use, to build, to dwell within the system becomes apparent. And the hackers often share the stories of interesting configurations and use of the system, such as scripts of

which they are particularly proud, incorporating stories and slang into their shared repertoire as well. As I will argue in chapter 6, it is through this shared history of acquiring the skills and internalising shared norms, that the Ubuntu hackers become members of a shared community of practice.

The sedimented system

As I argued in chapter 4, each Ubuntu hacker weighs his motivations for working on Ubuntu differently in their negotiation of the joint enterprise of Ubuntu, and for that same reason, they all find their personal adaption of the system to be a central benefit, as it allows them to configure the system to match their different goals. Ubuntu hacker Colin's answer to the question of what he gets out of contributing to projects like Ubuntu and Debian, proved quite typical:

> "I got a system that works for me." (he pauses, grins, then elaborates:) "...It's like moving from renting a house to buying a house – you begin to look at it in a different way. You begin to notice all the places where there is room for improvement. Part of it is commitment... having and taking the responsibility to make things work, but it is also having the opportunity to do so. You decide for yourself."

Colin's comparing his use of the Ubuntu system to inhabiting a house is remarkable, as it underlines that it is only once the hackers have taken up this state of dwelling within the system that they begin to notice how it can be improved, and begin to take responsibility for building, extending and improving it.

Just like how Michael, upon becoming more familiar with the Debian system, decided to adopt Synaptic as described in chapter 3, the open model of development allows the individual hacker, such as Colin, to improve the system, taking responsibility for making changes that will affect more than just themselves and their own use. Whenever the Ubuntu developers fix a bug or implement a new feature in a software package, they upload a new version of that package containing those changes to the main Ubuntu archive on-line, against which all on-line Ubuntu systems running the same version of Ubuntu are synchronized through Internet updates controlled by package managers such as Synaptic. This not only makes it very easy to distribute fixes as they

are produced, but also to introduce unexpected problems if these updates aren't thoroughly tested.
In order to be able to extend the Ubuntu system without risking to break the system for the many users through an ill-conceived software update, the Ubuntu developers work on at least three active versions of Ubuntu at any given time: Two stable versions which have been released within the past year for end-users to use and for which Canonical guarantees support – which requires the Ubuntu developers to maintain and update them as new security holes and integral weaknesses are exposed. And one version which is the current development version which, as described in chapter 2, merges the Ubuntu system with the latest unstable upstream software derived from Debian, which the Ubuntu developers then work to gradually stabilize, with the testing and bug fixing work intensifying to ensure the quality of the system in time to meet the release date.
The Finnish technology theorist Ilkka Tuomi has called this process of maintaining different versions of the same software a continuous *sedimentation,* comparing the older, stable versions of the system as layers of sediments left behind in the evolutionary pattern of software development (Tuomi 2001). And though the Ubuntu developers spend time maintaining the older, stabler sediments, they dedicate most of their time to working on the development version as that is where the software has not yet been sedimented for others to use. Like construction workers collaborating directly on-site, the Ubuntu hackers do their work directly within the virtual construction site of the development version of the system itself as it allows them to spot and fix problems in the system as soon as they occur.
With each new development cycle, the Ubuntu hackers start afresh, branching out from the Debian upstream, slowly rebuilding the system to incorporate the latest upstream changes and their own new features, and at the same time stabilising it for their own use. In this way, their work follows the two aspects of hacking described in chapter 3, resulting in two parallel tracks of work: One focused on *development*: the drafting, designing, developing, and *hacking up* new features and their eventual integration in the release; and one focused on *integration*: the merging of improvements from Debian and stabilising, maintaining, integrating, and *hacking on* the system to ward against the constant disruption caused as new features and redesigns

inevitably introduce new bugs and regressions to the code.
Illustration 11 shows how the system is developed. Vulnerable at first, it slowly matures and hardens through gradual *freezing* of the system code in order to avoid introducing new bugs. A *freeze* is the common hacker term signifying that the code is not open for change. Despite the name, a freeze merely indicates a different degree of control with changes made to the system, as hackers will need to apply for a Freeze Exception from one of the leading Ubuntu hackers employed by Canonical – such as Matt Zimmerman, the chairman of the Ubuntu Technical Board, or the release manager for that given release – in order to get their changes uploaded. During the development cycle, test releases are made as more and more of the system is frozen, in order to get the constant feedback in the form of bug reports and discussions from and with the wider community of interested users testing and using the evolving development version of the system. The input from these contributors are necessary to ensure that the shared environment of the Ubuntu system not only works for the Ubuntu developers, but also for the main body of users who will be using the system once it is released.
It is the leading Canonical-employed hackers who deal with the stress of making the development version ready for final release, which they release

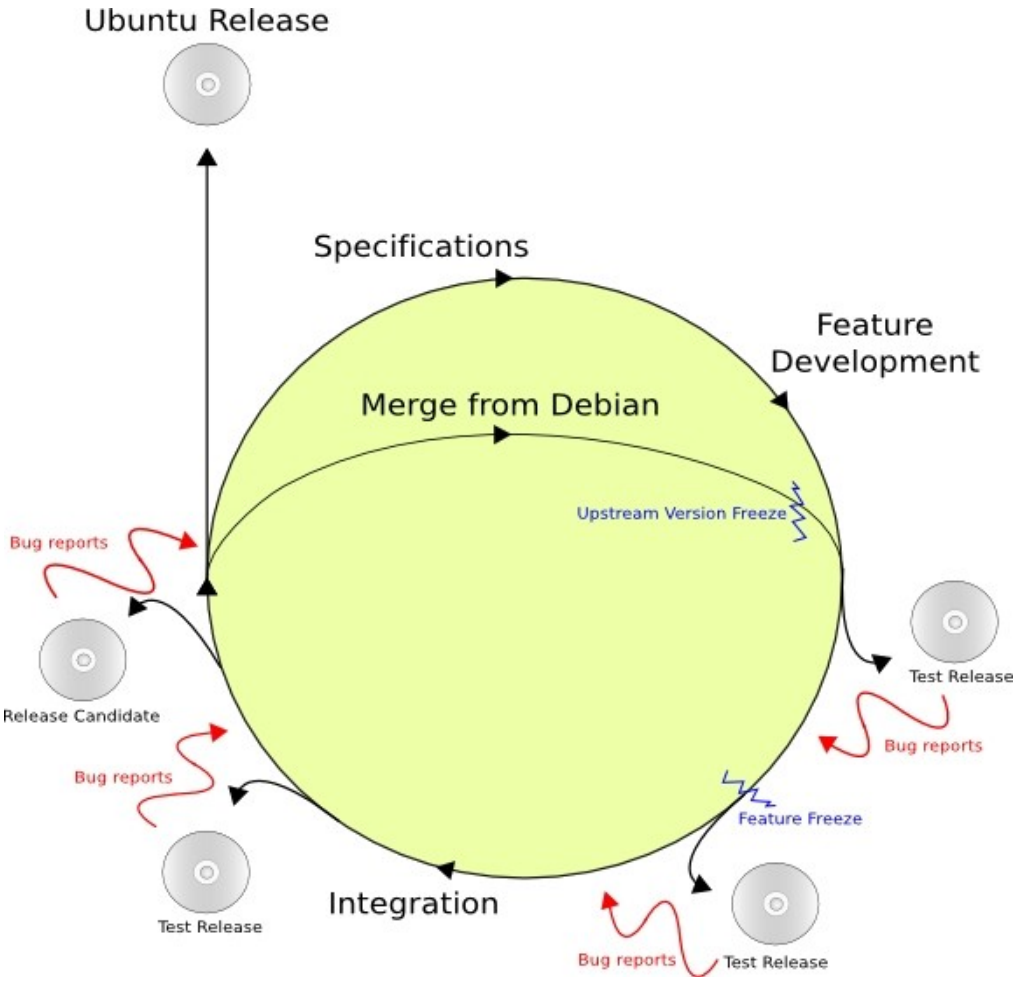

Illustration 11: A simplified diagram of the Ubuntu release cycle with its two parallel tracks. One focusing on the merging of the Ubuntu code with the Debian upstream (ending with the Upstream Version Freeze at which point the developers work to integrate the new code in the release), and one focusing on the specification, planning and development of new features for inclusion in Ubuntu. Feature development ends with Feature Freeze, at which point all developers focus on integrating the new features in the system for release.

when they consider it stable enough for all – both hackers and users – to use. When a new version of Ubuntu is released, a new development version is immediately initiated, and usually, soon afterwards, an old stable version of Ubuntu is retired. But as soon as the system has been released on CD-ROM, it is no longer frozen in the on-line archive, and the Ubuntu hackers continue to develop on it so as to support it by adding changes for later updates which users can download upon installation.
For all the contributors, from core hackers to casual bug reporters, the system being developed is not merely a product for somebody else to use – it is their own work environment. Therefore, the system becomes an end in itself, rather than just a tool to be wielded to solve other problems. It is through the continued successful shared use and development of the system that the Ubuntu hackers not only seek to fulfil their motivations for working within the community, it is also the nexus of their shared association.

The many dwellers of the Ubuntu system

Tim Ingold concludes his essay on dwelling by comparing a house to an oak tree as described in the Estonian biologist Jakob von Uexküll in his book "Stroll through the Worlds of Animals and Men" (Ingold 2000: 176). Von Uexküll shows the manifold inhabitants of the tree: There is the fox building his lair at its roots, the owl perched on high on its branches, the squirrel foraging nuts in the intricate maze of the treetop, the various insects making their home in the bark and many others. Each of these animals perceive and act differently in the environment of the oak tree, and von Uexküll defines the sum of each animal's actions and perceptions of the tree as its *Umwelt*. Von Uexküll argues that each animal is unable to detach its consciousness from its Umwelt to see the tree as a neutral object – they will invariably attach their own activities to it (Ibid. 176-177).
Ingold argues that a house is inhabited in much the same way as von Uexküll's oak tree – that houses, too, are living organisms, constantly being built and rebuilt through the dwelling of its many inhabitants – human as well as non-human – shaping their own Umwelten through the flow of intentional activity (Ibid. 187-188).
Like the house or the oak tree, the Ubuntu system is an incredibly big and complex entity with the potential to cover a constantly increasing number of

configurations, uses and needs. Each user of Ubuntu values different functions of the system and has different motivations for using it, and through this use, each user comes to adapt the overall, standard system to his own personal work flow and interests, shaping it into a personal Umwelt, distinct and separate in use and configuration from that of other users, drawing upon the thousands of software packages to fashion his own nest in which to dwell. These Umwelten can be as physically and mentally separate from one another as that of a squirrel might be to an owl. From high-powered servers offering critical services to hundreds of users to the cheap desktop PCs being used for games and socialising. From Ken, an artist using the system to design graphics and edit photos, requiring minute manipulation and specialized photo equipment, to Henrik who is paralyzed from the neck down and requires a specialized “head mouse” and dialing wand to navigate the system, shaping not only his work flow but also his configuration of the system.
In this way, each user adopts the system for his needs, some even contributing back based on that use, in order to scratch their own itches. Like Ken who, displeased with the look and feel of the desktop, works to make it as visually pleasing and consistent as possible, or Henrik who works to improve the accessibility features of Ubuntu to make the system easy to use for disabled users, or Og who submits Brazilian translations of text in the Ubuntu packages in order to improve Brazilian language support in Ubuntu, or Sebastian who, wanting a better graphical user interface for his favourite package manager, works to improve Synaptic. Thus, in order to grow, the system is dependent on

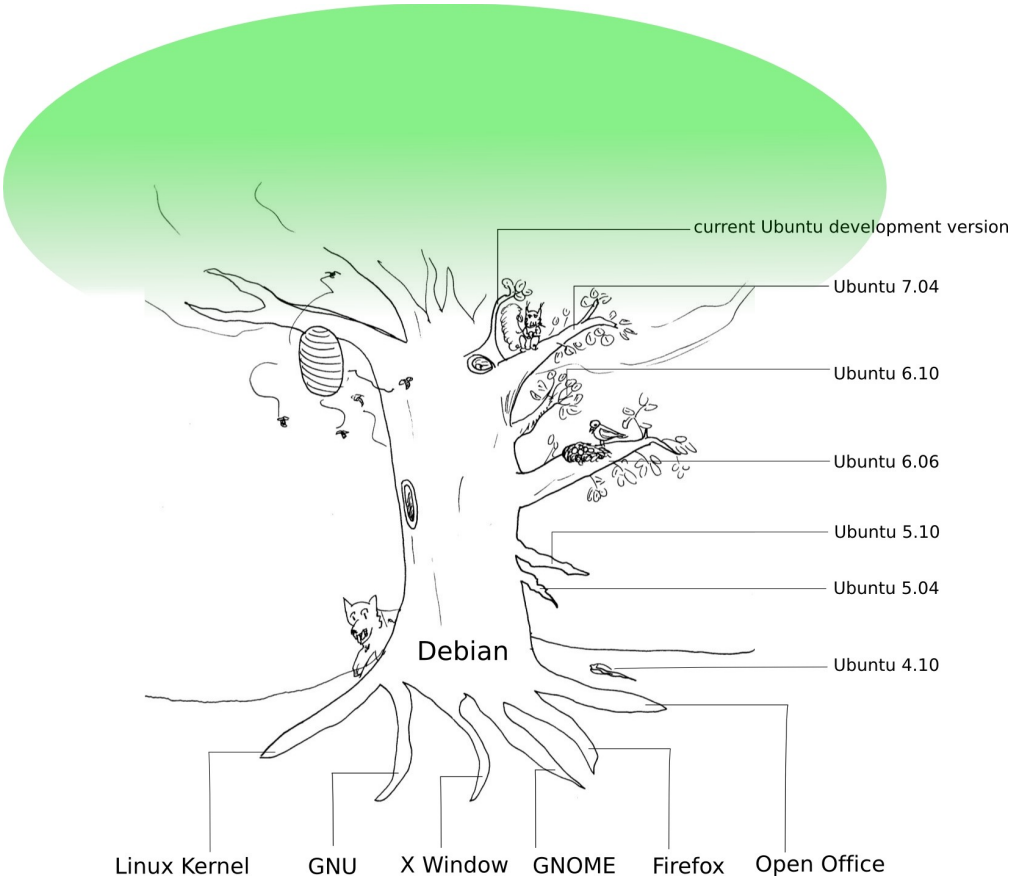

Illustration 12: Using the idea of the free software communities as an eco-system of interdependent upstreams and downstreams, the upstream projects are the roots, creating the foundation of the system united in the Debian distribution, from which each new version of Ubuntu is an offshoot, growing stronger and stronger through the development and maintenance of its life cycle, supporting a multitude of users and their diverse needs throughout the eco-system, before each branched version of Ubunu eventually reaches the end of its life cycle as a product and is shed from the tree as old, unsupported technology.

all of the hackers and users individually dwelling, sharing and collaboratively building the system into a constantly evolving organism, like all the different species inhabiting and supporting the eco-system of an oak tree, with each new version of the system building on top of the previous to reach further, supporting more uses and covering more needs (cf. Illustration 12).

Conclusions

This chapter set out to explore how the Ubuntu hackers use and customize the system. Like other expert craftsmen, the Ubuntu hackers spend much effort towards configuring, maintaining and extending their work environment to afford a shared repertoire of specialized needs in order to enable them to work efficiently. But not only does this involve customising work tools such as the command line, but also maintaining connections to other computers and test systems, organising personal files, and configuring relevant means of communication to fit the personal needs and mental pursuits of the individual hacker. I have compared this way of being at home in the built environment of the Ubuntu system to Tim Ingold's notion of dwelling, and I have argued that dwelling is the necessary prerequisite for developing the system. As Ingold remarks, "building, then, is a process that is continually going on, for as long as people dwell in an environment" (Ingold 2000:188).

The Ubuntu developers use the open model of development to collaboratively engineer their shared environment in a sedimented, almost organic fashion that affords not only their own destabilising practices of building and dwelling in a personal, trusted system of their own, but also the specialized needs and configurations of the millions of users who also come to dwell on the more stable branches of the Ubuntu system. In this way, the Ubuntu system is the scene of the community of collaborative practices through which the hackers develop and use the system itself as the means through which to fulfil their diverse social and social interests.

Chapter 6

Building trust online

– how the Ubuntu hackers collaborate on a system-wide scale

As I have argued in the preceding chapters, the Ubuntu hackers engage in a community of practice to develop the Ubuntu system. They are brought together by their own personal motivations in a shared domain of interest in and commitment to developing software to be freely shared in a joint enterprise towards world domination. In this domain, the hackers build relationships that enable them to learn from each other by collaborating, discussing, sharing their work, and helping each other both on-line and in-person, expressing a mutual engagement in their shared practices. Through these shared interests and practices revolving around the development of free software, the hackers come to share experiences, stories, tools, and ways of addressing recurring problems through their daily use and customization of the Ubuntu system – the result of their work and a shared repertoire of the means of their community of practice.

Yet as much as the Ubuntu hackers come to trust and depend upon their own configuration of the Ubuntu system in order to dwell and build a system that works for themselves, they must also come to trust each other's abilities and motivations in order to collaborate on the grander scale of building and maintaining the entire Ubuntu system.

In this penultimate chapter, I examine how the diverse small-scale collaboration based on the interests, motivations and needs of the individual hackers that I have examined so far, scales to the development of such an immensely complex system as Ubuntu: How do the Ubuntu hackers negotiate and take responsibility for their shared work on the Ubuntu system?

Each Ubuntu developer commits their changes to the system directly to the central archive of the Ubuntu system for all other developers and users to download, and I use Matt Elliott's concept of *stigmergic collaboration* to explore how the individual Ubuntu hackers are empowered to maintain and

extend the system in this way. Following this, I explore the processes through which Ubuntu hackers seek to build and maintain what Etienne Wenger calls a *mutual accountability* (Wenger 1998:81-82) through reciprocal trust built through both technical and social processes, in order ensure effective collaboration and attenuate the inherent anonymity of the on-line interaction.[35] Building on Christopher Kelty's work on reputation and trust within on-line communities as well as Marshall Sahlins' ethnography of Melanesian *big-men*, I argue that the handling of this communal trust is central to understanding the principally stated meritocracy, which characterise the political and social environment of the Ubuntu hackers. And that this communal trust is central to their maintaining the continuous positive development of the system, as it is through these technical means of ensuring trust that the Ubuntu hackers connect their individual use of the system to that of the wider community of users and co-developers.

Becoming an Ubuntu developer

As noted in chapter 2, all of the software packages in the Ubuntu system can be changed by any of the approved Ubuntu developers, unlike the Debian system where each package is generally maintained and owned by one developer who has near complete control and authority over that package. At first, the Ubuntu developers instituted this free-for-all system because there were so few Ubuntu developers to maintain so many packages that it simply wouldn't be feasible to discuss and approve all changes beforehand.
This means that, taking into account the various freezes that come into effect during the development cycle, the individual Ubuntu developer is not

[35] New media researcher Carolyn Miller remarks with regards to sociality on-line that "a rational social world is possible only with an irrational, presumptive trust" (Miller 2002: 272). This trust is easily broken due to the lack of visual clues and rapport of the on-line textual environment which forces all interaction to be typed out, even when such a discussion might be unnecessary. As Ubuntu hacker Michael explains: "If you say, 'Hey, I like those Nazi guys,' in a room, everybody will lift their eyebrows, turn backs and physically ignore that person. You can't do that in an electronic community. There people cannot use physical hints or clues. They have to take the full discussion." This lack of physical rapport along with the asynchronicity of hackers' preferred means of communication, as described in chapter 3, makes it difficult to build trust using the normal means of physical rapport.

restricted in his ability to upload changes to the Ubuntu system, apart from having the technical privileges to commit changes to the system in the first place. These privileges are granted by the Ubuntu Technical Board which consists of some of the most experienced hackers in the Ubuntu community. The process of gaining upload access to the Ubuntu repositories is markedly different than the corresponding process in Debian. While both processes seek to guarantee the abilities and trustworthiness of the hacker to be approved, the *Debian New Maintainer process* focuses intensely on testing the individual qualities and abilities – technical, legal as well as ideological – of the hacker seeking approval as to ensure that the approved hacker is of equal or comparable ability to the other Debian Developers. It is often a long and arduous undertaking (cf. Coleman 2005:373-387).

The process to become a new Ubuntu developer with rights to upload changes to the central archive is less formal. First, new contributors need to apply to become Ubuntu members through another governance body called the Community Council, also led by Mark Shuttleworth.[36] Ubuntu Membership can be granted to any contributor who has contributed a substantial amount of work within the community. This work does not have to be technical, but can be helping users on-line, creating artwork, writing and editing documentation, doing grassroots marketing and so forth.

As I argued in chapter 5, each user's dwelling within the Ubuntu system opens for deeper participation in the community of practice around the system. This is what Etienne Wenger calls a *legitimate peripheral participation* where newcomers legitimately can participate in the community on their own terms to some degree (Wenger 1998:100), through peripheral activities such as reporting bugs or writing translations which do not require a deep understanding of the mutual engagement and shared repertoire of the

[36] The Community Council takes care of community-related issues such as the forming of new teams and the growing of a volunteer support community, as well as being the central instance of conflict resolution within the community. Unlike the Debian community, where only approved Debian Developers with access change the system are allowed to vote on community issues, all Ubuntu members have an equal vote, no matter whether their contribution is technical or non-technical. But since actual access to the system is granted through the Technical Board, I have chosen to focus on that process here, as that continues to define the Ubuntu system to a much greater degree than any other kind of contributions.

Ubuntu hackers' work.[37]

When new contributors begin to make technical contributions which result in changes to the central archive, these contributions are reviewed and sponsored for upload by approved developers, like when Michael approved and uploaded Sebastian's work on Synaptic. If they continue to do good work, and their skill and experience increases, the sponsors may suggest that they apply to become official Ubuntu developers, or they can simply apply on their own. The application is the formal acknowledgement of the informal apprenticeship which the applicant has undertaken, participating in the shared history of learning the mutual engagement, joint enterprise, and shared repertoire within the Ubuntu community of practice.

The developer application process consists of the Technical Board interviewing the applicant on IRC, hearing testimonials from other developers and considering the applicant's contributions to the system. Based on this, as well as their own impressions of the social qualities of the hacker, the Technical Board vote on the application. As can be seen in this concluding moment from a Technical Board meeting where board members Matt Zimmerman (mdz) and Matthew Garrett (mjg59) consider and vote upon Ubuntu hacker Brandon Holtschaw's (imbrandon) application for Ubuntu core developer membership:

...		
17:03	**mjg59**	imbrandon: Are you happy with being able to upload stuff that could break everyone's systems?
17:04	**imbrandon**	mjg59, yup as i'm confident that i can NOT break everyones system
17:04	**mdz**	but I'm sure you understand that this is a great responsibility and it's important that we establish whether you understand that
17:04	**imbrandon**	we're all human but i do tend to check very well and do ask for opinions when its code i'm not familiar with
17:04	**imbrandon**	definately

[37] It was by participating in such peripheral activities that I began contributing to the Ubuntu system. Not only did it give me a way to interact socially with other members of the community, but it also made it possible for me to learn many of the technical terms and issues relevant for those discussions.

17:05	**mdz**	mjg59: I'm ready for a vote when you are
17:05	**Lure**	I can say that imbrandon typically publish his work on his server for review, particularly for larger changes (kdebase...) so I can say he is very conservative in this part
17:05	**mjg59**	mdz: Sure, go head
17:07	**mdz**	+1 from me. please remember to take this responsibliity very seriously, and be conservative with regard to the release process. when in doubt, ask. even if you don't doubt, asking for confirmation never hurts!
17:07	**mjg59**	I'm inclined to go with +1, for working closely with upstream and seeming to have a good sense of the responsibilities
17:07	**imbrandon**	;)
17:07	**imbrandon**	mdz, definately
17:07	**mdz**	imbrandon: congratulations and welcome

Rather than focusing on the specific technical abilities of the applicant, they look to make sure that he has an active interest in the packages he gets access to modify, as well as being both technically and socially capable to use that access: only making changes which he can account for, and being willing to ask for second opinions when unsure and to help others in similar situations – in short respecting the mutual engagement and accountability of the Ubuntu hackers' collaborative work. Once a hacker has been approved as an Ubuntu core developer, he is granted access to change the packages of the Ubuntu archive – and thereby to make or break the system for everybody else updating their system from that archive.

With so many developers involved in maintaining and extending Ubuntu directly, and so many packages being changed all the time, direct communication before modifying a package is rare. There are recognised areas of expertise where other developers rarely tread, but if a change to one developer's special interest packages is necessary in order to make some other change go through, and that developer isn't around to approve of it, it is fair game to change it without their consent, as they can always revert it later. This, of course, only applies to the development version of Ubuntu. Updates to the stable versions of the system require a period of thorough testing and peer-review before they are approved. Though it is technically possible for

any approved Ubuntu developer to upload a change to a stable version without prior review, it is almost solely the Canonical-employed developers who handle such stable version updates.

Building digital trust

This mode of system-wide collaboration on the integration of packages is markedly different than the close, one-on-one development collaboration between Michael and Sebastian as described in chapter 3. Thus, in order to secure the mutual trust and accountability necessary for developers, who may or may not have met each other in person, to be able to change and affect each other's systems, a central part of becoming a Debian or Ubuntu developer is having the requirement for an applicant to meet an approved developer in-person so that he can verify the applicant's identity and digitally sign his personal *PGP key* – a digital cryptographic key for signing and encrypting data, thus associating it with the official identity of the developer. I asked Ubuntu developer Martin what it means to sign a key:

> **Andreas:** So how well do you know these people that you've signed keys with?
> **Martin:** Mostly not at all. That's not the important thing about keysigning. For keysigning, the important thing is that whenever you sign someone else's key that you have to be absolutely sure that the name that is stated on the key belongs to the person that you have exchanged your identity cards with and so on. So it doesn't mean at all that you have to trust this person, you just have to trust this key.
> **Andreas:** That's curious.
> **Martin:** Yeah, this is why you can be relatively liberal in who you sign keys with. I mean, getting signatures from other people is of course fine, you can accept them happily, you just have to make sure not to screw up signing someone else's key. ... This is basically your network identity in such communities so it is very important to have those virtual passports.

In this way, trust, as created through the signing of PGP keys, does not mean trusting the person or the software they upload, but rather trusting that their on-line identity matches one that is officially authenticated, creating a purely technical verification of identity. In both the Debian and Ubuntu communities, it is considered extremely important that all of its members are connected in a *web of trust* in which each developer's identity has been verified by other

developers so that a path of trust can be found from any one member to any other in order to guarantee the identity, accountability, and responsibility of each developer involved.[38]

As Christopher Kelty notes, such a decentralised web of trust is remarkably totalising in that there are no strangers or middlemen possible in such a system: People with no key are just as untrustable as people who sign with untrusted keys (Kelty 2005c:139). Because of this, it is only once this technical trust has been established that social trust can be built by examining the work which a developer has signed with that key thereby taking credit and responsibility for it. Exceptions to this rule are developers who get their work reviewed and sponsored by already approved developers, such as the case with Michael reviewing and uploading Sebastian's work. But as noted in chapter 3, this depends on the personal communication between two developers, rather than the free access to change the system as granted with core developer membership.

[38] This works through the famous "six degrees of separation" model, where the hackers can use algorithms to prove that every developer has met and verified the identity of at least one other developer, who in turn has met at least one other developer, and so on, until every developer is connected in an intricate web of trust.

Illustration 13: A typical scene from a keysigning party. Each hacker brings his passport and a list of all the PGP keys he expects to sign at the signing. The hackers then divide up into two rows which file past one another, verifying in-person identities and matching them to the PGP keys as they go (picture taken by Noirin Plunkett at ApacheCon 2006. Available at http://flickr.com/photos/noirin/177599802/)

Though individual key-signings occur, and some hackers indeed collect key signatures much like how others would collect autographs, most key signings take place at big key signing parties at conferences such as the Ubuntu Developer Summits described in chapter 3 (cf. Illustration 13). Therefore,

signing PGP keys is a central point of overlap between the hackers' in-person and on-line identities where these identities are ritually and reciprocally matched and confirmed by the hackers present.

Communicating through changes

Whenever a Ubuntu hacker uploads a change to the system, he notes the change in the system *change log* – a central listing in which the developers register all of the changes to the system – and signs it with their individual PGP key to associate that change with their on-line identity. A typical changelog entry reads:

```
Accepted:
bluez-utils 3.7-1ubuntu4 was ACCEPTED.
        Component: main Section: admin

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

Format: 1.7
Date: Fri, 20 Oct 2006 17:47:23 +0200
Source: bluez-utils
Binary: bluez-pcmcia-support bluetooth bluez-cups bluez-utils
Architecture: source
Version: 3.7-1ubuntu4
Distribution: edgy
Urgency: low
Maintainer: Debian Bluetooth Maintainers <pkg-bluetooth-
maintainers at lists.alioth.debian.org>
Changed-By: Daniel Holbach <daniel.holbach at ubuntu.com>
Description:
 bluetooth  - Bluetooth stack utilities
 bluez-cups - Bluetooth printer driver for CUPS
 bluez-pcmcia-support - PCMCIA support files for BlueZ 2.0
Bluetooth tools
 bluez-utils - Bluetooth tools and daemons
Changes:
 bluez-utils (3.7-1ubuntu4) edgy; urgency=low
   * debian/bluez-utils.bluetooth.init:
     - pass -- $HCID_OPTIONS to 'restart' too, not only to
'start' (Malone:
       #67169), thanks James Henstridge <james.henstridge at
canonical.com>             for finding out.
Files:
 c3e11b4ae9b41c3942d32ce25f294c67 873 admin optional bluez-
utils_3.7-1ubuntu4.dsc
 9eb29248da63a33f80e38fda8e725bb6 24705 admin optional bluez-
utils_3.7-1ubuntu4.diff.gz

-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.3 (GNU/Linux)
```

```
iD8DBQFFOPBYr3O2CKlAUK8RAoUwAJ9ZokQCufA38l2OaZb2BINh41aRNACeI
/oi
tw1a8E/W+GyK+fITYh8SRDU==UUOu
-----END PGP SIGNATURE-----
```

In this mess of contexual data showing the exact version of the package and the files being altered, the main message is that the change has been accepted into the system archive. Acceptance is given based on the fact that the upload is associated with the name of a trusted Ubuntu developer (cf. the final lines labelled “PGP Signature”), who thus takes personal responsibility for the change, which is described under the field “Changes” containing a review of what necessitated a new upload. In this case, the change was to fix a bug, with the relevant bug registration number in the bug tracker included for reference. It also acknowledges the hacker who first called attention to the issue who otherwise wouldn’t receive credit for his work as it is incorporated in an upload signed by another developer. Thus, at this system-wide level, the Ubuntu hackers communicate their work by changing the system directly, thereby letting their actions and changes to the system tell of themselves. This corresponds well with the way the Ubuntu hackers' discussions on IRC and mailing lists generally focus on *how* to do things – the asking for second opinions and review mentioned above – rather than on *which* things should be done, as I showed in my discussion of the hackers' mutual engagement in chapter 3.

The new media theorist Mark Elliott calls this kind of collaboration "Stigmergic collaboration", borrowing a term from zoologist Pierre-Paul Grassé who first coined the term in the 1950s to describe the way termites interact by modifying their nest environment and reacting to such changes (Elliott 2006:1).

Elliott argues that human stigmergic collaboration has become possible through digital means, as it makes it possible to sidestep social negotiation, making it instantly possible for hundreds of developers to contribute and collaborate without having to get acquainted and maintain relationships with one another (Ibid.3). Elliott uses the on-line user-edited encyclopedia *Wikipedia* as his main case study for stigmergic collaboration, and it fits very well with his description as it allows all users to edit any article as they please, since it is easily possible to revert any change to a previous version as

needed. In the case of Ubuntu, I find that it is perfectly possible for any user to write bug reports and translation suggestions without any prior involvement or social negotiation, but since changing the system carries the potential risk of breaking all of the inter-connected Ubuntu systems – as opposed to merely displaying incorrect information in the case of Wikipedia – each Ubuntu developer needs to have acquired the formal trust of the community through direct approval from the technical board in order to be able to actively change the system.
Thus, stigmergic development is driven by an awareness that every change you upload affects the entire system. Not only an immense responsibility, it is a very empowering position, as it allows you to fix a problem that has been bothering you, not only for yourself, but for everybody else who might encounter the same problem. As Ubuntu hacker Martin explains:

> And I think that this thing [maintaining a high-profile software package in Debian and Ubuntu] mainly [...] taught me that people actually rely on the work I do. So whenever I screwed up and did a bug in the new version and then people who upgraded this said "hey, you just broke my installation" and so on. This really got me the feeling that everything I do is actually used outside there.... and it is important and people are relying on it...

This indirect social connection through the use and development of the system is central to Ubuntu hacker sociality. Since the system is shared and developed among several hundred hackers with access to change it, each of them come to depend on and trust that the developers are responsible enough only to make beneficial and well-considered changes to the packages they use.
In this way, the Ubuntu hackers maintain a communal regime of mutual accountability, aware of the responsibility they hold, as well as ready to let others know when their changes break each other's carefully configured system. As Ubuntu hacker Paul succinctly puts it, "You get a good reputation by doing stuff without fucking up."
Since the hackers' collaboration lacks any more telling and personal ways of

displaying ability and status, trust and reputation is built from the changelogs, associated with that developer's name and confirmed through his digital signature. In this way, it is the hacker's on-line identity itself, verified, approved and directly associated with all his work on-line, which becomes the focus of his on-line reputation.
As Eric Raymond has noted, one of the few taboos in the free software communities is altering change logs and taking credit for other people's work (Raymond 1998), much like how scientists are expected to use citations to acknowledge prior work in their field. Indeed, Christopher Kelty has compared the number of times scientists are cited in the Science Citation Index as a marker of peer recognition similar to hackers' *greputation* – that is, the number of times their name appears in the changelogs of a given project when searched for and filtered through the command line tool called *grep* (Kelty 2005b). Though somewhat simplistic, the amount of recognised and approved work one has done, as well as how willing others are to base their own work upon it, is at least to some degree an indication of the level of trust one's peers, whether hackers or scientists, extend to you.[39]

Breaking the users' trust

This trustful stigmergic collaboration does breakdown from time to time, as the developers upload changes which introduce new bugs to the system. But since most of such disruptive changes occur in the development version of Ubuntu where such disruptions is expected, it does not affect the main body of users who use the stable versions of Ubuntu. During my fieldwork, I witnessed only one instance of a hacker uploading a disruptive change to one of the stable versions of Ubuntu, which broke the X Window System, the software driving Ubuntu's graphical user interface, for the thousands of users who installed that update during the 17 hours that it was in the archive, leaving them with nothing but the command line and its unforgiving demand for technical knowledge. The reaction to this lapse in trust was an immediate

[39] Whereas Raymond uses this discussion to argue that free software communities are gift economies focused on the exchange and maximising of personal reputation rather than any monetary value, Kelty refutes this by arguing that though both scientific citations and greputation can be considered values of reputation, neither citations nor changelog entries are necessarily positive. The change or citation is registered even if people use it to refute your scientific argument or revert your changes.

disappointed outrage among Ubuntu users on web forums, blogs and mailing lists, including many stories such as this:

> You want to know why I'm so angry at this? I just spent all last weekend installing a fresh copy and configuring ubuntu to how I like it. Everything was going great. I was grinning from ear to ear. Then this morning, the update light comes on […] I installed it as i'm sure most would only to reboot this afternoon and find a pear-shaped xserver-xorg [the software package containing the core elements of the X system].
> This is one of the reasons I'm so angry at this. The other is how could "they" let this massive BALLS UP go live...?

These users invested their trust in the stable Ubuntu system, adopting and customising it in the same manner as the Ubuntu developers. But unlike the Ubuntu developers, they did not have the technical skills to fix the issue themselves, nor had they expected to be forced to, and the result was a deep sense of broken trust which was frustrated even more by the fact that few of the users knew the means through which the Ubuntu developers build and maintain their reputation, and thus realised that they had no concrete group to blame or direct their complaints at, only a vague, faceless “they.”
By chance, the upload took place while I was at a week's development sprint with the Canonical-employed Ubuntu developers in Wiesbaden, making it possible for me to see the flustered faces of the developers as they realized the mistake. And though tension was high, it was soon deflated as the hackers sat together and discussed the matter in-person. They found that it had been a comedy of errors which had led to the incident: A bad update, a bad review of that update before uploading it, a bad decision not to use the test infrastructure available, and a problem with the archive which let the update stay on-line for 17 hours before it could be replaced with a better version. Working quickly under the watchful eye of their employer, the hackers soon had the update removed as well as instructions on to how fix the system and the relevant apologies and promises posted on the Ubuntu website for all users to read.
That work done, Mark Shuttleworth asked the hackers to close their laptops, thus demanding their full attention, in order to discuss the lessons they were to learn from this incident. He reminded them of the great responsibility under which they work, as their work affects millions of users around the world.

Stating that “this is about becoming professional”, he underlined how their work is no longer some part-time hobby project, but an ambitious project that the users need to be able to trust. Following the meeting he communicated the same message from his much publicised personal blog under the headline “One on the chin”:

> This gave us the opportunity not just to analyse and fix the issue, and to talk about the sequence of events that led to the problem, but also to discuss the processes we must improve to further reduce the likelihood of a repeat. The team is now more aware than ever of the responsibility we assume given the extraordinary rate of adoption of Ubuntu.
>
> My goal is for the team to grow and learn from this experience without becoming paralyzed on future updates. We can’t afford to take risks with our users’ trust, but I balance that with the need to continue to improve the desktop.
> (Shuttleworth 2006)

With this, Shuttleworth strikes upon a central theme in how the Ubuntu system links the relatively few, highly capable Ubuntu hackers to the millions of Ubuntu users: That the formalised mutual trust built among the small group of core developers extends well beyond the people contributing actively to Ubuntu, to all of those users who also dwell within and depend upon the Ubuntu system.

As I argued in chapter 4, a central aspect of the developers’ personal motivation to work on Ubuntu is to connect socially with the users of their software: to win appreciation for the work they do, to be inspired to new features for the software and solve bugs, and to spread the dogmas of free software. This collaborative work is built through the shared use of the Ubuntu system which makes it easily possible for users to shape the system, should they take an active interest in learning how to do so. In this way, all of the work within the eco-system of Ubuntu depends on maintaining this extended trust. Not merely by building a system that works just for the individual hacker, but to keep the entire Ubuntu system usable for everybody in order to win over new users.

As the leader of the project, it falls to Mark Shuttleworth to take responsibility and guarantee the mutual accountability through which this

trust is built, and to be the public face of the community of Ubuntu developers in crises such as the one described here.

Hacker big-man governance and meritocracy

As the Norwegian anthropologist Lars Risan has noted, the kind of political authority that free software project leaders come to possess is remarkably similar to the authority of Melanesian big-men as described by Marshall Sahlins (Risan unpublished, Sahlins 1963). The authority of the Melanesian big-man is personal, it is not a political position into which one can be elected. Rather, it is attained "through a series of acts which elevate a person above the common herd and attract about him a coterie of loyal, lesser men" (Sahlins 1963:289). Such acts can be shows of skill - of magical powers, of oratorical style, of bravery in war – but perhaps most decisively is his ability to amass and distribute goods among his followers, winning their trust through selfless reciprocity (ibid. 291). Similarly, free software big-men such as Linus Torvalds attract followers by freely giving away their work,

garnering respect and building digital *greputation* trust through the quality of their efforts.
Typically, such big-men are the founders of new projects that draw in contributors with social values of their work, and the scope of the vision that drives them. Noting this, it is remarkable how often the personality and motivations of these big-men come to define, and even to some degree personify, the projects they lead.[40]
This is very much the case with Mark Shuttleworth and the Ubuntu project, which he founded and funded from the beginning. His role as leader of the project is clearly stated on the Ubuntu homepage describing the details of Ubuntu governance:

> This is not a democracy, it's a meritocracy. We try to operate more on consensus than on votes, seeking agreement from the people who will have to do the work. Mark Shuttleworth, as SABDFL [Self-Appointed Benevolent Dictator For Life], plays a happily undemocratic role as sponsor of the project. He has the ability, with regard to Canonical

[40] The Big-men leaders even represent the motivations of the contributors themselves to some degree. For instance, comparing to the three main motivational factors described in chapter 4, the Linux kernel project led and managed by Linus Torvalds is still dominated by his initial motivation of sharing an exciting technical challenge with other hackers. The GNU project is still dominated by Richard Stallman's unrelenting ethical dedication to the dogmas of free software. And the *qmail* and *djbdns* software projects are still led, developed and maintained solely by hacker and computer scientist Daniel J. Bernstein whose insistence on maintaining the high quality of his work has led him to license his software under terms that do not allow nobody else to distribute modified versions of his code, winning him not contributors to his project, but rather fans of his style.
It is worth noting that a handful of central free software projects such as Apache, FreeBSD, and Debian do not have these types of charismatic, big-men leaders but have vested power in various modes of community governance such as councils, annual elections or boards.

> employees, to ask people to work on specific projects, specific feature goals, and specific bugs.
>
> [...] In many cases, there is no one "right" answer, and what is needed is a decision more than a debate. The SABDFL should act to provide clear leadership on difficult issues, and set the pace for the project.
>
> It is understood that the divisive use of the SABDFL's authority could weaken the project. For that reason the authority is used carefully, in the hope that it will create momentum in the best direction for the project, breaking stalemates where otherwise competing views would fail to reach consensus.

In officially defining the Ubuntu community as a meritocracy based on each individual developers' contribution to the project, Mark Shuttleworth has opened the project to community involvement, but he has also secured his own power base as project leader by employing 25 highly capable core Ubuntu developers to work full-time on the project, investing much more time in shaping the system and winning more influence than any group of hobbyist contributors.[41] And while the employed Ubuntu developers certainly still have strong personal interests in their free software work, they have essentially sold their work and thus their political influence on the project to their employer. And since they would have to quit their jobs in order to shape the project directly against Shuttleworth's instructions, they are limited to the try to persuade him to change his mind.[42]

In this way, Shuttleworth has found a new way of winning influence by using his wealth to build his authority within Ubuntu rather than through a reputation built through the quality of his work, which is markedly different from most other free software projects. For though his hacking credentials are generally accepted, Shuttleworth rarely, if ever, contributes code to Ubuntu, instead preferring a role as "benevolent dictator", delegating tasks and

[41] This is supported by my quantitative on-line survey and my general impressions from my in-person fieldwork that many of the most active free software developers are students, academics or paid developers. as these are the people who are in a position to devote much time to free software development, but also to take the time to travel to hacker conferences around the world to socialize and build the in-person social relationships which are so crucial in the long-term development of free software.

[42] Several of the Canonical-employed Ubuntu developers humorously called this practice for "Mark-steering" – that is the subtle hinting and leading the SABDFL to certain conclusions based on their own ideas and their knowledge of his interests.

representing the project publically. He seeks to confirm his decisions as often as possible through the Technical Board and the Community Council, the two main governance bodies of the Ubuntu project, the members of which are among the most respected and well-reputed Ubuntu hackers who act as a sort of tribal elders, helping to define the technical and communal direction of the project through Shuttleworth's reciprocal authority (rather than his authority as an employer, even though most of the initial members of both bodies were employed by Canonical).

But as the crisis case described above shows, Mark Shuttleworth's leadership of the Ubuntu community is not merely dependent on his using his wealth to employ developers, sponsoring contributors to come to conferences, and send free CDs to anyone interested. It is just as much dependent on his ability to represent a coherent ethos as a benevolent, responsible big-man, which the members of the Ubuntu community – full-time employees and hobbyists alike – can trust and approve of, and which the millions of users of the Ubuntu system feel that they can depend upon. They all look to him to ensure an ambitious vision is planned and realised for the project, one which they can approve of, and which they feel they can help shape – in which regard he is exactly like other free software big-men.

Much like the Melanesian big-men described by Sahlins, Shuttleworth wins the trust and approval of the bigger community of users by offering them a system that is both *gratis* and *libre.* And for which he is willing to take responsibility and reciprocate the trust of the contributors by constantly reaffirming their freedom to use and contribute to the system as they please. By letting the Ubuntu hobbyist hackers stigmergically shape the system, deriving from the core Ubuntu system as they see fit to make new versions specialised for their needs (naturally, the employed hackers are not given similar freedom) and following the suggestions of the community's most respected hackers, Shuttleworth embraces the reciprocity through which he came into this position of authority in the first place, building trust in his ethos and his long-term vision of the world domination of free software.

Conclusions

In this chapter, I have explored how the Ubuntu hackers negotiate and take responsibility for their shared work on a system-wide scale, spanning all of

the more than 19.000 software packages that make up the detailed complexity of the Ubuntu system. Using Matt Elliott's concept of stigmergic collaboration I have argued that the Ubuntu developers rarely coordinate their changes to the system beforehand, but rather communicate directly by changing the system, which the hackers dwelling within that version of the Ubuntu system will experience directly as a change in their environment. This empowered collaboration depends on cultivation of a mutual accountability through a reciprocal trust built and reinforced through interdependent layers of decentralised and centralised means: Decentralised in the individual, yet shared use and dwelling within the system, the web-of-trust verification of identity through PGP key cryptography, the formal legal trust in the GPL license which allows each hacker to modify and control his system in every detail, and by extension the open, stigmergic collaboration through which the hackers can build their personal reputation, as has been described by Christopher Kelty. Trust in the project is centralised around the reciprocal governance of Mark Shuttleworth and the respected and well-reputed hackers of the Community Council and the Technical Board in a manner similar to Marshall Sahlins' description of the rule of Melanesian *big-men*. It is these leaders who grant new Ubuntu developers access to change the system based on an experienced examination of each hackers' ability and judgment; and who determine the goals and deadlines for each new release at the in-person conferences, which allow the Ubuntu hackers to create social ties of greater depth than is possible on-line. All of these goals are summed up in the ethos and vision of Mark Shuttleworth, inspiring the project's direction and seeking to match and enroll the interests of each individual Ubuntu contributor in the project. All of these means of trust are connected through the Ubuntu hackers' shared history of learning to dwell and continually build and trust the Ubuntu system itself – practices which are at the core of their community.

I argue that it is through this legitimate, yet at first rarely intentional, participation at the periphery of the Ubuntu community of practice, that new contributors find their way into that shared history of learning. As Etienne Wenger puts it, "we create ways of participating in a practice in the very process of contributing to making that practice what it is" (Wenger 1998:96). It is in the winning and maintaining of this reciprocal trust between users and developers that the complexity of the Ubuntu system is managed and tested in

times of crisis, and which in turn comes to shape the way the system is developed and used.

Conclusion

Unfolding the Ubuntu system

A hacker folklore dictum has it that an organization building a computer system is bound to produce designs resembling the organization itself, since the way the different parts of the system interact with one another reflects the social relations between the developers working on those different parts. This is known as *Conway's Law*, named after the programmer who first made the observation (Conway 1968).

Here I have sought to *unfold* the transparent *white box* of the Ubuntu system by exploring how the Ubuntu hackers relate to it in their individual and collaborative day-to-day practices, and I find myself agreeing with Conway's more than almost 40-year-old second-order observation.

With its more than 19.000 modular software packages integrating the work of thousands of free software projects within the same system, the Ubuntu system reflects how the Ubuntu hackers distributed across world can collaborate according to their interests and abilities in a community of practice centered around their use and configuration of the system – both as a shared goal and as a work environment of their own: With each new version of Ubuntu, every part of the system is revisited and challenged anew in the continuing work to maintain the system. Thus, the Ubuntu system remains unfinished – an unsealed box which has been developed through 40 years "oral" academic hacker tradition based on the Unix operating system. It is built to meet the constant technical challenge of every hacker who happens upon it. A city continuously being rebuilt, an oak tree sprouting new branches. All to meet the ever-changing needs of its inhabitants. This cumulatively improved Ubuntu system is part of a lively interdependent "eco-system" built on the reciprocal sharing of knowledge and source code which through its open on-line development and cleverly crafted copyright licenses has become a viable

Illustration 14: Unfolding the white box of the Ubuntu system to find it full of traces of shared practice: Discussions, collaborations, tools, trust, old disagreements, new goals, traditions and social occasions. These practices come to shape the source code of the system itself through the Ubuntu hackers' daily collaboration and continuous interdependence.

alternative to the mainstream IT industry. A fact which the Ubuntu hackers themselves have been among the first to recognise through their ambitious goal of breaking Microsoft's monopoly on computer operating systems and winning *World Domination* for free software.

The Ubuntu community of practice

By following Bruno Latour's methodology of *unfolding* the network of practices, processes and actors through which the Ubuntu system has been constructed, I have sought to break away from the focus on the political, ethical and economic implications of free software development which has drawn the attention of most of the anthropologists working in the field, and instead explore the everyday work and socialising of a fairly well-defined grouping of free software hackers over a longer fieldwork period. Based on this fieldwork, I have inductively posited the Ubuntu hackers' collaborative practices and relations to the Ubuntu system as a community of practice sharing the three main dimensions of relation through which Etienne Wenger suggests that a community of practice is defined: Joint enterprise, mutual engagement and shared repertoire.

Each hacker has different personal motivations for their involvement in the joint enterprise for world domination, most of these tending to be centered around the fascination of learning the details of technology, enjoying working together on technical challenges, and sharing an ethical commitment to write software that can be shared freely for the benefit of others. Following Alfred Gell's theories, I have argued that these diverse interests are reflected in every little part of the system and results in constant negotiations of the details of how to realise their joint enterprise of furthering free software.

They work towards this goal by building a mutual engagement in their complementary practice between what Lévi-Strauss calls *engineer work* and *bricolage*: Both visualising and designing new features as well as redesigning and reimplementing odd bits of left-behind work to fit their needs through a haphazard extending, fixing, fussing and testing. This engagement enables them to learn from on another by collaborating, discussing, sharing their work, and helping each other both on-line and in-person.

customization of the Ubuntu system – the shared built environment, which they adopt as their own in which to *dwell* - in Tim Ingold's terminology. Through this dwelling the system affords the hackers' work to such an extent that it becomes an extension of the hacker's mind, as suggested by Clark & Chalmers, thus becoming the all-encompassing means for their hacking work. The hackers communicate by changing their shared environment directly, in what Matt Elliott calls a *stigmergic* fashion, continuously building the system, *sedimenting* and stabilising it for others to use as well. This empowered collaboration rests on the cultivation of a *mutual accountability* through a reciprocal trust built and reinforced both through the individual, yet shared dwelling within the system, the hackers' personal on-line reputations built through technical and cryptographic means, and in the reciprocal governance of the respected and well-reputed hackers leading the project in a manner similar to Marshall Sahlins' description of the rule of Melanesian *big-men*. This community of practice, and the mutual trust which holds it together, is built through the hackers' shared history of learning to dwell and build within Ubuntu system, making it their own, and through coming to trust and collaborate with the other users dwelling there, keeping the door open for new contributors to enter and learn among them. In this way, the Ubuntu hackers come to share more than just the same system, they come to share a common history, and to a certain extent, a shared identity through their work. As Etienne Wenger concludes, "Communities of practice are more than purely instrumental purposes. They are about knowing, but also about being together, living meaningfully, developing a satisfying identity, and being altogether human" (Wenger 1998:134).

In this way, Conway's insight applies doubly to the relationship between the Ubuntu system and the developers working on it: Not only does the social organization of the Ubuntu hackers reflect the technical structure of the system, in turn, the system is shaping the practices and identity of the hackers who come to adopt and work on it through the reciprocal trust and passion for technology with which they continue to build it. Thus, when the Ubuntu individual Ubuntu hacker works to build "a system that works for me," his work is reflected not only in his own system, but in the system as a whole, as well as the shared practices through which it comes to be.

Implications of theories and methods on the analysis

I came to take part in this shared history of learning from the moment I first loaded the Ubuntu system on my computer. At first by using the system and learning to configure it to my needs, I came to dwell there in a manner somewhat similar to the hackers I interviewed. In this dwelling and learning, along with my active participation in contributing to the system, I took part in the core practices of the Ubuntu community.

I have sought to use my own experiences with the system as a basis to make sense of my discussions and interpretations of the Ubuntu hackers'own, more sophisticated, work with the system. The central part of my fieldwork data consists of the correlation of these two perspectives that form a coherent picture of the sum of relations extending from the Ubuntu system. In inductively building on these observations to pair the Ubuntu hackers' practices with Etienne Wenger's theoretical framework, I have come to rely on these correlations for my analysis. Yet they reflect to a much greater extent the Ubuntu hackers' mastery of the system that is so central to their practice than my learning experience of coming into that community. I haven't fully touched upon my initial frustrations at being unable to appreciate the technical details of the hacking work and my miscomprehensions of the nature of the Ubuntu hackers' online collaborations, which were only slowly dispelled as I learned more of the system and the hackers' shared practices. It wasn't until I was able to meet the hackers in person at the Ubuntu Developer Summit, after two months of fieldwork that I finally began to feel that I could connect socially with the Ubuntu hackers.

In building my argument around Wenger's somewhat totalising framework, I have limited my analysis to the Ubuntu hackers' collaborative practices, making it impossible for me to delve further into how power is attained and bigger conflicts are resolved through on-line political discussion and argumentation through technology, as has been explored by Gabriella Coleman and Christopher Kelty in other ethnographic contexts. Furthermore, by positing the Ubuntu community as a community of practice, I have focused on the social aspect of the Ubuntu hackers' practices, in part because the social interactions between the hackers were easier to examine and discuss with them. But even so, individual learning remains a big part of coming into the Ubuntu community, as there are many texts to be read, and skills to be

learned on your own before you can begin to contribute. And even though there are people willing to help, the gap left by the on-line lack of tactile "showing how" to do a task can only be bridged by individual ardour.

Perspectives for wider application and future research

It is my impression that many other free software communities share many of the characteristics of the Ubuntu hackers' practices I have described. But because of my inductive approach, I cannot generalise on this, though it may indeed turn out that the community of practice around the Ubuntu community is part of what information theorists Paul Duguid and John Seely Brown, extending on Wenger's concept, call a "network of practice" around the open development of free software. Such networks consist of more or less disconnected communities which all share a similar practice, but which to some extent are marked by different goals and histories, and shared repertoires of tools and routines (Seely Brown & Duguid 2000:141).

If the free software communities indeed share similar collaborative practices to a greater extent, then many of my conclusions here can be tested in the many similar projects within the free software eco-system. As one hacker said, "we are your fruitflies" – perhaps this is a way to examine to what extent this statement is true.

The Ubuntu community of practice appears to be the kind of informal and empowered learning and creative space that offers what Eric von Hippel has called a "democratization of innovation" (von Hippel 2005) where the development of new ideas and designs is displaced from inside the individual companies to wider networks of interested and capable users collaborating for a common interest - in this case the shared system.

To further examine this, it would be necessary to focus further on the role of Canonical within the Ubuntu community, since the company plays a huge part in the development of Ubuntu – both in the building of the Launchpad infrastructure, the sponsoring of the Ubuntu Developer Summits, and the employment of many of the core Ubuntu developers. Canonical's distributed organization with developers in more than 20 countries is an attempt to synthesize the open model of development of the free software communities with the closed model of development of the mainstream IT industry to create a community that can harness the energy and passion of interested developers

- both employed and volunteer - working with a common interest in furthering the cumulative "idea treasure" of free software.
To me, the central discovery of this thesis has been uncovering to some extent the manner in which the Ubuntu hackers come to learn, work, and manage their knowledge on-line. For while the system indeed is guaranteed by copyright licenses and reciprocal leadership to remain a white box that is transparently open for adoption and modification, other barriers to membership in the community, such as class, gender and depth of technical knowledge still remain. Perhaps most of all, as Gabriella Coleman suggests, because no elegant technical solution to these problems exist (Coleman 2004:517-518).
As I have argued here, it is through opening the system to new users and engaging them in the joint enterprise of the community that they will come to dwell and learn within the system. Exploring the details of how this learning process comes about and works within the free software communities of practice such as Ubuntu and their specialized technical infrastructure will be a good choice for further study.

References cited

Juan-José Amor-Iglesias, Jesús González-Barahona, Gregorio Robles-Martínez & Israel Herráiz-Tabernero:

2005 "Measuring Libre Software Using Debian 3.1 (Sarge) as A Case Study: Preliminary Results " in Upgrade – The European Journal for the Informatics Professional, Vol. VI, Issue 3, June 2005.

Michael Banck:

2004 "The Ubuntu Development Model" (2004). accessed at http://www.advogato.org/person/mbanck/diary.html?start=24 on June 6th, 2007.

Maurice Black:

2002 *The Art of Code* - PhD. Dissertation, University of Pennsylvania 2002.

Frederick P. Brooks:

1995 *The Mythical Man-Month*. (Boston: Addison-Wesley 1995 [1975])

Manuel Castells:

2001 *The Internet Galaxy: reflections on the Internet, Business and Society*. (Oxford: Blackwell, 2001)

Andy Clark & David J. Chalmers:

1998 "The Extended Mind" in: *Analysis* 58, 1998. pp. 10-23.

Gabriella Coleman:

2004 "The Political Agnosticism of Free and Open Source Software and the Inadvertent Politics of Contrast" in: *Anthropological Quarterly* 77(3) Summer 2004.

2005 *The social construction of Freedom in Free and Open Source Software: Hackers, Ethics and the LiberalTradition*. PhD. Dissertation, University of Chicago 2005.

Melvin E. Conway:

April, 1968. Accessed at http://www.melconway.com/research/committees.html on June 20th, 2007.

Mihali Csikszentmihalyi:

2002 Flow – The Classic Work on how to achieve happiness. (London: Rider, 2002 [1992]).

Chris DiBona:

1999 *Open Sources: Voices from the Open Source Revolution.* (Sebastopol CA:O'Reilly 1999)

Matt Elliott:

2006 "Stigmergic Collaboration: The Evolution of Group Work" in: M/C: A Journal of Media and Culture, vol. 9, issue 2, May 2006.

Joseph Feller, Brian Fitzgerald, Scott A. Hissam, and Karim R. Lakhani:

2005 "Introduction" in: Joseph Feller, Brian Fitzgerald, Scott A. Hissam, and Karim R. Lakhani (eds.): *Perspectives on Free and Open Source Software*, pp. xvii-xxxi (Cambridge, MA: MIT Press, 2005)

Karl Fogel:

2005 *Producing Open Source Software*. (Sebastopol CA: O'Reilly, 2005).

Bill Gates:

1976 "Open Letter to Hobbyists" in: *Homebrew Computer Club Newsletter*, Volume 2, Issue 1, February 1976. Accessed at http://www.digibarn.com/collections/newsletters/homebrew/V2_01/ on June 11th, 2007.

Alfred Gell:

1999 "The Technology of Enchantment and the Enchantment of Technology" in *The Art of Anthropology: Essays and Diagrams* pp. 159-186 (London: Berg Publishers 1999)

Rishab Aiyer Ghosh:

2005a Measuring free software: Cooking-pot markets and balanced value flows, in: Rishab Ghosh (ed.): *Collaboration, Ownership and the Digital Economy* (Cambridge, MA: MIT

Press, 2005)

2005b "Understanding free software developers: Findings from the floss study" in: Joseph Feller, Brian Fitzgerald, Scott A. Hissam, and Karim R. Lakhani (eds.): *Perspectives on Free and Open Source Software*, pp. 23-46 (Cambridge, MA: MIT Press, 2005)

2006 "Sustaining a software ecosystem with FLOSS: skills and local economic growth" – Presentation of the results of the FLOSSPOLS (Free/Libre/Open Source Software: Policy Support) Project given at 7th Asia Open Source Symposium, Kuala Lumpur, Malaysia, March 7, 2006 accessed at http://flosspols.org/dissemination.php on June 17th, 2007.

Carlo Ginzburg:

1986 "Spor: Indicie-paradigmets rødder" in: *Kultur og Klasser* no. 54, pp. 6-48.

Paul Graham:

2004 *Hackers & Painters* (Sebastopol CA: O'Reilly, 2004).

Benjamin Mako Hill & Gabriella Coleman:

2004 "The Social Production of Ethics in Debian and Free Software Communities: Anthropological Lessons for Vocational Ethics." In Stefan Koch (ed).: *Free and Open Source Development (*Hershey, Penn: Idea Group. 2004)

Benjamin Mako Hill, Jono Bacon, Corey Burger, Jonathan Jesse & Ivan Krstić:

2006 *The Official Ubuntu Book* (London: Prentice Hall, 2006).

Pekka Himanen:

2001 *The Hacker Ethic and the Spirit of the Information Age.* (London: Vintage, Random House, 2001).

Eric von Hippel:

2005 Democratising Innovation (Cambridge, MA: MIT Press, 2005).

Douglas Hofstadter:

1979 *Gödel, Escher, Bach: an Eternal Golden Braid (New York:* Basic Books,1979).

Tim Ingold:

2000 *The perception of the environment: essays on livelihood, dwelling and skill*. London: Routledge, (2000) xvi, 454 pp.

The Jargon File (v.4.4.7 – updated October 2003) Accessed at http://www.catb.org/~esr/jargon/ on June 7th, 2007.

Alan Kay:

1984 "Computer Software" in: *Scientific American*, 251(3):41--47, September 1984.

Christopher Kelty:

2002 "Hau To Do Things With Words" – Paper accepted for publication by the JAI Press quarterly series *Knowledge and Society*, but omitted from the final publication. Accessed at http://www.kelty.org/or/papers/Kelty.Hautodothings.2002.rtf on June 5th, 2007.

2003 "Qualitative Research in the Age of the Algorithm: New Challenges in Cultural Anthropology", presentation given at RLG conference, May 2003. Transcription accessed at http://www.rlg.org/en/page.php?Page_ID=2201on June 7th, 2007.

2004 "Culture's Open Sources" and "Punt to Culture" in *Anthropological Quarterly* 77(3) Summer 2004.

2005a "Geeks, Recursive Publics, and Social Imaginaries" in *Cultural Anthropology* 20.2 Summer 2005.

2005b "Free science," in Joseph Feller, Brian Fitzgerald, Scott Hissam, Karim Lakhan (eds.): *Perspectives on Free and Open Source Software. (*Cambridge MA: MIT Press, 2005).

2005c "Trust among the Algorithms: ownership, identity and the collaborative stewardship of information," in Rishab Ayer Ghosh (ed.): *CODE: Collaborative Ownership in the Digital Economy (*Cambridge, MA: MIT Press. 2005).

2006 "Emacs. Grep and UNIX: Authorship, Invention and Translation in Software" - unpublished presentation given at

Case Western Reserve University, Cleveland, OH, April 2006.

Martin Krafft:

2005 *The Debian System—Concepts and Techniques* (Munich: Open Source Press, 2005)

Bernhard Krieger, Dawn Nafus & James Leach:

2006 "Gender: Integrated Report Findings" (2006). Report part of the Free/Libre/Open Source Software: Policy Support (FLOSSPOLS) Project at the Maastricht Economic and social Research and training centre on Innovation and Technology, accessed at http://flosspols.org/deliverables.php on June 17th, 2007.

Donald E. Knuth:

1992 "Computer Programming as an Art" in D.E. Knuth, *Literate Programming* (Stanford: Center for the Study of Language and Information, 1992), pp. 1-16.

Ko Kuwabara:

2000 "Linux: A Bazaar at the Edge of Chaos" in*: First Monday*, volume 5, number 3 (March 2000). Accessed at http://firstmonday.org/issues/issue5_3/kuwabara/index.html on June 6th, 2007.

Bruno Latour:

1987 *Science in Action* (Cambridge MA: Harvard University Press, 1987).

James Leach:

2005 "Modes of Creativity and the Register of Ownership" in: *CODE – Collaborative Ownership and the Digital Economy* (Cambridge, MA: MIT Press, 2005)

Lawrence Lessig:

1999 *Code and Other Laws of Cyberspace*. (New York: Basic Books. 1999)

2001 *The Future of Ideas: The Fate of the Commons in a Connected World*. (New York: Random House, 2001)

2004 *Free Culture: The Nature and Future of Creativity*. (New York: Penguin Press, 2004)

Timothy Lethbridge, Susan Elliott Sim & Janice Singer:

2005 "Studying Software Engineers: Data Collection Techniques for Software Field Studies" in *Empirical Software Engineering* Vol. 10, Issue 3 (July 2005) pp. 311 – 341.

Claude Lévi-Strauss:

1994 *Den vilde tanke* (København: Samlerens Bogklub, 1994 [1962])

Steven Levy:

1994 *Hackers – Heroes of the Computer Revolution* (New York: Penguin Books, 1994 [1984]).

Yuwei Lin:

2004 *Hacking Practices and Software Development: A Social Worlds Analysis of ICT Innovation and the Role of Free/Libre Open Source Software* . PhD. Dissertation, University of York, 2004.

Geert Lovink and Ned Rossiter:

2005 "Dawn of the Organized Networks" in *Fibreculture Journal (2005). Accessed at* http://journal.fibreculture.org/issue5/lovink_rossiter.html *on June 6th, 2007.*

Annette N. Markham:

1998 *Life online: researching real experience in virtual space* (Lanham, MD: Altamira Press. 1998)

Martin Michlmayr & Anthony Senyard:

2004 "How to Have a Successful Free Software Project" In: *Proceedings of the 11th Asia-Pacific Software Engineering Conference*, pp. 84-91, December 2004.

Carolyn R. Miller:

2002 "Writing in a Culture of Simulation: Ethos Online" in Patrick Coppock (ed.): *The Semiotics of Writing: Transdisciplinary Perspectives on the Technology of Writing* (Turnhout,

Belgium: Brepols Publishing, 2002), pp. 253-279.

Eben Moglen:

1999 "Anarchism Triumphant: Free Software and the Death of Copyright" in: *First Monday* volume 4, number 8 (1999), accessed at http://firstmonday.org/issues/issue4_10/bezroukov/index.html on June 6 2007.

Glyn Moody:

2001 *Rebel Code – Linux and the Open Source Revolution*. (London: Penguin Press, 2001)

Ole Møller Markussen:

2002 "The Humanly Mediated Computer Interview." in *Den Vilde Tanke* No. 24 (October 2002), pp. 2-4.

Siobhan O'Mahoney:

2002 *The Emergence of a New Commercial Actor: Community Managed Software Projects.* Ph.D. dissertation, Stanford University 2002.

Gregers Pedersen:

2006 "The gift of sharing" - paper disseminated in relation to presentation given at 23rd Chaos Communication Congress, Berlin 2006.

Matt Ratto:

2003 *The Pressure of Openness: the hybrid work of Linux Free/Open Source kernel developers* . Ph.D dissertation, University of California, San Diego 2003.

Eric S. Raymond:

1996 New Hacker's Dictionary (Eric Raymond ed.) (Cambridge MA: MIT Press, 1996) – see also Jargon File reference, above.

1997 "The Cathedral and the Bazaar" - Accessed at http://www.catb.org/~esr/writings/cathedral-bazaar/ on June 7th, 2007.

1998 "Homesteading the Noosphere" in: First Monday, volume 3, number 10 (1998) - Accessed at

http://www.firstmonday.org/issues/issue3_10/raymond/index.html on June 6th, 2007.

2004 *The Art of Unix Programming* (Boston: Addison-Wesley, 2004)

Joseph Reagle:

1999 "Why the Internet is good: community governance that works well." Unpublished paper accessed at http://cyber.law.harvard.edu/people/reagle/regulation-19990326.html on June 6th 2007.

2004 "Open content communities" in *M/C: A Journal of Media and Culture*, 7, July 2004. Accessed at http://journal.media-culture.org.au/0406/06_Reagle.rft.php on June 6th, 2007.

Patrice Riemens:

2002 "Some thoughts on the idea of "Hacker Culture"" in: *Cryptome* (June 3rd, 2002) – Accessed at http://cryptome.org/hacker-idea.htm On June 6th 2007.

Lars Risan:

Unpublished "Hackers produce more than software, they produce *hackers"* Paper disseminated on-line at http://folk.uio.no/lrisan/Linux/Identity_games/ - accessed there on June 6th 2007

2005 "Kodens sakramentale karisma — linuxsamfunnets gavekultur" - presentation at the *NETTVERK! SOSIALE – DIGITALE* seminar at Oslo University on April 18th, 2005.

Marshall Sahlins:

1963 "Poor Man, Rich Man, Big-Man, Chief: Political Types in Melanesia and Polynesia" in *Comparative Studies in Society and History, 5, 1963. pp. 285-303.*

1972 S*tone Age Economics* (Chicago: Aldine, 1972).

Peter Salus:

1994 *A Quarter Century of Unix* (Reading MA: Addison-Wesley, 1994)

John Seely Brown & Paul Duguid:

2000 The Social Life of Information (Boston MA: Harvard Business School Press, 2000)

Mark Shuttleworth:

2006 "One on the chin" – blog entry accessed at http://www.markshuttleworth.com/archives/54 on June 20th, 2007.

Richard Stallman:

2002 "On Hacking" - article accessed at http://www.stallman.org/articles/on-hacking.html on June 6th, 2007.

2005 "Copyright and Globalization in the Age of Computer Networks" in: Rishab Ghosh (ed.): *Collaboration, Ownership and the Digital Economy* (Cambridge, MA: MIT Press, 2005)

Neal Stephenson:

1999 *In the Beginning was the Command Line*. (New York: Avon Books, 1999).

Linda Stone:

2006 "Attention: The *Real* Aphrodisiac" - presentation given at the Emerging Technology Conference 2006, audio recording of the presentation accessed at http://www.itconversations.com/shows/detail739.html on June 6th, 2007.

Ilkka Tuomi:

2001 "Internet, Innovation, and Open Source: Actors in the Network" in: *First Monday*, volume 6, number 1 (January 2001), accessed at http://firstmonday.org/issues/issue6_1/tuomi/index.html on June 6th, 2007.

Sherry Turkle:

1984 *The Second Self – Computers and the Human Spirit* (New York: Simon & Schuster, 1984).

Ellen Ullman:

1995 "Out of Time: Reflections on the programming life" in: James Brook & Iain Boal (eds.): *Resisting the Virtual Life (San Francisco: City Lights Publishers, 1995),* pp. 131-144.

Steven Weber:

2004 *The Success of Open Source* (Cambridge: Harvard University Press, 2004)

Gerald M. Weinberg:

1971 *The Psychology of Computer Programming* (New York: Van Nostrand Reinhold, 1971).

Etienne Wenger:

1998 *Communities of Practice: Learning, Meaning, and Identity* (Cambridge: Cambridge University Press, 1998)