IPPOLITA

OPEN NON È FREE

COMUNITÀ DIGITALI
TRA ETICA HACKER E MERCATO GLOBALE

elèuthera

il nostro sito è www.eleuthera.it
e-mail: info@eleuthera.it

INDICE

Prefazione 7

INTRODUZIONE

Codici, metodi, tattiche e amenità varie 15

I. Storie informatiche 25
II. Licenze, politica, mercato 41
III. Comunità 59
IV. La strategia economica dell'Open Source 83
V. ICS Sunt Leones 103

Appendici 115
Presentazione oggetti grafici 118
Disclaimer 126

PREFAZIONE

Questa prefazione – come di consueto, del resto – è in realtà una postfazione, nel senso che è stata scritta dopo il libro stesso. Il fine è parlare fuori dai denti, rendere il discorso il più possibile comprensibile, ampliare le prospettive, dare delle indicazioni utili al lettore, soprattutto quello non specialistico e alieno a concetti come software, computer, internet.

Questo libro traccia dei percorsi e delle linee di fuga in una materia complessa: la scrittura di codici informatici, l'agire quotidiano di legioni di coder e hacker di vario tipo. Il mondo digitale, la tecnocultura pervasiva, la matrice – tutte immagini di gran moda – devono molto a questi individui manipolatori di codici, ovvero coloro che detengono il potere tecnico di intervenire direttamente sui processi di creazione dei codici che modellano le realtà.

Tuttavia, malgrado il loro enorme potere, raramente queste persone se ne fanno carico, difficilmente lo gestiscono, in pochi prendono posizioni dal punto di vista politico, o per meglio dire al di fuori dei mondi digitali. Si tratta di una minoranza.

I media di massa ripropongono regolarmente, e in maniera concertata, banalizzazioni ridicole dell'attivismo digitale. Questo atteggiamento di sufficienza e spettacolarizzazione rende difficile una cartografia anche solo vagamente oggettiva di quanto si muove nelle reti: i pirati informatici sono uno spauracchio utile al pensiero totale, non importa di quale colore politico, e funzionale alle risposte preconfezionate. Per correre ai ripari, per difendersi

da questa malvagia incarnazione piratesca, sono stati costituiti corpi polizieschi internazionali con giurisdizione anche virtuale, sono state lanciate campagne sulla sicurezza informatica, sequestrate migliaia di macchine in tutto il mondo, arrestate centinaia di persone; i superstati, dagli USA alla UE, fanno a gara nell'approvare corpus di leggi liberticide (DMCA, Digital Millennium Copyright Act, del 1998; EUCD, European Union Copyright Directive, del 2001)[1] che finalmente permettano loro di prendere il controllo delle reti.

Un effetto lampante di questa politica è stata la criminalizzazione, avvenuta nell'indifferenza generale, di larghe fasce della popolazione che ha accesso alle reti: tutti quelli che scaricano materiali protetti da copyright, audio, video, testi, qualsiasi cosa, al momento compiono un illecito penale, alla faccia della riproducibilità tecnica! Questi tentativi, in parte già riusciti, di imbrigliare, irreggimentare, castrare la libertà creativa delle reti informatiche riguardano dunque la vita concreta di tutti. Tutti usano carte di credito e bancomat, cellulari e computer, pochi si preoccupano della costante chiusura di orizzonti, delle continue limitazioni delle libertà sulle reti, che guarda caso corrispondono a tagli drastici delle libertà civili più tradizionalmente intese: più controlli ovunque, più paranoia per tutti, più polizia, più armi (naturalmente, «nel vostro interesse», «per la vostra sicurezza»). I mondi digitali, di cui la rete di internet è la manifestazione più nota, non sono completamente altro dai mondi reali: sono semplicemente differenti, spesso in movimento più rapido e convulso, ma sostanzialmente riflettono e a volte anticipano i movimenti che si verificano fuori di essi. Perciò la mitizzazione manichea dell'hacker come individuo pericoloso che si muove in un territorio senza leggi (magari!), onnipotente, quasi fosse un essere distribuito con terminali senzienti in ogni capo del mondo, in rapporto con oscure comunità di supertecnici, è un'immagine decisamente nostalgica di soluzioni facili, desiderosa di stabilire confini chiari e netti, di separare i buoni dai cattivi. Il mito fortemente modellato dalla cultura cyberpunk rappresentava gli hacker come individui pericolosamente interfacciati con la realtà, tra il virtuale e il reale, con il giubbotto di pelle e gli occhiali a specchio. Effettivamente, gli hacker creano codici e aprono nuove strade nella tecnosfera, ma non hanno gli occhiali a specchio, forse non li hanno mai avuti. Hanno una pas-

sione per i codici, per le macchine, un desiderio di capire come funzionano e di comunicarlo agli altri. Creano comunità molto stratificate e spesso fortemente gerarchizzate, dove la meritocrazia ha un ruolo centrale, ma difficilmente parlano «al mondo»: nel complesso, da un punto di vista meramente politico, sono neutri, non schierati, non attivi.

Una delle ragioni di questa disaffezione per la vita reale, la *real life* schematicamente contrapposta alla *virtual life* (campo di azione e di costruzione della propria individualità per un numero sempre più imponente di individui), risiede probabilmente nelle caratteristiche stesse dei mondi e dei linguaggi digitali. Il cyberspazio, la matrice digitale, già di per sé è fatta di codice. La scrittura e l'uso di codici informatici può dunque sembrare del tutto autoreferenziale, interna all'espansione della matrice stessa, senza relazioni con la realtà non-digitale. La realtà esterna, invece, non è completamente codificata, perdurano enormi sacche che resistono a qualsiasi tentativo di codifica. Mentre scrivere codice crea di fatto, e completamente, la realtà della matrice, e si configura in quanto azione omogenea alla natura stessa della matrice, usare una lingua naturale non crea tutta la realtà, è un'azione eterogenea, perché crea solo il mondo condiviso da chi comprende quel linguaggio[2].

Inoltre, se paragoniamo i linguaggi informatici alle lingue naturali, l'aspetto che più ci preme sottolineare è la radicale differenza di finalità e funzionalità: una lingua naturale viene codificata a posteriori, viene scritta una grammatica da esperti basandosi sull'utilizzo della lingua; invece un linguaggio digitale viene pensato per raggiungere un determinato scopo: per scrivere interfacce grafiche, per mettere in relazione altri programmi scritti in linguaggi differenti, per programmare una macchina a basso livello, ecc. La finalità è dettata a priori, anche se ovviamente si possono aggiungere usi e funzionalità impreviste. Finalità e funzionalità: il fine di un codice è che funzioni. Poi ognuno lo userà a modo suo. L'attitudine hacker è tutta qui: ho un bisogno o desiderio, applico la mia passione per soddisfarlo, scrivo un codice che funzioni a quel fine. Banalizzando: ho un computer, un microfono, un telefono, desidero parlare con un amico lontano, scrivo un codice, un programma che metta in relazione gli elementi tecnologici per raggiungere il mio scopo. La politica diventa personale al mas-

simo grado: uso il mio potere tecnico per raggiungere i miei obiettivi in maniera funzionale.

Abbiamo imparato molto dallo stile hacker. Abbiamo imparato a giocare e a condividere, a immaginare nuovi possibili usi della tecnologia. Vorremmo dare qualcosa in cambio, influenzare come siamo stati influenzati: condividere un immaginario radicale. Smetterla una buona volta con la strategia della resistenza e della difesa di minuscoli interstizi di libertà, di piccole aree autogestite a fatica connesse tra loro, sempre pronti a cambiare aria se la repressione alza il tiro; abbandonare le strategie di pura sopravvivenza, le economie di autosussistenza, e cominciare ad ampliare le aree di libertà. La creazione di TAZ (Temporary Autonomous Zone) è solo il primo passo, ma non basta: deve diffondersi come un virus, moltiplicarsi in una miriade di progetti. I mezzi ci sono: la tecnica è nelle mani di chi la sa usare, e adesso è il momento di promuovere un uso sovversivo della tecnica. Negli anni Ottanta gli hacker venivano processati e sbattuti in prima pagina (e, spesso, cooptati subito dopo dai servizi più o meno segreti per spiegare a ottusi funzionari come usare le macchine) perché osavano penetrare nei sistemi delle grandi compagnie telefoniche americane. Era ridicolo, visto che chiunque sapesse leggere e usare i manuali tecnici delle compagnie, non certo segreti, avrebbe potuto fare altrettanto. Ma diffondere le conoscenze e le informazioni, nell'età in cui l'informazione è il bene più prezioso, l'unica vera moneta di scambio e fonte di potere, è già di per sé sovversivo. Oggi gli hacker detengono senz'altro il potere tecnico per costruirsi le loro reti telefoniche o reti di qualsiasi altro tipo, senza chiedere il permesso a nessuno, negoziando invece con i soggetti interessati i possibili scenari. Dovrebbero solo sporcarsi di più le mani con la vita reale, prendere la parola e imparare a parlare anche a persone che non hanno la loro competenza tecnica. Non è facile, non è automatico, non ci sono ricette di sicuro successo. L'incomprensione è sempre dietro l'angolo, la traduzione può risultare oscura e inefficace. Però si può giocare, e metterci tutto il proprio desiderio. Non sarà comunque fatica sprecata.

Questo libro, quindi, è un'azione diretta, un modo di chiamarsi in causa, di non restare a guardare il divenire vorticoso della tecnocultura, ma di metterci su le mani. È stato scritto a otto mani, at-

traverso strumenti di *open publishing* in rete[3], da una comunità scrivente che si è costituita in diversi mesi di lavoro comune; ma, in pratica, sono molte di più le mani che sono intervenute: ognuno con le sue competenze, abbiamo dovuto confrontarci e cercare di capirci fra di noi, mediare e trovare linguaggi condivisi, prima di poter dire qualcosa. Questo libro non è solo un libro perché continua sulla rete, nei percorsi che si sono aperti man mano che ci guardavamo intorno, chiedendo a chi ne sa di più, ma magari non ha tempo, voglia e capacità di raccontare agli altri. Questo libro è un'autoproduzione di un autore collettivo che ha coagulato intorno a sé degli interessi precisi, una volontà chiara di immergersi nella realtà, consapevole dei propri mezzi tecnici.

Fra tante azioni, ci sono state anche parecchie riflessioni. Innanzi tutto su chi siamo e cosa vogliamo, sui nostri desideri. Sul modo di relazionarci fra di noi, nei confronti degli altri, delle comunità di cui facciamo parte. Sul primato del processo, del metodo, rispetto ai risultati. Nessuna indagine sociologica, economica, linguistica che pretenda di essere oggettiva: ma tutto questo, e molto altro insieme, in un divenire fluido. Ogni capitolo può essere letto a sé: si susseguono una discussione sull'uso dei codici (Introduzione), una panoramica storica dell'emergere dei concetti di Free Software e Open Source (cap. I), una disamina delle licenze software (cap. II), un'analisi delle comunità digitali (cap. III), un approfondimento sulle relazioni tra Open Source e mercato (cap. IV), una focalizzazione sulle possibilità di azione degli individui (cap. V). Un altro livello di lettura, più immediato rispetto alla narrazione, è quello grafico; infatti, sono state inserite delle infografiche, cioè oggetti grafici, mappe di vettori senza alcuna legenda che evidenziano le connessioni fra le comunità digitali e fanno il punto sulle relazioni tra Free Software e Open Source, per facilitare il posizionamento del lettore nel discorso del testo. In rete, oltre al libro completo liberamente scaricabile, rilasciato sotto una licenza Creative Commons[4], si trovano sitografie, link e approfondimenti vari aperti a contributi futuri.

In questo libro tanto eterogeneo vi sono molte carenze e molti punti di forza, nessuno sviluppato a fondo: questo perché la teorizzazione perfetta, le teorie piene e lisce, senza alcun punto debole, perdono subito contatto con la realtà e si traducono in pratiche catastrofiche, autoritarie, non condivise. Preferiamo abbozzare,

rilasciare una versione alfa, e attendere contributi. Questo libro è pensato come un software modulare: abbiamo dei desideri da realizzare, vorremmo far funzionare e implementare le nostre reti, quindi abbiamo pensato di scrivere delle librerie, dei pezzi che possano essere riutilizzati in altri contesti, dei brani di codice che possano servire da collegamento tra diversi tipi di comunità e soggetti eterogenei: hacker, tecnici, attivisti, utenti a qualsiasi livello delle tecnologie informatiche. L'ordito e la trama della tela che possiamo tessere sono molto più complesse di quanto non possa restituire un libro, ma un libro è quello che ci serve per iniziare a costruire. Questa struttura modulare è funzionale inoltre all'intervento esterno: chiunque può scrivere la sua implementazione, proporre migliorie, ideare e realizzare un suo plugin che svolga funzioni specifiche.

Annoveriamo tra i punti deboli, quelli che possono facilmente essere attaccati, almeno tre linee di fuga che ci piacerebbe seguire, o meglio che qualcuno seguisse.

Innanzi tutto, un discorso sul mondo del lavoro. Elaborare cioè pratiche di autoformazione e la condivisione delle competenze come modello di autodifesa digitale, esportabile in qualsiasi campo, anche al di fuori dell'ambito qui affrontato: prospettive di biosindacalizzazione dei soggetti precari, per i quali le tattiche del welfare tradizionale (e di qualsiasi presunto welfare «alternativo») sono del tutto obsolete e inappropriate.

Inoltre, le tematiche legate all'ergonomia. A partire dal software e dal rapporto uomo-macchina, progettare oggetti, servizi, ambienti di vita e di lavoro, affinché rispettino i limiti dell'uomo e ne potenzino le capacità operative, con la massima attenzione al comfort, all'efficacia, alla sicurezza. Buone pratiche per vivere con un certo stile, per usare le tecnologie e non esserne usati.

Infine, immaginare nuovi modi per attraversare i livelli delle realtà in cui viviamo, nuove declinazioni collettive e individuali che prendano forma, diventino azioni concrete e di quando in quando riescano, attraverso pratiche di scrittura comunitarie, a fermare il tempo e il flusso dell'azione, a teorizzare, a individuare nuove vie di fuga, per spingere al massimo i propri desideri.

Una precisazione: in questo libro si accenna appena a Microsoft, perché sparare sulla Croce Rossa è troppo facile e persino noioso. Le posizioni monopolistiche e di chiusura pressoché totale

dei codici del colosso di Redmond non possono nemmeno essere prese in considerazione tanto sono lontane dallo spirito hacker. Se addirittura l'antitrust americano si accorge che qualcosa non va, non ci vuole un grande intuito.

È più interessante prendere in considerazione il sottile slittamento di significato che ha portato la pratica del Free Software a diventare più semplicemente Open Source, un movimento che sostituisce la pratica della libertà con una meno imbarazzante «apertura»: come ricompensa, viene appoggiato da governi e da corporations come IBM e Sun, insomma da poteri forti che ora si fanno improvvisamente paladini dello sviluppo dei metodi di condivisione e apertura elaborati nelle comunità digitali. Perché non avvalersi della collaborazione appassionata di persone a cui piace il lavoro che fanno, invece che costringere persone poco motivate a produrre merci che non gli interessano? Il controllo morbido, l'insistenza sulla creatività e sul lavoro di équipe, le pacche sulle spalle, le gratificazioni, sono da tempo patrimonio delle tecniche aziendali: si realizzano prodotti migliori in minor tempo e a costi inferiori. Per molti settori, persino i militari preferiscono usare lo sviluppo aperto, piuttosto che la chiusura, per perfezionare i loro strumenti di dominio e sterminio. Vogliamo allora mettere il dito sulla piaga, evidenziare le incapacità politiche del Free Software, l'insufficienza della GPL, la necessità di estendere il copyleft e insieme l'ipocrisia, molto redditizia in tutti i sensi (ma che nonostante tutto ha dato una scossa al monopolio Microsoft), che ha portato al successo del termine Open Source.

Infine: se questo libro vi darà delle risposte e lo chiuderete colmi di sicurezze e gratificazioni, avremo fallito. Speriamo che questo libro vi deluda: siamo certi che non sia abbastanza e perciò speriamo che vi spinga a dire la vostra, ad agire in prima persona, magari a prendervi uno spazio di elaborazione e scrittura collettiva, usando e migliorando gli strumenti che qui abbiamo testato. Questi strumenti e molti altri sono a disposizione, fra l'altro, presso il server Ippolita (www.ippolita.net), che incidentalmente è anche l'autore di questo libro. Solo così il meccanismo della delega, almeno per una volta, sarà accantonato: confidiamo nell'assunzione diretta di responsabilità, per la creazione di dinamiche impensate di autogestione.

Note alla Prefazione

1. Alcuni approfondimenti in italiano: EUCD http://www.softwarelibero.it/progetti/eucd/; DMCA http://www.annozero.org/nuovo/pages.php?page=Sklyarov+DMCA.

2. Il discorso qui accennato è ovviamente assai più complesso. La realtà è idiota, nel senso etimologico del termine di proprio, privato, particolare; questo aspetto è assolutamente alieno alle codifiche totalizzanti che rendono invece la matrice digitale una sequenza, per quanto gigantesca, di impulsi discreti, di zeri e di uno.

3. Matthew Arnison, *L'open publishing è la stessa cosa del software libero*, Indymedia FAQ #23, http://italy.indymedia.org/news/2002/07/64459.php. Lo strumento principale che abbiamo usato è un wiki, un software collaborativo per scrivere, che potete trovare qui: www.ippolita.net

4. In particolare abbiamo scelto una licenza Creative Commons copyleft di tipo by-nc-sa 2.0, ovvero: «Tu sei libero: di distribuire, comunicare al pubblico, rappresentare o esporre in pubblico l'opera; di creare opere derivate. Alle seguenti condizioni: by (attribuzione): devi riconoscere la paternità dell'opera all'autore originario; nc (non commerciale): non puoi utilizzare quest'opera per scopi commerciali; sa (condividi sotto la stessa licenza): se alteri, trasformi o sviluppi quest'opera, puoi distribuire l'opera risultante solo per mezzo di una licenza identica a questa. Maggiori informazioni: http://www.creativecommons.it

INTRODUZIONE

CODICI, METODI, TATTICHE E AMENITÀ VARIE

In questo libro si parla spesso di codici. Un codice è un insieme di regole che permette la conversione di una porzione di informazione (una lettera, una parola, una frase) in un'altra forma di rappresentazione, non necessariamente dello stesso tipo. Una lingua naturale, come l'italiano, è un codice, in quanto è composta da segni arbitrari che, combinati con altri segni dello stesso tipo, costituiscono un sistema di segni complesso, ovvero un codice. La lingua quindi può essere considerata un codice in quanto mette in relazione un universo di significati e l'insieme dei significanti di quella lingua stessa.

Allo stesso modo, nel contesto informatico, un codice mette in relazione le proprietà di un particolare linguaggio (C, PHP, perl, ecc.), composto di segni arbitrari, con i significati che l'utilizzatore del linguaggio desidera esprimere. La scrittura di codice, l'atto del codare (da *to code*), in questo senso, è un'operazione di passaggio da un significato a un significante. La decodifica, viceversa, è il passaggio dal significante al significato[1]. Nella stesura e nella composizione del libro, operazioni successive di codifica e decodifica, abbiamo cercato di tenere presente e di rendere questo continuo trapasso, questo movimento.

1. Canguri dimensionali

Saltando in una tensione continua da un piano all'altro, dalla

rete al reale, dall'individuale al comunitario, e ritorno, è inevitabile porsi in un'ottica di *traduzione*, e dunque di tradimento.

Finora con la stesura di questo libro, che non riusciamo a considerare concluso, abbiamo saltato in continuazione da dimensioni virtuali a dimensioni reali: siamo passati dalle microrealtà individuali (l'hacker solitario che scrive il suo codice) ad ampliamenti comunitari (la condivisione delle conoscenze), a proiezioni globali (l'uso di licenze che rilasciano un permesso, che concedono un determinato potere a un certo utente riguardo a una precisa porzione di codice).

Naturalmente, questi stessi esempi sono semplificazioni: i flussi si possono analizzare solo perdendo qualcosa della loro complessità.

Questi passaggi non solo non sono affatto scontati, perché richiedono una certa elasticità e molteplicità di punti di vista, ma sono evidentemente anche del tutto soggettivi e situati.

Abbiamo cercato di fotografare una realtà complessa e in divenire vorticoso, il mondo digitale, con il suo pullulare di comunità, individui, codici, nelle relazioni con il mondo commerciale. Questa fotografia, è quasi inutile dirlo, non poteva che risultare mossa, dunque un tradimento dell'oggettività che esiste (forse) là fuori da qualche parte, ma che le parole e i codici possono appena abbozzare.

2. Analisi pure e oggettive – Desideri e linguaggi

I livelli di analisi macrosociale scadono presto in costruzioni di poco o nullo interesse perché perdono di vista la materialità degli individui che danno vita allo spazio sociale, sia esso digitale o reale (quanto poi i livelli siano mescolati, è un ulteriore elemento di complessità).

Se consideriamo invece i singoli individui, e i loro desideri, si svela l'ovvia parzialità del proprio punto di vista. Tradurre nella realtà ciò che sta in rete è un'azione *impura* (per fortuna), cioè di tradimento, perché ogni traduzione si confronta con la complessità del non–detto (i codici non sono mai equivalenti anche se sembrano dire la stessa cosa): allo stesso modo, un codice ben scritto non si esaurisce nella purezza delle sue routine e subroutine, e nemmeno nello stile di indentazione o nella ritmica dei commenti

al codice stesso. Il codice contiene in sé la possibilità di un uso, che può tradursi in un accrescimento, in una perdita, comunque in una trasformazione di ciò che già esiste.

3. Codici

Limitiamo momentaneamente il termine codice all'ambito informatico.

Un codice non utilizzato è monco, o forse inutile. La questione dell'utilità o meno di un codice è decisamente mal posta, spinge un parallelo con i linguaggi naturali eccessivamente forzato[2]. Prescindendo dall'utilizzo che ne viene fatto (o *non* ne viene fatto), possiamo dire che non esistono codici inutili.

Esistono sicuramente codici monchi, non finiti, abbandonati, non utilizzati, accatastati in qualche cimitero di sorgenti sparso da qualche parte nella rete. Non si scrive mai codice iniziando dallo zero assoluto, ma studiando i codici sorgenti di altri prima di scrivere il proprio. Dunque accade che codici ritenuti monchi da chi li ha scritti tornino a essere sfogliati, studiati, utilizzati, implementati.

Il codice può essere riciclato e riutilizzato, studiato, ammirato, smembrato e ricomposto, oppure semplicemente tenuto da qualche parte per i posteri o per il piacere di avere un archivio.

Infine, non è nemmeno detto che sia possibile analizzare l'utilità o meno di un codice. Ovvero, i codici possono essere scritti per il «grande pubblico», per gli «addetti ai lavori», per il «programmatore stesso». Non per forza acquistano valore nella loro compilazione ed esecuzione, ma a volte addirittura si attualizzano solamente e completamente nella loro semplice scrittura. Per non parlare dei codici che sono scritti per essere il più possibile non utilizzati oppure semplicemente per il gusto di essere i primi a riuscire in una qualche impresa.

L'utente medio può apprezzare un programma per gestire la posta elettronica come *Thunderbird*, «pochi eletti» o «addetti ai lavori» apprezzano *Blender* per la grafica 3D, alcuni coder sorridono all'idea delle librerie *dmalloc*. Esistono poi linguaggi di programmazione esoterici, il cui fine non è tanto l'usabilità quanto il desiderio di esplorare i limiti delle macchine, o dimostrare un con-

cetto, o giocare semplicemente con le possibilità del codice ed espanderle[3].

4. Virtù(alità) del codice

L'uso di un codice, in ogni caso, apre la strada al salto dimensionale: è virtuale, sì, e proprio per questo non è solo virtuale. Un codice usato non è solo codice: entra in relazione con la realtà, e la realtà non è fatta solo di codici e parole, ma anche di cose che proprio non ne vogliono sapere di rientrare nella semplicità del nostro codice, di essere fedelmente rispecchiate, di adattarsi. Anzi di solito accade che la realtà, le cose, trasformino il codice e viceversa: l'esempio più banale è l'uscita di nuove release dei programmi che impone l'aggiornamento del proprio sistema, quindi un impiego di tempo da parte di individui in carne e ossa, e a volte un mutamento nella struttura stessa delle macchine (più potenti, più performanti).

L'uso, la prassi, fra l'altro, trascina con sé l'etica. L'etica è la responsabilità degli esseri umani, la loro capacità relazionale, e non solo. Non si tratta affatto di riscoprire il grande valore aggiunto dell'umanità, la centralità delle scelte dell'uomo, che si costruisce nel quadro di un umanesimo in fondo malinconico: ah, certo, le macchine servono, risparmiano fatica, ma come si stava bene senza! L'etica riguarda anche le macchine, e non solo gli esseri umani, perché, che piaccia o meno, attualmente esistono macchine che agiscono, che si relazionano con altre macchine, con altri esseri più o meno viventi e senzienti. Esistono ad esempio macchine armate (dagli esseri umani, certo) che decidono della vita di esseri animati[4]. Vi sono passaggi continui tra i mondi. Mancano, perché devono essere create, parole adeguate (che non siano solo codici) per esprimere questi passaggi, dal virtuale al reale e ritorno, e dal collettivo all'individuale e ritorno, dove non c'è mai un punto di partenza né di arrivo, ma il percorso *dipende dal punto di entrata*. Insomma, «realtà» non significa affatto «ambito d'azione degli esseri umani, mondo umano dei corpi e delle cose», né «virtuale» sta per «mondo delle macchine, delle menti e degli oggetti immateriali». Questa dicotomia fra reale e virtuale, cavallo di battaglia di una certa vulgata pseudoscientifica, è del

tutto inappropriata, oltre che orfana di altre ben più potenti e storicamente influenti separazioni (fra corpo e mente, fra mondo delle idee e mondo della materia, ecc.)[5]. Immaginare possibili hacking della realtà non c'entra nulla con una ridefinizione e riproposizione dell'ambito reale come «migliore perché umano e umano dunque migliore».

Le cose sono assai più complesse, i mondi variamente mescolati, e osservare i punti di passaggio fra di essi implica delle identità sempre in gioco e negoziabili. Volerle fissare e legare a un livello a ogni costo è una semplificazione indebita. Chi vuole definire un livello «migliore», un approccio «migliore», di solito, vuole semplicemente mantenere lo status quo perché gli fa comodo, ovvero per mantenere il potere che ha.

L'umanesimo che ribadisce allarmato che il progresso tecnico mette da parte l'essere umano ed esalta le macchine, inumane e dunque cattive (come se invece gli esseri umani fossero «di natura buoni»), è solo la faccia luddista della tecnofilia repressiva, che immagina di scaricare le menti umane in software puri. Sogni, incubi di tecnofascisti puritani, che si troverebbero a loro agio in un universo disincarnato: come se potessero darsi software senza hardware o senza wetware. Entrambe le posizioni hanno scarse capacità di cambiare, ma soprattutto di immaginare e desiderare.

5. Visioni differenti

Capacità di immaginare e desiderare: in questo senso, ogni capitolo può essere letto come un tentativo di far vedere ciò che non si vede tra le «semplici» righe di un codice. Ad esempio, nel capitolo dedicato alle licenze software appare evidente che, adottando il «permesso d'autore» (copyleft), le licenze come la GPL (General Public Licence) funzionano con un meccanismo virale e cercano così di garantire ed estendere la libertà del codice. Un codice rilasciato sotto GPL è libero, e tutti i codici che da esso derivano saranno altrettanto liberi; queste licenze quindi cercano di esplicitare ciò che non si può esplicitare, far vedere la libertà intrinseca in quel codice. Tuttavia questo non è sufficiente per rendere oggettivo e ineliminabile un uso etico del software.

Infatti l'etica, come la libertà, non si può garantire per decreto

o per licenza, dando o togliendo permessi, ma solo confrontando, traducendo, adattando. Lì sta il tradimento, nell'uso, che può essere diverso e lontano (volendo persino di mercato, anche perché la licenza GPL non nega affatto la possibilità di profitto commerciale) rispetto a quello che si era previsto scrivendo quel codice. È come dire a un autore (che scriva romanzi, saggi, poesie o software a questo punto non è cruciale): io nel tuo testo ho letto questo. L'autore si potrà anche risentire, sostenendo di aver avuto tutt'altro intento, ma l'interpretazione non può che essere libera. La libertà dell'utente in questo caso sta nella possibilità di uso (etica) che gli viene concessa. Dare una libertà significa anche prendersela.

Scrivere uno stesso concetto con il linguaggio del liberismo, dello strutturalismo, del postmodernismo, del marxismo, dell'avanguardismo (o di qualunque altro «ismo», maggiore o minore che sia) non è metodologicamente diverso dallo scrivere un codice che in linea di principio compie le stesse cose in un linguaggio procedurale come il C, oppure con uno script perl, o PHP, o in un linguaggio a oggetti, o in un linguaggio macchina assembly, ecc. Si tratta di visioni differenti di una medesima, molteplice realtà. Ognuno ha familiarità e competenza per un certo linguaggio e non per altri, quindi formulerà il proprio approccio nel linguaggio che padroneggia meglio o che ritiene più adatto.

Un programma può essere scritto in un linguaggio qualsiasi, in linea di principio, sempre che *i mezzi linguistici siano adeguati ai fini che si intende raggiungere,* e dunque: una sequenza di boot all'accensione di un computer, il cui fine è bootare, sarà probabilmente scritta in assembly, un sito web dinamico, il cui fine è servire pagine web agli utenti che interagiscono e le richiedono, potrà essere scritto ad esempio in PHP+mysql[6], e così via.

Tuttavia alcune cose si possono scrivere solo con certi linguaggi, o meglio, la maggior parte delle idee che un programmatore desidera realizzare si possono codare con un linguaggio adeguato al target di utenti che si intende raggiungere[7].

Dunque, la scelta del codice o dei codici utilizzati non è indifferente.

6. Traduzioni e poteri: modelli chiusi, modelli aperti, modelli free, ovvero liberi.

Tradurre significa inoltre mescolare le carte in tavola. Significa mettere a confronto linguaggi spesso del tutto eterogenei che descrivono (codificano) la stessa realtà e cercare di trovare ciò che non quadra, l'errore, il bug, che può anche essere un elemento di nuova forza e spinta a migliorare linguaggi e codici.

Ovviamente, considerato quanto abbiamo detto poco sopra, tradurre significa posizionarsi in un luogo di potere dal quale si dà la parola a un certo linguaggio e in un certo modo. Tradurre è contraddittorio perché significa dare a chi non conosce quella lingua o quel codice la possibilità di accedere a un determinato significato, ma nello stesso tempo forzare il significato ad adattarsi al proprio particolare punto di vista. Significa prendersi la responsabilità della fiducia, affermando: fidati di me, anche se non conosci questa lingua, io ti posso spiegare. Nel caso della scrittura di codice vi sono ulteriori passaggi che rendono ancora più indispensabile un investimento di fiducia nella traduzione proposta, perché l'utente medio «legge» il codice già «tradotto» dall'interfaccia grafica[8].

Ma sarebbe ipocrita fingere che i codici possano davvero spiegarsi da soli, per quanti manuali, how-to e tutorial (o dizionari e prontuari) si scrivano: anzi, ogni manuale in più è un nuovo punto di vista, una nuova operazione di posizionamento in cui chi scrive, per spiegare, domanda la fiducia a chi ascolta la spiegazione[9].

Il codice si spiega quando qualcuno prende coraggio e dice: secondo me funziona così, l'ho scritto (oppure l'ho debuggato, migliorato, ecc.) e credo che questo sia l'uso migliore che se ne possa fare. Se questo codice è proprietà privata, o puro oggetto di scambio commerciale, nessuno può in teoria spiegarci meglio, modificando e migliorando il codice stesso, come stiano le cose. In questo senso, possiamo dire che un software close, è nella stragrande maggioranza dei casi un'operazione di posizionamento assoluta, in cui qualcuno si arroga il diritto di spiegarci come stiano le cose, riducendole a un solo punto di vista, il suo, che generalmente non è affatto disinteressato, visto che il codice l'ha scritto per far soldi e basta.

Il software proprietario, ipocritamente, riduce le possibilità e

afferma: le cose stanno così, proprio così, nessun privilegio, nessun luogo di potere, solo la pura e semplice verità. Non è una visione differente, è una visione totalizzante che richiede fiducia cieca da parte dell'utente senza dare in cambio l'accesso agli strumenti per interagire e implementare quella traduzione, quel particolare punto di vista.

Per rendere ancora più saldo questo monopolio interpretativo, il software close si avvale di minacce, ricatti e repressioni: bisogna aspettare il prossimo rilascio del programma se ci sono dei bug, ma se ci azzardiamo a metterci su le mani rischiamo denunce civili e penali. Ma ci sono anche strategie più morbide di controllo e persino sfruttamento degli utenti: ad esempio, si può contribuire allo sviluppo di applicazioni close facendo betatesting, ovvero testando versioni non ancora stabili (beta version) e comunicando le proprie difficoltà e impressioni d'uso alle aziende produttrici; oppure inviando segnalazioni di errore quando qualcosa non funziona. In cambio, potremo acquistare una nuova versione migliorata grazie ai nostri suggerimenti!

Adottare un modello open, o meglio free, libero, significa invece senz'altro affermare la propria posizione, scrivere il proprio codice, ma dare la possibilità ad altri di spiegarlo, nella maniera più condivisa possibile. Poi sta alla comunità che recepisce il codice trovare nuovi usi, interpretazioni e interazioni, nuove spiegazioni, modificare e tradurre, creare. Ma senza nascondersi dietro una presunta trasparenza del codice che hai scritto.

Se il codice si può usare, in un modo qualsiasi, anche smontandolo a pezzi (e non solo accettando passivamente le condizioni stabilite una volta per tutte da chi ha scritto la licenza), la trasparenza è meglio lasciarla a chi è davvero convinto che un giorno, quando il mitico *Progresso del libero mercato* avrà fatto il suo corso, chiunque schiacciando un bottone farà la cosa giusta e sarà libero. I codici non rispecchiano fedelmente e in maniera trasparente la realtà: sono traduzioni imperfette, punti di vista e prese di posizione parziali.

L'uso del codice implica l'etica, un certo stile, un certo codice di comportamento; la libertà d'uso implica l'imprevisto, e la continua rinegoziazione di ciò che significa «fare la cosa giusta» con quel pezzetto di codice. In ogni caso, vuol dire fiducia negli altri e nelle capacità di condivisione degli esseri umani, nelle loro ca-

pacità di interazione con le macchine (che non si costruiscono da sole – non ancora, almeno), nella loro capacità di conoscere, comprendere: nella possibilità di cambiare, e divenire Altro.

Questa fiducia non è per nulla incondizionata, anzi: non è fiducia nell'Uomo, nella Macchina, nella Ragione, ma più semplicemente in alcuni individui che si sentono affini, comunità di riferimento, situazioni in cui si è a proprio agio e che procurano piacere. È fiducia in alcune macchine che si ritengono migliori di altre, o che stimolano maggiormente il proprio senso estetico. Nessuna grande idea quindi, nessuna verità da svelare, nessuna bandiera da sventolare: al contrario, molte idee, molti individui, molte verità, molte etichette, raggruppamenti, comunità, sempre in divenire.

Tra questi soggetti e oggetti vogliamo stabilire delle relazioni, e non si tratta di volare basso, perché per quanto piccole, temporanee, revocabili, queste aperture di fiducia possono creare livelli di complessità enormi.

Note all'Introduzione

1. In realtà la questione è assai più complessa, sia per quanto riguarda la lingua naturale, sia nel caso dei linguaggi informatici. SITOGRAFIA: *Codici e Linguaggi*.

2. Infatti, avrebbe poco senso paragonare l'uso, nel 2005, di una lingua arcaica e desueta, ad esempio la scrittura cuneiforme sumerica, con l'uso di codici informatici «antichi». Basti pensare a quanti usano i codici di mplayer 0.1, Linux 0.01, xchat 0.1 ecc. Non ha senso definirli codici monchi o inutili.

3. Linguaggi di programmazione esoterici: http://en.wikipedia.org/wiki/Esoteric_programming_language

4. L'espressione più compiuta dell'incubo delle macchine armate è probabilmente il film *Il Dottor Stranamore, ovvero come imparai a non preoccuparmi e ad amare la bomba* di Stanley Kubrick, GB, 1964. Dati alcuni input iniziali, le macchine scatenano l'apocalisse nucleare. E, se non fosse per la pervicacia di alcuni esseri umani nel raggiungimento degli obiettivi di distruzione, nell'adempimento militaresco degli ordini ricevuti, potrebbero essere fermate! Per un'analisi più recente si veda: Manuel De Landa, *La guerra nell'era delle macchine intelligenti*, Feltrinelli, Milano, 1996 (ediz. orig.: *War in the Age of Intelligent Machines*, Zone Books, New

York, 1991). Sicuramente la biocenosi contemporanea è un continuum di interazioni e sinergie caotiche fra esseri di specie diverse, tra le quali le macchine svolgono una funzione non secondaria.

5. In particolare, si vedano i paradigmi dell'IA nella sua formulazione forte (non a caso ispirata da oltre mezzo secolo di ricerca militare). Si tratta in un certo senso di una rivisitazione tecnofila del platonico mondo delle idee, trasformato in un mondo di software in cui finalmente i corpi non saranno più il carcere di gigantesche menti inumane. Si veda Naief Yehya, *Homo Cyborg*, pp. 141 ssg., Elèuthera, Milano, 2005.

6. Non tutto si può scrivere con ogni linguaggio, perché ogni linguaggio ha una sua finalità. Il C è stato ideato per scrivere a medio livello sistemi operativi, PHP è stato fatto per il web, l'assembly è stato scritto perché era un po' più semplice del linguaggio macchina in codice binario. Ovviamente poi ci sono linguaggi particolarmente fortunati da essersi accaparrati altre fette di utenti. Non crediamo che quando nel 1974 Dennis Ritchie ha rilasciato il primo compilatore C si immaginasse che oggi qualcuno potesse programmare applicazioni grafiche...

7. Ad esempio, si può benissimo scrivere un window manager in perl, ma lo userebbero quattro persone e tutti e quattro perlisti esperti. Se faccio un portale web in C, lo faccio semplicemente per esibire i miei attributi (senza considerare che aumenterei esponenzialmente la possibilità di essere attaccabile).

8. Vedi nota 1. Un codice informatico, a seconda del linguaggio in cui è scritto, subisce passaggi radicalmente diversi. I linguaggi informatici si dividono in tre tipologie principali: linguaggi interpretati, linguaggi compilati, linguaggi macchina. SITOGRAFIA: *Codici e Linguaggi*.

9. Un percorso di approfondimento a proposito della nozione di verità, della necessità di stringere patti di fiducia e di «sospensione dell'incredulità» (si veda in proposito Samuel T. Coleridge, «*suspension of disbelief*»), al fine di comprendere le traduzioni e il rispecchiamento del reale da parte di un linguaggio, prende le mosse senz'altro da Friedrich Nietzsche, *Su verità e menzogna in senso extramorale*, Newton Compton, Roma, 1981 (ediz. orig.: *Über Wahrheit und Lüge im aussermoralischen Sinn*, 1873).

I

STORIE INFORMATICHE

1. In principio era Unix, ma lo sapevano in pochi

Agli inizi degli anni Novanta la Free Software Foundation (FSF) distribuiva un sistema operativo derivato da Unix, ma libero, gratuito, stabile e soprattutto (ancora) poco famoso: GNU/Linux. Il panorama digitale di quegli anni, lato utente e piccole-medie imprese, era scandito dal lancio sul mercato dei processori Intel e dalle release[1] di MS-DOS e, successivamente, di Windows.

Il computer, la scatola magica della rivoluzione informatica, era appena diventato un fenomeno di massa[2] e la gente veniva indottrinata sullo standard di accessibilità economica dei Personal Computer (un Computer per tutti! Un PC, un Computer Personale!).

Le nuove leve del digitale non conoscevano altri sistemi operativi oltre a quelli di casa Microsoft, i programmi erano piccoli e

a pagamento, ma anche facilmente copiabili su floppy disc e distribuiti quindi di mano in mano.

I nuovi programmatori nati sui PC lavoravano per lo più in BASIC (Beginner's All-Purpose Symbolic Instruction Code) e pascal, pochissimi in assembly[3]. Nascevano i virus, i giochi grafici, la computer music e le riviste di settore. GNU/Linux non era conosciuto, così come si ignorava la possibilità, e persino l'utilità, di avere i codici sorgente di un applicativo. Semplicemente, i programmi venivano crackati. L'aspetto commerciale del mondo digitale era dato per scontato da chiunque perché quasi nessuno sapeva cosa si celava all'altro lato di questa rivoluzione. In realtà, se consideriamo che le pratiche di condivisione del codice sono nate insieme all'informatica, piuttosto che di origini del Free Software dovremmo parlare di origini del software close o software proprietario.

Erano interessati al Free Software accademici, informatici consapevoli o programmatori dotati di collegamenti a internet che si sarebbero diffusi presso l'utenza non specializzata solo cinque-dieci anni più tardi. Questi personaggi, insieme a tutti coloro che sviluppavano applicativi su piattaforma Unix, rappresentavano i principali depositari del termine hacker: colui che scrive codice per passione, lo migliora, lo condivide e aiuta chi ha problemi nell'utilizzarlo[4].

Il termine hacker ha un ampio spettro semantico e sicuramente acquisterà nuove accezioni in futuro. Non è dunque errato sostenere che gli hacker esistevano anche su piattaforma PC, legati per lo più al movimento cyberpunk della seconda metà degli anni Ottanta. Qualcuno usava Linux, ma pochissimi avevano contatti con le comunità virtuali di sviluppo software, utilizzando per lo più le BBS (Bullettin Board System[5]) come strumento di incontro e confronto. In particolare, i gruppi che nascono in Europa in quegli anni sono orientati alla sperimentazione tecnica, come il mitico Chaos Computer Club di Amburgo[6], ma spesso sono anche più spiccatamente luoghi di elaborazione politica e culturale, che di fatto c'entrano molto poco con il computer hacking, come l'italiano Decoder[7] o l'olandese Hacktic[8], fino alla rete europea ECN (European Counter Network)[9]. Questi gruppi cercano nelle nuove tecnologie degli strumenti di comunicazione per gruppi di affinità sociale e/o ludica, per esperimenti di partecipazione pubblica e politica alla vita delle comunità reali; il mondo della rete è percepito come una frontiera di libertà nella quale rielaborare sconfitte e vittorie della contro-

cultura underground[10]. Un momento cruciale, di forte aggregazione delle culture hacker, fu la tre giorni di Amsterdam *Icata 89*[11]. In quell'occasione Lee Felsestein, il «papà» dei Personal Computer, che già negli anni Settanta aveva fondato il Community Memory Project e poco dopo l'Homebrew Computer Club, portò l'esperienza della Bay Area: il confronto con i gruppi americani diventava sempre più serrato, si elaborava con entusiasmo l'etica hacker.

Negli Stati Uniti, la novità nell'approccio della FSF risiedeva soprattutto nel taglio politico col quale esigeva l'accesso al codice sorgente. Mentre si diffondevano programmi crackati, la FSF insisteva sulla necessità politica di rilasciare software libero all'origine. Del resto, fino alla fine degli anni Settanta, quando il mercato del software non si era ancora affermato (il mercato era praticamente limitato all'hardware, e i produttori facevano a gara a distribuire software gratuitamente, pur di vendere le macchine), negli ambiti universitari o laboratoriali possedere i codici degli applicativi in uso rappresentava la base per l'evoluzione stessa dell'informatica; nessuno avrebbe rivendicato diritti sulla propria opera se non in termini professionali di riconoscimento dei meriti e dei contributi individuali. La protezione e la chiusura dei codici era dunque un fenomeno del tutto «nuovo», e da combattere con determinazione.

La cultura digitale dell'epoca traeva le proprie origini dalla ricerca scientifica condivisa e dall'eccellenza accademica statunitense (ovvero da strutture come il MIT o l'Università di Berkeley); di qui vengono mutuate le successive codifiche, regole e abitudini sociali all'interno dei network di sviluppo tecnologico, trasformandosi in quella cultura comunitaria virtuale che ha gettato le basi per l'approdo di un'intera società nella rete globale.

Il metronomo storico di questa cultura e delle evoluzioni che seguirono è stato il sistema operativo Unix[12], l'ispiratore di GNU/Linux. L'introduzione dei sistemi operativi aveva reso i programmi sempre più portabili: lo stesso sistema operativo veniva offerto dal produttore di diversi modelli di hardware. Unix nasceva alla fine degli anni Sessanta nei laboratori Bell presso l'AT&T, società telefonica americana, da programmatori che lo avevano pensato e scritto nel tempo libero. Per quanto semplice, questo sistema conteneva innovazioni tecniche o implementazioni che l'hanno reso immortale: il linguaggio con cui è stato programmato, il C; un

nuovo file system[13] e una nuova struttura dei file e delle directory; un'integrazione con l'ambiente di sviluppo che i sistemi operativi commerciali di uso domestico/individuale non erano ancora in grado di offrire (in realtà non esistevano affatto!).

Una famosa causa antitrust contro la AT&T vietò alla società di entrare nel settore dell'informatica. Unix venne allora distribuito, a un prezzo simbolico, a buona parte delle istituzioni universitarie, le quali si ritrovarono ad avere una piattaforma comune, completa di codice sorgente, ma senza alcun supporto da parte del produttore. Si creò spontaneamente una rete di collaborazioni attorno al codice di questo sistema operativo, coordinata dall'Università di Berkeley (che avrebbe in seguito sviluppato la versione BSD[14] di UNIX).

Personaggi chiave dell'evoluzione informatica (da Vinton Cerf a Richard Stallman, fino a Tim-Barnes Lee, insieme a molti altri) hanno sperimentato con grande autonomia nel periodo tra gli anni Settanta e Ottanta dando vita ai principali protocolli di comunicazione di rete (SMTP – invio di posta elettronica, FTP – trasferimento di file in rete, HTTP – trasferimento di ipertesti), ai linguaggi di programmazione ancor oggi fra i più utilizzati, come il C, e alla suite di protocolli di rete TCP/IP.

L'etica hacker, nata dalla spinta congiunta dei movimenti libertari dell'America negli anni Settanta e dall'ottimismo scientifico delle élite accademiche[15], costituiva l'energia di cui si nutrivano i laboratori e gli studenti universitari crescendo nel brodo di coltura di un sistema operativo aperto, Unix, vivificato da uno strumento appena nato: internet.

Il potere politico derivato dalle conoscenze tecniche, unito al metodo della collaborazione paritetica, forniva la libertà necessaria a uno sviluppo dal ritmo incalzante; strumenti quali mailing list e chat line cominciavano a diffondersi anche nei primi gruppi non puramente tecnici, ma affascinati da questo nuovo paradigma comunicativo.

Nel frattempo, la suddivisione della AT&T in ventisei società, le cosiddette BabyBell, favorì l'uso di logiche biecamente commerciali nella distribuzione di Unix. AT&T chiuse il codice e iniziò a distribuirlo solo compilato, innalzando notevolmente i costi delle licenze e impedendo la pratica delle patch. Nel 1982 ebbe inizio la storia delle diverse versioni commerciali di Unix, legate ai singoli

produttori di hardware, che, differenziando anche solo di poco la propria versione, riuscivano a legare a sé i clienti acquisiti, in quanto programmi scritti per una specifica versione di Unix solitamente non funzionavano su versioni concorrenti. Unix, senza una copertura legale, si modellava quindi secondo le esigenze delle comunità di ricerca, creando sistemi spesso non compatibili fra loro: nascevano BSD, Solaris, IRIX[16] e altri sistemi operativi di cui alcuni tutt'ora utilizzati. Sulla scia di questo esempio, le università e i laboratori di ricerca cominciarono a regolamentare l'accesso ai codici e adottarono manovre di riservatezza e chiusura, facendo sottoscrivere ai coder accordi di non divulgazione[17] che sostanzialmente li espropriavano delle loro creazioni. In questo modo gran parte della filosofia hacker iniziò a perdere di valore.

Lo standard POSIX[18], con il primo rilascio nel 1988, fu la conseguenza naturale della nascita di diversi Unix: per far comunicare fra loro diverse versioni del sistema vennero stese regole abbastanza ferree su cosa poteva e doveva essere un sistema operativo per definirsi Unix. L'operazione di standardizzazione fu compiuta in gran parte dalla IEEE[19], un ente relativamente neutrale (se si tralasciano le operazioni di lobbying, assolutamente comuni negli Stati Uniti), che non privilegiò nessuno dei concorrenti Unix proprietari; in questo modo fu sostanzialmente raggiunto l'intento di fornire soltanto linee guida per rendere portabili gli applicativi tra le versioni e le varie implementazioni.

L'informatica stava uscendo dalle università, anche se l'informatica di massa era ancora ben lontana; con l'abbattimento dei costi dell'hardware molti neofiti si avvicinavano alle tecnologie grazie ai PC. Paradossalmente, da una situazione di scambio e libera circolazione dei saperi nei laboratori di ricerca, l'epoca d'oro degli hacker del MIT negli anni Sessanta, si andava verso la chiusura dei codici e la mera commercializzazione: con l'espansione delle tecnologie digitali la direzione sembrava ormai tracciata, il cracking stava diventando più diffuso dell'hacking. In attesa del boom internet negli anni Novanta, le comunità di sviluppo costituirono la riserva di caccia delle softwarehouse per assoldare coder, gente che scrivesse programmi commerciali, abbandonando di conseguenza lo sviluppo dei sistemi aperti (ai quali comunque molti coder continuarono a dedicarsi nel tempo libero).

Nessun privato poteva acquistare un PDP-11 della DEC, una

macchina su cui giravano gli Unix, ma i PC IBM 8088, tecnicamente molto inferiori, erano compatibili col portafoglio della gente. Sui PC IBM 8088 e successivi girava un sistema operativo molto primitivo, se confrontato allo Unix System V commercializzato dall'AT&T, l'MS-DOS. Questo limitato e quindi semplice sistema operativo era della Microsoft, all'epoca piccola softwarehouse con appena due dipendenti, e veniva fornito assieme al BASIC, linguaggio di programmazione implementato da Bill Gates e Paul Allen. L'MS-DOS era basato sul DOS, un sistema operativo compatibile con CP/M[20], l'86-DOS, subito acquistato dall'ex hacker e neo imprenditore Gates.

E mentre l'informatica diventava accessibile anche a chi desiderava un network unito per affinità sociali e culturali, e non solo informatiche, all'interno delle prime comunità di hacker molto stava cambiando. Negli anni Ottanta ormai da tempo le università avevano partner commerciali: per questa ragione ritennero ragionevole bloccare la condivisione di codice tra laboratori e utilizzare pratiche di controllo sullo sviluppo dei programmi.

2. E venne il Free Software

Fu in questo periodo che Stallman intraprese la sua battaglia politica. Abbandonati i laboratori di intelligenza artificiale del MIT, si dedicò a scrivere codice per un sistema operativo libero, avviando nel 1984 il progetto GNU (GNU's Not Unix, un acronimo ricorsivo in stile hacker): «L'obiettivo principale di GNU era essere software libero. Anche se GNU non avesse avuto alcun vantaggio tecnico su Unix, avrebbe avuto sia un vantaggio sociale, permettendo agli utenti di cooperare, sia un vantaggio etico, rispettando la loro libertà». Nel 1985 fondò la Free Software Foundation (FSF), un'organizzazione senza fini di lucro per lo sviluppo e la distribuzione di software libero: i software del progetto GNU (molti scritti da Stallman stesso, come il compilatore GCC e l'editor di testo Emacs) sarebbero stati rilasciati sotto la General Public License (GPL), licenza scritta da Stallman che rese *de facto* un applicativo libero, accessibile a tutti, modificabile e distribuibile in qualsiasi modo, purché accompagnato da codice sorgente. Approfondiremo il fondamentale discorso sulle licenze nel cap. II.

Con la diffusione di massa dei PC e di internet, il progetto GNU cominciò a essere conosciuto e si creò una nicchia di utenti attivi, convinti e motivati. Si andava così formando una comunità indipendente dal nascente mercato del software e animata da precise finalità politiche: obiettivo principale della battaglia era la libertà di espressione in campo digitale.

Nel combattere la strategia di penetrazione socioeconomica del software close, Stallman ipotizzò un sistema operativo basato su Unix, ma libero in ogni suo aspetto. Coniando il termine Free Software per definire gli applicativi liberi di essere scambiati, modificati e distribuiti, diede inizio a una campagna contro i brevetti, ma soprattutto contro il copyright, sostenendo l'uso del cosiddetto copyleft, il permesso d'autore: nessun diritto riservato[21].

In sostanza, il copyleft ribalta il copyright. Teoricamente, il livello massimo della libertà con cui può essere distribuito il software è quello che corrisponde alla denominazione di «pubblico dominio» (*public domain*), che da molti anni viene largamente adottata nella comunità degli informatici. Il problema è che un prodotto software di pubblico dominio può anche essere utilizzato per la realizzazione di software proprietario, così come è avvenuto anche per molti programmi liberi, che essendo distribuiti con licenze permissive (ad esempio BSD) sono stati trasformati in prodotti chiusi e proprietari.

Stallman non ritenne opportuno rendere GNU di pubblico dominio, anche se quella soluzione avrebbe consentito la massima diffusione del prodotto: un programma completamente di pubblico dominio avrebbe rischiato di estinguersi – limitandosi a essere un'entusiasmante esperienza creativa – una volta che le sue varianti fossero state utilizzate per un uso proprietario. In tal modo le relazioni cooperative si sarebbero spezzate, oscurando anche il valore insito nel prodotto iniziale, in quanto l'ultima versione aggiornata avrebbe avuto molte più probabilità di diffondersi, vanificando tutto il lavoro precedente. Il copyleft invece rende inalienabili e virali le libertà del Free Software. Molti programmatori aderirono al progetto e allo sviluppo dei software di Stallman, scrivendone di nuovi o rilasciando i propri sotto licenza GPL.

Si definiva così la GNU Economy: uno spostamento dell'interesse economico dal prodotto (software) al servizio (assistenza). I programmatori furono messi nella condizione di modificare un

loro programma per specifiche esigenze commerciali, e quindi divennero in grado di vendere una copertura di servizio difficile da ottenere per un software acquistato a scatola chiusa. Parecchi anni più tardi la New Economy non ha fatto altro che appropriarsi di questo e di molti altri aspetti dello sviluppo libero. Parte del fallimento della New Economy[22] è da attribuirsi sicuramente alla banalità con la quale ha utilizzato queste intuizioni, senza alcuna cognizione di causa, come se si trattasse di una semplice strategia per conquistare nuovi mercati e non di un movimento essenzialmente politico. Infatti, bisogna ricordare che le posizioni della FSF fin dall'inizio erano tese alla promozione della libera circolazione dei saperi e, attraverso la centralità dell'assistenza, alla liberazione dei programmatori dalla noia frustrante della routine lavorativa imposta dalla committenza, invece che dal proprio desiderio di creare nuovi strumenti.

Dopo dieci anni di lavoro, agli inizi del 1990, la comunità GNU aveva quasi ultimato il suo sistema operativo. Buoni compilatori, interfacce grafiche, editor di testi, giochi, software di rete, tutto ciò che esisteva per un sistema Unix degli anni Ottanta era disponibile anche nel lungo elenco degli applicativi GNU.

Ma non esiste un sistema operativo se non esiste il kernel. Il kernel[23] è il motore software per ogni computer ed è ciò che distingue un sistema operativo da un altro. La FSF tentò per anni e senza alcun successo di scrivere questo ultimo fondamentale elemento.

Molteplici le ragioni del fallimento del kernel HURD[24], l'elemento software mancante alla GNU per creare un sistema operativo completo: Stallman accusò l'architettura modulare scelta, palesemente troppo complessa da gestire e tuttavia già troppo sviluppata per decidere di accantonarla; altri invece considerarono responsabili i programmatori a causa dell'approccio gerarchico imposto alla comunità scrivente (coder, debuggers, ecc.).

Qualunque siano le cause di questa sconfitta, la storia volle che un giovane studente finlandese, Linus Torvalds, risolvesse la situazione con risultati che andarono ben al di là del cuore del sistema operativo. Nell'agosto del 1991 Linus iniziò la scrittura di un kernel ispirato ai sorgenti di Minix – un sistema operativo che Andrew S. Tannenbaum aveva concepito a scopo didattico – con una vera e propria chiamata alla armi sui principali newsgroup di programmazione. In due anni formò una comunità che con veloci

rilasci di nuove versioni, un'ampia cooperazione e la totale apertura delle informazioni consentì di testare estensivamente il codice.

Mentre Stallman si affannava inutilmente sul kernel HURD, Linus aveva spontaneamente dimostrato che per ottenere una performance migliore, quando si è svincolati da istituzioni premianti (le università che gratificano attraverso lauree e titoli di riconoscimento i ricercatori, oppure le aziende che offrono gratifiche monetarie, ecc.), è necessaria un'adesione a un complesso di valori capaci di coniugare il coinvolgimento comunitario e la passione tecnica.

Pubblicare i codici sorgenti prima ancora che l'applicativo fosse terminato, chiedendo immediatamente un feedback ai propri pari, agli individui che si ritengono in grado di comprendere e collaborare a qualsiasi livello, rappresentava per la FSF un processo nuovo e non privo di rischi, ma ebbe un successo tale da venire assunto collettivamente sotto il nome di metodo «bazar»[25], termine coniato da Eric S. Raymond, uno degli ex sviluppatori GNU, per sottolineare l'aspetto di scambio quasi caotico dello sviluppo.

Con l'ingresso della metodologia «bazar» usata per il kernel Linux, la comunità assumeva una configurazione completamente diversa. Al ruolo degli sviluppatori si affiancava quello di una miriade di funzioni periferiche con finalità di testing, documentazione, traduzione. Non occorreva più essere un mago del codice per diventare parte di una comunità, perché ogni aspetto era stato reso pubblico e trasversale. Anche il singolo utente poteva sapere come stava procedendo il progetto e contribuire con test e suggerimenti.

Il bazar di Torvalds aveva segnato il passaggio definitivo degli hacker dal luogo chiuso degli atenei e dei network di soli sviluppatori allo spazio sconfinato della rete di internet – uno spazio misto e spurio, in cui la contaminazione fra reale e virtuale diventa la norma e non l'eccezione[26].

In questo nuovo contesto la produzione di applicativi si apriva alla cooperazione di centinaia di potenziali partecipanti attivi. L'immagine di un centro pulsante di sviluppo a cui si può collaborare in maniera marginale, derivata in ultima analisi dal mito dello scienziato chiuso nel suo laboratorio, veniva spazzata via dal modello bazar. Le reti favorivano la deterritorializzazione dello sviluppo, per cui un luogo fisico univoco, il laboratorio, non era più indi-

spensabile: il lavoro deterritorializzato delle periferie, che producevano feedback a tutti i livelli (dalla documentazione al debug) si muoveva con lo stesso ritmo del cuore dello sviluppo (d'altra parte nemmeno gli sviluppatori codavano in luoghi vicini, certo non tutti nello stesso laboratorio). E non ci volle molto perché lo stesso metodo reticolare venisse esportato in nuovi progetti comunitari.

In brevissimo tempo si materializzarono comunità autonome per specifiche esigenze. Sono del 1995 i progetti Apache, uno dei server web più utilizzati, PHP, un linguaggio per web dinamico, e The Gimp, il programma di manipolazione di immagini della GNU. I principali desktop GNU/Linux, KDE e GNOME, nascono rispettivamente nel 1996 e nel 1997.

Fu infatti la grafica a rendere Linux così attraente ai non tecnici. Il porting del window server X[27] fu sicuramente un duplice successo: senza un ambiente grafico il nuovo sistema operativo non avrebbe mai avuto un'ampia diffusione e il server X sarebbe rimasto confinato ai sistemi Unix.

Con l'uscita delle prime distribuzioni[28], si affermò subito Linux come nome del sistema operativo. I software della FSF erano largamente in uso, ma non sembravano più avere particolare importanza politica: il ruolo della FSF veniva relegato a una battaglia passata e in parte vinta, almeno dal punto di vista della diffusione dei modelli aperti. La libertà passava in secondo piano.

Stallman dovette spesso ribadire che il nome corretto del nuovo sistema operativo era GNU/Linux e non Linux, poiché quest'ultimo è solo il nome del kernel, una delle tante parti essenziali di un sistema completo – in particolare l'unico che la FSF non era riuscita a creare. Ancora oggi questo errore concettuale viene commesso da molti[29].

Il 1998 fu l'anno del cambiamento. Per molti membri della comunità, GNU/Linux era pronto per diffondersi sui sistemi desktop fino ad allora monopolizzati da Microsoft (e da Apple, per quanto riguarda una ristretta ma fedele nicchia di utenti). Dopo la coniazione del termine Open Source e la pubblicazione di alcuni testi firmati da Raymond, O'Really e altri guru, si ottennero i primi risultati. In quello stesso anno la Netscape, famosa softwarehouse in concorrenza con il browser Explorer, rilasciò i codici sorgenti del proprio applicativo insieme a un nuovo nome: Mozilla.

3. E finalmente solo soldi! Nasce l'Open Source

Il termine Open Source nasce quindi per ragioni prettamente economiche. In inglese la parola *free* ha il duplice significato di libero e di gratuito. Tutte le comunità di sviluppo erano favorevoli alla libertà del codice, ma per alcuni *free* poteva essere inteso come gratuito ed essere quindi un ostacolo per la vendita. La soluzione a questo problema, non solo linguistico, fu la creazione ex novo di un termine più neutro politicamente e più appetibile per le imprese. Tuttavia, si trattava di un problema solo apparente: bisogna infatti notare che il progetto GNU, finanziato dalla FSF, non aveva particolari preclusioni al mercato. I software venivano sviluppati da programmatori appositamente stipendiati. Furono inoltre programmate altre componenti di sistema Unix, alle quali si aggiunsero le applicazioni più disparate, fino ai giochi. Questi programmi furono distribuiti a un prezzo di circa 150$, una cifra che, oltre a coprire i costi di riproduzione (il costo delle memorie di massa era centinaia di volte superiore a quello attuale), garantiva un servizio di supporto al cliente. L'unica condizione era che tutte le modifiche eventualmente effettuate su tali programmi venissero notificate agli sviluppatori. Il termine Open Source non ha dunque altra finalità se non far passare in secondo piano le libertà propugnate dalla FSF. E infatti la FSF non riconobbe mai il nuovo nome perché avrebbe dato adito all'ambigua idea secondo cui il software può essere accessibile, ma contemporaneamente non libero: semplicemente, aperto.

Dal 1998 in avanti, spinti dall'esempio di Netscape/Mozilla, alcuni progetti Open Source iniziarono a modificare il metodo di scrittura aperta, eliminando la relazione di dipendenza con gli altri progetti e strutturando le comunità in gerarchie sempre più verticali. Questi cambiamenti di metodo implicarono mutamenti radicali anche nel software finale (vedremo come nei prossimi capitoli). Banalizzando, lo status del software passò da creazione, frutto di un investimento creativo da parte di un coder o di un'intera comunità di sviluppo, a prodotto generato per soddisfare una richiesta di mercato (solitamente un bisogno indotto).

Fino ad allora le relazioni «a piramide» costituivano una parte minima della gestione dei rapporti interni alle comunità (per quanto la pratica della meritocrazia comporti di fatto dei rapporti

gerarchici). Con la svolta di Mozilla la gerarchizzazione divenne una condizione imprescindibile per quei progetti che si rendevano completamente autonomi dalle modalità di condivisione maturate nel periodo precedente.

Dopo la decisione della Netscape, molte altre aziende cercarono di stabilire legami con il mondo Open Source: IBM, SUN, Novell e HP sono di certo le più note. Ed è appunto quando il software libero comincia a essere chiamato più semplicemente e facilmente software aperto, in modo da entrare a pieno regime nella catena commerciale, che la nostra storia inizia.

Note al capitolo

1. È significativo notare che *release*, in inglese, significa in prima istanza «scarcerazione, liberazione» e solo in seguito ha assunto l'accezione di «rilascio» di una versione di codice o di una determinata caratteristica hardware. Tradurre quindi *release software* con «scarcerazione» o «liberazione di codice» non è affatto esagerato.

2. Naturalmente parliamo di masse di utenti nel cosiddetto «primo mondo»: gran parte della popolazione rimane ancora oggi esclusa dal paradigma digitale; si veda http://www.digital–divide.it/

3. BASIC: http://it.wikipedia.org/wiki/Basic; pascal: http://it.wikipedia.org/wiki/Pascal_%28linguaggio%29; assembly: http://it.wikipedia.org/wiki/Assembly

4. Più precisamente, si tratta di «computer hacking»; per una storia del termine hacker, si veda Sam Williams, *Codice Libero – Richard Stallman e la crociata per il software libero*, Apogeo, 2003, disponibile in rete all'indirizzo http://www.apogeonline.com/ebooks/2003/90045/CodiceLibero/

5. http://it.wikipedia.org/wiki/BBS

6. Il CCC nasce nel 1981 per iniziativa di Wau Holland e Steffen Wernery ed è poi formalizzato nel 1984; si veda http://www.ccc.de/

7. www.decoder.it

8. www.hacktic.nl

9. Il progetto ECN, di cui si discuteva dal 1988, coinvolgeva fin dall'inizio gruppi italiani, inglesi, tedeschi e olandesi; nasce di fatto nel 1990, in particolare in diversi nodi italiani; si veda www.ecn.org

10. SITOGRAFIA: *La cultura cyberpunk*; si veda il portale http://www.dvara.net/HK/ e l'ipertesto http://www.decoder.it/archivio/cybcult/index.htm

11. Dichiarazione iniziale: http://www.dvara.net/HK/icata89bis.asp; dichiarazione finale: http://www.strano.net/wd/cr/cr001.htm

12. SITOGRAFIA: *Unix*. http://it.wikipedia.org/wiki/UNIX; repository: ftp://ftp.autistici.org/Unix-mirror (primi codici di Unix); si veda anche la storia di Unix nel Corso Base di Unix, documentazione dei corsi di Unix del LOA Hacklab di Milano http://www.autistici.org/loa/web/doc/corsounix.pdf

13. In informatica un file system è, informalmente, un meccanismo con il quale i file sono immagazzinati e organizzati su un dispositivo di archiviazione, come un hard disk o un CD-ROM. Più formalmente, un file system è l'insieme dei tipi di dati astratti necessari per la memorizzazione, l'organizzazione gerarchica, la manipolazione, la navigazione, l'accesso e la lettura dei dati.

14. BSD è un sistema operativo libero largamente utilizzato da sistemisti, hacker e programmatori. Ispirato anch'esso da Unix, come GNU/Linux, rispetta lo standard POSIX; http://www.ibsd.org

15. Basti pensare all'influenza di personaggi come Marvin Minsky o John MacCarthy, teorici e fautori dell'intelligenza artificiale, entrambi professori al MIT a partire dagli anni Sessanta; si veda anche: Steven Levy, *Hackers*, ShaKe, Milano, 1996. Elaborazioni molto più note al grande pubblico, come quelle di Pekka Himanem e dello stesso Linus Torvalds, derivano da analisi assai più tarde e sostanzialmente debitrici dell'esplosione del fenomeno Linux; si veda Pekka Himanem (con Linus Torvalds e Manuel Castells), *L'etica hacker e lo spirito dell'età dell'informazione*, Feltrinelli, Milano, 2001.

16. Solaris: http://www.sun.com/software/solaris/; Irix: http://freeware.sgi.com/

17. Gli accordi di non divulgazione sono sempre più diffusi in ogni settore; si veda NDA (Non-Disclosure Agreement) http://en.wikipedia.org/wiki/Non-disclosure_agreement

18. SITOGRAFIA: *Lo standard POSIX* e la guerra tra IBM, SUN, Novell e altre società che commercializzavano Unix; si veda anche http://www.lilik.it/~mirko/gapil/gapilsu8.html

19. IEEE: Institute of Electrical and Electronics Engineers, www.ieee.org

20. CP/M, Control Program for Microcomputers, sistema operativo sviluppato nel 1975 dalla Digital Research. Nel 1979 Tim Paterson, della Seattle Computer Product scrisse l'86-DOS. Contemporaneamente l'IBM stava pensando di costruire un Personal Computer. Fu allora che Bill Gates, presidente della Microsoft, si incontrò con l'IBM (la quale nello stesso momento contattò anche la Digital Research). L'IBM non raggiunse un accordo con la Digital Research, mentre la Microsoft acquistò i diritti per la distribuzione dell'86-DOS di Tim Paterson e trattò con successo con IBM: fu così che l'MS-DOS (leggi 86-DOS) divenne il sistema operativo del nuovo Personal Computer IBM. Infine, nell'aprile del 1981, l'IBM an-

nunciò l'uscita del Personal Computer IBM (che adottava la CPU Intel 8088), con il sistema PC-DOS (MS-DOS con alcuni programmi aggiuntivi).

21. La definizione originale di copyleft, formulata nel 1988, si trova all'indirizzo http://www.gnu.org/copyleft/copyleft.html; in italiano, una buona introduzione è http://www.wumingfoundation.com/italiano/outtakes/copyleft_booklet.html

22. Non si parla più oggi di New Economy, ma sicuramente pratiche come regalare i decoder satellitari (magari grazie a contributi governativi) per poi far pagare i programmi sono applicazioni su vasta scala del principio della vendita dei servizi (le trasmissioni televisive, i servizi legati alla telefonia cellulare, ecc.) piuttosto che dei prodotti. In fondo, l'esplosione stessa dei call-center deriva appunto da una spinta alla vendita di servizi, al *customer care*, ecc.

23. Questo software ha il compito di fornire ai processi in esecuzione sull'elaboratore un accesso sicuro e controllato all'hardware. Inoltre regola la gestione del filesystem, i permessi degli utenti, l'avvio di nuovi processi. Poiché possono essere eseguiti simultaneamente più processi, il kernel ha anche la funzione di assegnare una porzione di tempo-macchina e di accesso all'hardware a ciascun programma (multiplexing).

24. Il progetto kernel della GNU (HURD) si basa su una struttura a micro-kernel tanto complessa quanto elegante. A cavallo fra gli anni Ottanta e Novanta furono pubblicati molti scritti sulla progettazione di kernel, tra cui vale la pena citare Andrew S. Tannenbaum (l'inventore di Minix, da cui Linus Torvalds ha sviluppato Linux), *Operating Systems: Design and Implementations*, Prentice Hall, 1987 (n.e. *Modern Operating Systems*, Prentice Hall, 2001). La struttura a micro-kernel era definita un'architettura molto più performante rispetto alla monolitica approcciata da Linus Torvalds. SITOGRAFIA: *Linux is obsolete*. http://www.kde.org/history/Linux_is_obsolete.php

25. Eric S. Raymond, *The Cathedral and the Bazar*, http://www.catb.org/~esr/writings/cathedral-bazar/ [trad. it.: *La cattedrale e il bazar*, http://www.apogeonline.com/openpress/doc/cathedral.html].

26. Nel passaggio da comunità omogenee (coder GNU, progetti monolitici) a comunità eterogenee (GNU/Linux, progetti distribuiti) i linguaggi si moltiplicano necessariamente e di conseguenza si moltiplicano anche le occasioni di incontro. In questa espansione anche il mercato, e il suo linguaggio, trova spazio e possibilità di sviluppo.

27. X nasce nei laboratori del MIT come gestore di finestre grafico. Era rilasciato gratuitamente e senza una licenza che limitasse l'utilizzo o ne regolasse il copyright. SITOGRAFIA: *La storia di Xfree*.

28. Una distribuzione (distro) è una raccolta di software scelti e preconfigurati.

Le principali usate oggi sono: Debian, Redhat, SuSE, Mandrake, Gentoo e alcune altre centinaia.

29. Curiosa la posizione di Linus a riguardo: sostenne che non tutto il resto del sistema operativo era GNU quindi chiamare GNU/Linux il tutto significava non riconoscere adeguatamente l'apporto di altri applicativi come X.

II

LICENZE, POLITICA, MERCATO

1. Free Software != Open Source

Free Software[1] (software libero) e Open Source[2] (sorgente aperto) sono sintagmi usati spesso come sinonimi per indicare codici o porzioni di codici; rispecchiano tuttavia prospettive radicalmente differenti. Free Software è un termine nato agli inizi degli anni Ottanta per iniziativa di Richard Stallman (vedi cap. I): si riferisce alla libertà dell'utente di usare e migliorare il software. Più precisamente, può essere riassunto in quattro libertà fondamentali:

1) Libertà di eseguire il programma, per qualsiasi scopo.

2) Libertà di modificare il programma secondo i propri bisogni (perché questa libertà abbia qualche effetto in pratica è necessario avere accesso al codice sorgente del programma, poiché apportare modifiche a un programma senza disporre del codice sorgente è estremamente difficile).

3) Libertà di distribuire copie del programma gratuitamente o dietro compenso.

4) Libertà di distribuire versioni modificate del programma, così che la comunità possa fruire dei miglioramenti apportati.

L'espressione Open Source, invece, nasce alla fine degli anni Novanta per iniziativa, in particolare, di Bruce Perens ed Eric S. Raymond, che nel 1998 fondano la Open Source Initiative[3] (OSI); la quale fa riferimento alla Open Source Definition, a sua volta derivata dalle Debian Free Software Guidelines, ovvero una serie di punti pratici che definiscono quali criteri legali debba soddisfare una licenza per essere considerata effettivamente «libera»: o meglio, con il nuovo termine, Open Source.

È evidente quindi che da una parte il Free Software pone l'accento sulla libertà: come sottolineato nella definizione, «il Software libero è una questione di libertà, non di prezzo»[4]. Dall'altra parte, l'Open Source si occupa esclusivamente di definire, in una prospettiva totalmente interna alle logiche di mercato, quali siano le modalità migliori per diffondere un prodotto secondo criteri open, cioè aperti. Il Free Software ha un senso che va ben oltre il mercato, pur non escludendolo a priori; l'Open Source esiste, come specificato dai suoi stessi promotori, per adattare un modello preesistente, quello free, al mercato.

Comunità hacker, FSF, progetto GNU: erano questi i primi soggetti protagonisti di un percorso che dopo aver rilanciato nelle menti e nelle reti l'idea di condivisione, insistendo sul concetto di permesso d'autore (copyleft), avevano messo in crisi l'apparente inattaccabilità della proprietà privata, almeno nella sua applicazione al software.

Negli anni Ottanta e Novanta l'importanza dello sviluppo di software crebbe in maniera esponenziale (mentre in precedenza era l'hardware la componente principale e più costosa di un computer): aumentò di conseguenza l'attenzione delle aziende nei confronti del Free Software, delle comunità hacker e dei suoi metodi. Il mercato di riferimento, infatti, dopo aver gettato le basi per la diffusione del software close (vedi cap. I), vedeva crescere il colosso Microsoft che, attraverso la stipula di contratti commerciali capestro e accorte manovre di lobbying, aveva condotto il mercato a una fase sostanzialmente monopolistica[5].

Le implicazioni di questo fenomeno non sono solo tecniche, ma permeano l'etica, la filosofia e necessariamente la sfera del politico, oltre che quella dell'economico. L'avvicinamento del mondo commerciale al mondo delle comunità hacker non poteva infatti non intaccare l'etica che sta alla base del movimento Free Software[6]. La radicalità politica del Free Software, questa insistenza sulle libertà, era piuttosto scomoda e controproducente dal punto di vista della sua diffusione al di fuori dall'ambito hobbistico o universitario. Inoltre in inglese era facilmente fraintendibile per via del doppio significato di «software libero» e di «software gratuito»[7]; a ragione o a torto, il termine Free era visto come fumo negli occhi perché legato a correnti di pensiero poco apprezzate nel mondo aziendale. Troppa «libertà» fa male al «libero» mercato: l'Open Source entra allora in gioco.

Il cambiamento – dapprima lessicale – presupponeva punti di vista e possibili evoluzioni diverse rispetto al periodo precedente: non solo il termine free non piaceva alle aziende che cercavano quote di mercato, ma anche la licenza GPL non era soddisfacente, perché il suo sistema di diffusione virale rischiava di lasciare fuori dall'accelerazione digitale molte imprese, che vedevano nelle comunità hacker uno sbocco naturale delle proprie strategie economiche.

Open Source era un termine che rispondeva perfettamente a queste esigenze: una volta avvicinate le imprese, si trattava solo di dimostrare come le logiche organizzative e produttive delle comunità da una parte e il profitto dall'altra potessero andare d'accordo. Diventava così molto semplice ampliare il bacino delle aziende impegnate a entrare nel «fantastico mondo dell'Open Source».

Il nuovo movimento, meno politicizzato e più orientato alle possibilità commerciali di quello che era allora definito Free Software, si impose. Molte le ragioni concomitanti che facilitarono il passaggio concettuale e pratico: lo scossone dovuto alla nascita di Linux; alcuni passaggi a vuoto della FSF, spesso chiusa su posizioni piuttosto ingenue e di nostalgia per un'età dell'oro di condivisione e libertà delle conoscenze; la paralisi delle comunità hacker più radicali (che negli anni Settanta avevano dato luogo a mitici gruppi come l'Homebrew Computer Club nella Bay Area di San Francisco); la potenza intrinseca del movimento Open

Source, spalleggiato fin da subito da imprese che avevano subodorato l'affare. Lo scenario politico del mondo hacker veniva modificato rapidamente e profondamente.

Lo strappo si completò con la formulazione di nuove licenze che si ponevano in concorrenza con la GPL. L'outing di alcune società che rilasciarono il loro codice sorgente, Netscape *in primis*, rappresentò il contributo decisivo: il Free Software e la sua filosofia, con una forzatura palese ma funzionale, vengono relegate nell'angolo dell'estremismo, dei progetti non sostenibili economicamente; dall'altra parte l'Open Source conquista il mercato anche a livello di credibilità in termini di business.

Nonostante queste differenze palesi, i fraintendimenti sono molti. Proviamo allora a specificare e delimitare meglio termini e concetti. Si può definire l'Open Source come quel metodo di progettazione e produzione (di software, ma non solo) basato sulla condivisione di ogni aspetto e passaggio (tipico delle comunità hacker e del Free Software) che, nell'ottica di sviluppare un prodotto open, accetta al proprio interno componenti e passaggi closed (comunità ristrette e chiuse, in alcuni casi anche software proprietario) per garantire maggiore competitività sul mercato ai propri prodotti. Nulla di troppo differente da quanto le imprese avessero fatto fino ad allora: il lavoro di équipe e la condivisione di obiettivi comuni non è certo una novità[8].

La differenza tra queste due componenti delle comunità virtuali hacker è comprensibile seguendo le evoluzioni delle licenze che nel tempo hanno cercato di regolare la distribuzione di prodotti Free Software, garantendone da un lato i principi di riproducibilità, modifica, distribuzione e dall'altro cercando, con l'avvento dell'Open Source, di linkarlo più agevolmente al mercato e alle sue regole[9].

Quello sulle licenze è il primo livello di investigazione circa le effettive novità e l'originalità dell'approccio hacker – nel mondo del software ma non solo. Immaginiamo per un attimo una comunità di individui che si definiscono hacker: le licenze d'uso rappresentano il livello più esterno, di relazione verso coloro che non appartengono alla comunità, che per lo più non sono hacker e non condividono un'etica e una storia comune. Quando dalla GPL si passa a licenze formulate esclusivamente per il mercato, ovvero del tutto aliene allo spirito hacker, appare evidente che bisogna

scavare molto più a fondo. Useremo allora una sorta di lente d'ingrandimento, passando dalle licenze alle comunità, fino all'individuo: abbozzare una cartografia di questi tre differenti livelli ci permetterà di delineare possibili vie di fuga dal virtuale al reale, cioè di immaginare nuove possibili ricadute dell'hacking sulla realtà.

2. La licenza: che cos'è?

La parola licenza deriva dal latino *licentia*, da *licire*, «essere lecito»; nell'accezione che ci riguarda significa: «concessione, da parte di un organo competente, di una determinata autorizzazione; anche, il documento che comprova l'autorizzazione concessa: licenza di esercizio...»[10].

Le licenze infatti specificano una concessione da parte di un soggetto nei confronti di qualcun altro. Nel caso del software «concedono» l'utilizzo del codice alle condizioni stabilite da chi redige la licenza: «licenziare» significa «concedere il potere specificato».

In ambito informatico la licenza è un contratto tra il detentore del copyright e l'utente. Si parla quindi propriamente di «licenza d'uso». La licenza è una sorta di «certificato» che l'autore pone al proprio prodotto per salvaguardarne la sicurezza, la distribuzione e naturalmente rendere noti i propri meriti. Un software rilasciato senza alcun testo che «determini» un utilizzo del prodotto stesso rischierebbe infatti di finire nelle mani di qualcuno che potrebbe arricchirsi indebitamente, farne un uso non etico, non rispettare la volontà del creatore del prodotto.

Le licenze software, nel loro complesso, rappresentano una cartina tornasole per misurare l'evoluzione delle comunità hacker e verificare l'impatto della nascita e dello sviluppo del concetto di Open Source.

Ogni movimento ha bisogno di qualcosa che sancisca i passaggi salienti della propria vita: le licenze software hanno spesso segnato qualcosa di più di un cambiamento, coagulando un *modus operandi*, uno stile, un background culturale e politico.

Le licenze, dalla GPL alle ultime rilasciate, ripercorrono la storia del Free Software e dell'Open Source in alcuni casi incidendo

profondamente su di esse (non solo la GPL, ma anche altre: si pensi solo alla Mozilla Public Licence, MPL[11]): differenti per specificità e finalità, prendono le mosse da percorsi etici e culturali anche molto eterogenei. Le licenze adottate sottolineano con forza le molte differenze tra le varie comunità ed evidenziano l'intreccio di finalità diverse in un breve lasso di tempo. Gli anni di rilascio delle varie licenze, infatti, tracciano la mappa temporale e concettuale del processo che ha portato all'egemonia del termine Open Source e delle sue pratiche.

Le discussioni in merito alle licenze, sul «permesso», sui «poteri», spaziano ovviamente ben oltre i limiti delle reti informatiche. Innumerevoli le visioni, le note, le caratteristiche riscontrate, gli usi, le preferenze. Nell'ottica di analizzare i rapporti di potere e le evoluzioni dell'Open Source le licenze sono da un lato tentativi di regolamentare qualcosa che fino a quel momento non aveva delle regole, dall'altro veri e propri scatti concettuali da parte di particolari gruppi di «potere» all'interno del mondo delle comunità hacker.

In un racconto di Philip K. Dick, già rivisitato dal cinema[12], un ingegnere elettronico ha il compito di carpire i segreti della concorrenza e provvedere a concepire nuovi prodotti tecnologicamente migliori. I suoi datori di lavoro si tutelano a loro modo dalla possibilità che l'ingegnere venga «assoldato» da altre aziende, cancellando al loro dipendente la memoria.

Dick, straordinario nel suo oscuro scrutare[13], percepisce con largo anticipo la nascita di un'economia in cui la «protezione» delle conoscenze sarebbe diventata fondamentale[14], anche a costo di limitare, castrare e distruggere gli individui stessi.

L'economia del XX secolo si è basata sempre più sul concetto di «protezione delle idee» e «proprietà intellettuale»: un miscuglio assurdo di copyright, brevetti, marchi registrati e accordi di segretezza che ha prodotto tra l'altro infinite e costose cause legali. In teoria tutti questi strumenti erano stati approntati per tutelare le creazioni e diffondere meglio le conoscenze; nei fatti si sono sempre rivelati armi a doppio taglio, come del resto le stesse licenze apparse negli ultimi anni, che vanno esattamente in questa direzione: evitare un eccessivo arricchimento economico da parte di chi si impossessa del software, garantendo quindi la libertà del software e allo stesso tempo tutelando l'autore del programma da

tentativi di appropriazione (anche a scopi economici) del proprio software. Nel XXI secolo, la cooptazione da parte delle aziende dei metodi di sviluppo free, diventati open, dimostrano in maniera lampante come la «protezione» debba cedere il passo alla libera circolazione dei saperi.

3. Free Software e GPL

Abbiamo visto che la licenza GPL (General Public Licence) nasce dal progetto GNU di Richard Stallmann.

Stallmann si pose il problema di come distribuire il software libero, provando dapprima attraverso un semplice spazio ftp del MIT. Questo risolveva il problema della distribuzione, ma non garantiva che il software rimanesse libero. Chiunque infatti da quel momento poteva comprare il software e adottarlo come prodotto della propria azienda, commercializzandolo e appropriandosi dei diritti. Proprio questo era l'inconveniente del *public domain* che Stallmann e migliaia di coder volevano evitare: la possibilità che il proprio software finisse per essere utilizzato in progetti commerciali e proprietari, o ancor peggio, in progetti eticamente discutibili. Mancava cioè la possibilità per il programmatore di avere delle valide difese rispetto a un uso non solo commerciale, ma anche irresponsabile, del proprio prodotto. Fin dall'inizio quindi il problema di Stallmann non era la «sostenibilità» economica del Free Software (era ovvio, dal suo punto di vista, che il FS fosse migliore, anche commercialmente, del software close), quanto piuttosto la salvaguardia della sua libertà attraverso una sorta di copyright del proprio autore.

Stallmann adottò allora quello che definì permesso d'autore, giocando sul significato della parola copyleft: anziché privatizzare il software, lo rendeva libero, fornendo tutta una serie di garanzie nei confronti dell'autore del software.

Così nasce la GPL[15]: l'idea centrale è di dare a chiunque il permesso di eseguire, copiare, modificare e distribuire versioni modificate del programma, ma senza dare la possibilità di aggiungere restrizioni. La GPL è virale perché le libertà di cui è intriso il software sono garantite a chiunque ne abbia una copia e diventano «diritti inalienabili»: il permesso d'autore infatti non sarebbe

efficace se anche le versioni modificate del software non fossero libere.

In questo modo Stallmann «garantiva» il creatore del software e la comunità di riferimento stessa, perché ogni lavoro basato sul prodotto originale sarebbe stato sempre disponibile, libero, aperto per tutta la comunità.

La GPL e Stallmann in quel momento rappresentavano lo stato avanzato dell'etica hacker e il primo vero tentativo di «dichiarare autonoma e replicabile» l'esperienza della condivisione e del superamento del settecentesco copyright; era inappropriato e stupido applicare il concetto di proprietà intellettuale al mondo software. Abbiamo visto però che Stallmann, uno dei personaggi chiave della storia del movimento hacker, non riuscì a vincere la sfida tecnologica che si poneva: creare un sistema operativo potente e flessibile come Unix, ma libero.

Dopo l'esplosione di Linux, all'inizio degli anni Novanta, apparve chiaro che Stallmann e il suo Free Software non erano più gli unici soggetti in movimento nel mondo delle comunità digitali, anzi: erano considerati improvvisamente quasi arcaici relitti dell'epoca d'oro dell'hacking, surclassati dal successo mondiale di Linux. Ormai il processo di espansione di internet, principale veicolo della diffusione del software, era inarrestabile, e contestualmente i limiti e l'agilità con cui la comunità hacker si apriva al commerciale venne incarnata anche dalla nascita del termine Open Source. Eppure sappiamo che la GPL stessa non escludeva applicazioni commerciali con finalità di business.

4. Open Source e nuove licenze

Nel 1997 Bruce Perens ed Eric Raymond, in una conferenza in posta elettronica, sanciscono la definizione di Open Source, specificando che il termine non significa solo «accesso al codice sorgente», ma si estende a numerosi altri aspetti. Il radicalismo politico del Free Software non era attraente per molte imprese produttrici di software, che tra l'altro temevano di perdere il proprio capitale intellettuale; la Open Source Definition (OSD) cambia decisamente le carte in tavola.

Alcune precisazioni. Innanzi tutto la OSD sottolinea di non pre-

vedere nessuna discriminazione dei settori che possono utilizzare software Open Source: la licenza non proibisce ad alcuno l'uso del programma in uno specifico campo o per un determinato proposito. Nessuno può proibire che il software venga utilizzato per scopi commerciali o scopi non considerati etici. La precisazione degli autori della OSD è piuttosto chiara: il software dev'essere impiegabile allo stesso modo in una clinica che pratichi aborti e in un'organizzazione antiabortista.

A chi fa notare l'incongruenza o, quanto meno, i rischi di tali affermazioni, viene fornita una risposta di una sconcertante apoliticità: queste discussioni politiche sono di pertinenza del Congresso degli Stati Uniti, si sostiene, non delle licenze del software. Delega totale. D'altro canto per i fautori dell'Open Source, la GPL, che pure viene consigliata in molti casi, viene considerata con malcelato disprezzo un mero «manifesto politico»[16].

Lo scossone del 1998, quando Netscape rende pubblici i propri codici sorgenti e nasce il progetto Mozilla, è strettamente legato alla questione delle licenze: la MPL, pur essendo approvata dalla OSI come licenza Open Source, specifica che chiunque «è libero di creare un prodotto commerciale utilizzando il codice Mozilla», e nella licenza il primo punto è dedicato proprio a questo aspetto («*1.0.1. 'Commercial Use' means distribution or otherwise making the Covered Code available to a third party*»).

Il passaggio da Free Software a Open Source è compiuto, non solo a livello semantico e linguistico ma anche a livello di implicazioni commerciali: dalla libertà al primo punto, siamo giunti a mettere il mercato avanti a tutto. Se infatti accettassimo come reale la distinzione operata da Raymond tra «fanatici» del Free Software e «moderati» dell'Open Source, il percorso concettuale apparirebbe lineare: inizialmente il Free Software porta alla ribalta la parte «fanatica» delle comunità hacker, coloro cioè che hanno sempre specificato la libertà del codice senza accettare compromessi commerciali, se non a livello di distribuzione.

La distinzione di Raymond è semplicistica ed è evidentemente sbilanciata a favore dell'Open Source. Egli infatti si pone tra i «moderati» (termine conciliante e sempre buono per chi non vuole apparire scomodo) e definisce gli altri «fanatici» (letteralmente, «ispirati da un dio» o «presi da delirio»: nulla di positivo, in genere). Fanatiche sarebbero, per Raymond, quelle comunità

hacker che avevano appoggiato la FSF e il suo percorso e che mal digerivano la piega conciliante con il mercato intrapresa dal movimento Open Source. Lo stesso Raymond, per avvalorare la propria tesi, richiama a sé proprio Linus Torvalds: questi, al pari degli estensori della OSD, ha specificato di apprezzare anche l'utilizzo di software proprietario, per compiti specifici, deridendo «gentilmente quegli elementi più puristi e fanatici all'interno della cultura hacker»[17].

La distinzione è quindi più che altro funzionale all'Open Source, non certo oggettiva. La realtà è ben più complessa di quanto un'etichetta possa racchiudere: nella storia del movimento hacker si è assistito alla nascita di un'etica che ha messo in crisi il concetto di proprietà intellettuale, in un'epoca e in un settore, il software, di riproducibilità tecnica totale. Una delle conseguenze maggiori di questa forte politicità, ovvero di questa prassi fortemente orientata e innovativa, è stato l'avvicinamento di alcune comunità ad aziende e imprese che hanno fiutato nei metodi e nello sviluppo di software delle comunità una valida opzione per creare mercato e profitti. Nel prossimo capitolo ci occuperemo di questo incontro tra comunità digitali e mondo commerciale.

Distinguere tra fanatici e moderati significa mettere in un angolo i primi, considerare nel giusto i secondi, mentre naturalmente la questione è più sfumata: Free Software e Open Source sono entrambe componenti di peso delle comunità hacker, e la seconda è un'evoluzione della prima, ma non necessariamente un miglioramento.

La distinzione non tiene inoltre conto del mondo hacker nella sua complessità: esiste infatti una «massa grigia» che possiamo considerare come una parte poco interessata non solo alle implicazioni politiche e filosofiche che il Free Software aveva portato alla luce, ma anche alle stesse implicazioni *business oriented*. Unica esigenza di questa «massa»: poter codare in maniera sicura avendo garantita la GPL e poter praticare quella condivisione connaturata al mondo delle comunità di software libero.

Linux, come del resto la MPL, rappresentavano dunque queste parti delle comunità che vedevano nella GPL solo un mezzo, uno strumento, e non una riflessione etica, politica e filosofica applicata al software. E si sono rapidamente mosse verso una «democratizzazione» (a fronte del monopolio Microsoft) del mercato del digi-

tale, partendo dal software commerciale, ma comunque libero, e cabotando pericolosamente lungo la zona commerciale non-libero.

L'Open Source ha sviluppato fin da subito marketing in grado di irretire queste componenti; nello stesso tempo, il movimento Free Software, poco in grado di cogliere immediatamente questi cambiamenti, si è trovato presto in grave difficoltà nel proporre soluzioni al passo con i tempi.

In realtà, anche tralasciando divisioni artificiose che quasi mai sono applicabili a una realtà in costante evoluzione, pare invece più fondato e realistico il passaggio concettuale che le imprese sono riuscite a far attecchire all'interno delle comunità: l'Open Source infatti non rappresenta altro che un'assunzione del rischio imprenditoriale da parte tanto dei programmatori quanto delle aziende. Le licenze nate dopo il 1998 sanciscono questo passaggio.

Una società in crisi può «guarire» passando all'Open Source, perché le licenze (che per definizione cercano di regolare il mercato, non uscendo, in pratica, dalla sua logica) permettono affidabili recuperi di denaro e competenze in una sana, apparentemente democratica, economia di mercato. In fondo anche la monopolista Microsoft, in un certo senso, si sta muovendo verso l'Open Source (certo non verso il Free Software). Infatti Microsoft ha risposto alla crescente richiesta da parte delle pubbliche amministrazioni e delle grosse aziende di poter visionare il codice che utilizzano varando l'iniziativa denominata *shared source*: a pagamento (ma gratuitamente per le pubbliche amministrazioni), consente di avere accesso ai sorgenti[18].

Le licenze quindi possono essere definite addirittura «metronomi» del processo di avvicinamento del mondo commerciale all'Open Source: più il legame si è fatto stretto, più sono nate licenze in grado di permettere, mantenendo il principio di libertà di accesso al codice, una collaborazione effettiva tra comunità Open Source e progetti commerciali.

5. Non siamo fanatici: «Ci va bene qualunque licenza»...

«Ma non intendiamo farne una questione ideologica. Ci va bene qualunque licenza che ci permetta il controllo del software che usiamo, perché questo ci mette in grado a nostra volta di of-

frire ai nostri clienti e utenti il vantaggio del controllo, siano essi ingegneri della NASA o sviluppatori di applicazioni gestionali per studi dentistici»[19].

Robert Young, spiegando la nascita di Red Hat, una delle principali distribuzioni commerciali di Linux, è molto chiaro: ci va bene qualunque licenza, il mercato lo abbiamo già. Questo per l'economia globale è decisamente un bel passo in avanti, significa partire con il vento in poppa, un po' come pescare in una vasca da bagno già colma di pesci: gli utenti – anzi i consumatori – ci sono, dateci gli strumenti per raggiungerli...

Alcune licenze costituiscono semplicemente una sorta di «lasciapassare» che una comunità più incline al richiamo commerciale recepisce per favorire il passaggio all'Open Source, visto come un modello di capitalismo alternativo, magari addirittura dal volto umano. La differenza tra software proprietario e Open Source nel «fare soldi» è infatti minima: si crea un grande prodotto, si fa del marketing creativo e accattivante, si genera «il bisogno», ci si cura degli utenti/clienti e si costruisce l'affidabilità del prodotto e di chi lo produce.

L'Open Source in questo modo diventa un marchio qualunque, un logo, un simbolo di affidabilità, qualità e coerenza, e come tale porta profitti. Robert Young chiarisce molti aspetti relativi alle licenze e al loro attuale valore, formulando una dichiarazione di principi elastici quanto basta per consentire all'Open Source di affermarsi come modello economico di salvezza per un capitalismo in crisi di crescita.

Dal closed all'Open Source: cambia poco nei metodi di affermazione di un prodotto, ma cambia moltissimo nel mondo monopolistico del software. Il vantaggio di una marca che produce in un determinato settore è infatti l'ampiezza del proprio mercato, prima ancora delle quote che la società produttrice detiene: più utenti GNU/Linux in circolazione significa più clienti potenziali per Red Hat.

I rapporti di potere all'interno delle comunità hacker si rispecchiano quindi anche nell'ambito delle licenze; nel giugno del 2003 ad esempio la Red Hat è stata accusata di «uscire dalla GPL»: alcune parti dell'End User License Agreement (EULA) del software Red Hat Advanced Server sembrerebbero essere in netto disaccordo con la licenza GPL[20].

Questo chiarisce inoltre come il capitalismo sia disposto ad assumere l'Open Source come alternativa economica all'attuale sistema monopolistico di Microsoft Windows e come parti di questo mondo cerchino costantemente strade per aprire nuovi mercati, per elaborare nuove strategie, per fare nuovi profitti.

L'Open Source in definitiva è solo il tentativo (fino a ora riuscito) di adattare il metodo altamente produttivo delle comunità hacker, che rispetti la condivisione, ma che sia più elastico rispetto a ingerenze economiche commerciali. Questa elasticità, o flessibilità, è ormai usuale nei metodi di lavoro ed è sancita, verso l'esterno, con le licenze. L'Open Source rappresenta la vittoria del modello liberale sul modello libertario del Free Software: l'unica libertà che si preserva, in fondo, è quella del mercato.

Ogni tipo di licenza si presta a usi commerciali, si tratta di scegliere quella più adatta al proprio business. Tra il 1997 e il 1998, infatti, Linux, FreeBSD, Apache e Perl hanno visto crescere l'attenzione di personaggi provenienti dal mondo commerciale (manager di progetto, dirigenti, analisti d'industria e venture capitalist). La maggior parte degli sviluppatori ha accolto quest'attenzione con molto favore: un po' per orgoglio, un po' per motivare i propri sforzi e, non ultimi, i propri compensi.

6. Nella pratica... modelli di business molto «aperti» e molto poco «liberi»

I moderati dunque spopolano. Molti ritengono ormai che il modello Open Source sia davvero un modello affidabile per la conduzione dello sviluppo software a scopi commerciali. Un'analisi ad ampio raggio delle licenze disponibili rileva che la BSD è, da un punto di vista commerciale, la migliore quando si debba intervenire su un progetto preesistente.

La BSD[21] in pratica specifica che il codice può essere utilizzato come meglio si crede, salvo darne credito all'autore. La licenza BSD fu scelta inizialmente da Apache, progetto nato da webmaster di ambiti commerciali che cercavano un web server in grado di ottimizzare le proprie esigenze di business. La BSD, molto semplicemente, permette l'inserimento di prodotti all'interno di software proprietari: per molti è una licenza pericolosa, volta a

minimizzare l'impatto etico della cosiddetta economia del dono.

La licenza BSD, o X e affini, permette di fare ciò che si vuole del software licenziato. Questa «liberalità», spesso utilizzata contro la viralità «politicante» della GPL, ha radici precisamente politiche: il software originariamente coperto dalle licenze X e BSD era sovvenzionato con sussidi del governo degli Stati Uniti. Dal momento che i cittadini statunitensi avevano già pagato il software con i soldi delle tasse, fu loro garantito il diritto di fare del software tutto ciò che volevano...

La BSD è un esempio calzante anche della possibilità di modifica delle licenze. Un dettaglio negativo era costituito da una clausola che fu poi espunta: la BSD prescriveva infatti che, quando si faceva cenno in una pubblicità a una caratteristica di un programma sotto BSD, si esplicitasse il fatto che il software era stato sviluppato all'Università della California. Era evidente la problematicità di questa clausola per distribuzioni contenenti migliaia di pacchetti software.

La prima licenza, vicino alla BSD, che ha portato novità importanti è stata quella di Mozilla: tale licenza prescrive che le modifiche alla distribuzione siano rilasciate sotto lo stesso copyright della MPL, rendendole nuovamente disponibili al progetto. Evidente in questo caso il rischio di utilizzo di software proprietari: se per distribuzione si intendono i file così come vengono distribuiti nel codice sorgente, questo consente a un'azienda di aggiungere un'interfaccia a una libreria di software proprietario, senza che quest'ultimo sia sottoposto alla MPL, ma solo l'interfaccia.

Anche la GPL consente utilizzi commerciali, per quanto solo di riflesso e non esplicitamente nel testo: diciamo che il mercato non è il problema principale. Infatti, la GPL prescrive che «gli incrementi, i derivati e perfino il codice che incorpora altri codici» vengano rilasciati sotto GPL; per questo motivo la GPL viene definita «virale», per consentire che il codice che nasce free rimanga free.

Tuttavia anche la GPL può agire come «eliminatore» di concorrenti in un determinato progetto: con la GPL, come dimostra l'esempio di Cygnus, citato da Brian Behlendorf (uno dei leader del progetto Apache), «non c'è speranza che un concorrente possa trovare una nicchia tecnico-commerciale che sia sfruttabile con la struttura GCC senza dare a Cygnus la stessa opportunità di approfittare anche di quella tecnologia. Cygnus ha creato una

situazione in cui i concorrenti non possono competere sul piano della diversificazione tecnologica, a meno che non intendano spendere quantità ingenti di tempo e di denaro e usare una piattaforma completamente diversa da GCC»[22].

La FSF di fronte ai cambiamenti in atto ha tentato di ovviare in qualche modo e recuperare terreno: nel 1999 nasce la LGPL (Lesser General Public Licence)[23] in relazione al problema delle librerie e per diminuire l'impatto considerato un po' troppo limitante della GPL.

La LGPL, come la GPL, si presenta contemporaneamente come una licenza e come un manifesto politico: viene infatti specificato che le licenze della maggior parte dei programmi hanno il solo scopo di togliere all'utente la possibilità di condividere e modificare il programma stesso. Al contrario le Licenze Pubbliche Generiche GNU sono intese a garantire la libertà di condividere e modificare il software libero, al fine di assicurare che i programmi siano liberi per tutti i loro utenti.

Nella licenza viene messa in evidenza la finalità principale, ovvero garantire che anche un programma non libero possa utilizzare delle librerie open (ad esempio la libreria C del progetto GNU in modo da consentire a più persone di usare l'intero sistema operativo GNU), ma allo stesso tempo viene specificato di usare la licenza con parsimonia, mentre si viene sollecitati a sviluppare librerie libere in modo che chiunque possa ridistribuirle o modificarle[24].

Infine, nata nel 2000, la licenza GFDL (GNU Free Documentation Licence)[25] è una licenza libera, copyleft, ma per distribuire documentazione e non software; l'elaborazione è della FSF. Un esempio di applicazione concreta della GFDL è l'esperimento di Wikipedia[26], un'enciclopedia libera e multilingue open publishing: nella versione inglese è arrivata a contare oltre 500.000 voci, quella italiana al momento ne conta circa 40.000.

Le licenze software sono una modalità di relazione verso l'esterno di creazioni per lo più comunitarie. Vediamo ora come si relazionano al loro interno queste comunità.

Note al capitolo

1. http://www.gnu.org/philosophy/free–sw.it.html
2. http://www.opensource.org/docs/definition.php
3. http://www.opensource.org/
4. http://www.gnu.org/philosophy/free–sw.it.html
5. Il monopolio commerciale della Microsoft si è ampliato a dismisura: dalle patenti europee del computer (ECDL), all'email su hotmail, passando ovviamente per MSN, i sistemi operativi windows, le certificazioni per sistemisti e programmatori. Tanta strada è stata fatta da quando Bill Gates vendette il porting per il linguaggio BASIC per Altair 8800 nel 1975 a oggi; la softwarehouse Microsoft ha cambiato in questo lasso di tempo svariate facce. In origine nacque come società di sviluppo software (furono scritte infatti diverse versioni del linguaggio BASIC – non di sua invenzione – come fortran, cobol, pascal, Visual Basic e il nuovo C#), ma si specializzò ben presto in applicativi gestionali (visiCalc, MS-word, MS-excel) e sistemi operativi (Xenix e MS-DOS). L'anno decisivo fu sicuramente il 1990 con il rilascio di Windows 3.0: un sistema operativo abbastanza usabile, dotato di interfaccia grafica, a un costo nettamente inferiore rispetto al più avanzato Apple Macintosh. Windows 95, Windows NT, Windows 98, Windows 2000, Windows XP e molti altri sistemi operativi hanno poi invaso il mercato. Oggi Microsoft è la più grande società di software del mondo (http://it.wikipedia.org/wiki/Storia_di_Microsoft_Windows).
6. Richard Stallmann definisce Free Software e Open Source «due partiti politici» (http://www.gnu.org/philosophy/free–software–for–freedom.html).
7. Più correttamente, software gratuito, ma non libero, si traduce con Freeware.
8. Molti «osservatori esterni» confondono i piani: Negri e Hardt nel loro ultimo lavoro dedicano una parte all'Open Source descrivendolo come si descriverebbe il Free Software. A conferma di questa confusione, nelle note, viene citata un'opera di Stallmann da cui sono ripresi alcuni brani. Stallmann Open Source! (Michael Hardt, Antonio Negri, *Moltitudine. Guerra e democrazia nel nuovo ordine imperiale*, Rizzoli, Milano, 2004, p. 350)
9. http://www.Linux.it/GNU/softwarelibero.shtml
10. http://www.garzanti.it
11. http://www.mozilla.org/MPL/MPL–1.1.html
12. John Woo, *Paycheck*, 2003, tratto dal racconto di Philip K. Dick, *I labirinti della memoria*, Fanucci, Roma, 2004.
13. Si veda ad esempio Philip K. Dick, *Ubik*, Fanucci, Roma, 1995 (ediz. orig.: *Ubik*, 1969): i personaggi utilizzano già dei telefoni cellulari.

14. Certo l'Ufficio brevetti si occupa da troppo tempo di «proteggere» le conoscenze, costituendo di fatto un ostacolo sempre più insopportabile per le stesse aziende; per una lettura di *Paycheck* alla luce della situazione attuale, si veda http://www.quintostato.it/archives/000611.html

15. Da sempre oggetto di aspri dibattiti, la GPL è stata accusata nel 2003 da SCO di violare la costituzione degli Stati Uniti e le leggi sul copyright, sull'antitrust e sulle esportazioni; si veda http://www.Linuxworld.com/story/35679.htm. È invece di fine luglio 2004 il riconoscimento legale della licenza GPL da parte del tribunale tedesco di Monaco di Baviera.

16. Bruce Perens, *The Open Source Definition*, http://www.apogeonline.com/openpress/libri/545/bruceper.html

17. Eric S. Raymond, *Colonizzare la Noosfera*, 1998, http://www.apogeonline.com/openpress/doc/homesteading.html

18. Ironicamente, una falla nella sicurezza del sito della società delegata a gestire gli accessi al codice sorgente ha causato nel febbraio del 2004 la diffusione di una parte del sorgente di una versione preliminare di Windows 2000 (circa 200 MB di codice), da cui derivano anche Windows XP e Windows 2003.

19. www.apogeonline.com/openpress/libri/545/young.html

20. David McNett scrive infatti: «*Would appear to be at odds with the also included GNU General Public License*»; http://www.smh.com.au/articles/2003/06/19/1055828414057.html.

21. Testo originale della Licenza BSD versione 4.4 (inglese): http://www.freebsd.org/copyright/license.html

22. Brian Behlendorf, *Open Source come strategia commerciale*, http://www.apogeonline.com/openpress/libri/545/brianbeh.html

23. Testo completo della licenza LGPL, in inglese: http://www.gnu.org/copyleft/lesser.html

24. Un breve accenno all'Artistic License, http://www.opensource.org/licenses/artistic–license.php. Questa licenza è considerata incompatibile con la GPL e il progetto GNU: farraginosa e imprecisa nelle specifiche, non rappresenta né una novità né una rilettura di altre licenze; è mal formulata, perché impone dei requisiti e fornisce poi delle scappatoie che rendono facile aggirarli. Per questo molto software che utilizza questa licenza si avvale di una seconda licenza, quasi sempre la GPL.

25. http://www.gnu.org/copyleft/fdl.html. Da notare che Debian sta eliminando tutti i software rilasciati sotto GFDL per incompatibilità con le proprie linee guida; si veda: http://people.debian.org/~srivasta/Position_Statement.html

26. http://it.wikipedia.org/wiki/Pagina_principale

III
COMUNITÀ

1. Etica: pratiche non scritte

Le connotazioni etiche del software, come abbiamo visto nel capitolo precedente, sono definite da interazioni fra le modalità di distribuzione, le metodologie di sviluppo e le licenze applicate. Non dipendono dunque dal software in sé, non sono inscritte nel codice, ma sono frutto di un sistema complesso di sinergie.

In sostanza il discrimine è costituito dalla comunità di sviluppo che col proprio lavoro marca le caratteristiche peculiari del software sia in senso tecnico che culturale. Se la licenza di un software rispetta le caratteristiche espresse nei punti OSI o FSF, allora questo applicativo si può definire «amico» del mondo Open Source o Free Software: è ragionevole aspettarsi che riceva tutti i benefici derivanti dall'uso di strumenti messi a disposizione dalla comunità.

Sin dalla nascita di internet, e nella maggior parte dei casi ancora oggi, è stata regola non scritta (ma fondante dei principi collaborativi tra sviluppatori) l'accordo a non sviluppare mai applicativi o librerie già esistenti. Al contrario: è pratica corrente avere sempre presenti gli aggiornamenti circa gli sviluppi degli altri progetti[1].

Non si tratta di una banale «usanza», ma di una pratica consolidata per ragioni precise. Infatti, è ritenuto più utile e interessante aderire a un progetto già avviato piuttosto che iniziarne uno nuovo, poiché la rete per sua natura si sviluppa in una miscela complessa nella quale l'emersione di grandi progetti avviene attraverso la collaborazione reale e non competitiva tra sviluppatori, debuggers e utenti.

Il processo attraverso il quale si raggiunge l'obiettivo spesso è più importante dell'obiettivo stesso; o meglio, se non si fosse sperimentato il metodo di sviluppo aperto, paritetico e di interdipendenza, il mondo del software non avrebbe avuto le stesse ricadute politiche sulla vita reale.

Molti interventi analizzano l'aspetto metodologico in questione avvalendosi di termini come «passione», o «economia del dono»[2]: viene così descritta e spiegata la ragione di tanta dedizione a progetti anche estremamente impegnativi nell'assoluta mancanza di una gratificazione economica.

Sicuramente il desiderio di aderire a un progetto comunitario è un elemento fondamentale: è un'energia che deriva dalla propensione al gioco, dalla scelta ludica peculiare dello spirito hacker. Per noi si tratta di una combinazione sinergica tra il piacere della creazione fine a se stessa e il senso di appartenenza a un gruppo, una comunità, capace di gratificazione e stimoli nei confronti dell'individuo. In ogni caso, più che di «economia del dono» si tratta di una «ecologia del desiderio»: un discorso fluido, non un paradigma strutturato di regole normative.

2. Come nasce e si sviluppa un software: dall'humus alla comunità

Una delle tante novità apportate dalla nascita del kernel Linux al Free Software è stata quella di espandere l'équipe di sviluppo del programma. Vengono aggiunti al core nuovi e diversi gruppi de-

dicati: ad esempio, alla documentazione, alle distribuzioni complete, o alla creazione di temi grafici, fino alla nascita di *community* di semplici utilizzatori e appassionati.

Attualmente molti progetti di applicativi giungono al mondo Open Source da un passato commerciale di sviluppo close, affascinati o costretti a liberare i codici sorgenti proprietari per ragioni di business; altri invece, che nascono con precisi intenti commerciali, iniziano da subito a muoversi nel mondo del codice libero.

In qualunque caso, siano le comunità libere o interamente gestite da un'amministrazione aziendale, identifichiamo almeno cinque macro nodi, cinque tipologie che si intersecano variamente e compongono morfologie complesse: comunità di sviluppo, di distribuzione, di documentazione, di utenti, di sicurezza.

Cominceremo ad analizzare alcune diverse tipologie di comunità, a partire da come viene originariamente costruito un manufatto digitale. Per evitare noiose tassonomie, le chiameremo genericamente comunità Open Source, comprendendo anche le comunità che più si riconoscono nell'originario movimento Free Software.

3. Comunità di sviluppo

Quando nasce un applicativo, generalmente la spinta iniziale si deve alla volontà di un piccolo gruppo o di un singolo sviluppatore che ne abbozza i codici; capita molto di frequente, infatti, che gli hacker si dedichino per gioco ai propri progetti, pur portandoli a livelli di elevata comprensibilità, fruibilità e professionalità.

Attorno a una nuova idea di applicativo troviamo sempre un ribollire di micro progetti, software completi e funzionanti, idee e codice. L'emersione di ogni singolo elemento da questo magma è frutto di una scrematura a ognuno dei grandi nodi; una volta affermata la stabilità dell'applicativo nessuno generalmente sceglie la via della competizione (imbarcandosi magari nella scrittura di qualcosa di analogo), piuttosto si collabora alla messa a punto di nuove idee presso il software «dominante»[3].

Di regola l'implementazione di un software avviene attraverso la scrittura di plugins, cioè per mezzo di aggiunte funzionali per programmi a gestione modulare. Pensiamo per esempio a Xmms,

il clone del noto lettore di file audio Winamp per piattaforma Windows; la struttura di questo applicativo è pensata per supportare moduli che leggano nuove tipologie di file, ulteriori output audio, effetti grafici più avanzati.

Tramite la compilazione di plugins si formano piccole comunità o reti informali di singoli programmatori che forniscono il proprio contributo relazionandosi occasionalmente al progetto.

Questo fenomeno di cooperazione aperta è in corso da oltre dieci anni ed è particolarmente visibile anche in campi nuovi come il multimedia, il video e l'audio, ove un numero incredibile di programmi sta iniziando a prendere piede con il desiderio di sfondare presso il grande pubblico (che a sua volta è in crescita verticale, data la disponibilità sempre maggiore di strumenti di riproduzione audio-video)[4]. Nella maggior parte dei casi i software che emergono sono quelli il cui team riesce ad affrontare con successo e rapidità le questioni tecniche e relazionali.

La velocità di risposta ai debuggers, che utilizzano l'applicativo e segnalano i problemi, e alle implementazioni delle nuove funzioni, possono mettere in luce la capacità dei coder di sostituirsi alla figura del/dei mantainer condividendone il lavoro. Il mantainer, dal canto suo, dovrà essere in grado di puntualizzare o gratificare prontamente la mailing list di sviluppo e/o della *community* allargata, confermando le modalità d'uso condivise degli strumenti e la netiquette.

In sintesi, si può affermare che la possibilità di livellare la struttura gerarchica insita nelle comunità (vedi cap. I) dipende dalla capacità di scomposizione dei ruoli a opera del core di sviluppo. Vi sono casi in cui software validi e complessi si formano attraverso la summa di progetti minori nati non tanto attraverso delle comunità quanto piuttosto dal dialogo e dalla messa in relazione di patch individuali e scheletri di progetti abbandonati[5].

Le relazioni tra una specifica équipe di sviluppo e il resto della rete sono spesso agevolate dalla presenza di individui chiave che aderiscono parallelamente a più progetti e sono capaci di travasare non soltanto le proprie competenze da un progetto all'altro, ma anche le informazioni e gli aggiornamenti sullo status di ciascuna elaborazione.

In maniera analoga esistono progetti che legano trasversalmente le comunità tra loro in un rapporto di interdipendenza.

Questo particolare legame tra applicativi non è da ritenersi una relazione di subalternità, bensì chiarisce l'aspetto metodologico reticolare: la collaborazione paritetica è la modalità attraverso la quale ogni nodo della rete è collegato esponenzialmente a un altro.

L'esempio migliore di correlazione tra progetti è costituito dalle librerie, ovvero porzioni di codice compilate che forniscono strumenti a programmi che abbiano la necessità di funzioni semplificate. Le librerie grafiche GTK, QT e FLTK implementano gli elementi standard di un'applicazione visuale (pulsanti, menu, icone...), semplificando il lavoro dei programmatori che, appoggiandosi alla libreria preferita, scriveranno solo le funzioni reali del programma. Saranno infatti le librerie a disegnare i pulsanti, gestire i click del mouse, tracciare le ombre e molto di ciò che siamo abituati a vedere come utenti (si vedano le Appendici in merito alle librerie QT).

Le librerie sono progetti sviluppati in maniera diffusa da comunità il cui intento è fornire strumenti generici e trasversali per risolvere problemi complessi (connessioni di rete, comunicazione tra applicativi, gestione del testo, immagini, compressione). Esattamente come un software viene scritto con l'obiettivo di raggiungere più utenti possibile, una libreria è fatta per arrivare a quante più comunità possibile (chiariremo più avanti le contraddizioni, e le tensioni, insite nello sviluppo di queste applicazioni).

Abbiamo detto che all'interno delle comunità di sviluppo si trovano precise modalità organizzative. La maggior parte dei progetti ha una struttura mobile che ruota attorno a un elemento centrale: il mantainer, colui che dà inizio al progetto, che svolge la supervisione di ogni proposta, codice o suggerimento proveniente dalla comunità; ed è sempre il mantainer a decidere sull'uscita delle release del software.

In alcuni progetti particolarmente ampi e complessi può accadere che alcuni dei ruoli del mantainer siano gestiti da soggetti diversi. Un prototipo è la struttura cosiddetta «a stella» della comunità Linux ove esistono dei capi progetto per ogni singola sezione del kernel. Compito dei capi progetto (che nella fattispecie sono detti «luogotenenti») è quello di vagliare ogni materiale proveniente dai sotto-gruppi combinandolo per la consegna a Linus Torvalds, che farà un ultimo check prima dell'uscita della release. In questa imponente struttura anche il ruolo della «messa on line»,

della pubblicazione effettiva delle nuove versioni, è delegata a un responsabile. In questo modo il mantainer, oltre alla scrittura di codice, può dedicarsi a una supervisione generale di parti già testate e controllate da altri.

Esistono poi dei meta progetti che si possono descrivere come la collezione di un numero, a volte molto elevato, di ulteriori progetti o software. È il caso dei desktop grafici GNOME e KDE, formati da tante librerie e programmi, ognuno dei quali gestito da una comunità spesso totalmente indipendente dalla suite grafica stessa.

Editor, giochi, applicativi di rete, strumenti di configurazione sono alcuni degli elementi che formano i desktop grafici. Il ruolo di congiunzione per questi singoli progetti è ricoperto dalle librerie, che attraverso una veste grafica omogenea fanno interagire gli elementi controllandone l'esecuzione e passando ai vari programmi autonomi funzionalità automatizzate.

4. Comunità di distribuzione

Una distribuzione (GNU/Linux, ma non solo) è un insieme di programmi, file di configurazione, utility di amministrazione e un kernel, il tutto installabile su un computer e aggiornabile da repository appositamente dedicati sulla rete internet. Inoltre, almeno nella maggior parte dei casi, l'aggiornamento è gratuito.

Un'attività imprescindibile delle comunità di distribuzione consiste nel passare al setaccio la rete internet con l'obiettivo di trovare sempre nuovi applicativi che, compatibilmente con la policy della «distro» (cioè con le regole scritte della comunità di distribuzione), possano essere «pacchettizzati», ovvero resi installabili dall'utente finale attraverso i tools della distribuzione stessa.

Debian[6] è una distribuzione storica, da sempre indipendente dalle grandi case commerciali; nel corso degli anni ha saputo brillantemente investire sulla formula della comunità aperta, includendo al proprio interno esclusivamente software di standard OSI. Come si legge sul sito ufficiale della distribuzione:

> Debian è un sistema operativo (OS) libero (free) per il tuo computer. Un sistema operativo è l'insieme dei programmi di base e utilità che

fanno funzionare il tuo computer. Debian utilizza Linux come kernel (la parte centrale di un sistema operativo), ma molti dei programmi di utilità vengono dal progetto GNU; per questo usiamo il nome GNU/Linux. Debian GNU/Linux fornisce più di un semplice OS: viene distribuito con oltre 8.710 pacchetti, programmi già compilati e impacchettati in modo tale da permettere installazioni facili sulla tua macchina.

Tuttavia, paradossalmente, per preservare la propria specificità «free», il team di Debian ha creato una struttura altamente gerarchica; per accedere alla comunità infatti occorre essere presentati da qualcuno già membro e superare alcune prove sulla storia, l'organigramma e i rapporti della comunità Debian con il mondo GNU/Linux[7]. Questo passaggio di «iniziazione» è lungo e complesso, ma una volta entrati si appartiene a una delle comunità più ampie ed eterogenee possibili, con un'etica saldamente legata ai parametri del software libero. Certamente Debian non è l'unica distribuzione slegata dal mondo commerciale; ve ne sono molte altre, e di diverse tipologie.

Slackware[8] è una fondazione no profit; per scelta stilistica si presenta come una distribuzione minimale: infatti è spesso difficile trovare software pacchetizzati o appositamente configurati per le installazioni. Slackware prende spunto da un modello di boot simile a quello dei sistemi BSD: per questa ragione si differenzia parecchio dalla stragrande maggioranza delle altre distribuzioni GNU/Linux che invece sono livellate su un'organizzazione di script e directory detta System V[9].

Invece nel caso di Gentoo[10], una distribuzione recente realizzata da giovani hacker in ambiente universitario, l'installazione di un software si compie attraverso la compilazione ad hoc dei codici sorgenti. La compilazione di un programma direttamente sulla propria macchina rende l'applicazione maggiormente performante rispetto a un'installazione precompilata, ma i tempi dell'operazione si dilatano enormemente. Il successo di questo stile si deve all'illusione di possedere una distribuzione personalizzata creata passo dopo passo sul proprio computer.

Infine, anche se non sono basati su un kernel GNU/Linux, esistono anche i port di BSD, che presentano un sistema di compilazione parziale del codice sorgente dei programmi richiesti.

Queste le principali distribuzioni libere.

Le distribuzioni commerciali, come Redhat, SuSE, Mandrake, capofila di un discreto elenco, funzionano diversamente. Generalmente il core di sviluppo, documentazione ecc. della distribuzione viene retribuito dalle società finanziatrici del progetto; le propaggini secondarie vengono lasciate invece al lavoro volontario. Il team delle distribuzioni commerciali, per ovvi motivi di mercato, si impegna soprattutto nel creare tools di amministrazione semplificata e software di installazioni più grafici possibili[11].

5. Servizi

Tra comunità di sviluppo e comunità di utenti la comunicazione avviene attraverso il web, ma sono i repository delle singole distribuzioni a essere il mezzo maggiormente utilizzato per l'installazione o l'aggiornamento dei programmi.

Esistono poi degli specifici fornitori di servizi che mettono a disposizione, a chiunque ne faccia richiesta, computer per testing, repository, mailing list, spazio web e altro ancora. Di questi paradisi digitali solo due possono vantare una fama mondiale e un numero di progetti ospitati superiore a ogni altro concorrente: sourceforge.net e freshmeat.net. Dietro a questi nomi, però, si trova un'unica società: la Va Software Corporation, che fornisce al mondo dell'Open Source computer estremamente potenti e banda larga, hard disk e assistenza non alla portata di tutti.

Il prestigio della Va Software è dovuto in particolare ai propri portali, che ospitano gratuitamente il codice sorgente delle comunità e dei singoli programmatori e permettono a chiunque di effettuare ricerche sul database dei progetti ospitati. Questi portali hanno una tale risonanza che anche i progetti più piccoli, una volta comparsi sulle prime pagine, arrivano tranquillamente a contare centinaia di visite al giorno.

E facile quindi immaginare come, con un database utilizzato gratuitamente da migliaia di coder, la Va Software possa garantire un ottimo servizio di *business to business* per aziende legate al mondo Open Source ma non solo: un data mining di particolare interesse per business miliardari. Tra gli sponsor e advertiser di sourceforge e freshmeat troviamo infatti dalla Red Hat fino alla Microsoft.

6. Comunità di sicurezza

Le comunità nate attorno al tema della sicurezza informatica si costruiscono attorno a mailing list, portali web e canali di chat; non hanno un aspetto progettualmente tecnico, o meglio la loro finalità è quella di fare ricerca tecnica generica su bug critici.

L'obiettivo del security hacking è quello di trovare modalità di intrusione, prevenire l'esistenza di virus e ipotizzare nuovi percorsi di attacco ai sistemi; per questa ragione si analizza ogni tipo di software, incluse le applicazioni close.

Gli ambiti di incontro e discussione su questi temi sono altamente compositi: vi si trovano ad esempio autorevoli «cani sciolti» che collaborano alle discussioni per qualche tempo per poi andarsene dedicandosi ad altro, sistemisti ed esperti di sicurezza, giovani hacker della scena underground, ovviamente la polizia e i servizi di ogni tipo, professionisti freelance oppure legati a società di penetration test o sistemistica[12].

Questo circuito della sicurezza è tra quelli che maggiormente suscita fascinazioni e ambiguità nell'immaginario, spesso superficiale, dell'opinione pubblica. Ma i cattivi pirati informatici sono una banalità mediatica, che riesce addirittura a confondere gli *script kiddies* (solitamente ragazzini che lanciano codici altrui per fare danni, ignorandone il funzionamento) con criminali comuni che usano le proprie capacità informatiche per semplici furti. La scena della sicurezza è ben più complessa e articolata, ma in ogni caso le comunità che scrivono gli exploit con i quali è possibile penetrare nei server per effettuare crimini informatici sono anche quelle che forniscono supporto alle applicazioni per quel debug altamente tecnico che rende un'applicazione sicura e realmente affidabile[13].

Scendendo un poco più nel dettaglio, possiamo distinguere tre tipologie di soggetti connessi alla sicurezza informatica: gli utilizzatori di exploit, che si pongono come fine la semplice violazione dei sistemi, sono la categoria più ampia, perché essenzialmente costituiscono un fenomeno legato alla formazione di un'identità digitale e all'affermazione del sé. Questi soggetti spesso non conoscono nessun linguaggio di programmazione, e ciò nonostante svolgono anch'essi un importante lavoro di debug.

Coloro che scrivono l'exploit devono invece comprendere il

funzionamento delle architetture di un computer ad alto livello. Hanno la capacità di comunicare col processore quasi a linguaggio macchina e possiedono uno stile proprio, ma non per forza sono veri e propri coder, anzi normalmente nutrono per questi ultimi un profondo rispetto. I loro codici, nella maggior parte dei casi, sono brevi ed eleganti, ma non particolarmente articolati rispetto alle creazioni dei coder.

Infine, troviamo i soggetti che studiano i bug e creano nuove tipologie di exploit. Si tratta di un numero estremamente esiguo di ricercatori informatici con competenze e lampi di genio fuori dal comune che, con la loro intelligenza, creano nuovi percorsi e spazi nello scenario non solo della sicurezza ma della stessa programmazione.

Senza la resa pubblica del lavoro dei security hacker, su siti appositi come securityfocus.com[14], gli sviluppatori non potrebbero correggere al meglio i propri codici lasciando i dati alla mercé di chiunque sia in grado di sfruttarne i bug. Normalmente la scrittura di un exploit coincide con la scoperta del corrispettivo bug; entrambi i passaggi vengono realizzati e resi pubblici dalla stessa persona.

Dal punto di vista della condivisione dei codici, esistono diverse tipologie di exploit. In alcuni casi i bug non vengono immediatamente resi noti, oppure sono conosciuti ma si ritiene che non esistano metodi per sfruttarne la vulnerabilità. Sono comunemente detti «0day» o «zero-day» quegli exploit che vengono usati ma di cui non si conosce subito né il bug né la relativa patch. Questo accade per ragioni commerciali o di altra natura. Si tratta di exploit dalle qualità strategiche capaci di riconfigurare i rapporti di forza interni alla scena di sicurezza; il giro di condivisione di questi codici è molto elitario e limitato a un tempo di poche settimane entro le quali vengono resi pubblici presso le comunità di riferimento. La ragione di questa particolarità è dovuta generalmente a errori negli algoritmi utilizzati[15].

Nella realtà questi ambienti sono altamente apprezzati sia per la ricerca puramente tecnica sia per l'aiuto offerto ai coder e agli utenti finali. Per quanto spesso malvisti dalle società commerciali, i security hacker svolgono un ruolo fondamentale per l'intero mondo del software libero[16]. La scrittura di exploit, per cercare una metafora «organica», equivale infatti a una poderosa scossa di

adrenalina per la comunità del software «colpito»: tale comunità reagisce attivandosi immediatamente nella realizzazione di un'apposita patch (codice correttivo); frattanto, a seconda della propagazione dell'exploit, tutti gli utilizzatori del software non potranno che attendere la soluzione o la propria capitolazione.

7. Comunità, azienda, Stato: casi interessanti

Come già accennato (vedi cap. I), Mozilla è il primo progetto commerciale impiantato nell'Open Source e rappresenta uno dei più poderosi colpi inferti al dominio di Microsoft, nonché il primo passo delle comunità verso le contraddizioni del mondo economico.

I risultati sono stati tecnicamente eccellenti, ma le ragioni politiche della società Netscape sono facilmente individuabili. Quando la Netscape nel 1998 ha deciso di rilasciare il proprio browser sotto una licenza aperta e di investire su una comunità ad accesso libero, l'idea di fondo appare chiara: ampliare quella fetta di mercato che sta pian piano perdendo spazio per colpa della Microsoft.

Il duo Mozilla-Firefox ha velocemente minato il dominio del browser Internet Explorer: nella versione suite (Mozilla, che comprende anche un client di posta e altre funzionalità), o standalone (Firefox), è disponibile praticamente per ogni sistema operativo e architettura e ha funzioni uniche nel suo genere, dal tabbed browsing (molteplici finestre aperte in contemporanea) all'espansione di funzionalità tramite plugins.

La Netscape ha certamente vinto la battaglia sui browser, diventando un logo nascosto dietro alle quinte di una comunità libera, tirandone i fili e indirizzandola verso ciò che ritiene commercialmente utile grazie a investimenti economici diretti ai programmatori.

Quando ha iniziato la propria mutazione, Netscape era l'unico soggetto commerciale a investire in maniera cospicua nel mondo Open Source; se si fosse presentata in forma troppo visibile come progetto commerciale, non avrebbe potuto beneficiare della partecipazione spontanea intrinseca alle comunità.

La comunità di Mozilla-Firefox, ormai attiva da otto anni, presenta un'amministrazione molto verticale, basata su un team centrale coordinatore di singole unità più o meno tecnocratiche; que-

sta struttura gerarchizzata tende a sviluppare tutti i codici necessari all'interno del proprio gruppo senza appoggiarsi, se non in rari casi, ad altri software o librerie, contravvenendo in questo modo all'assioma di interdipendenza tra progetti come forma di autovalutazione.

Un esempio eclatante è il supporto SSL, ovvero le connessioni criptate secondo uno scambio di chiavi pubbliche e private (RSA)[17], sviluppato e compilato come blocco unico dal core dei programmatori di questo progetto. La scelta di Firefox suscita diffidenza perché la proposta sugli algoritmi di criptazione più conosciuta e apprezzata sulla scena digitale, ovvero le librerie OpenSSL, sono sviluppate da una comunità molto attiva che da anni segue tutte le novità matematiche sulla criptografia: dallo streaming di rete alla criptazione di file e all'hashing di dati[18].

A un rapido controllo si noterà che il browser in questione si appoggia a poco più di tre progetti: X server, le librerie grafiche, le librerie base del sistema. Quest'aspetto monolitico è il principio della grande portabilità su altri sistemi operativi, poiché in questo caso l'applicativo viene modificato solo in minima parte (giusto quella grafica) lasciando la maggior parte del codice inalterato. Di contro, questo approccio sarà causa di una pesantezza e lentezza notevoli rispetto a un software che sappia comunicare maggiormente con ciò che è già presente e attivo all'interno del sistema.

In ragione di queste scelte tecniche, progetti simili a Mozilla-Firefox, come vedremo in seguito, privilegiano un preciso target comunitario: quello dell'utenza medio-bassa, la quale si prodigherà nel debuggare il browser ma a livello di sola funzionalità del programma; mentre coloro che in un applicativo considerano prima di tutto l'affidabilità del codice saranno decisamente poco stimolati dalla ridondanza proposta o addirittura disturbati dalla mancata valorizzazione delle competenze sviluppate da altre comunità.

Da Netscape in poi, con la nascita del movimento Open Source, nel corso di qualche anno lo scenario è cambiato a tal punto che attualmente esistono intere aziende nate con lo scopo di interagire con il metodo proprio della libera condivisione. Uno degli episodi più emblematici degli ultimi anni è stato l'acquisto da parte della Novell, un'azienda produttrice di hardware, di una piccola software house, la Ximian, nota per il client di posta Evolution, un clone di Ms Outlook per Windows. Novell è una delle tante so-

cietà che ha giostrato i copyright commerciali dello Unix rimbalzati di mano in mano come il testimone in una gara economica. Ximian era un team di programmatori Open Source come altri, e in pochissimo tempo si è trasformata in uno dei poli centrali per progetti aperti dal valore strategico. Oggi la Ximian detiene il core di sviluppo di applicazioni come GNOME, uno dei desktop più utilizzati sotto Linux, il già citato Evolution, e Gnumeric, foglio di stile compatibile con Ms Excel. I software Ximian infatti sono scaricabili dal sito internet della Novell e le comunità degli applicativi si basano su server che come desinenza hanno sempre novell.com.

Per molti la possibilità di lavorare con un contratto per un'azienda che sviluppa progetti open sotto GPL ha rappresentato il raggiungimento completo di un obiettivo etico; per molti altri però, che ne intravvedevano le ambiguità politiche, la soddisfazione è stata solo parziale.

Succede sempre più spesso che gli sviluppatori, consapevoli delle posizioni equivoche assunte dalle multinazionali, scelgano un doppio passaporto: da una parte svolgono un lavoro remunerato presso le *majors* che investono nell'Open Source; dall'altra proseguono un'attività creativa non retribuita presso progetti indipendenti.

Interessante anche la vicenda di Open Office, il clone della suite per ufficio Microsoft Office. Acquistato nel 1999 dal colosso Sun Microsystems, i suoi codici vennero liberati a partire dall'anno successivo. Attualmente, in ragione della sua estrema portabilità, rappresenta una delle migliori teste d'ariete puntate verso i cancelli dell'impero Microsoft.

Ogni release del software è posta sotto una duplice licenza: LGPL e SISSL (Sun Industry Standards Source License), con la quale la Sun commercializza l'applicativo.

Al pari di Mozilla, Open Office è un applicativo completamente indipendente e allo stesso modo reca con sé le problematiche di lentezza e pesantezza. Programmi come questi possono infatti funzionare al meglio solo su architetture realativamente recenti. Per questa ragione rappresentano un ostacolo per chi non ha modo di comprare agevolmente hardware all'avanguardia o cerca di dimostrare che con la tecnologia libera è possibile riciclare le vecchie architetture.

La velocità di avvio in grafica di un programma è estremamente favorita dalla compatibilità di questo con le librerie preinstallate; mancando quel rapporto di interdipendenza che abbiamo descritto, quegli applicativi completamente separati dal sistema risulteranno inutilizzabili sotto la soglia del Pentium III.

Ammantandosi della filosofia propria del Free Software e appoggiando le battaglie europee per la lotta ai brevetti, il progetto OpenOffice.org riceve la collaborazione spontanea e gratuita di centinaia di sinceri democratici, che ritengono così di contribuire all'apertura orizzontale dei software. In molti, infatti, ritengono sia più importante mettere a segno una vittoria dell'Open Source sul mercato piuttosto che affrontare un ragionamento sulle risorse disponibili per la popolazione esclusa dal paradigma digitale. La crescita parallela – concordata (il duo Intel-Microsoft è solo il più noto) e del tutto assurda – di hardware sempre più potenti, utili solo a far girare software pesanti con milioni di funzioni superflue, è la dimostrazione lampante dell'ipocrisia delle analisi in merito al *digital divide*.

In molti sostengono che in questa fase sia prioritaria la diffusione su larga scala della consapevolezza della non brevettabilità del codice, e quindi auspicano la sistematizzazione dei cloni di programmi proprietari come Ms Office. Dal nostro punto di vista, la lotta contro il copyright è certamente imprescindibile, ma riteniamo vada supportata a partire da una conoscenza critica degli strumenti e non da un generico «uso di applicativi aperti». Non siamo disposti a rinunciare alle libertà in nome di una più conciliante apertura.

Naturalmente c'è anche chi non nasconde i propri intenti meramente commerciali. È il caso di Snort[19], uno tra i più famosi e utilizzati Intrusion detector; questo tipo di software serve per rilevare e segnalare intrusioni nel sistema. Marty Roesch, sviluppatore e mantainer del progetto, in molte interviste ha chiarito la sua posizione di fronte al mondo del codice aperto, definendo il proprio lavoro un bell'esperimento commerciale: sviluppare con strumenti Open Source semplifica il lavoro, permette di avere codice più sicuro e di averlo più in fretta.

Snort è distribuito sotto licenza GPL, ma attorno a esso è nata una società diretta dallo stesso Roesch che distribuisce moduli aggiuntivi close a pagamento. I moduli sono aggiunte funzionali al

software e permettono di estendere regioni del programma in maniera quasi illimitata. In poche parole l'utente può scaricare la versione base di Snort gratuitamente e sondarne il codice, ma se desidera considerare le evoluzioni del software dovrà pagarne i moduli senza poter accedere ai nuovi codici.

Diametralmente opposta è la visione della comunità KDE, la quale, nel suo manifesto[20], scrive di definirsi free in ogni aspetto del termine. KDE, però, dal 2003 è finanziata dal governo tedesco, ovvero da quando la Germania ha iniziato a investire nell'Open Source per passare tutta l'amministrazione pubblica a GNU/Linux. In questo senso KDE e SuSE sono state scelte abbastanza obbligate, dal momento che SuSE è una distribuzione nata e sviluppata in Germania e la comunità tedesca KDE è una tra le più attive.

KDE propone un desktop grafico free basato su server X, ma va ricordato che la sua grafica è frutto delle librerie QT, sviluppate close dalla Trolltech[21] e rilasciate solo nel 1999 con una licenza Open Source. Quindi, per quanto KDE si sia sempre definito libero, ha avuto per oltre quattro anni di sviluppo una contraddizione difficile da gestire; ancora adesso le librerie QT (vedi Appendici: *Le librerie QT*) non sono sviluppate da una comunità libera, ma dal team ufficiale della Trolltech, società con relazioni commerciali talmente ramificate da coprire ogni aspetto del mondo dell'IT.

Per la somma di 210 milioni di dollari nel gennaio 2004 Novell ha acquisito SuSE: con questa mossa è diventato il primo fornitore di soluzioni GNU/Linux per l'azienda, dai desktop ai server. I toni sono stati trionfalistici: «Novell sarà l'unica azienda software da 1 miliardo di dollari con una distribuzione Linux e con personale tecnico in tutto il mondo in grado di fornire supporto».

Sempre nel gennaio 2004 Novell ha rinnovato il proprio accordo con IBM per circa 50 milioni di dollari. L'accordo permette a IBM di distribuire SuSE Linux Enterprise Server con i server IBM. Novell apre così un nuovo canale per distribuire SuSE Linux in tutto il mondo.

È interessante seguire le peripezie finanziarie di questi colossi e le loro alleanze al di fuori del mondo specificamente informatico: aziende che si fanno paladine dell'apertura dei codici, contro la chiusura di Microsoft, ma che nei fatti sono mosse dai medesimi

interessi commerciali di profitto a qualunque costo e non certo da uno sfrenato desiderio di condivisione del loro know-how tecnologico.

Gli investitori di SuSE Linux sono e-millenium1, AdAstra Erste Beteiligungs GmbH e APAX Partners & CO. Apax (un termine greco che significa «una volta sola») è una società di servizi finanziari di «venture capital e private equity» che gestisce 12 miliardi di euro nel mondo (dato 2001), con uffici a New York, Londra, Parigi, Milano, Monaco, Madrid, Tel Aviv, Stoccolma e Zurigo. Ultimamente ha fatto shopping in Italia stringendo un accordo per 8,6 milioni di dollari per finanziare l'italiana Kelyan Lab, azienda del gruppo Bernabè (partner TIM), per Net solutions avanzate. Con le Telco italiane sembrano proprio essere affari d'oro per la Apax, che ha incrociato anche Telecom nei propri affarucci israeliani. Apax infatti è nota (spesso accostata alla Jerusalem Global Ventures) anche per le sue operazioni in Israele dove investe soprattutto in microelettronica e materiali avanzati, settori indissolubilmente legati alla produzione di armi nonché al controllo militare del territorio; per questo è stata oggetto di boicottaggio.

AdAstra Erste è una società di venture capital con sede a Monaco di Baviera: *lead investor* è la HypoVereinsbank, la seconda banca in Germania. Quest'ultima fa parte del consorzio di collocamento della Deutsch Post di cui azionista di maggioranza è il governo tedesco. Inoltre AdAstra Erste nell'ottobre 2004 ha investito parecchi milioni di dollari a sostegno della Biomax, una società di bio-informatica. Insomma dietro alla «libera» SuSE, ora la «aperta» SuSE-Novell, c'è la crema del venture capital tedesco.

In Francia invece il ministro della Funzione pubblica Renaud Dutreil ha dichiarato nel giugno 2004 che, per aggiornare una parte dei computer dell'amministrazione statale, intende chiamare in causa i fornitori di programmi Open Source, con un'iniziativa mirata a contenere il deficit pubblico. «Non abbiamo intenzione di dichiarare guerra a Microsoft o alle compagnie informatiche statunitensi», ha precisato Dutreil in un'intervista. «La competizione è aperta e ci auguriamo di riuscire a realizzare un taglio dei costi di almeno il 50%» ha aggiunto il ministro[22].

Come in Germania il consorzio di aziende si è dato disponibile ad agevolare il business è formato da IBM e Novell-SuSE.

8. Gare

Sfruttando la meritocrazia e la competizione insite nel mondo dell'hacking, il desiderio di creare un hack sempre migliore, vi sono state anche particolari derive che sono riuscite a influenzare persino la FSF. I sistemi di infiltrazione e finanziamento più o meno nascosti promossi dagli investitori commerciali – in un mondo regolato essenzialmente dalle licenze – sono sempre più palesi.

Fra le novità indicative del cambiamento in corso nel mondo dell'Open Source le gare a progetto sono certo la più notevole. Si tratta di vere e proprie sfide sulla scrittura di intere parti di codice con premi economici ai vincitori: chi prima consegna o chi trova e corregge più velocemente i bug riceve infatti una somma tra i 200 e i 1.200 dollari.

Le applicazioni più eclatanti di questa strategia sono state messe in atto dalle comunità di MONO e DotGNU, progetti apparentemente[23] in competizione fra loro per la clonazione del framework .NET, sviluppato da Microsoft, su piattaforma GNU/Linux.

MONO è gestito dalla XIMIAN e quindi di proprietà della Novell, mentre DotGNU è curato dalla Free Software Foundation.

.NET è un'idea e una implementazione Microsoft di un terreno digitale (framework) sul quale possono girare programmi scritti nei più diversi linguaggi (C++, Visual Basic, C# e altri), con l'obiettivo di rendere gli applicativi portabili da un computer all'altro, anche in streaming, su ogni sistema Windows[24].

MONO e DotGNU sviluppano un medesimo framework su piattaforma GNU/Linux con l'idea di far girare applicazioni Microsoft e in genere per Windows anche su sistemi operativi liberi. Avere un terreno digitale su cui è possibile programmare con un linguaggio intermedio tra il linguaggio macchina e il linguaggio scelto dal programmatore rende ogni programma teoricamente portabile su qualsiasi sistema operativo dotato di tale framework.

Il linguaggio intermedio è un qualcosa di commestibile dal framework che lo esegue in linguaggio macchina per il processore e il sistema operativo su cui sta girando. In altre parole il programmatore potrà scegliere il suo linguaggio, il compilatore lo trasformerà in linguaggio intermedio, il framework lo eseguirà su ogni architettura.

L'idea è già stata ampiamente implementata e utilizzata con alterne fortune. Java, ad esempio, si basa su una Virtual Machine che rende un applicativo scritto in questo linguaggio portabile su ogni sistema operativo dotato del framework opportuno. In sostanza, lo stesso programma Java può girare su un PC, su un Mac, su un'Alpha Station, ma anche su un cellulare o su una lavatrice[25].

Un altro modo per conseguire il traguardo della portabilità è quello di scrivere applicazioni in linguaggi interpretati, che non essendo compilati girano ovunque ci sia un software capace di comprenderne la sintassi.

L'innovazione di .NET è quella di aver reso la portabilità un elemento nativo del compilatore. In questo modo una qualunque applicazione scritta in un qualunque linguaggio girerà su ogni sistema dotato del framework .NET.

Progettare questi compilatori implica notevoli modifiche a progetti da tempo già affermati (in particolare GCC), e per il mondo Open Source la scrittura da zero di nuovi compilatori per C# e Visual Basic che sono linguaggi proprietari di Windows.

Entrambi i gruppi si sono confrontati con la complessità del lavoro intrapreso e hanno optato per una scelta operativa similare: pagare chi consegnava per primo una delle parti più «noiose» da effettuare. Questi codici riguardano per lo più la grafica e le comunicazioni di rete, mentre lo sviluppo delle parti più complesse e interessanti viene lasciato alle comunità. Ricorre a questo metodo di sviluppo anche Mozilla: in questo caso però le gare sono limitate ai bug e il contributo economico è proporzionato alla complessità del problema risolto.

Gli esempi di comunità spurie o metodi eterodossi sono innumerevoli. Potrebbe sembrare del tutto incidentale, superflua, la quantità di dati tecnici posta sul piatto. In fondo, sembrerebbe un banale scambio di tempo-uomo per programmare macchine in cambio di denaro. Vedremo nel prossimo capitolo come questo sia in realtà un problema relativamente marginale, o comunque derivato: del resto, abbiamo già sottolineato come lo spirito della FSF non sia contro il mercato; lo contempla fin dalle origini come una delle molteplici possibilità, ma certo non la più importante. Il problema, dunque, non è il denaro, bensì le relazioni di potere.

La questione si articola riguardo all'individuo e alla comunità di riferimento.

Siamo arrivati fin qui perché scoprendo i nervi, le viscere delle comunità in alcuni passaggi critici, l'elemento comunitario sembra essersi dissolto. Le gare indette anche da DotGNU sono un elemento di verifica lampante di come l'equilibrio complesso, l'humus delle comunità, si stia riducendo a una prestazione seriale di competenze in cambio di identità. Dilatare il tempo dell'emergenza, continuare a insistere sulla necessità di risultati, subito, adesso, e sempre, significa mutuare una metodologia propria della produttività capitalistica. Ma dilatare il tempo dell'urgenza perenne (centrare gli obiettivi a ogni costo) corrisponde a un movimento di contrazione compulsiva nella quale il desiderio non ha più spazio. Il potere tecnico dell'individuo (semplice atomo e non parte di una comunità) viene ceduto in cambio di un riconoscimento troppo ristretto, ridotto, non condiviso, senza adesione a un piacere più ampio. Il desiderio deve allora essere sostituito con il denaro: questa è la nuova identità trovata. Scambio di know-how tecnico per alimentare una dipendenza dall'ultimo ritrovato, dallo stare sulla breccia a qualsiasi prezzo: sopravvivenza immediata, non più hacking creativo.

A questo livello, la ricaduta sulla realtà è completamente priva di stile. I piani vengono schiacciati, banalizzati, ridotti a un puro, semplice, scambio di tempo, denaro, potere.

Si apre uno scenario a cui si aggiunge, oltre alla sfida tecnica, un confronto e una lotta serrati sui rapporti di forza nel lavoro telematico e sui diritti digitali in seguito alla vittoria *de facto* della GPL. La difficoltà ora sta nel comprendere che l'etica di un software non sta solo nella licenza, ma nell'interazione tra le modalità di distribuzione e la metodologia di sviluppo, poiché il sistema si è fatto più complesso. Da una parte i programmatori dovranno fare i conti con il dilagante vortice di precarietà del quale, se isolati da un confronto critico, saranno facilmente preda; dall'altra i futuri consumatori-utenti di prodotti Open Source dovranno affinare le proprie capacità autoformative al di là della veste user-friendly di un qualunque applicativo.

Si tratta cioè di diventare consapevoli degli aspetti multi-identitari che il mondo digitale reca con sé, sviluppandone al massimo la tensione ricombinatoria. Ad esempio, per i coder, utilizzare un doppio passaporto (triplo, ecc.), costruendo la propria identità intorno a molteplici scelte e attività: compiti magari noiosi (la gra-

fica dei pulsanti) ma retribuiti; compiti non creativi ma utili a un progetto a cui si aderisce; compiti creativi svincolati da riconoscimenti monetari, e così via, giocando a diversi livelli.

Più in generale, anche in senso non tecnico, è necessario rendere pubblici i modelli organizzativi della propria équipe di lavoro, in un'ottica di apertura, condivisione, miglioramento collettivo. È necessario implementare le mediazioni economiche che singoli o comunità pongono a garanzia della propria indipendenza. Sabotare in qualità di utenti smaliziati le semplici barriere ideologiche che impediscono la comunicazione diretta oltre lo steccato del proprio posto di lavoro.

Tuttavia questi sarebbero temi buoni per un manuale di autodifesa digitale dagli sciacalli del know-how nell'era della mezzadria globalizzata; gente ridicola che si riempie la bocca di parole come *skill*, *mission*, *framework*, *workaround*, *development*, naturalmente senza aver mai messo le mani su una macchina né tanto meno aver mai lavorato davvero a un progetto condiviso; perciò ci riserviamo di scriverne in altro ambito.

Le pratiche delle comunità, in ogni caso, sono di capitale importanza per la comprensione delle dinamiche in atto. Prima di vedere come gli individui agiscono e interagiscono nei contesti comunitari, diamo un'occhiata alle relazioni che il mondo delle comunità Open Source ha stabilito con il mercato.

Note al capitolo

1. È curioso che nel mondo del codice le pratiche siano spesso non codificate, non scritte, o meglio vengano codificate ed esplicitate solo quando si pone un problema concreto da risolvere... Un approccio sicuramente improntato alla funzionalità, al desiderio di creare qualcosa che dia soddisfazione piuttosto che alla realizzazione di una solida costruzione coerente e inattaccabile, o riconducibile in modo inequivocabile a una qualsiasi corrente di pensiero.

2. Si vedano in particolare sull'economia del dono riguardo al software: Eric S. Raymond, *op. cit.*; la micro introduzione al tema di Francesco Varanini, http://www.bloom.it/vara39.htm. Il testo di riferimento rimane comunque Marcel Mauss, *Saggio sul dono*, Einaudi, Torino, 2002 (ediz. orig.: *Essay sur le don*, PUF,

Paris, 1950 [1924]). In sostanza, la descrizione antropologica dei rituali del dono compiuta da Mauss (come l'anello di kula polinesiano e il potlach fra le tribù dei nativi americani) viene applicata all'attuale congiuntura economica: spesso, va detto, con eccessivo semplicismo e toni profetici sull'avvento di un capitalismo «postmoderno» o di un'economia della complessità. Sicuramente lo studio delle teorie sul dono in ambito economico è un passo in avanti rispetto alla chiusura concettuale del liberismo classico; inoltre, riscoprendo antichi rituali, queste teorie contribuiscono alla formulazione di un pensiero critico sull'identità, poiché il legame sociale appare il fine e non il mezzo per l'affermazione dell'io e la costruzione dell'identità stessa. Il dono è però un concetto ambiguo e ambivalente, colmo di aporie come tutto ciò che riguarda l'antropologia. I discorsi sull'Open Economy (si veda ad esempio http://www.open-economy.org/) dovrebbero come minimo tener presente le riflessioni di Jacques Derrida: «Se la persona cui dono qualcosa ne è consapevole e mostra quindi la sua riconoscenza dicendomi ad esempio: 'ho ricevuto questo da parte tua, grazie', questa semplice gratitudine e questa presa di coscienza mettono in moto un circolo economico e un gesto di restituzione che, come tali, distruggono il dono», http://lgxserver.uniba.it/lei/rassegna/990218.htm; si veda anche Jacques Derrida, *Donare il tempo*, Cortina, Milano, 1996 (ediz. orig.: *Donner le temps*, Editions Galilée, Paris, 1991).

3. Lo strumento cardine della condivisione nel lavoro in rete è il CVS (Concurrent Versions System), un programma che permette di modificare un insieme di file tenendo traccia di ogni singola modifica, in modo da poter risalire con facilità a una versione precedente in caso di errori. Parlando di CVS si intende un metodo di lavoro ma anche un software vero e proprio. Tuttavia, come è usuale nel mondo del software libero, non esiste mai solo un applicativo che copre una funzione, ma si può scegliere tra una vasta gamma di possibilità. Oltre al CVS (http://www.gnu.org/software/cvs/), citiamo quindi anche SVN (SubVersion – http://subversion.tigris.org), un software altrettanto valido e utilizzato.

4. SITOGRAFIA: *Il mercato Macintosh*.

5. Fetchmail è un esempio di software nato dagli scheletri di altri. Si tratta di un vero e proprio patchwork realizzato da Raymond. SITOGRAFIA: *Fetchmail*.

6. Sito ufficiale: http://www.debian.org

7. Debian sostiene la FSF anche nello sviluppo di HURD: http://www.debian.org/ports/hurd/

8. Sito ufficiale: http://www.slackware.com/

9. System V e BSD. Unix System V (spesso chiamato semplicemente System V) era una delle versioni del sistema operativo proprietario Unix. Sviluppato dalla AT&T e rilasciato nel 1989, ne furono sviluppate quattro versioni successive: System V Release 4 (o SVR4), che unisce caratteristiche del System V, di Xenix e di BSD, è si-

curamente stata la più famosa, poiché introduce numerose caratteristiche tipiche dei sistemi UNIX-like, quali il sistema di Sys V init scripts (/etc/init.d), usato per l'avvio e lo spegnimento del sistema, e il System V Interface Definition, uno standard che definisce quello che dovrebbe essere il funzionamento dei sistemi System V. Berkeley Software Distribution (più noto come BSD) è invece il primo e l'ultimo *flavour* (versione) che tenta il distacco dalla linea concepita da AT&T; BSD soprattutto ha significato: nuove chiamate al kernel, una nuova concezione dei processi di boot, configurazione dei servizi, gestione di account, dischi e dispositivi, nuovi comandi. Timeline aggiornata sulla storia di UNIX: http://www.levenez.com/unix/

10. Sito ufficiale: http://www.gentoo.org

11. Esistono poi altre tipologie di distribuzioni chiamate livecd (Live Distro), ovvero CD-ROM autoavvianti (ma ormai anche minidistribuzioni per penne USB o altre memorie di massa) che rendono subito utilizzabile un sistema GNU/Linux senza bisogno di installazione; ne esistono per varie architetture (PC, PowerPC, ecc.). Knoppix, Gentoo livecd, ecc. (ma ormai anche le principali distribuzioni commerciali offrono una versione live) sono alcuni esempi fra i più noti di una lista molto lunga: www.frozentech.com/content/livecd.php

12. Le società di penetration test producono questi codici o semplicemente vendono come servizio alle aziende test per implementare la sicurezza e svelare violazioni dei sistemi informatici.

13. Il server di posta Qmail, oggi largamente utilizzato, non è soggetto ad aggiornamenti dal 1997 e non perché il progetto sia morto, tutt'altro: da quella data non necessita di correzioni legate a problemi di sicurezza, ovvero nessuno fino a oggi è riuscito a trovare dei bug. Questa insolita circostanza ha destato curiosità al punto da suscitare il desiderio di una vera e propria sfida, infatti per il primo che riuscirà a trovare una falla di sicurezza nel programma c'è un simbolico premio in denaro (1.000 dollari). L'eccentrico programmatore Daniel J. Bernstein, autore del codice di Qmail, ha pubblicato al link http://web.infoave.net/~dsill/dave/qmail/qmail–challenge.html le regole per partecipare alla sfida, tutt'ora aperta.

14. La media degli exploit pubblicati su securityfocus.com si aggira intorno ai cinque giornalieri. La società è stata fondata da Elias Levy meglio conosciuto come aleph1, alias l'hacker che ha redatto il primo testo sulla scrittura di exploit. Si veda http://www.phrack.org/phrack/49/P49–14. Nell'agosto 2002, dopo meno di un anno dalla fondazione (ottobre 2001), il portale è stato acquistato dalla Symantec, nota società di antivirus.

15. La trasmissione di queste conoscenze è, curiosamente, orale. Non esistono siti che riportino aneddoti dettagliati circa gli zero-day e il loro ciclo vitale.

16. SITOGRAFIA: *SHA1, Secure Hashing Algoritm*. L'algoritmo SHA1, usato in molti programmi di criptazione, restituisce in alcuni casi lo stesso hash date infor-

mazioni differenti: questo bug potrebbe essere utilizzato per la scrittura di exploit.

17. http://en.wikipedia.org/wiki/RSA

18. Sito ufficiale: www.openssl.org. Il gruppo rilascia un software stabile e disponibile per qualsiasi programma dal 1996 a oggi che voglia usufruire di criptazione dati, ed è *de facto* uno standard nel mondo del codice libero.

19. http://www.snort.org/

20. Il manifesto etico di KDE: http://www.kde.org/whatiskde/kdemanifesto.php

21. SITOGRAFIA: *Trolltech*. Vedi Appendici: *Le librerie QT*.

22. http://www.unita.it/index.asp?SEZIONE_COD=HP&TOPIC_TIPO=&TOPIC_ID=35567. Una curiosità: nell'aprile 2004 Sun e Wal Mart hanno stretto un accordo per distribuire al dettaglio i PC Microtel dotati della distribuzione GNU/Linux di Sun Microsystems.

23. I progetti MONO e DotGNU non competono fra loro ma coprono due aspetti del panorama generato dall'avvento di . NET. Mentre MONO rappresenta il porting quasi completato del *framework* .NET di Microsoft sotto GNU/Linux, DotGNU crea l'alternativa sfruttando le stesse tecnologie. Contrastando una politica di centralizzazione dei dati portata avanti dalla Microsoft, la GNU vuole utilizzare gli stessi strumenti per decentralizzare le informazioni e i programmi. In pratica entrambi i progetti hanno completato lo sviluppo di compilatori pressoché identici per i linguaggi C, C# e stanno procedendo con la scrittura di quelli per Java e Visual Basic. Mentre DotGNU ha rilasciato il suo webserver per realizzare groupware, la MONO ha concluso anche il supporto per i linguaggi ADO.NET e ASP.NET, oltre alle librerie grafiche GTK. Segnaliamo un interessante dibattito sulle ragioni per cui proseguire lo sviluppo di progetti che di fatto aumentano l'influenza di Microsoft nel panorama Open Source: http://www.theregister.co.uk/content/4/35481.html

24. La portabilità di un software consiste nella sua idoneità di funzionamento su sistemi operativi diversi, architetture dissimili e processori differenti, senza bisogno di ricompilazioni o modifiche. Non bisogna confondere la portabilità col fatto che molti programmi esistano sia in versione Windows che GNU/Linux (o per Apple e altre architetture ancora), perché spesso questa possibilità è dovuta a modifiche anche sostanziali del codice.

25. Vi sono diverse ragioni per evitare l'utilizzo di Java, *in primis* la lentezza e l'eccessivo dispendio di CPU; rimane comunque un ottimo esempio della possibilità di assicurare una portabilità elevata attraverso un framework condiviso, anche se nello specifico è necessaria una Java Virtual Machine diversa per ogni macchina e installata per eseguire i codici Java.

IV

LA STRATEGIA ECONOMICA DELL'OPEN SOURCE

1. GNU Economy e Open Source Economy

Tra la GNU Economy e la Open Source Economy, metaforicamente, c'è la stessa differenza che passa tra una bottega di artigiani e una multinazionale. A parte le differenze di scala, le due economie differiscono anche a livello di obiettivi e pratiche (e spesso a livello etico), ma conservano un tratto in comune: entrambe insistono su una situazione di mercato di tipo capitalistico. Entrambe si situano nel mercato e rispondono alle sue leggi.

I caratteri salienti della GNU Economy sono la metodologia, il permesso d'autore, il codice libero e lo spostamento dell'attenzione dal prodotto al servizio. I software GNU sono liberi e gratuiti: chi volesse concentrarsi sull'assistenza (customizzazione, risoluzione problemi, modifiche ad hoc) potrebbe generare profitto.

Non si vende il prodotto, si vende l'assistenza, perciò il gua-

dagno non deriva dalla vendita pura e semplice del software. Questo eticamente giustifica la parte più politico-economica del manifesto GNU: è legittimo chiedere un riconoscimento monetario per la propria creatività e il proprio software, ma il modo in cui questa legittima richiesta viene gestita attualmente dal mercato secondo il manifesto GNU non è corretto.

«Spremere» denaro dagli utenti di un programma, imponendo restrizioni sull'uso, è infatti distruttivo perché riduce i modi in cui il programma può essere usato. Questo diminuisce la quantità di ricchezza complessiva che si ricava dal programma[1].

Alla base di questo approccio c'è una scelta personale tanto da parte del programmatore, che sceglie la strada attraverso la quale distribuire la sua creazione, quanto dell'utente, che sceglie il prodotto libero: infatti in questo modo l'utente promuove anche tutti i processi di autoformazione che il software libero comporta, anziché il preconfezionato prodotto «close» proprietario (sui processi di autoformazione: vedi cap. V).

Questo naturalmente non significa affatto negare la possibilità di profitto per i programmatori; tuttavia, è chiaro che rispetto a sistemi di sviluppo close i guadagni saranno tendenzialmente inferiori. D'altra parte, è altrettanto evidente che nel lungo periodo l'approccio etico del progetto GNU, oltre a garantire la possibilità di vivere degnamente, consente anche un incremento complessivo, un'espansione delle libertà.

Rendere i programmi liberi significa soprattutto uscire dall'epoca del «bisogno», liberando i programmatori dal «dover programmare per forza», attraverso uno spostamento delle fonti di reddito sulla formazione e sui servizi.

Un'autentica libertà di scelta, il codice libero e lo spostamento dell'attenzione (anche economica) al servizio, secondo gli autori del manifesto GNU, spingono a diminuire il peso della burocrazia e delle attività non produttive, *in primis* di marketing.

La minore concorrenza porterebbe infatti a una diminuzione della mole di lavoro per i programmatori. Non ci sarebbero tempistiche ferree da rispettare o fette di mercato da conquistare con rilasci di software nuovi e più performanti; di conseguenza, diventerebbero sempre meno importanti quelle attività di marketing e advertising che in un circolo virtuoso per il mercato, vizioso per i «consumatori», creano bisogni e nuove dipendenze.

Senza concorrenza e senza scadenze, i programmatori possono inoltre gestirsi il proprio tempo e le proprie idee. Non è la concorrenza e la segretezza del codice a creare il denaro, ma il servizio di supporto al prodotto. Data questa centralità del servizio, il pericolo è che si affermi la tendenza a sviluppare prodotti che necessitano di molto servizio di assistenza, di lunghi tempi di formazione per poter essere utilizzati; per questa ragione è necessario implementare le dinamiche di condivisione dei progetti a ogni livello, a partire dalla mappatura dei bisogni reali degli utenti, affinché non si creino software senza alcuna attenzione per la funzionalità e l'ergonomia.

Tempo e bene relazionale durevole: sono questi i paradigmi hacker della GNU Economy che traducono nei progetti della FSF un diverso approccio al mercato, senza però metterlo mai in discussione nei suoi paradigmi fondanti.

Il manifesto GNU, partendo dalla disamina delle critiche che vengono fatte al Free Software, pone in essere un'analisi, seppure frammentaria e incompleta, sul problema dei diritti del lavoro nel campo della produzione di software e più in generale sull'approccio lavorativo e il suo impatto sociale.

Meno concorrenza significherebbe tra l'altro apertura di nuovi mercati sia per i consulenti autonomi, sia per le piccole società di assistenza e i centri di formazione.

I consulenti autonomi non potranno coprire da soli la massa di richieste di servizi degli utenti/clienti: dunque anche le società di assistenza potranno avere un loro mercato. Non a caso una simile strategia è propria di Red Hat e SuSE nel mondo dell'Open Source. I fautori della GNU Economy ritengono quindi che questo genere di servizi diffusi a tutti i livelli ponga fine alla presunta necessità di software proprietari, poiché le economie basate sugli interventi di assistenza permettono un profitto «etico» in grado di far vivere dignitosamente sia gli assistenti autonomi, sia le grandi società[2].

L'Open Source, in definitiva, ha cooptato molti elementi dal Free Software tranne le sue implicazioni più etiche e politiche. In questo modo è stato possibile da un lato tranquillizzare le comunità riguardo alle intenzioni delle aziende, dall'altro rassicurare queste ultime sulle rosee prospettive di crescita economica.

Alcune pratiche come la chiusura delle comunità, il fatto di sti-

pendiare alcuni membri delle comunità stesse e di finanziare progetti che offrano gratuitamente spazio per i coder[3] si stanno affermando a macchia d'olio: in precedenza, era la comunità stessa a offrire assistenza ai propri membri e ai membri di altre comunità.

Il modello proposto è tipicamente non comunitario, di assistenza aziendale: «impacchettamento del prodotto», customizzazione totale, disponibilità a qualsiasi richiesta 24 ore su 24. Basta pagare. Si ritorna dunque a un'economia concorrenziale, in cui i tempi di rilascio diventano inesorabili appuntamenti per il lancio del nuovo prodotto e in cui l'utilizzo di software chiusi viene gestito con licenze permissive e «liberali»[4] (solo per i detentori della proprietà, non per i creatori né tanto meno per i fruitori) meno etiche della GPL; sopra tutto, naturalmente, svetta l'imperativo del profitto.

2. Open Source come strategia commerciale

Distinti i campi semantici del Free Software e dell'Open Source anche dal punto di vista meramente economico, si possono analizzare i rapporti tra quest'ultimo e il mercato.

I finanziatori e le grandi imprese che investono nell'Open Source non sembrano muoversi tutti allo stesso modo: alcuni rispettano il principio di libertà del software, altri lo utilizzano per dare vita a una sorta di «sistema misto» che permette l'utilizzo di software proprietario anche in logiche produttive Open Source.

Lo scarto è evidente e spinge le comunità hacker che si rifanno al concetto originario di Free Software a re-inventare dinamiche e strategie, confrontandosi in particolare con il nodo economico e politico legato alle recenti dinamiche mercantili e mediatiche dell'Open Source.

Il termine Open Source ormai ha conquistato anche i media *mainstream* ed è entrato nel linguaggio comune. Per contrasto, quindi, il Free Software deve cercare di uscire dal «ghetto culturale» in cui rischia di trovarsi invischiato attraverso un'evoluzione del proprio approccio al mercato. Anche perché, necessariamente, la diffusione di metodologie «aperte», in senso generico, coinvolgerà anche altri settori, non solo quello informatico-tecnologico; si tratta dunque di una sfida ad ampio raggio.

Il software Open Source viene considerato ormai comunemente affidabile, capace di performance elevate e offerto a costi sostenibili; in pratica, oggi è spesso considerato decisamente migliore dei software proprietari, proprio perché in grado di aumentare l'affidabilità di un prodotto grazie soprattutto a una metodologia differente.

Il metodo cooperativo «bazar» di Linux sembra infatti smentire la «legge di Brooks» che regolerebbe lo sviluppo dei software. La legge di Brooks afferma che, aumentando i programmatori impegnati in un progetto software, aumenta proporzionalmente la quantità di codice, ma aumentano geometricamente le complessità e i bug, rendendo in breve il progetto ingestibile. Questo inconveniente non è presente nelle dinamiche di sviluppo dei software Open Source[5].

3. All we need is software

Open Source e libero mercato convergono nel nome del software: ormai l'emersione dell'economia informazionale ha raggiunto un'importanza commerciale critica e imprescindibile in qualsiasi segmento di mercato.

Non si spiegherebbe altrimenti il fatto che tanto le multinazionali dell'IT quanto i soggetti più genericamente legati al mondo della comunicazione o delle telecomunicazioni, abbiano assunto nel proprio core business anche lo sviluppo di software. Tutti, sia chi ha sempre svolto attività nel campo dell'elettronica, sia i colossi delle telecomunicazioni che decidono di investire, ad esempio, su un progetto editoriale sul web, hanno la necessità di software.

Basti ricordare la corsa di fine anni Novanta per accaparrarsi i programmi gestionali di marketing manager più avanzati (integrati con i CMS, Content Management System): i più ricercati erano quelli in grado di supportare applicativi in grado di profilare altamente gli utenti in nome del CRM (Customer Relationship Management). Una delle forme più note di CRM è il Permission Marketing. Si tratta in sostanza di una forma evoluta di *direct* marketing che sfrutta i bassi costi di entrata di internet (rispetto a sistemi di catalogazione analogici, come le banche dati cartacee) per profilare gli utenti e fidelizzarli. Questi processi basati sul data

mining richiedono ovviamente un'alta intrusività del software proprietario o freeware, che traccia e mappa ogni movimento on line dell'inconsapevole utente, il quale, non potendo accedere al codice sorgente, non può nemmeno sapere di essere osservato e porvi rimedio[6]. La protezione dei dati personali, l'importanza della privacy, delle tecnologie di criptazione, sono senz'altro tematiche di primaria importanza, che esulano tuttavia dagli intenti di questo testo e richiedono tutt'altro spazio per poter essere sviluppate. È chiaro, in ogni caso, che l'attività (per quanto coerente con il suo ruolo istituzionale) del garante della privacy non è certo sufficiente a proteggere i soggetti che vivono i mondi digitali. Il controllo globale, a ogni livello, traccia, registra e profila i movimenti delle reti, dalle operazioni di intelligence, intercettazione e repressione alle più banali (ma fastidiose e lesive) statistiche e analisi di mercato, di voto e altro, fino alle odiose violazioni della privacy sui luoghi di lavoro. Nessuno ci darà gli strumenti adeguati, nessuno ha interesse a svilupparli: è necessario che comunità e individui si pongano il problema della sicurezza dei dati, sperimentino e creino strumenti per la criptazione, l'anonimato della navigazione, insomma per l'autodifesa digitale[7].

La corsa al software in atto da alcuni anni è, secondo alcuni osservatori, una delle conseguenze della cosiddetta «convergenza tecnologica», uno slogan ormai divenuto quasi un paradigma della nuova era: l'avvicinamento e la sinergia di varie tecnologie, precedentemente ritenute estranee, studiate e sviluppate in ambiti separati.

Dinanzi a queste trasformazioni, la creazione di standard aperti ha creato un varco: «cooperate on standars, compete on solution» è il motto della IBM, una delle principali imprese che investe nell'Open Source.

Per molte imprese l'Open Source è infatti una delle poche possibilità per contrastare i monopoli o uscire dai pericoli del mercato. È il caso di SAP, che oggi offre alcuni dei suoi servizi di business in modalità «Open», avendo deciso di rilasciare i codici sorgenti di alcuni prodotti per limitare i costi di sviluppo e quindi diminuire il «prezzo» dei propri servizi[8].

Le imprese conoscono da tempo e apprezzano il valore di una dinamica di sviluppo e di alleanze reticolare: è la stessa vecchia legge di Metcalfe, l'ideatore delle reti ethernet, ad affermare che il

valore di una rete è proporzionale al quadrato delle persone/nodi che collega[9]. L'Open Source sembra offrire alcune garanzie rilevanti nello sviluppo di reti ad alto valore aggiunto: da una parte consente al software di rimanere in qualche modo un bene «pubblico» (adotta lo sviluppo aperto e si avvale di comunità di supporto); dall'altra mantiene molto bassi quelli che vengono definiti gli switching costs, ovvero i costi derivanti dal passaggio da un sistema all'altro, in particolare dai modelli close a quelli open.

4. Open lock-in

Tuttavia l'Open Source non sfugge alle logiche di «lock-in», quel fenomeno che avviene quando i costi di cambiamento sono talmente elevati da scoraggiare il passaggio da un prodotto a un altro. L'Open Source in realtà prospetta una forma di lock-in ancora più esasperato, almeno nelle zone di mercato in cui si afferma, tale da rendere difficile, se non impossibile, il ritorno a standard chiusi una volta passati a quelli aperti.

In pratica i sostenitori dell'Open Source e le imprese che guardano con attenzione al fenomeno hanno sciolto il dubbio principale, ovvero la contraddizione apparente che sembra legare un prodotto non proprietario alla possibilità di generare profitto.

Non che con la GPL questo non fosse possibile (come abbiamo già sottolineato), ma alcuni elementi hanno facilitato la scelta della strategia Open Source al posto di quella Free Software. In primo luogo il management, tutto proveniente dal mondo commerciale, fin dall'inizio ha sponsorizzato l'Open Source, sottolineando la propria capacità creativa di intravvedere business e di indirizzare verso il mercato i propri prodotti. Al contrario, i più naïf sostenitori del Free Software sono stati immediatamente catalogati come fanatici (vedi cap. II): gente invasata con cui era meglio non fare affari. In secondo luogo la GPL stessa era un limite: la sua viralità non poteva essere sostenuta da imprese ibride alla ricerca di qualcosa di più performante in termini di «libertà» imprenditoriale.

Sono numerosi gli attori «convergenti» nell'Open Source, da quelli che ci si sono trovati a operare fin da subito con logiche commerciali, a quelli definiti «ibridi» che ben presto hanno fatto proprio, contribuendo anzi all'elaborazione dello standard Open Source.

Imprese *pure players* (ovvero quelle nate e cresciute in parallelo con l'Open Source) come Red Hat hanno dimostrato di saper generare profitti elevati anche solo basandosi sulla distribuzione di Linux: ci si sposta dal prodotto al servizio. Il valore aggiunto che la Red Hat commercializza è quello di «confezionare» il prodotto, venderlo come sicuro e affidabile grazie ad accurate verifiche di compatibilità e a controlli dei codici.

L'affare è ottimo, anche se l'approccio delle imprese all'Open Source è molto diversificato: si passa da strategie di difesa e attacco (Microsoft), a quelle di supporto (Sun, ma anche SAP), a quelle *pure players*, da sempre impegnate nella diffusione dell'Open Source, nelle vesti di un prodotto commercialmente valido, presso le pubbliche amministrazioni.

5. E adesso: concorrenza!

Se inizialmente l'obiettivo del software libero era diminuire la concorrenza, le ultime vicende dell'Open Source sembrerebbero dimostrare l'esatto contrario. Le aziende impegnate nell'Open Source non stanno infatti a guardare: HP (insieme a IBM il maggior concorrente sul mercato) ha ottenuto da poco la certificazione Evaluation Assurance Level 3 (EAL3), che permetterà ai propri sistemi Open Source di essere utilizzati anche da uffici governativi e militari statunitensi[10].

Anche i *pure players* provano a reagire: è dell'ottobre 2004 la notizia che Red Hat ha acquistato da America On Line alcuni asset di Netscape Security Solution. Il valore dell'operazione non è stato svelato, ma Red Hat ha detto di aver sborsato meno di 25 milioni di dollari.

Queste operazioni chiarificano il passaggio concettuale da Free Sofware a Open Source, perché Red Hat, oltre ad avere acquisito un prodotto, ha anche fatto proprio un «metodo». Infatti, alle critiche di Sun riguardo l'acquisto di qualcosa considerato «obsoleto», la Red Hat ha risposto che spera di migliorare e aggiornare il prodotto grazie all'aiuto della comunità di Linux e di un piccolo team di sviluppatori ereditato da Netscape[11].

Non mancano le note folcloristiche e lievemente ironiche di chi per la prima volta si avvicina al tema in modo superficiale.

Ormai dell'Open Source se ne sono accorti tutti: *Linux entra nel salotto buono*, ha titolato un noto quotidiano italiano, sottolineando come la libertà decantata dalla FSF fosse «utopica» e valutando invece positivamente la «felice declinazione» che l'Open Source ottiene nel mondo degli affari; si esplicita così l'importanza lessicale di «open» rispetto al pericoloso «free». La libertà, strano a dirsi, fa paura al «libero mercato»[12].

Il mondo degli affari si rivolge all'Open Source per rigenerarsi: imprenditori offrono finanziamenti per scoprire bug dei software Open Source[13]; Malesia, Sudafrica e Cina guardano con interesse agli esperimenti relativi all'uso dell'Open Source nell'ambito delle pubbliche amministrazioni europee[14] (vedi il caso Germania, cap. II).

Proprio per quanto riguarda le pubbliche amministrazioni è in corso un dibattito sull'opportunità o meno di finanziare progetti Open Source. Le ragioni di interesse delle pubbliche amministrazioni sono evidenti: l'Open Source assicura costi minori (le pubbliche amministrazioni sono costrette ad acquistare parecchio software, anche solo per gestire la burocrazia), maggiore sicurezza (necessaria dopo scandali come Echelon[15], quando è apparso evidente che gli USA utilizzavano prodotti close per monitorare persino i propri partner commerciali) e maggiore facilità di manutenzione[16].

Il ministero dei Trasporti francesi, ad esempio, nel luglio 2004 ha migrato su 1.500 server – localizzati all'interno di vari uffici pubblici e ministeriali – il sistema operativo Windows NT con la MandrakeLinux Corporate, la versione server della distribuzione GNU/Linux della società parigina Mandrakesoft. Le motivazioni sono le usuali: diminuzione dei costi dell'IT e volontà di aprire il mercato pubblico a una maggiore competizione[17].

Gestione gerarchica delle comunità, concorrenza, rilasci dettati da tempi non più «volontari»: il panorama dell'Open Source si affaccia nel mondo «dei grandi», scopre il marketing (o è scoperto dal marketing) e svilisce, appiattendola al semplice livello di mercato, una storia di hacking che le nuove generazioni di programmatori saranno invitati a non investigare. Troppo free, libera, anarchica, pericolosa.

6. Open Source e resto del mondo...

Il software è un artefatto digitale: la sua natura peculiare è di essere perfettamente riproducibile, infinitamente replicabile senza perdere nulla della propria originalità, e a costi praticamente inesistenti. Anzi, come gran parte degli artefatti della tecnologia digitale, il valore del software cresce di pari passo con la sua replicabilità e diffusione.

A metà degli anni Trenta Walter Benjamin, in *L'opera d'arte nell'epoca della sua riproducibilità tecnica*, analizzava l'avanzamento della tecnica di riproduzione e copia delle opere d'arte. Il filosofo tedesco leggeva questa trasformazione come la caduta «dell'aura», ovvero dell'unicità e irrepetibilità che sembrava contraddistinguere il prodotto artistico.

Lo stesso Benjamin intuiva le prospettive rivoluzionare di questa perdita dell'unicità e del valore dell'originale: replicabilità infinita significa anche rendere popolari l'arte e la conoscenza in genere, minarne alla base la «tendenza aristocratica» ed elitaria[18]. La diffusione e la riproducibilità riguardano non solo le opere d'arte, ma anche altri campi come il software o la ricerca scientifica: vi sono enormi implicazioni economiche e culturali in senso lato.

Mentre l'Open Source si appresta a conquistare sempre più quote del mercato del software, anche altri settori guardano con interesse alle metodologie che lo contraddistinguono. La ricerca scientifica, e in particolare la medicina, sembrano percorrere, per certi versi, la strada dell'Open Source, dopo aver rilevato i successi che questo approccio ha ottenuto nel campo del software.

Ancora una volta è necessario un chiarimento linguistico: anche in questi casi si tende a un uso non corretto del termine Open Source, che abbiamo visto connotarsi con metodi e finalità ben precise. Il termine viene spesso ripreso nella sua accezione parziale e corrispondente soprattutto a due caratteristiche: il codice «aperto» e la sua metodologia di condivisione.

In realtà il termine Open Source non identifica con precisione semantica questo processo: in molti settori si parla di Open Source quando non si ha a che fare con «codici sorgenti» e dunque l'analogia si rivela del tutto inappropriata[19].

Si può parlare propriamente di Open Source (specificando l'uso linguistico per indicare un metodo condiviso di sviluppo e un rila-

scio aperto) per diversi settori della ricerca scientifica, molto affini all'informatica. Al contrario, in altri settori, il termine più appropriato per indicare la tendenza a utilizzare le dinamiche e la base teorica delle comunità hacker è la parola che ha caratterizzato proprio le prime istanze del movimento: il copyleft, ovvero il permesso d'autore[20].

In molti ambiti, infatti, come per il software, il metodo di lavoro è di équipe, fondato su principi meritocratici di eccellenza e basato su una spinta volontaristica e una precisa «etica», che si coniuga alla volontà di permettere a chiunque di accedere a una risorsa, usarla, modificarla, distribuirla, senza il permesso di aggiungere restrizioni: è questa la base del copyleft. Mentre il copyright permette la privatizzazione di un bene, il copyleft lo mantiene libero e appare più elastico nella sua applicazione ad altri settori (scientifici ma anche umanistici). Il copyleft utilizza il copyright per trasformarlo da una restrizione in una garanzia assoluta della libertà del bene: l'autore mette il proprio copyright a un suo prodotto, diventandone proprietario e decidendo come tale di lasciare la possibilità di copia, modifica, distribuzione. Soprattutto, il copyleft impedisce a chiunque di impossessarsi del bene facendolo esclusivamente proprio e negando così ad altri la possibilità di accedervi.

Pur senza la benedizione iniziale dei fautori del termine Open Source[21], il copyleft sta dilagando in molti settori, non solo quelli propriamente scientifici: dalla Open Cola alla musica, fino ad arrivare all'americana Open Laws[22], un'associazione di avvocati che rilascia pubblicamente i materiali giuridici con licenza copyleft e che si avvale di centinaia di debuggers. Non mancano i limiti: ad esempio, essendo materiali pubblici, gli avvocati di Open Laws non possono contare sull'effetto sorpresa in tribunale, ma allo stesso tempo il materiale è oggetto di *sharing* (condivisione) da parte di molti cittadini[23]. Le potenzialità sono enormi e ancora tutte da sperimentare.

Il concetto di copyleft costituisce inoltre il *trait d'union* tra campi che invece con la terminologia Open Source rischiano di rimanere ambiguamente separati. In particolare nella produzione letteraria collettiva il copyleft si avvicina a quel concetto di *fair use*[24] di cui esistono esempi di straordinaria funzionalità che si basano proprio sulla durevolezza del bene relazionale; tali applicazioni si distinguono dal copyleft informatico specifico (la licenza d'uso),

vista come un'applicazione del copyleft e non come il copyleft stesso[25].

Il successo e il tentativo di applicare le logiche del software libero anche in altri settori porta inevitabilmente a coniare nuovi termini e a formulare nuovi auspici. Oggi infatti l'idea di *società open source* è divenuto un paradigma della nuova era volta alla ricerca di uno strumento comune per una prospettiva politica «possibile». Per *società open source* si intende infatti una società il cui codice sia libero e in cui tutti possano partecipare liberamente al suo miglioramento[26]. Messa in questi termini, non si può che essere d'accordo. Sorprende però la leggerezza con cui si mutua un termine da un percorso particolare, tecnico e informatico, per renderlo generale, applicandolo a teorie filosofiche, economiche, sociologiche, senza considerare le possibilità di modificare quel concetto o utilizzarlo con le dovute precisazioni.

Infatti, nel campo del software in cui è nato, il termine Open Source (vedi cap. II) significa anche concorrenza, gare, finanziamenti, operazioni miliardarie di acquisto e vendita, un grande business orientato a una forma più «democratica» e morbida di capitalismo rispetto all'attuale situazione monopolistica. L'aspetto quasi ridicolo di questa benevolenza delle corporations verso l'Open Source è la grande disponibilità di quest'ultimo a piegarsi alle leggi del mercato, quando dall'altra parte perfino il Free Software non ha ancora trovato una via d'uscita che vada oltre le prese di posizione di Stallman e le sue invettive con l'aureola di Sant'Ignuzio.

Il termine Open Source di per sé indica qualcosa che può essere comunemente percepito nella sua accezione positiva, ma si dimenticano troppo presto le ragioni politiche per le quali il termine è nato, e cioè sostituire un ingombrante riferimento alla libertà con un aggettivo più neutro e appetibile: aperto.

Il termine copyleft rappresenta meglio di Open Source ciò che unisce svariati settori dell'agire umano, e può diventare un concetto estensivo in grado di comprendere la necessità di cambiamento, la capacità di sognare e le passioni, oltre a costituire un attacco alla maggiorità in nome di un auspicato divenire minoritario della società dell'informazione.

7. Spazi aperti e spazi chiusi

Abbiamo visto che la strategia di rivendicare la propria capacità di fare mercato condanna anche la GNU Economy a stare dentro a un determinato orizzonte e spazio, a cabotare sulle onde dei tempestosi flussi finanziari.

Affrontare con la GPL gli squali della finanza è come usare una cerbottana contro i carri armati. Ingenuo e inutile. Nonostante eccezionali possibilità e prospettive, la GNU Economy diventa così un'altra rivisitazione del «libero mercato».

Vedere la rete come uno spazio aperto – spazio di libertà, spazio del possibile – significa, da una parte, mettere l'accento sull'apertura di nuove e inedite combinazioni: quando siamo connessi entriamo nei mondi virtuali, ma anche i mondi virtuali entrano nel nostro spazio vitale, lo influenzano, lo trasformano in maniera imprevedibile.

Dall'altra parte, cercando di focalizzare una possibile lettura economica della galassia internet, bisogna notare che lo spazio aperto, in movimento e dinamico, è sempre passibile di chiusura. Nessuna conquista è per sempre, nemmeno i diritti che sembrano «acquisiti», e sono necessarie continue riattualizzazioni. Lo dimostra l'avanzata dei modelli di sviluppo close, o solo nominalmente open. Nei fatti, i milioni di righe del codice di Mozilla sono un paradosso: nessuno può intervenire «aprendo» il codice a nuove possibilità, a meno che non sia «organico» a una struttura gerarchica niente affatto trasparente e orizzontale.

Il modello dell'economia di mercato – assolutamente «close», perché basato sull'accumulo privatistico del cosiddetto «individuo razionale», che opera «nel suo interesse» e non in base alla condivisione comunitaria e alla volontà di riconoscimento – si è imposto più per ragioni culturali e storiche che per ragioni strettamente razionali o per calcoli di sottili economisti, come invece i profeti del libero mercato si ostinano a predicare[27].

Il mercato capitalista non è una necessità storica, ma un accidente, un caso, e per di più un evento rovinoso per la sua pervasività distruttiva in tutti i settori della vita umana: tutto ha un prezzo, il tempo è denaro, libero mercato ovvero la guerra di tutti contro tutti, e simili amenità, sono moneta corrente o peggio dogmi incontestabili. In questa prospettiva, pensare che realmente l'unico

sbocco plausibile anche per la rete e per coloro che la animano sia ascoltare le sirene del mercato – che insistono a gran voce a dire: è finito il tempo dei giochi, rimboccatevi le maniche e cominciate a lavorare, è ora di diventare grandi – è probabilmente un errore di valutazione, una chiusura di orizzonti, la negazione del possibile.

Riconoscere il valore economico del prodotto digitale, riconscere il valore anche monetario del flusso informazionale, ma ricondurlo a una dimensione regolabile e regolata a misura dell'umanità che genera questo valore diventa allora una necessità. Ridimensionare l'importanza dell'economia, ovvero della legge, la regola dell'ambiente umano (oikos-nomos), e riportarla a un'ecologia (oikos-logos), cioè a un discorso sull'ambiente nel suo complesso, che non può essere altro che un discorso complesso sulla tecnosfera. In un certo senso, la necessaria sovversione gerarchica del rapporto fra fattori economici e fattori sociali che ne segue è già avvenuta nell'ambito delle comunità hacker: le persone che si muovono nella rete hanno da tempo messo avanti i loro desideri (dunque i fattori sociali), ampliando a dismisura l'apertura di possibilità intrinseca allo spazio aperto della rete. Codice libero, libertà, significa allora anche resistenza alla colonizzazione economica messa in atto da individui e comunità: sono questi i soggetti capaci di usare tecnologie avanzate e reti come campo di sperimentazione per i loro bisogni, desideri, bio-sogni, e quindi come nuovi territori di spazio e tempo politico in senso quanto mai personale, collettivo e vitale.

8. Beni, merci, relazioni

Questa sovversione di senso ha altri aspetti: la fiducia, che sembra tanto fondamentale nello scenario di condivisione, di apertura a molteplici livelli di gioco del codice, non ha infatti la minima importanza dal punto di vista del mercato. Il consumatore (educato e formato per questo) finge di credere che in cambio di soldi il produttore gli venda qualcosa che gli serve per renderlo felice, per soddisfare desideri e bisogni. La riduzione di prospettive del software close, ma anche dell'Open Source e della GNU Economy derivata dal Free Software, nel momento in cui sembra che l'unica questione aperta sia la relazione con il mercato (e non piuttosto i metodi di creazione e uso dei software), non potrebbe essere più netta.

Tolta di mezzo la fiducia, ovvero ciò che rende possibile lo scambio, la relazione nel contesto free, rimangono le merci, la cui circolazione è resa possibile dal movimento virtuale (ma sempre concretissimo) del denaro.

Nel linguaggio del mercato le merci sono il fulcro di ogni movimento e flusso. Il mercato prospera con le merci, che sono beni non durevoli. Infatti il mito del progresso infinito del mercato implica una merce sempre «nuova», dunque il consumismo.

La comunità Free Software ha però un valore aggiunto non da poco, essendo composta di individui che si relazionano fra loro. La relazione è l'elemento in più. Il bene relazionale, portato dall'apertura di fiducia nella relazione fra individuo e comunità, ha la caratteristica di essere durevole. La «razionalità» mercantile peraltro non contempla cose tanto inafferrabili come le relazioni complesse fra gli individui: al massimo, può sfruttare le relazioni fra gli individui per aumentare i profitti.

Semplificando, mentre un prodotto chiuso non è altro che una merce, un prodotto aperto come il Free Software è un bene. Per produrre merce è necessario avere denaro; vendendo la merce si otterrà altro denaro per produrre nuova merce. Questo ciclo può essere più o meno complesso, ma ciò che conta è l'inizio (il denaro), lo sviluppo (la merce), e la fine (ancora il denaro). Sembra non esserci via d'uscita.

Viceversa, per creare un software libero non è necessario denaro. È necessario tempo investito come volontà di cooperare da parte di sviluppatori e coder che vogliono partecipare a un progetto libero, o che creano usando liberamente il loro tempo un software libero e lo mettono a disposizione della comunità attraverso il meccanismo della licenza virale (GPL, LGPL, FGDL, ecc.).

Il ciclo non parte più dall'investimento di denaro, ma dall'investimento di tempo e volontà di singoli; questi possono anche guadagnare denaro dalla loro creazione (perché GNU Economy e Open Source Economy non escludono il mercato), ma in realtà trovano una gratificazione essenziale, non monetaria, nel fatto che la comunità di riferimento si arricchisce di un nuovo bene, il software aperto. Esso inoltre non scade nel tempo, anzi acquisisce sempre nuovi usi, poiché circola nelle comunità e migliora in continuazione.

Le corporations da tempo hanno capito che il software aperto

non impone investimenti monetari, o meglio richiede investimenti minimi e può avere ricadute commerciali; dal loro punto di vista è dunque solo un ottimo affare: investimento quasi zero (al massimo un manipolo di programmatori indiani, tra i migliori in assoluto, ovviamente sottopagati) e prodotti eccellenti da rivendere.

L'investimento capitalistico è l'unica forma di autoreplicazione del mercato, sempre identica a se stessa nelle sue finte innovazioni, sotto forma di rappresentazioni pubblicitarie di bisogni indotti. L'investimento relazionale, di tempo, volontà, desiderio, dialogo, sfugge a questa logica: lo spostamento dell'investimento provoca un cortocircuito nella definizione stessa della ricchezza, per cui non è più ricco chi ha (più merci, più denaro), ma chi dà (più tempo, più desiderio). Chi dà di più, creando software aperto, sarà riconosciuto e gratificato all'interno delle comunità; e all'esterno, volendo, potrà essere gratificato dal riconoscimento monetario dato dalla vendita del servizio, non del prodotto (è free). Del mercato si sa tutto (è la religione corrente), e molti si impegnano a spiegarci in ogni momento come e perché sia essenziale, insostituibile, inevitabile, anzi addirittura bello; ci siamo concentrati sulle licenze, sulle comunità: rimane ora l'individuo e le scelte che può operare per immaginare uno scenario in cui il mercato perde, a poco a poco, la sua importanza. Si tratta di scelte individuali, di accollarsi responsabilità senza scaricarle sulla società.

Note al capitolo

1. http://www.gnu.org/philosophy/software-libre-commercial-viability.it.html

2. http://www.gnu.org/philosophy/software-libre-commercial-viability.it.html

3. Tra i finanziatori di Sourceforge.net, il portale della Va Software Corporation, troviamo anche IBM.

4. Il termine «liberale» è coerentemente utilizzato dai sostenitori dell'Open Source: sono moderati, non certo «amanti fanatici della libertà».

5. Sulla legge di Brooks si veda anche http://www.apogeonline.com/openpress/libri/545/raymondb.html

6. Del tutto prevedibilmente, e la cosa è davvero poco incoraggiante, esistono anche software Open Source per attività di CRM: http://www.sugarcrm.com/home/

7. Si vedano per una prima introduzione all'argomento: le mini guide su sistemi di anonimato e criptazione a cura di Autistici/Inventati, http://www.autistici.org/it/howto/index.html; la vecchia ma sempre valida storia di Joe Lametta, http://www.ecn.org/kriptonite/kriptonite/sommario.html; lo strumento di criptazione per eccellenza GPG (GNU Privacy Guard), http://www.gnupg.org/(it)/

8. Si veda Moreno Muffatto, Matteo Faldani, *Open source. Strategie, organizzazione, prospettive*, il Mulino, Bologna, 2004; SAP è il leader mondiale delle soluzioni per il business, con un'offerta completa di software e servizi in grado di soddisfare ogni necessità. La sua produzione è definita di *software ibrido*.

9. SITOGRAFIA: *Le reti*. Si tratta di una legge matematica formulata alla fine degli anni Settanta da Robert Metcalfe, studente della Harvard University e poi fondatore di 3Com, oltre che pioniere del networking (e inventore del protocollo ethernet). Ecco la traduzione *ad hoc* del passaggio saliente: «Il valore di un network cresce esponenzialmente rispetto al numero dei computer connessi al network stesso. Quindi, ogni computer che si aggiunge al network da una parte utilizza le risorse di questo (o le informazioni o le competenze) e dall'altra porta nuove risorse (o nuove informazioni o competenze) al network, facendone incrementare il valore intrinseco». Da questo principio generale deriva che: 1) il numero di possibili relazioni, o meta-relazioni, o connessioni all'interno di un network, cresce esponenzialmente rispetto alla crescita di computer a esso collegati (di qui il valore strategico dei link all'interno di una rete); 2) il valore di una comunità cresce esponenzialmente rispetto alla crescita degli iscritti a questa comunità (di qui il valore strategico delle comunità virtuali). Si veda http://en.wikipedia.org/wiki/Metcalfès_law

10. EAL3 è un'attestazione che fa parte di uno standard di valutazione internazionale noto come Common Criteria for Information Security Evaluation (CC). La rivale diretta di IBM nel settore, ovvero HP, ha ottenuto questo livello di certificazione per alcuni suoi sistemi, in particolare i server ProLiant e i computer basati su processori Itanium 2, su cui girano i sistemi operativi enterprise di Red Hat e SuSE Linux (http://punto-informatico.it/p.asp?i=49993).

11. http://punto-informatico.it/p.asp?i=49824

12. Negli ultimi anni aziende, scuole e pubbliche amministrazioni hanno dimostrato di essere sempre più disposte a migrare i loro sistemi informatici nell'universo Open Source: si veda http://www.corriere.it/Primo_Piano/Scienze_e_Tecnologie/2004/09_Settembre/22/Linux.shtml

13. Mark Shuttleworth, imprenditore americano, metterà a disposizione dell'iniziativa i primi 5.000 dollari per chi scova bug di Mozilla (http://punto-informatico.it/p.asp?i=49251).

14. http://www.apogeonline.com/webzine/2004/09/29/05/200409290501

15. Su Echelon si veda su internet il testo originale di Duncan Campbell, che

ha fatto esplodere la questione (http://duncan.gn.apc.org/echelon–dc.htm), e il suo libro, rivisto e ampliato in italiano, *Il mondo sotto sorveglianza. Echelon e lo spionaggio elettronico globale*, Elèuthera, Milano, 2003.

16. Gli osservatori più attenti sostengono correttamente che l'Open Source è l'estensione del mercato, non una sua alternativa. Si veda Muffato Moreno, Faldani, *op. cit*.

17. http://punto-informatico.it/p.asp?i=48945

18. Bisogna però notare come l'aura sia a volte ricreata proprio dal mondo informatico iperveloce e iperconnesso, come rileva il dromologo Paul Virilio in diversi interventi, distinguendo fra *arte attuale* e *arte virtuale*, e parlando anche di *arte terminale*, arte innervata di morte dopo la morte dell'aura. Sulla rinascita «pubblicitaria» dell'aura nell'opera d'arte si veda tra l'altro Paul Virilio, Enrico Baj, *Discorso sull'orrore dell'arte*, Elèuthera, Milano, 2002.

19. *Medicina aperta*, «Internazionale», 16-22 luglio 2004.

20. Abbiamo visto nel cap. I che l'espressione inglese copyleft, gioco di parole su copyright, indica un tipo di licenza libera che, pur garantendo le libertà previste dalla definizione, impone delle restrizioni sul rilascio di opere derivate in modo tale che queste si mantengano sempre libere, generalmente sotto la stessa licenza dell'opera originale. Esempi di licenze copyleft per il software sono la GNU GPL e, in minor misura, la GNU LGPL, per altri ambiti le Creative Commons License con la clausola *share alike*. Per un'introduzione alle questioni relative al copyright, si veda: Raf Valvola Scelsi, *No Copyright*, ShaKe, Milano, 1994.

21. Si veda l'ammonimento ad applicare metodi Open Source a prodotti non software in Eric S. Raymond, *La cattedrale e il bazar*, cit.: «La musica e la maggior parte dei libri non sono come i programmi informatici, perché in generale non hanno bisogno di essere corretti o aggiornati». Senza questo bisogno i prodotti guadagnano poco dall'esame e dal rimaneggiamento di altre persone, perciò un sistema Open Source offrirebbe scarsi benefici. «Non voglio indebolire l'argomento vincente dei programmi Open Source legandolo a un possibile argomento perdente», sostiene ancora Raymond (www.newscientist.com/hottopics/copyleft).

22. http://cyber.law.harvard.edu/openlaw/, ma anche, www.openlaw.co.uk in Gran Bretagna e www.openlaw.ch in Svizzera.

23. http://internazionale.it/home/primopiano.php?id=2241

24. Il *fair use* è una dottrina legale statunitense che permette limitazioni ed eccezioni alla tutela legale del copyright. Se l'uso di un lavoro è definito *fair use*, il proprietario del copyright non ha il diritto di controllarlo e non sono necessari né licenze né permessi. Il *fair use* è presente esclusivamente negli Stati Uniti, ma principi simili sono in vigore anche in altri stati; in Italia al momento non vi sono leggi che possano essere considerate assimilabili al *fair use*.

25. La posizione di Wu Ming 1: http://www.wumingfoundation.com/italiano/outtakes/nota_copyleft.html

26. Negri e Hardt suggeriscono di raffigurarci «la democrazia della moltitudine» come una «società open source» (Michael Hardt, Antonio Negri, *op. cit.*). Preferiamo piuttosto immaginarci una «democrazia diretta di individui» come una «società libera».

27. Si veda ad esempio Karl Polany, *La grande trasformazione*, Einaudi, Torino, 1974 (ediz. orig.: *The Great Transformation*, New York, 1944). La grande trasformazione è quella verificatasi dopo la crisi del 1929, con il rovesciamento della tendenza che aveva dominato il mondo a partire dalla rivoluzione industriale e l'affermarsi dell'economia di mercato. Lungi dall'essere un approdo naturale e necessario, il mercato autoregolato segna, per Polanyi, una fase eccezionale, una parentesi rovinosa nella storia, le cui contraddizioni e la cui crisi non a caso hanno prodotto il fascismo. Da questo punto di vista l'Ottocento, l'epoca del trionfo dell'economia di mercato e delle istituzioni liberali, si concluse propriamente con gli anni Venti del Novecento. La tesi dell'eccezionalità e insieme dell'artificiosità dell'economia di mercato viene argomentata grazie a un'analisi comparata di società diversissime: da quelle primitive a quelle feudali, moderne e contemporanee, in Europa e altrove. Studioso di antropologia e di economia comparata, Polanyi non è fautore di una pianificazione rigida, ma auspica il ritorno a forme di protezione della società dai meccanismi distruttivi di un'economia non regolata.

V

ICS SUNT LEONES

1. Formazione permanente

Educazione e formazione sono due elementi cardine dell'economia dell'era informazionale.

È evidente, ad esempio, che una scuola situata in una condizione di isolamento relazionale, in un mercato sempre più improntato alla trasmissione dell'informazione, non riuscirà a fornire le capacità di interagire con il mondo circostante: si trasformerà quindi in un terreno fertile per la cosiddetta «irrilevanza sociale»[1].

Simili analisi sono valide e condivisibili per quanto riguarda la sistematizzazione di alcuni processi; decisamente meno convincenti sono però le conclusioni.

Sottolineare l'importanza della condivisione e del networking, ribadendo però continuamente il ruolo di garanzia delle istituzioni, conduce sempre a sostenere la possibilità di formare persone per

un «altro capitalismo possibile». L'ottica proposta rientra a pieno titolo nei meccanismi di mercato e di produzione, anche informazionale: è improntata al consumo e basata pur sempre su modelli statici e repressivi, come solo lo possono essere le istituzioni e il loro sistema di rappresentanza e delega.

La rivalutazione della «democrazia reale» sembra essere il leitmotiv dell'intellighenzia socialdemocratica, ben decisa a conquistarsi la maggiorità ancor prima della maggioranza; questo orientamento invece non si pone affatto problemi riguardo a percorsi che – ben più profondi e critici – tendono a invertire il senso: non una base più ampia, che garantisca maggior consenso e dunque maggior potere, ma più basi autogestite, fittamente intrecciate fra loro in contesti di affinità[2].

La formazione, una delle chimere dell'era informazionale, considera l'utente come produttore e consumatore: sfuma le differenze, finge di ampliare le scelte, salvo poi ridurle a quelle realizzabili, praticabili sul mercato. Questa formazione serializzata e l'educazione alla disponibilità totale tende a trasformare anche il tempo dell'ozio – in generale il tempo «improduttivo» e «libero» – in una produzione «sistematica» attraverso il consumo, l'enfasi sull'occupazione e la chiusura della proprietà delle informazioni.

Nell'ambito tecnologico un esempio paradigmatico è quello delle patenti europee informatiche (ECDL): un attestato che rivela non capacità critiche ma un utilizzo da automa del computer, con Windows come solo sistema operativo. La patente di guida del computer è limitata quindi solo all'uso e non certo alla comprensione di come funziona una macchina, al porsi domande, a percepire la possibilità di una formazione permanente di ben altro livello, che le comunità invece sono in grado di creare.

Viviamo in un'epoca di produzione su larga scala, anche relativamente alle competenze, tesa alla creazione di produttori e consumatori degli stessi prodotti, alla rincorsa costante degli aggiornamenti, perché la formazione permanente – ma, come per le merci, non durevole nel tempo, nonostante i buoni auspici socialdemocratici – rimane poco interessata alle competenze e all'interdisciplinarietà. Ciò che conta non è l'individuo, ma piuttosto il circolo «virtuoso» (per il mercato) di creazione di nuove identità (profiling) e bisogni, tanto per chi deve formarsi quanto per i consumatori finali[3].

La formazione permanente, ai valori del mercato e della globalizzazione nelle sue accezioni più becere, è uno degli strumenti più sottili ed efficaci attraverso i quali le odierne società di massa producono l'atomizzazione e il silenzioso consenso per una gestione più comoda, ma democratica... della cosa nostra pubblica[4].

Le comunità digitali in questo senso sono un esempio di formazione permanente autorganizzata che si sostituisce in parte allo stato, al pubblico. Il concetto stesso di welfare, da lungo tempo comatoso, non pare possa essere resuscitato da quelle componenti formative (le comunità, l'autoapprendimento scelto dal singolo) che in pratica sono in grado di sostituirlo, senza creare dipendenza da un organo istituzionale che forma per produrre e al contempo esclude la libertà di cercare forme alternative. Né la socialdemocrazia, né tanto meno le prospettive liberali di promozione del terzo settore, del no profit, e di sviluppo di economie civili[5] offrono alternative soddisfacenti.

La direzione intrapresa dall'educazione globalizzata produce atomizzazione e alienazione: è centrale la conoscenza privata nel mondo del lavoro, fatto di *skills* e forte verticalizzazione delle competenze, senza alcuna possibilità di condivisione; anzi, proprio l'esclusività delle conoscenze offre o preclude la possibilità di accedere al lavoro.

Formazione al sistema da un lato, atomizzazione individuale dall'altro. Gli individui sono guidati, attraverso consigli suadenti e luccicanti (pubblicità e advertising in genere), ma anche grazie a strumenti di coercizione e terrore piuttosto rozzi (lo spauracchio della disoccupazione, della povertà, della recessione e dell'esclusione), su un terreno di accettazione formale dello status quo. In questo modo, nessuno è spinto a mettere in dubbio i dogmi della produzione a ogni costo. Eppure, più produciamo, meno abbondanza sembra esserci: stranamente, siamo sempre in una società della scarsità. Le specificità, irrigidite in compartimenti stagni, favoriscono una situazione di consapevole assenso a qualunque proposta. Va tutto bene, caro consumatore utente! Stai tranquillo!

Il consumatore consuma un prodotto. L'utente usa un servizio. Il servizio è fornito dallo Stato, dalle istituzioni. Nel caso della rete di internet non solo consumatore è un termine inadatto, perché propriamente parlando non si consuma nulla, ma anche utente è

improprio per chi nella rete si relaziona con comunità, aziende e istituzioni più o meno strutturate e intrecciate fra loro.

La pericolosa scommessa della strategia Open Source è proprio questa: oscillare continuamente fra la rassicurante proposta di democrazia controllata della maggioranza e l'efficacia, rapida e adeguata, del modello aziendale.

L'Open Source infatti riesce a coniugare una concreta possibilità di scelta fra strumenti «aperti» e «razionalità» economica classica. Nessuna contraddizione, nessuno scontro. L'esempio più chiaro è la strategia di Apple: offrire non solo prodotti tecnicamente all'avanguardia, ma anche altamente usabili dal grande pubblico, con uno studio ergonomico perfetto alle spalle e la possibilità di personalizzare la propria macchina fin nei minimi dettagli, utilizzando addirittura software libero, come nella migliore tradizione hacker[6].

Come abbiamo anticipato nel capitolo precedente, in uno scenario desolante di «capitalismo dal volto umano», una via d'uscita sembra intravedersi: la scelta personale.

2. Origini: metodi e scelte

La formazione attuale si picca di istruire, ma certo non mette in discussione origini e modalità di trasmissione dei saperi: è molto concentrata sull'esito, poco sui metodi e processi.

L'hacker, nella rete priva di Stato e organi istituzionali, trova nella comunità e nel concetto basilare di meritocrazia continui stimoli alla preparazione e alla formazione, sia sotto forma di aggiornamenti tecnici ottenuti dalla condivisione di competenze, sia sotto forma di rimandi alla storia di cui fa parte, ovvero quella delle comunità hacker e delle sue caratteristiche etiche.

Idealizziamo la figura dell'hacker: studio appassionato, scelte di autoformazione fuori dal mercato, curiosità e scambio con le comunità di riferimento, reti di relazioni ampie e variegate. L'hacker non si accontenta di racconti, veritieri o meno, ma ha bisogno di scorgere la fonte, di toccare con mano la sorgente, l'origine. Metterci sopra le mani.

Questo è lo spirito della formazione che ci piace: poter aprire, vedere dentro quanto ci viene raccontato, con la possibilità di ri-

salire alla fonte e poter reinterpretare quanto appena appreso, stratificare le conoscenze individuali e contribuire a quelle collettive.

La scelta di un utente nell'utilizzo di un software libero, rispetto a uno proprietario, va nella stessa direzione, quella di riappropriarsi dell'origine, del codice.

3. Autoformazione

Si delinea un modello di formazione molto distante da quello proposto dalle élites tecnoburocratiche. Cerchiamo di scandirne i momenti: riappropriazione, rispecchiamento (modifica e personalizzazione), rappresentazione; nel complesso questo processo di formazione crea un equilibrio omeostatico[7] fra i singoli individui e le comunità a cui appartengono.

Lo scarto fondamentale rispetto alla formazione permanente globalizzata è la scelta. Restringiamo momentaneamente la prospettiva al coder puro, colui che scrive codice. Il coder sceglie di partecipare a un progetto, o di crearne uno nuovo, spinto essenzialmente dal proprio gusto personale, da un desiderio di creazione nel quale investe il proprio tempo e la propria intelligenza. Il primo passo è senz'altro «metterci sopra le mani», in senso hacker: un movimento di riappropriazione, smontaggio e comprensione di un oggetto. L'individuo, in questo contesto, non è affatto una tabula rasa da formare secondo un paradigma calato dall'alto, ma si autoforma sperimentando, ampliando in continuazione la propria «cassetta degli attrezzi». Perciò è innegabile che un coder venga profondamente influenzato dallo stile di altri coder, dalle modalità di risolvere un dato problema, dalla struttura di alcuni algoritmi che trova particolarmente interessanti e funzionali.

In un secondo momento, lo stile personale che il coder sviluppa, e che rispecchia la sua personalità, trova posto in un contesto di condivisione: il codice che crea entra in un circolo comunitario, viene condiviso e influenza a sua volta lo stile di altri coder. I codici rispecchiano gli individui che li creano, e informano di sé, modificandole, le comunità che implementano o anche semplicemente utilizzano quel codice.

Naturalmente le cose si fanno assai più complesse se consideriamo che i coder puri sono animali rari, come del resto la purezza

in sé non è una caratteristica saliente di nessun individuo particolare (a meno che costui non l'abbia coscientemente scelta per autodefinirsi). È difficile che un coder scriva solo e sempre codice nella sua vita; ovvero, estendendo il concetto, è quasi impossibile che un individuo possa essere descritto completamente da una sola qualificazione. Probabilmente farà anche altre cose, diverse in diversi momenti della sua vita; come ogni individuo è indescrivibile da un solo punto di vista: sarà situato fisicamente, sessualmente, linguisticamente, politicamente, storicamente, ecc.

Banalmente, mentre un atomo formato in permanenza a essere il più possibile intercambiabile con qualsiasi altro atomo deve sviluppare caratteristiche standard per essere appetibile (al mercato globale), un individuo sarà invece tanto più interessante per le comunità e per gli altri individui quanto più sarà unico, dotato di caratteristiche particolari, miscela irripetibile di differenti ingredienti ed esperienze. È probabile dunque che un individuo appartenga a diverse comunità contemporaneamente (culturali? politiche? tecniche?).

Appartenere a una comunità significa allora sentirsi rappresentati da quella comunità, e non certo perché si ha diritto di veto o potere di voto, ma perché si influenza direttamente l'immagine della comunità stessa, si influenzano gli altri individui attraverso le proprie creazioni, e ci si fa influenzare da loro. Si cambia, e si inducono cambiamenti. Autoformazione significa farsi individui nel farsi comunità.

Nel momento in cui la comunità, per una serie di ragioni, non rispecchia più la propria individualità, le possibilità sono molte: creazione di una nuova comunità (*forking*)[8], introduzione di nuovo codice che ci rappresenti maggiormente, ecc. Si tratta in ogni caso di un equilibrio dinamico, di una omeostasi e di una rinegoziazione continua della propria individualità e della conformazione della comunità nel tempo.

Non si possono immaginare individui statici che intervengono in comunità perfettamente e compiutamente codificate, aderendo totalmente a un manifesto o a una dichiarazione di intenti (o a una costituzione e relativo corpus di leggi, se ci forziamo a immaginare uno Stato come una comunità) e sottomettendosi al diktat della maggioranza, o peggio a leggi teoricamente uguali per tutti, ma mai effettivamente negoziate dai soggetti. La formazione in-

vece deve necessariamente investire l'individuo e la comunità, avere il coraggio di immaginare nuove vie di fuga, nuove possibilità di individuazione – poiché si cresce, si muta – che non siano flessibilità totale all'inflessibile mercato, ma operazioni di riappropriazione, rispecchiamento e dunque rappresentazione di sé in un contesto comunitario.

4. Scarti e vie di fuga: lo stile

Immaginare percorsi di autoformazione significa dunque operare scarti rispetto a una presunta norma. In ambito software, il pensiero va immediatamente all'organizzazione di corsi a diversi livelli; alla scrittura di manuali, how-to, guide; all'utilizzo di strumenti di scrittura e creazione comunitari; a possibili laboratori ed esperimenti per risolvere problemi pratici o anche solo per giocare con una nuova tecnologia. L'idea che attraverso metodi condivisi sia possibile aumentare e stratificare il livello complessivo di competenze – ma soprattutto di piacere che una determinata attività crea – riposa sulla convinzione che le conoscenze possano e debbano essere il più possibile diffuse e contaminate, e non appannaggio di un'élite di tecnocrati. Se una conoscenza non è comunicabile, nella migliore delle ipotesi è un bel gioco solitario, nella peggiore è uno strumento di potere potenzialmente pericoloso e repressivo. La condivisione è la migliore garanzia che abbiamo.

Naturalmente, poiché non tutto si può dire con un codice qualsiasi, ma esistono codici più o meno appropriati a seconda del target che si vuole raggiungere, il «come» si comunica una conoscenza è fondamentale almeno quanto il «cosa» si comunica. O, per meglio dire, si tratta di due facce della stessa medaglia.

Questo «come», cioè lo stile, il metodo, è il discrimine essenziale nell'utilizzo di un codice. Dal punto di vista dell'utente, che al di fuori dell'ambito informatico diventa il «ricevente» del messaggio, lo stile è la modalità con cui un contenuto gli viene passato. Nella realtà, spesso il messaggio passa nei due sensi, ovvero accade che l'utente risponda al creatore del codice. Nell'ambito di mercato, si tratta del noto meccanismo per cui l'utente (*consumer*), specie se evoluto, debugga, avanza richieste di migliorie, ecc., insomma diventa anche produttore (*productor*), cioè *prosumer*.

Spesso è difficile stabilire dove inizi il circolo fra proposta di un codice, accettazione e modifica, ritorno al creatore, e così via. È interessante notare che più lo stile cerca di facilitare l'utente, rendendo semplice e fruttuoso un suo feedback, più si creano dinamiche collaborative che migliorano il codice.

Dal punto di vista del coder, invece, lo stile è una specie di biglietto da visita che si trova inscritto in ogni riga del suo codice, disponibile per chiunque abbia le competenze tecniche per leggerlo e apprezzarlo.

Analizzando lo stile si possono identificare due estremi: un aspetto prettamente ergonomico, cioè come il programma si presenta e come interagisce con l'utente, e uno relativo al codice col quale è stato scritto. Il primo aspetto definisce i canoni di una interazione utente-software e può presentarsi come la sintassi di un file di configurazione, una interfaccia testuale, un sistema di login, o con differenti approcci a seconda della funzionalità e del target degli utilizzatori. Il secondo elemento, invece, riguarda il rapporto tra coder e software: il suo stile si manifesterà nell'indentazione, nella struttura delle routine, nella qualità dei commenti, nella pulizia complessiva del codice, ecc.

Si può dire quindi che lo stile risponda al complesso dei propri immaginari, al metodo di realizzazione concreta del desiderio di creazione di un oggetto.

5. Tracciare rotte, costruire ponti, declinare immaginari

Evidenziare l'importanza dello stile, cioè del «come» si fanno le cose, implica senz'altro una grande assunzione di responsabilità da parte dei singoli. Non è indifferente dire o fare in un modo piuttosto che in un altro. E non esistono certificazioni che garantiscano la «riuscita» di un codice. Se però osserviamo, come abbiamo cercato di fare finora, come si muove l'ambito informatico, notiamo una dinamica ricorrente.

Tendenzialmente gli informatici amano costruire ponti. Amano immaginarsi collegamenti inediti fra gli strumenti che usano e dai quali traggono piacere. Questi «ponti» normalmente servono per farci transitare gente.

La gente siamo noi quando scopriamo che da oggi Open Office ha il supporto per il database mysql (ODBC) e permette di estrarre dati da tabelle di un database anche remoto. L'utilità è dettata dalle necessità che ogni utilizzatore riesce a immaginarsi: stampare etichette di una rubrica, gestire un layout grafico di documenti fiscali, ecc. L'unione di due elementi apparentemente troppo distanti provoca uno stimolo a ricercare utilizzi di questa nuova tecnica.

Si possono immaginare un numero enorme di applicazioni di questi «ponti»: la curiosità, l'immaginazione, sono sempre in primo piano.

Esistono poi ponti che la gente non percorre ma che i programmatori percorrono. Un ponte può unire al linguaggio di scripting PHP il supporto per le librerie grafiche GTK. PHP nasce come linguaggio per il web. Le librerie GTK servono per scrivere interfacce grafiche in C e C++. Grazie a questo ponte, altri programmatori potranno scrivere applicazioni grafiche, basta che conoscano il PHP: non è più necessario che conoscano a fondo linguaggi complessi come il C.

In ultimo, esistono ponti fatti per non essere percorsi. Costruiti forse solo per gioco, perché l'idea di un ponte, una volta immaginata, è piacevole da realizzare a prescindere dai possibili usi. Un esempio assurdo: inserire in brainfuck[9] la possibilità di usare le syscall di Linux. Ovvero poter utilizzare le chiamate proprie del kernel (connessioni a internet, gestione del filesystem, accesso a periferiche e altro) in un linguaggio Turing-completo come brainfuck, che ha solo otto istruzioni possibili e che per sua natura nasce come pippa mentale (da cui il nome).

La strategia del «ponte» è un modo di declinare la propria immaginazione e di collegarla agli altri. Quanti ponti si possono costruire, quanti codici si possono collegare?

Abbiamo molto da fare, molto da imparare, molti desideri da realizzare.

Note al capitolo

1. Manuel Castells, *La città delle reti*, Marsilio, Padova, 2004.

2. SITOGRAFIA: *Modelli: Rete - Tela. Da Hakim Bey alla metafora del tessere nel discorso femminista contemporaneo*.

3. Il metodo di profilazione di nuove identità nel mondo del software è addirittura banale: campagne sulla proprietà del software, sul tempo che corre rapido e sulla necessità di essere aggiornati. Si va dalle nuove release dei software proprietari, nuovamente da imparare e conoscere, oltre che da acquistare, fino ai nuovi virus da cui ci si deve proteggere, o al nuovo gadget tecnologico assolutamente irrinunciabile per essere *update*. Pare necessario essere rapidi nel capire i tempi e, soprattutto, correre da soli: il mondo del lavoro è la giungla, l'oceano infestato da squali che bisogna saper azzannare con più ferocia per non soccombere. Sopravvivenza è la parola d'ordine; la progettualità, la capacità di immaginare e creare non ha spazio alcuno.

4. SITOGRAFIA: *Christopher Lasch, da sorvegliare e punire al controllo morbido*. Christopher Lasch (1932-1994) è stato professore di storia all'Università di Rochester. Nelle sue opere, in particolare *L'io minimo. La mentalità della sopravvivenza in un'epoca di turbamenti*, Feltrinelli, Milano, 2004, si delinea fra l'altro la tragicomica vicenda dell'io occidentale, l'ex homo faber del proprio destino: se la vita quotidiana diventa un esercizio di mera sopravvivenza, è necessaria la costruzione di una micro-identità (l'io minimo, appunto) funzionale alle difficoltà del presente e alla perpetuazione del sistema. Tragicommedia, perché le chiavi di lettura del mutamento approntate da Lasch si riferivano al desolante panorama del reaganismo rampante, eppure si attagliano alla perfezione anche al panorama odierno, oltre vent'anni dopo.

5. Si veda l'impostazione liberale ad esempio in Stefano Zamagni, *Per un'economia civile nonostante Hobbes e Mandeville*, http://www.pust.edu/oikonomia/pages/2003/2003_ottobre/studi_2.htm

6. La maggioranza che accetta gli standard – a proprie spese, ben inteso – è spinta a imparare la vulgata, duratura quanto la nuova release degli strumenti su cui si forma (gli artefatti digitali presentano un altissimo grado di depauperabilità); chi si posiziona fuori da questo raggio è costretto a cercare altrove le proprie istanze formative.

7. SITOGRAFIA: *L'omeostasi*. Il concetto di omeostasi, per indicare un equilibrio dinamico, viene utilizzato, al di là dell'ambito biologico che l'ha coniato, anche nella sistemica e nella cibernetica in particolare. Avendo a che fare con le relazioni fra macchine ed esseri umani, non è un caso che la cibernetica si interessi

altrettanto di «riflessività» (intesa come meccanismo di retroazione) e di «emergenza» (l'esplosione della complessità, dati elementi «semplici»). Tuttavia, dato l'evidente obiettivo e interesse militare di questi studi, si preferisce qui tralasciare ogni parallelo eccessivamente stringente. Più interessante potrebbe essere un riferimento al pensiero sulla tecnica e la modernità di Bruno Latour, ma soprattutto di Gilbert Simondon a proposito dei processi di individuazione e scambio individuo-comunità, al di fuori naturalmente delle forzature postmarxiste a cui è già stato ampiamente sottoposto. Si vedano in particolare Gilbert Simondon, *L'individuation psychique et collective*, Aubier, Paris, 1989; *Du mode d'existence des objets techniques*, Aubier, Paris, 1989. Per esempi di forzature improprie: http://multitudes.samizdat.net/article.php3?id_article=1563

8. SITOGRAFIA: *Forking*. A volte capita anche il contrario: una comunità che si divide perché alcune sue parti arrivano a tensioni non sanabili. Il *forking* si verifica appunto quando una parte di una comunità occupata in un progetto già avviato si distacca dal nucleo di programmatori per creare una versione diversa del software. Il *forking* è successo a molti progetti conosciuti come Emacs, l'editor di Stallman, Xfree, il server grafico di Linux, The Gimp, software di manipolazione di immagini. Le ragioni di questi scissioni sono varie: una parte della comunità di Xfree, ad esempio, si è staccata dal progetto padre per generare xorg a causa di un cambio di licenza. Oppure, si veda il caso di The Gimp: da una versione di Gimp si è evoluto Cinepaint per problemi relazionali tra mantainer. Ora The Gimp è il programma più utilizzato dai grafici sotto GNU/Linux, mentre Cinepaint è il software di effetti su animazioni utilizzato a Hollywood.

9. Brainfuck: http://en.wikipedia.org/wiki/Brainfuck; http://www.muppetlabs.com/~breadbox/bf/

APPENDICI

I – Da cattedrale a bazar: glibc

Per avere una panoramica delle tipologie di comunità, bisogna tornare al 1991, quando Linus Torvalds inizia il suo kernel e fonda la prima comunità basata sul concetto di «bazar». Abbiamo visto che il metodo precedente a «cattedrale», tipico della comunità GNU, muta rapidamente in questa nuova direzione.

Ancora oggi sussistono esempi sporadici di progetti strutturati a «cattedrale»; si tratta però di casi davvero rari e a volte folcloristici: citiamo il progetto Qmail, un server di posta distribuibile solo in versione sorgente e modificabile solo dall'autore stesso, poiché in caso contrario occorrerebbe scindere il progetto.

Il passaggio da «cattedrale» a «bazar» non è stato certo morbido e indolore, come testimonia ad esempio la crisi sulle glibc (librerie di base che permettono ai programmi di interagire al più basso livello con il sistema) del 1991.

La GNU stava combattendo una guerra soprattutto politica e ogni rilascio di codice di un nuovo programma rappresentava la vittoria di un'altra battaglia verso il Free Software; la nascente comunità Linux, invece, era completamente concentrata sulla tecnica: stava scrivendo un kernel e il suo obiettivo era che questo fosse il più possibile completo, performante e fruibile.

La comunità Linux era molto attiva e in espansione vertiginosa; tra l'altro, continuava ad aggiungere chiamate di sistema al nuovo kernel,

anche se lo sviluppo delle glibc non riusciva a stare dietro allo sviluppo complessivo.

Dopo un paio di anni, alcuni programmatori del kernel si spinsero dunque a ipotizzare un forking del progetto glibc, in modo da assumere questo progetto all'interno della nuova comunità.

Per comprendere l'estrema novità di una simile ipotesi, bisogna ricordare che in un sistema POSIX i software comunicano col kernel grazie alle librerie base del sistema. Sul sistema GNU/Linux questo rapporto di comunicazione viene espletato da Linux e dalle glibc. Le glibc sono state sviluppate dal gruppo della FSF con lo scopo tecnico e politico di creare l'humus necessario al rapporto di interdipendenza, e quindi di massima condivisione, tra gli sviluppatori del sofware applicativo e quelli del kernel.

Nel 1993 la comunità delle glibc, strutturata ovviamente a «cattedrale», non si aspettava che il progetto Linux sfornasse codice a tale velocità; poiché non riusciva a stare dietro agli aggiornamenti, i software esistenti si trovavano in difficoltà quando desideravano usare le nuove funzionalità via via proposte dal kernel.

Il forking poi non è avvenuto, e ora abbiamo un sistema operativo che funziona grazie a questi due elementi quasi in simbiosi con risultati più che discreti; però si può ben dire che c'è mancato poco e non era affatto scontato. È stata l'opera di diplomazia tecnico-politica delle due comunità e sicuramente dei due mantainer, Torvalds e Stallman, a evitare una spaccatura che non si sarebbe più potuta sanare.

II – Le librerie QT

Nel 1996 Matthias Ettrich avviò il progetto KDE basandolo sulle librerie grafiche QT. All'epoca, non esisteva nessun framework per sviluppare un desktop in grande stile, fatta eccezione per le appena nate librerie GTK, immature e per questa ragione ancora non completamente affidabili. KDE scelse di usare alcune librerie commerciali totalmente compatibili per X, sviluppate close dalla TrollTech, assumendosene l'onere tecnico e politico.

Nonostante la KDE si fosse prodigata nel fondare la KDE-QT Free Foundation – che impegnava la TrollTech a rilasciare le QT sotto licenza BSD nel caso in cui lo sviluppo si fosse fermato – una parte del mondo free, e in particolare i più intransigenti della GNU, non videro di buon occhio questo progetto; tuttavia, già dalla versione 1.0 le QT si guadagnarono meriti tecnici.

All'uscita della nuova versione, infatti, Stallman attaccò ferocemente il progetto KDE, accusando il gruppo di avere illegalmente legato codice libero a codice commerciale compilandoli insieme: questo di norma è permesso solo in relazione a librerie fondamentali di un sistema operativo.

L'obiezione fu subito: come è possibile decidere che cosa è fondamentale? Chi prende questa decisione? Il panorama dei framework grafici si stava evolvendo lentamente e la versione 1.0 delle GTK non era assolutamente paragonabile alle QT 2.0.

Inoltre, a partire da questa versione la licenza close utilizzata da Troll-Tech passò a QPL, che per quanto limitasse ancora molto i metodi di sviluppo era compatibile con lo standard Open Source (ovvero approvata dalla OSI).

Intanto era nato il progetto Harmony o Free-QT dagli stessi sviluppatori KDE, un tentativo di recuperare lo strappo nei confronti della GNU attraverso l'ulteriore scrittura di liberie compatibili con le QT e che svolgessero la stessa funzione.

Dopo pochi mesi la FSF si occupò in prima persona di portare avanti il progetto Harmony. A bloccare Harmony, facendo contemporaneamente decadere le nuove polemiche sorte intorno alla QPL, fu la stessa Troll-Tech, che dopo pochi mesi cambiò nuovamente licenza alle sue liberie portandole a GPL.

La risposta di Stallman fu chiedere al progetto KDE di fare pubblica ammenda nei confronti dell'intero mondo Open Source, cosa che peraltro non avvenne mai (si veda http://www.kde.org/announcements/announcement.php).

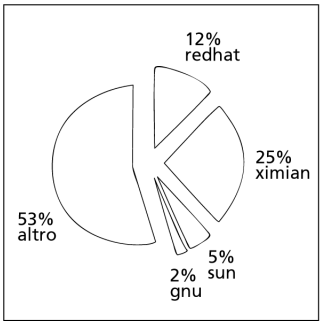
PRESENTAZIONE OGGETTI GRAFICI

Gli oggetti grafici qui riprodotti sono mappe delle relazioni che abbiamo cercato di mettere in luce nel testo: rappresentano alcuni dei nodi problematici più interessanti, alcuni degli ingranaggi più significativi delle gigantesche macchine che popolano i mondi digitali.

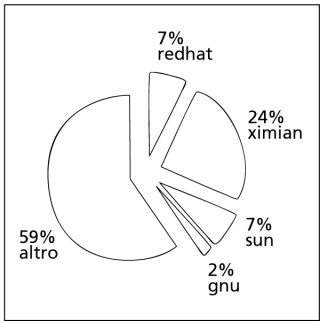
La forma degli oggetti non è casuale: sono reticolari se descrivono comunità aperte; più o meno orientate in maniera unidirezionale se si tratta di comunità maggiormente gerarchizzate.

La mappa sinottica vuole evidenziare le relazioni fra i molteplici attori in gioco. Una precisazione: la forma risultante ci sembra quella di un assedio. In effetti, la libertà del Free Software, come ogni libertà, è frutto di continui mutamenti, aggiustamenti, contaminazioni e rinegoziazioni di senso fra comunità e individui. Tuttavia, quando la contaminazione viene semplificata e i piani vengono schiacciati e ridotti all'unità, nello specifico l'unità dell'approccio mercantile, siamo di fronte a un tentativo di fusione, di normalizzazione (e infatti si parla di primato dell'economia, ovvero della «norma»), di riduzione: è il pensiero unico che chiama a sé.

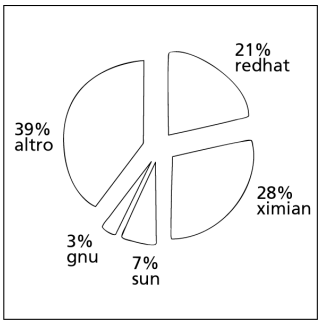
Per questa ragione, ricomponendo le varie mappe, crediamo che l'immagine complessiva che ne risulta sia quella di un assedio: l'assedio normalizzante, totalitario e banalizzante dell'economia sopra tutto e a ogni costo, dell'ideologia del mercato, che media ogni cosa e cerca di riassorbire in un'unica noiosa direzione i movimenti delle tribù nomadi, desideranti e libere, delle reti.



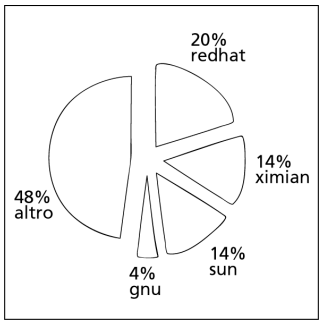
01: grafico percentuale dei programmatori delle librerie di Gnome



02: grafico percentuale dei programmatori dei software Gnome

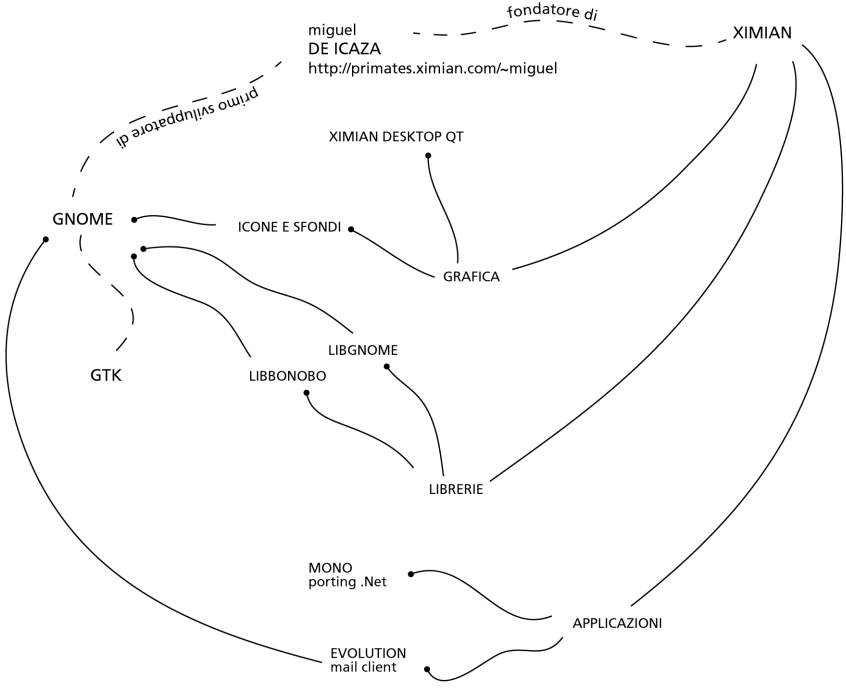


03: grafico percentuale dei mantainer delle librerie Gnome

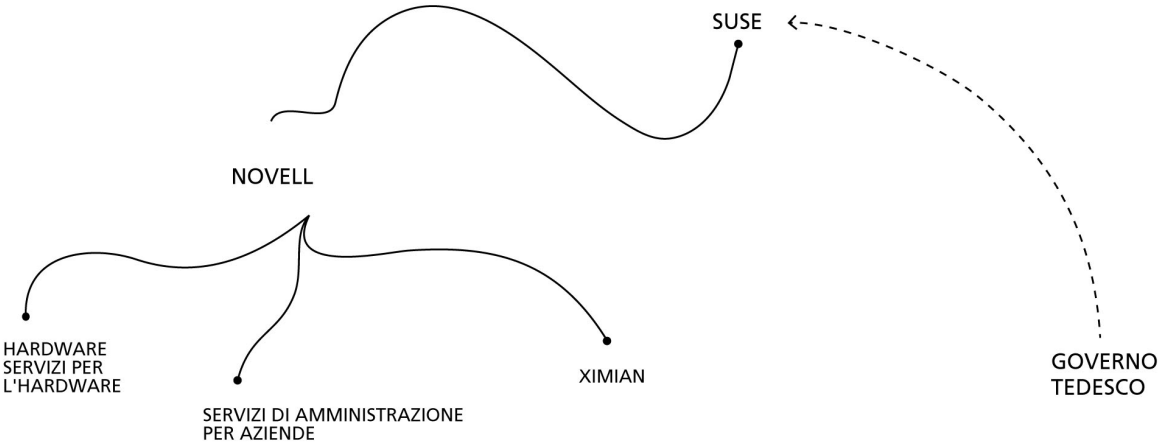


04: grafico percentuale dei mantainer dei programmi Gnome

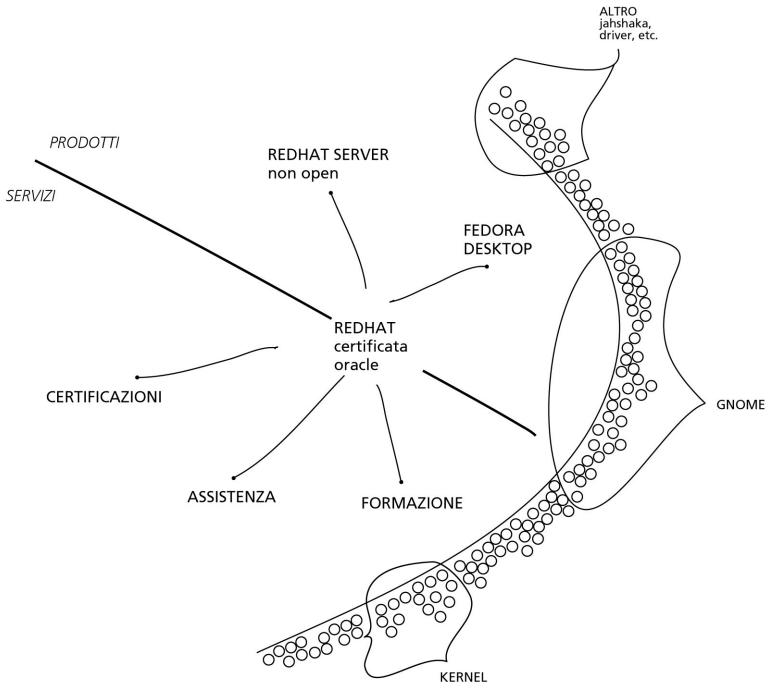
Schema delle relazioni di autonomia o appartenenza aziendale degli sviluppatori della comunità aperta GNOME.



Schema delle relazioni di Ximian.

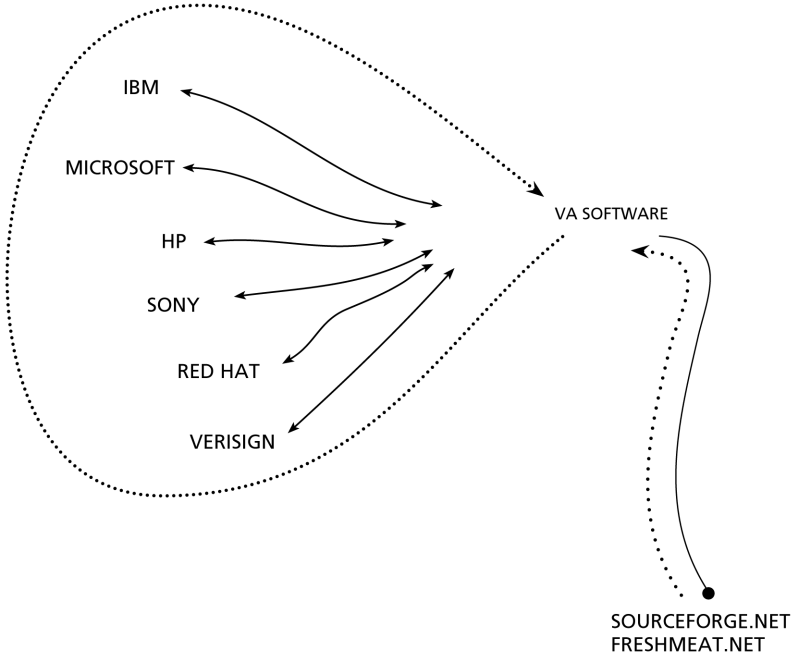


Novell è una società di servizi hardware e software per medie/grandi aziende. Ha acquistato la Ximian e ne gestisce tutti i progetti software (Gnumeric – Mono – Evolution e altri). Promuove e amministra la distribuzione Linux SuSE. Il governo tedesco finanzia lo sviluppo della distribuzione Linux SuSE e del desktop grafico KDE.



Schema delle relazioni di autonomia o appartenenza aziendale degli sviluppatori di Red Hat.

REDHAT
GNOME
GTK
NOVELL
XIMIAN
SUSE



Va Software è la società che gestisce i server di FreshMeat e SourceForge. Oltre a ospitare progetti Open Source, distribuisce i software e fornisce strumenti utili allo sviluppo e alla gestione della comunità. Tutti i codici e le schede tecniche sono in realtà beni commerciali per Va Software Corporation, che vende interrogazioni al suo sterminato database e informazioni sull'accesso a note aziende multinazionali.

creative commons
COMMONS DEED

Questo libro è distribuito sotto licenza Creative Commons 2.0 (Attribution, Share-Alike, Not Commercial) di cui riportiamo il testo in linguaggio accessibile.

Puoi trovare una copia del testo integrale della licenza all'indirizzo web http://creativecommons.it/Licenze/Deed/by-nc-sa, oppure richiederlo via posta a:

Creative Commons, 559 Nathan Abbott Way,
Stanford, California 94305, USA.
Attribuzione - Non Commerciale -
Condividi allo stesso modo 2.0 Italia

Tu sei libero di riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire o recitare l'opera, e di creare opere derivate alle seguenti condizioni:

BY
Devi riconoscere il contributo dell'autore originario.

NC
Non puoi usare quest'opera per scopi commerciali.

SA
Se alteri, trasformi o sviluppi quest'opera, puoi distribuire l'opera risultante solo per mezzo di una licenza identica a questa.

Potete scaricare la versione digitale del libro dalla sezione progetti del sito http://ippolita.net/

Ippolita a otto mani:
info@ippolita.net

Oggetti grafici:
urijoe.org

grx a:
wadada, ilvagabondo, garogno, alessia. giucas, tx0, blicero, tutta ReLOAd.
le comunità, pbm, dudley smith, gli hacker tutti.
manfre, la rossa, adisolata, tutta elèuthera. v., skrim, giaco, la cascinetta.
dino, françois, fra.
le ragazze e xxy.
tutti quelli che hanno pungolato la volontà e che ora, ovviamente, sono insoddisfatti.

Finito di stampare nel mese di giugno 2005
presso Grafiche Speed, Peschiera Borromeo, su carta Bollani,
per conto di Elèuthera, via Rovetta 27, Milano