Il sociologo, la sociologia e il software libero: open source tra società e comunità

Stefano De Boni

2010

Questo testo è completo.

Università degli Studi di Padova

Facoltà di Scienze Politiche
Corso di Laurea in Scienze Sociologiche

Il sociologo, la sociologia e il software libero: open s tra società e comunità

Autore: Stefano De Boni

Relatore: Valerio Belotti

Anno accademico 2009 – 2010

Indice

Nota a questa versione

In questa tesi si critica la descrizione data da Pekka Himmanen un decennio fenomeno dell'open source. In particolare si cerca di dimostrare come l'open nella sua accezione di software libero sia in realtà non un fenomeno romar postmoderno o postindustriale, quanto, all'opposto, un fenomeno razionа ipermoderno capace di riflessione ricorsiva sulla stessa idea di razionali

Con la ricerca e le riflessioni proposti in questa tesi si cercherà di indagare il fenomeno del software libero e dell'open source cercando di capirne innanzi tutto la rilevanza sociologica partendo, necessariamente, da una domanda cognitiva ampia: *come l'open source ed il software cosiddetto libero favoriscono lo sviluppo di un piano riflessivo collettivo, elaborando diverse modalità di accesso ai saperi esperti e di riappropriazione della tecnologia*. Il concetto di accesso e riappropriazione divengono centrali e particolarmente fecondi ai fini di questa indagine. Questi sono i termini della domanda cognitiva che permettono il collegamento ai diversi approcci tipici della sociologia.

L'accesso ai saperi esperti e la riappropriazione sono due concetti tipici della modernità radicale (Giddens, 1994). Oltre a ciò, l'accesso ha anche a che fare con i confini che delimitano il sacro, ma anche con i confini che delimitano ciò che è permesso, oltrepassare tali limiti significa porre in *pericolo* l'equilibrio ed esporre la società al rischio (Douglas, 2003). La riappropriazione, d'altro canto, ha a che fare anche con il capitale simbolico. Si vedrà come la riappropriazione del controllo tecnologico segua delle dinamiche del tutto simili a quelle che hanno portato all'appropriazione da parte del protestantesimo delle *sacre scritture*, almeno sotto due aspetti: Ascetico e razionale. Dal Punto di vista ascetico entrambi hanno a che fare con una rivelazione; dal punto di vista razionale entrambi comportano una trasformazione a cui segue l'acquisizione di conoscenza: saper leggere le Bibbie volgari da una parte, e saper controllare i sistemi informatici dall'altra; in entrambi i casi si tratta di mettere in discussione la mediazione tra l'uomo e il trascendente, in un caso, tra l'uomo e la tecnologia informatica dall'altro. Si vedrà più avanti come questo aspetto di riappropriazione della conoscenza assuma in entrambi i casi una dimensione testuale in contrapposizione ad una dimensione iconografica. Questo approccio richiede attenzione e precisione, si tratta di un campo di indagine molto articolato e le relative argomentazione rischiano di essere fraintese. È quindi necessario avvertire sin d'ora che queste regolarità tra etica hacker ed etica protestante riguardano il processo di riappropriazione e non i valori in sé.

D'altro canto la riflessione nasce dal bisogno di rielaborare la fiducia a seguito dall'esperienza concreta del rischio e dell'ansia legato in modo particolare all'informatica che spesso non mantiene ciò che promette in termini di velocità, di effetti attesi, di precisione, di usabilità e accessibilità, rilevante a tal punto che il tecno-stress diviene oggi uno dei campi di ricerca più indagati dalla psicologia del lavoro e delle organizzazioni. Oggi l'informatica riflette più che altri ambiti la

necessità di una riflessività ritenuta inderogabile dagli stessi produttori di software. Diversamente da un tempo, termini come bug e debug sono divenuti familiari anche a chi non sviluppa software. Il fatto che questi termini ricorrano spesso nelle discussioni, sia specialistiche che non, sta ad indicare una peculiarità informatica non in comune ad altri ambiti di consumo e cioè la sua particolare criticità. Per contro il bug[1] può essere eliminato ma questo richiede un continuo monitoraggio dei sistemi, una continua interazione tra programmatori e utilizzatori e questo è proprio il dispiegarsi di un atteggiamento riflessivo continuo. Si aggiungano a questo problemi come i malware[2], i virus[3], lo spamming[4], il phishing[5] e via dicendo. Quelle informatiche sono attività ansiogene ad alto rischio che necessitano di rielaborare in continuazione la fiducia attraverso la riflessività.

Tale domanda cognitiva impone un discorso altrettanto ampio sulla modernità, sul rapporto tra economia formale ed informale, su come cambia nella contemporaneità questo rapporto tenendo come riferimento il discorso sulla modernità su cui diversi autori (Giddens 1994, Bauman 2000 , Robertson 1991, Beck 1999) offrono un quadro teorico vasto, e più che adeguato per riferirvi il fenomeno open-source.

Aspetti come l'accesso ai saperi esperti, riappropriazione, fiducia, rischio, disgregazione e ri-aggregazione sembrano essere più che coerenti con l'emergere di una tecnologia aperta che incontra la sua operazionalità sociale sul piano del controllo prima ancora che su quello del suo utilizzo. Questo riguarda, o meglio, influenza altri campi del sapere e non solo la tecnologia informatica, come la farmaceutica, il sapere scientifico, umanistico e via dicendo. A differenza di altri campi del sapere, si cercherà di dimostrare come per l'informatica, e in particolare per lo sviluppo software, il codice, le istruzioni che in definitiva fanno funzionare i sistemi, assumano, oltre a ciò, significato di medium nelle relazioni e divengano oggetto di riflessione in un contesto di modernità radicale. Nonostante la comunicazione mediata attraverso il WEB, in determinate situazioni intensamente partecipate, ad esempio nello sviluppo di un progetto, diventa possibile un *luogo sacro virtuale*.

Si parla di, e ci si autodefinisce, community e si fa spesso appello alla solidarietà, mentre il focus attentivo è la tecnologia nella sua accezione dischiusa in qualche modo *rivelata*. La domanda cognitiva di partenza impone quindi di verificare se il fenomeno open source nella sua dimensione comunitaria rappresenta anche un fenomeno di ri-aggregazione. Questo aspetto ri-aggregativo

non può definirsi vasto tanto quanto la dimensione della diffusione dell'open-source, riguarda gruppi abbastanza limitati che lavorano attorno a progetti specifici e che attraverso il *codice* condividono un'esperienza di effervescenza e di gratificazione intellettuale propria di chi produce software senza riceverne alcun vantaggio formale o apparente. Si riporta a tale proposito quanto asserisce Richard Stallman (2000):

[...] La mia regola d'oro è: se un programma mi piace devo condividerlo con altre persone che lo apprezzano. Chi vende software adotta il metodo divide et impera con gli utenti, facendo in modo che ogni utente non voglia condividere nulla con gli altri. Io rifiuto una tale mancanza di solidarietà.

Diviene naturale aspettarsi processi di istituzionalizzazioni che offrono spunti coerenti con l'approccio fenomenologico e ideal-tipico dell'ascesi intra-mondana. Innanzi tutto il tema del carisma, e quindi il suo processo di istituzionalizzazione. Sarà quindi necessario individuare un ideal-tipo, spiegarlo, o meglio, comprenderlo, attraverso la caratteristiche che emergono da alcuni personaggi storici come Linus Torvalds o Richard Stallman, ma riscontrabile in maniera più o meno attenuata in altri persone che operano in questi ambiti.

A questo punto diventa necessario investigare anche le implicazioni organizzative del processo di istituzionalizzazione. È possibile, a questo scopo, cogliere un'analogia tra la strategia divulgativa di Lutero attraverso i *pamphlet sediziosi* e la pubblicazione nella *rete* delle prime implementazioni di Gnu/Linux da parte di Linus Torvalds. In realtà, come si è già storicamente dimostrato, i paesi protestanti sono in testa al processo di istruzione della popolazione, in quanto i fedeli dovevano *vedere con i loro occhi* la parola di Dio (Bagnasco, Barbagli, Cavalli, 2007). Anche questo in fondo è un processo di riappropriazione del capitale simbolico costituito dai testi sacri, svolto attraverso un appello alla responsabilità *individuale* che compensa la mancanza di struttura di potere e quindi di un'organizzazione gerarchica come poteva essere la Chiesa cattolica in contrapposizione alle sette protestanti, frammentate, diffuse, distribuite e organizzate in modo policentrico. Distribuire il capitale simbolico, darlo in custodia alla responsabilità dei singoli individui sembra quindi l'unica possibilità per compensare la carenza di organizzazione. Anche nella fattispecie dell'open source e del software libero ci troviamo di fronte ad una dualità di strutture: quella industriale delle grandi software house e quella policentrica e diffusa del software libero.

Se Weber aveva mostrato come un certo atteggiamento religioso ascetico portasse ad un agire razionale, qui sembrerebbe esattamente l'opposto, e quindi l'atteggiamento razionale che determina un agire ascetico non propriamente mondano ma globale. Ma ciò che ne consegue, almeno in una certa fase, è la compresenza di entrambi questi aspetti tra loro connessi. Lo scopo di questo lavoro è quello di individuare ambiti di indagine plausibili che, attualmente, si ritengono trascurati dalle scienze sociali benché sia l'economia, la psicologia e anche la sociologia vi abbiano ad oggi investigato in altre direzioni. A questo riguardo vale la pena anticipare come l'aspetto iconografico del software proprietario e l'aspetto invece testuale del software libero si ripropongano nella modernità radicale. Anche questa analogia si inserisce nel dualismo organizzativo che vede da una parte chiesa cattolica all'epoca della riforma e organizzazioni industriali moderne versus sette protestanti e software libero dall'altra. Quindi l'intensificazione del culto dei santi attraverso le immagini da una parte e la diffusione di bibbie volgari dall'altra. Tra l'altro questo dualismo, sia esso casuale o causale, viene ripreso proprio da un personaggio, Eric Steven Raymond, altrettanto ideal-tipico, quand'anche controverso all'interno della stessa community, con il suo saggio *The Cathedral and the Bazaar* (1997).

Altro aspetto che conferma questo dualismo è poi l'identificarsi come *smanettoni* da parte di chi si occupa di diffusione del software libero e quindi utilizza il computer con abilità installando e testando programmi liberi e funzioni open source. *Smanettone* riguarda non un'abilità generica, ma un'abilità specifica negli ambienti testuali a loro volta tipici dei sistemi GNU-linux. I server Gnu/Linux solitamente non vengono equipaggiati con interfaccia grafica a differenza di sistemi server proprietari. Benché le interfacce grafiche siano disponibili anche nei sistemi server open source il fatto che non vengano installate ha nel contempo un significato in parte razionale, volto ad aumentare l'efficienza e la robustezza, e in parte simbolico come segno di distinzione dai sistemi server proprietari che montano l'interfaccia grafica e quindi, di conseguenza, di distinzione per la community. Un sistemista Gnu/Linux si distingue perché usa comandi testuali da linea di comando. Nonostante si tratti di un atteggiamento razionale, volto ad aumentare le prestazioni del sistema, questo comporta nel contempo anche un'idea di purezza ed essenzialità coerente con la dimensione del sacro da delimitare e proteggere dalle *icone sediziose*. Un sistema server non deve essere amichevole ma ontologicamente puro, robusto e performante, nonché essenziale. Si cercherà quindi di verificare il grado di coerenza tra razionalità, etica hacker, dimensione emozionale, gratificazione intellettuale e tecnica. l'idea è che esista uno schema di senso del tutto coerente con quello indagato nell'etica

protestante pur partendo da presupposti non religiosi e non necessariamente di massa. Non religioso non significa che non sia plausibile l'atteggiamento religioso: dimensione comunitaria; *rivelazione delle scritture* del codice aperto; i simboli; i guru; i processi iniziatici; le prescrizioni formali, informali e morali; l'ascetismo e l'etica.

Si useranno diversi approcci di analisi strutturalisti, fenomenologici, finanche il tentativo di utilizzare i grafici dell'economia classica per verificare come il mercato delle licenze sia una costruzione volta a rendere, quasi funzionalmente, scarsa, e quindi compatibile con la legge della domanda e dell'offerta, una risorsa che per sua natura non lo è, cioè il software, mentre trascura ciò che realmente è scarso, cioè le competenze che vedono ridotto il loro spazio. Si cercherà di vedere alla luce della teoria dei sistemi di Luhmann come questi due sistemi apparentemente contraddittori tendano all'auto-referenzialità e all'auto-poiesi ma anche si compenetrino, come la tecnologia, che d'ora in avanti chiameremo proprietaria, tenda maggiormente dell'open source e del software libero all'entropia. Ma anche su questo piano la coerenza è forte, finanche nell'uso delle grammatiche che sono le stesse della tecnologia: sistema, ambiente, ricorsività (ricorsione in gergo informatico), ma più che altro informazione. Si aggiunga a questo la necessità della devianza e l'amplificazione della stessa al fine di permettere al sistema di elaborare soluzioni ed informazioni e che trova la sua espressione ideal-tipica nell'immagine dell'hacker. Solo da un punto di vista sociologico la fisiologia della devianza, e la sua funzione sistemica, può essere coerente con l'accezione positiva che il termine *hacking* assume tra i sostenitori del software libero e dell'open source.

Di tutto questo, non si daranno delle risposte esaustive, si individueranno dei temi di indagine tentando di misurarne euristicamente la portata sociologica. Si farà uso di bibliografie, di cui si avverte già da subito che alcune forniranno gli schemi interpretative tipici della sociologia, altre serviranno come riferimenti teorici specifici per la trattazione del fenomeno del software libero, altre ancora avranno lo scopo di fornire materiale documentale assieme a qualche osservazione etnografica e qualche intervista per analisi di secondo e terzo livello, in particolare si andrà ad analizzare come lo stesso movimento del software libero interpreta il fenomeno open source. Si sono già spesso usati i termini software libero e open source, ma non indistintamente. Infatti Richard Stallman (2000) precisa spesso la differenza semantica di questi due epiteti. Al fine di renderne comprensibile la differenza usiamo la stessa distinzione che ne fa Richard Stallman (2000):

[...] la retorica di "open source" si focalizza sulla possibilità di creare software di buona qualità e potente ma evita deliberatamente le idee di libertà, comunità, principio [...]

Al di là del fatto che l'omissione delle idee di libertà possa essere o meno intenzionale o moralmente rilevante, l'intenzione è quella di porre l'accento sulla rilevanza epistemologica che assume la diversa intensità etica tra software libero ed open source. Nella stessa edizione dello stesso libro, nel capitolo successivo Michael Tiemann non usa mai il termine *software libero*. Nella pratica è molto difficile separare questi due termini. Open source si riferisce al sorgente aperto, cioè al fatto che sia possibile modificare i programmi, si riferisce in pratica ad una caratteristica del software di cui si ha a disposizione anche il codice sorgente che può quindi essere modificato e quindi ricompilato. La compilazione del codice riveste un duplice significato: migliora le prestazioni del software ma impedisce che possa essere modificato.

Quindi open source non significa che il software venga interpretato dal computer senza essere compilato, anche se questo accade per molti linguaggi informatici, ma che oltre al programma compilato, quindi chiuso, viene reso disponibile anche il codice sorgente. La definizione *software libero* nasce quindi nel 1985 con la Fondazione per il software libero (Free Software Foundation) da parte di Richard Stallman e il suo gruppo. Il problema sta nella stessa parola *free* che in inglese significa sia libero che gratuito e questo sembra un problema insormontabile. Questa fondazione si mantiene sia con donazione che con la vendita di manuali, codice sorgente e programmi compilati, accompagnati al codice sorgente, di cui è permessa la rivendita quindi in contrapposizione al copyright compare il copyleft. Anche questa volta ci si trova di fronte ad un'ambiguità, left significa *permesso,* ma significa anche *sinistra* in contrapposizione a right che significa *diritto* ma anche *destra*, e attraverso questa ambiguità, come è ovvio, copyleft assume a volte anche un significato politico. In questo contesto viene elaborato un tipo di licenza d'uso, la *GNU General Public License* indicata come GNU-GPL o semplicemente GPL.

Questa licenza esprime, dal punto di vista contrattuale, le prerogative copyleft. Quindi un software rilasciato con licenza GPL può essere copiato e quindi distribuito dietro pagamento o meno. Esistono però restrizione volte a proteggere la persistenza della caratteristica del software libero, cioè deve essere distribuito con il codice sorgente e con adeguata documentazione. Altra caratteristica di questo contratto è il fatto che altri prodotti software distribuiti con

un software coperto da licenza GPL assorbono le stesse peculiarità del software libero. Quindi il software proprietario distribuito assieme a software libero GPL diviene software libero o in alternativa non può essere distribuito affatto. Questo tipo di licenza è quindi persistente e propagativa. Per ora possono bastare questi concetti, l'aspetto contrattuale legato all'open source è molto più articolato e rispecchia diversi paradigmi economici che si vedranno più avanti.

Se la definizione *software libero* fa spesso riferimento ad un movimento globale con determinati riferimenti etici[6]. La definizione *open source* ne rappresenta piuttosto il suo aspetto organizzativo e tecnico. Il software libero comporta che il software sia open source, ma la definizione open source non necessariamente implica la libertà del software. Entrambi questi aspetti sono la conseguenza dell'importante consapevolezza che esistono risorse umane intellettive di cui la stessa umanità si può avvantaggiare e che il mercato non è in grado di ottimizzare. Tale consapevolezza esce dai *conventi* coinvolge gli utenti, chiede la partecipazione attiva anche dei non specialisti che possono tradurre o comporre manuali d'uso, organizzare eventi ed arriva a coinvolgere anche altri aspetti del sapere umano fino a divenire una visione del mondo e una specifica etica.

Note

1. ↑ Difetto di programmazione
2. ↑ Tutte le attività e i prodotti relativi all'informatica deviante posti in essere allo scopo di danneggiare , truffare o spiare.
3. ↑ Programmi che agiscono nel sistema ospita in modo indesiderato e causando danneggiamenti software, il loro termine deriva anche dalla capacità di replicarsi.
4. ↑ Deriva dal termine inglese "spam" che significa carne in scatola, si riferisce al bombardamento indiscriminato di messaggi ottenendo gli indirizzi sulla rete o attraverso software che scansionano per tentativi i domini dei provider di posta.
5. ↑ Spillaggio di dati sensibili attraverso tecniche ingannevoli.
6. ↑ 0) La libertà di utilizzare qualunque software, a qualunque scopo. 1) La libertà di studiare il funzionamento del programma, e di cambiarlo a piacimento. Ovviamente il presupposto di questa libertà è avere accesso al codice sorgente. 2) La libertà di ridistribuire copie di qualunque software a

chi ti pare. 3) La libertà di distribuire copie del tuo software modificato a chi ti pare. Il tutto in una visione altruistica, cioè per aiutare chi ti sta intorno.

Non è possibile ai fini di questo studio descrivere nemmeno a grandi linee la vasta storia dell'informatica, sebbene recente e breve in termini temporali. Vale la pena accennare che è, innanzi tutto, uno strumento che si inserisce nel bisogno dell'uomo di classificare, calcolare e memorizzare lo scibile umano. Lo si può vedere da un punto vista mnemonico come l'estensione della stampa, oppure come evoluzione delle prime calcolatrici di Blaise Pascal e di Gottfried Leibniz. Già questa fase "preistorica" anticipa aspetti che diventeranno costanti con l'invenzione dei primi calcolatori elettronici. Mentre la macchina pascalina poteva essere riprodotta industrialmente, la macchina di Gottfried Leibniz eseguiva, oltre che somme e sottrazioni, anche moltiplicazioni e divisioni grazie al trasposítore, ma non poteva essere riprodotta in serie. Già qui si coglie un dilemma a pieno titolo filosofico, come si vedrà più avanti, tra potenza e riproducibilità, ai nostri giorni, tra software open source e software industriale. Pensare all'inizio della storia partendo da Blaise Pascal (non a caso anche il nome di un linguaggio di programmazione) e da Gottfried Leibniz ci permette di derivare, non troppo arbitrariamente, la nascita dell'informatica dalla filosofia per poi ricondurla ai giorni nostri, con la legittimazione di Richard Stallman, nuovamente alla filosofia.

Avvicinandoci ai nostri tempi, e facendo sicuramente torto, per necessità di brevità, ad una storia vasta e complessa, troviamo centri di ricerca, in particolare di università USA che necessitano di elaborare grosse quantità di dati, nonché enti governativi, in particolare il «ministry of war» USA, che hanno l'ambizione di integrare risultati di ricerche e sperimentazione dei centri di ricerca. Seguendo questo ramo della storia, più confacente ai nostri scopi, troviamo il Massachusets Institute of Tecnology (MIT) che acquista il suo primo computer PDP-1 attorno alla quale la cultura accademica crea i primi linguaggi - istruzioni o grammatiche che avvicinano la logica circuitale delle macchine a quelli che Luhman chiama sistemi cognitivi e sistemi di comunicazione, cioè gli uomini e la società - che hanno nomi come lisp, FORTRAN, Pascal, C e C++.

Chi al MIT si dedica a questa attività viene chiamato, o si definisce hacker, ma è importante sottolineare che, seconda l'etica hacker, tale è chi viene così definito da altri. Quindi hacker, anche per il prosieguo di questa ricerca, sarà in buona sostanza un programmatore, mosso nel suo fare principalmente da una gratificazione intellettuale e scientifica, oppure definiti «aficionados dedicated to the craft of computing » (Coleman, 2010), cioè appassionati dedicati all'artigianato informatico, anche se la traduzione italiana non rende la finezza del concetto, in quanto «to craft somenthing» significa creare qualcosa in modo parsimonioso lasciando trasparire anche un'idea di dedizione che a mio vedere si avvicina ad un'idea di vocazione e predisposizione intensa, come si cercherà di dimostrare, coinvolgente anche sul piano etico, quindi il «beruf».

Fondamentale è la nascita del progetto ARPAnet nel 1969, finanziato dal «ministry of war» USA, nell'ambito del quale le università USA, in particolare il MIT di Boston, Carnigie Mellon University, Stanford University, Barkeley University, iniziano a collaborare. I relativi team di sviluppo sono ovviamente composti da hacker che entrano in comunicazione stabile, continuativa ed operativa proprio attraverso ARPAnet che si evolverà poi in internet. Altro elemento importante che si inserisce in questa dinamica sono le imprese di elettronica che fabbricano i computer e mettono a disposizione i driver di base e i kernel su cui si costruiscono i sistemi operativi. Il concetto di driver e di kernel assumono per i nostri scopi un'importanza fondamentale. Sono le parti software più vicine alla logica circuitale, cioè quella parte minima di intelligenza artificiale

che permette agli sviluppatori, quindi agli hacker di implementare sui computer i sistemi operativi e quindi compilatori e linguaggi di programmazione che sono gli strumenti con i quali si costruiscono i veri programmi operativi per l'utilizzo finale.

Sempre ai nostri scopi è necessario capire come l'informatica si sviluppi a partire dai chip che contengono le funzionalità logiche di base, ai sistemi operativi ed infine agli applicativi (programmi per utenti finali non informatici) a strati. Negli strati più bassi si opera a livello molto astratto, i linguaggi rispecchiano la logica circuitale e non quella umana (linguaggio macchina). L'assembly è già un primo livello di avvicinamento al linguaggio umano. Più istruzioni macchina vengono «assemblate» in comandi singoli più immediati. Così si progredisce fino ad arrivare ai veri linguaggi di sviluppo come il FORTRAN, il Pascal, il C++, il lisp e via dicendo. Quindi a livello basso l'operatività è molto più complessa e specialistica, in gergo viene indicata con «scrivere sul ferro», mentre a livello alto le logiche operative rispecchiano i sistemi cognitivi umani, sono quindi più semplici e l'operatività tende ad essere meno specialistica ed inizia a riguardare di più le competenze estetiche, comunicative e organizzative. Controintuitivamente quando si parla di basso livello si intende pertanto un livello specialistico, astratto, lontano dai sistemi cognitivi e vicino alla logica circuitale. Ad alti livelli di operatività si può dire l'uomo conquista la semplicità operativa ma pagando un prezzo importante: la perdita di controllo sulla macchina.

È su questo punto, sulla base di questa convinzione, che a priori non diamo per scontato sia giusta, che il contesto hacker sviluppa la propria etica e si passa dall'entusiasmo, dall'effervescenza collettiva all'elaborazione di un'etica volta in primo luogo a preservare e difendere i confini di questa dimensione altra che gli hacker hanno sperimentato dal PDP-1 in avanti, fino al momento in cui questo spazio-tempo sembra finire, minacciato dalle logiche di mercato. Questo accade nel 1967 con la produzione del PDP-10 da parte di DEC (Digital Equipment Corporation) che nasce già equipaggiato di sistema operativo TOPS-10 e assemblatore MACRO-10 (macro indica proprio l'assemblaggio di istruzioni circuitali in comandi singoli). Il MIT decide di fare a meno del sistema operativo della DEC e per ragioni etiche, possiamo dire a questo punto, decide di costruire un proprio sistema operativo ITS (Incompatible Time sharing).

Il tema del controllo diviene centrale perché in funzione del livello in cui questo viene posto viene modificato il rapporto uomo/macchina. Un'immagine analoga ci è offerta da Pirsig (1992) che descrive il diverso approccio alla tecnologia tra motociclisti e automobilisti:

[...]*I proprietari di automobili solitamente non ci mettono mano, ma in ogni centro abitato c'è un garage dotato di ponti costosi, di attrezzi speciali ed apparecchiature diagnostiche che il proprietario medio di un automobile non può permettersi. E poi il motore di una macchina è più complesso e inaccessibile di quello di una moto, quindi la delega è più sensata. Ma per la moto di John, una BMW R 60, scommetterei che non c'è un meccanico da qui a Salt Lake City.* Pirsig (1992, p. 237)

Tanto quanto il controllo della logica circuitale viene mediato da strati software, tanto l'uomo diviene un esecutore, incanalato in percorsi decisi da altri. È pur vero che questi percorsi debbano essere decisi da qualche esperto se si vuole che anche i non esperti usino l'informatica, ma è diverso se questi esperti sono legittimati in un contesto comunitario che non dall'impersonalità del mercato. L'esperienza di Richard Stallman nel non poter più, da un certo momento, aggiungere funzionalità alla «sua» stampante è di fatto un'espropriazione di controllo. La risposta di un bisogno, come la funzionalità di avviso dell'inceppamento della carta, viene delegata al mercato anche se il mercato spesso non risponde adeguatamente, ed è su questo preciso aspetto in cui si trovano le argomentazioni più forti nella tensione tra software proprietari e software libero e le domande che emergono da questa dialettica sono: Il mercato delle licenze risponde a tutti i bisogni? Il mercato delle licenze sfrutta tutte le potenzialità della logica circuitale? È possibile che molti bisogni di automazione non emergano o vengano delegittimati o liquidati come impossibili solo perché non sono riproducibile industrialmente?

La community non è solo un contesto di legittimazione, è un qualcosa di concreto, in cui è possibile partecipare a qualsiasi livello oltre alla programmazione: organizzazione di iniziative, condivisione di visioni del mondo, traduzione di manuali, traduzioni di menu, finanche semplicemente usare software libero anziché proprietario e questa strategia è centrale perché risolve sul nascere il

problema dei free riders. Ed è attraverso i ruoli e l'impegno concreto che la comunità virtuale costruisce il «selves» pur con una limitata compresenza fisica, che comunque mantiene, come si vedrà più avanti, un'importanza non marginale.

Galbraith (1967) osserva che le grandi imprese riescono a indurre il comportamento dei consumatori. In sintesi la questione verte sul fatto che l'informatica «costruita» dal mercato è qualcosa di diverso da ciò che potrebbe essere se prodotta in una dimensione comunitaria, di conseguenza quello che «l'agente economico» chiede all'informatica non riguarda ciò che potrebbe essere ma ciò che corrisponde alle sue aspettative determinate dall'informatica delle licenze. Il tema di ciò che è oggi l'informatica contro ciò che potrebbe o dovrebbe essere non si limita solamente sulla questione «migliore» o «peggiore», che può essere riduttivo quanto relativo, ma si espande su questioni profonde e in particolare la libertà. La tesi di Galbraith (1967) sulla sequenza ritenuta e sulla sequenza aggiornata risulta maggiormente visibile nel settore informatico. Ciò che in ultima analisi viene messo in discussione è la plausibilità dell'industrializzazione dei processi di produzione di software su due piani: quello organizzativo della produzione e quindi dell'offerta, dove abbiamo visto è necessaria la limitazione delle risorse di conoscenza e la forzatura dell'informatica da «techinca» a «tecnica»; e quello della domanda dove la grossa componente del rischio già menzionata richiede il ripristino della fiducia attraverso la visibilità dei processi. A questo proposito, in termini più generali, Baudrillard (1976) sostiene che l'uomo moderno finisce per smarrirsi, perché non riesce a vedere le relazioni economiche che producono i beni di consumo. Allo stesso modo se il concetto di «folla solitaria» elaborato Riesman (1953) è necessario a sostenere il consumo di massa, al contempo non è sufficiente a sostenere il bisogno di fiducia che un bene complesso, mutevole e ad alto rischio come il software richiede.

Avendo spiegato cosa si intende per livello informatico ora possiamo asserire che la preoccupazione degli hacker è quella di preservare uno spazio sacro, - o controllare, in termini informatici ed ingegneristici, il livello basso - in cui si è sperimentata l'effervescenza collettiva. Se ciò è vero tale ipotesi deve essere confortata dal realizzarsi storicamente di una effervescenza collettiva autentica ed intensa. È possibile ricorrendo al concetto di stato di flusso di Csikszentmihaly, che spiega come l'energia psichica ed emozionale vengano integrate e mobilitate per risolvere le sfide che pone l'ambiente (o il sistema stesso, se visto da un punto

di vista costruttivistico) provocando un intenso grado di gratificazione amplificato dalla compresenza virtuale di altri che provano altrettanto attenuando la necessità di compresenza fisica.

Sicuramente non sbaglia Gabriella Coleman (2010), quando riprendendo Victor Turner, afferma che attraverso la compresenza in una conferenze di hacker e nelle hacker feasts, viene costruito il «selves» a partire dai «self», ma è però evidente che questa costruzione di «selves» inizia, anche cronologicamente, nella rete, nel luogo virtuale. Inoltre bisogna considerare che solo una piccola parte di coloro che partecipano ai progetti presenzia alle conferenze e anche coloro che non portano il loro corpo ma partecipano intensamente ai progetti non possono essere estranei alla costruzione del «selves»; a volte si partecipa una volta sola nella vita e tanto basta; il capitale simbolico rimane nella rete; la solidarietà si concretizza e si sperimenta nel network. Il linguaggio non verbale non è possibile in situazioni di non compresenza e non può essere surrogato, ma altrettanto la compresenza non può surrogare la rete e la comunicazione virtuale-non-verbale insita nella potenza comunicativa delle soluzioni e negli algoritmi.

Lo stesso acronimo GNU che indica il progetto fondato da Richard Stallman per costruire un sistema operativo diverso da UNIX, indica una ricorsione (GNU is Not Unix), che è una tecnica usata nei linguaggi strutturati per ordinare liste o risolvere problemi complessi in un modo non consueto ai sistemi cognitivi umani ma che fa parte della forma menti hacker, in più la ricorsione consente di scrivere codice in modo conciso risparmiando righe di comando e quindi tale acronimo contiene precise prescrizioni su quello che, nell'ambito del software libero, viene ritenuto un modo di programmare corretto e pulito, cioè puro (Douglas 2003) perché deve essere esposto al mondo intero.

Sempre Richard Stallman quando enumera dei concetti anziché partire da «1», parte da «0», pur comunicando con altri uomini e non con dei microprocessori. Chiaramente questo modo di comunicare è cercato e voluto, indica chiaramente un'appartenenza, un'identità, una prescrizione e un'ammonizione di corretta programmazione che potrebbe significare: «chi partecipa ad un progetto di sviluppo di software libero non si azzardi a costruire liste partendo da 1».

Se per il primo assioma della scuola di Palo Alto è impossibile non comunicare si può asserire che nella comunicazione di una soluzione (algoritmo) nell'ambito di un progetto open source è impossibile non comunicare anche qualcos'altro di intimo, profondo e personale. A questo proposito diviene illuminante la stessa immagine che Gabriela Coleman usa del mago di Oz come costruzione di un impostore che manovra da dietro ad una tenda, effettivamente nel modo di produrre dell'open source non ci sono tende.

Questo spiega come nella rete si possa sperimentare, in una certa misura, virtualmente, cioè senza compresenza fisica, il sacro. Solitamente le distinzioni riguardano il tipo di mezzo utilizzato, e quindi la contemporaneità tra emittente e ricevente, diversa ad esempio tra telefono o lettera, o la direzione, a senso unico ad esempio nella comunicazione di massa come televisione e giornali. Spesso le comunicazione mediate sono complementari a relazioni reali. Va precisato che si è spesso usato, e si userà nel prosieguo il termine virtuale, per indicare qualcosa a cui riferire le rispettive categorie sociologiche del mondo reale. Questo non è arbitrario, anche se spesso questo termine viene abusato, in quanto, ad esempio un luogo virtuale, può essere tale nel momento in cui, pur non essendo delimitato da confini fisici, è suscettibile di possedere caratteristiche «reali» che lo rendono idoneo ad essere trattato da un punto di vista sociologico come un luogo fisico, ad esempio, perché idonei a contenere i manufatti della cultura hacker, cioè il codice aperto del software.

In questo modo possiamo spiegare la dimensione sacra secondo Durkheim (1962). L'esperienza del sacro inizia con un raduno, nel luogo virtuale, che è la rete. Queste persone sono mosse non semplicemente da interessi comuni ma da un comune sentire. Due elementi importanti per l'innesco dell'effervescenza collettiva sono la contemporaneità, cioè la compresenza spazio-tempo, e il coordinamento.

Spesso si pensa alla comunicazione mediata, in particolare nella rete, come autonomizzazione dal tempo ed in questo senso la contemporaneità sembra non sussistere. Possono però essere sottovalutate due condizioni: l'autonomia dal tempo come semplice possibilità e non necessità; la sfumatura della compresenza. La seconda questione richiede una breve spiegazione. In qualsiasi comunicazione

vi è un certo coordinamento, una certa gestione dei tempi in cui, ad esempio tra due persone educate, mentre l'uno parla l'altro ascolta e viceversa. Non è possibile di fatto stabilire un tempo prossemico limite al di fuori del quale la reazione ad uno stimolo non rientra più in una dinamica di compresenza spazio-tempo. È chiaro che a buon senso oltre un certo limite viene meno la compresenza ed è tanto minore quanto maggiore è il tempo di attesa di una risposta (reazione). Quindi si potrebbe anche ipotizzare che la compresenza sfuma con il passare del tempo. Questo processo di sfumatura potrebbe anche non avere la stessa intensità, infatti il tempo è un concetto relativo. È plausibile inoltre supporre che lo scemare della compresenza sia tanto immediato quanto minore è il coinvolgimento e viceversa. In termine pratici la situazione potrebbe essere la seguente come si può riscontrare abbondantemente nella rete:

Graham Collins - 12/05/2009 Sono un nuovo arrivato e ho provato ad installare Twiki 4.3.1 (seguendo le sistruzioni ...) e ho gli stessi errori che John De Stefano descrive ...

Peter Thoeny - 13/05/2009 anziché lottare con CPAN install puoi manualmente sistemare il file twiki/lib/TWiki/Render.pm ...

Graham Collins - 13/05/2009: Grazie questo funziona ....

Paul Reiber - 14/05/2009: ho trovato lo stesso errore ...

Tomasz Grzegurzko - 14/05/2009: su Debian o Ubuntu apt-get libunicode-string-perl aggiusta questo problema

Tomasz Grzegurzko - 22/05/2009: Scusate, il comando precedente dovrebbe essere apt-get install libunicode-string-perl

(tratto da http://twiki.org/cgi-bin/view/Support/SID-00291 il 10/09/2010)

Le discussioni come queste sono tantissime nei progetti open source, e ciò che è importante notare, e relativizzare, è il tempo particolarmente breve che intercorre tra una prima segnalazione di bug (difetto) e la sua soluzione: 2 giorni; sono dinamiche impensabili nel software chiuso. Innanzi tutto un primo aiuto viene dato il giorno dopo da Peter Thoeny che spiega dove modificare il codice allo stesso utente. Due giorni dopo Tomasz Grzegurzko, probabilmente lo stesso sviluppatore, annuncia che la nuova installazione risolve tutto (apt-get install). Il 22 maggio Tomasz Grzegurzko si accorge che aveva scritto solo «apt-get» anziché «apt-get install» e quindi precisa meglio tutta l'istruzione. Durante questo arco di tempo non è venuto meno il focus attentivo sul progetto, la tecnologia non solo può contrarre il tempo, ma lo può anche dilatare (rallentare o sospendere).

Si tenga poi presente che è il progetto che costituisce l'evidente focus attentivo di questa dinamica, il quale viene modificato, mutato in qualcosa di migliore e questo processo entra nella storia della community. Costituisce un vero e proprio oggetto sacro specialmente se usato da un gran numero di utenti in tutto il mondo. Con questo non si vuole asserire che lo spazio-tempo virtuale non ponga dei limiti alla comunicazione ma semplicemente che in, qualche misura, è plausibile recuperare un certo grado di compresenza in situazioni di forte coinvolgimento. A questo proposito basta qualche semplice esempio:

un programmatore sperimenta nella rete ogni giorno la solidarietà. Problemi complessi che non si riuscirebbe a risolvere in tempi ragionevoli vengono risolti dalla comunità di colleghi hacker. Ma in queste situazioni l'esperienza catartica della solidarietà è intensificata da una condizione emotiva negativa che la precede, cioè l'ansia da consegna, la contrazione del tempo, e quindi la solidarietà è percepita intensamente. Questo spiega, come a livello micro, in condizioni di iper-modernità, sia possibile, anzi indispensabile, un rapporto tra economia formale ed informale e come queste si sostengano e nel caso specifico della tecnologia informatica non possano sussistere l'una senza l'altra. Questo rapporto, tra economia formale e non, è evidente oltre che nei progetti, dove operano volontari che costruiscono strumenti che vengono usati nel mercato delle consulenze e nello sviluppo di sistemi, anche nella situazione in cui viene data assistenza volontaria ad un problema che riguarda un'attività regolarmente remunerata. Spesso l'economia ha risolto questo paradosso imputando al costo «zero» l'effettivo punto di incontro tra domanda ed offerta, ma questo verrà

argomentato più avanti. Inoltre il software libero entra in diretta concorrenza con il software proprietario e certamente contribuisce ad abbatterne i costi.

Si pensi a quanto possa essere gratificante vedere una propria soluzione tecnologica, approvata da un intera comunità, vedere il proprio codice ripreso e perfezionato o implementato da altri, o mescolato ad altre soluzioni e via dicendo, e quanto questo soddisfi la desiderabilità sociale.

A questo livello le discussioni assumono un carattere esoterico e ciò se da un lato conferisce una tonalità elitaria dall'altro aumenta il suo tenore religioso. Queste discussioni hanno un pubblico più vasto, possono essere seguite anche da chi ha minori competenze che comunque percepisce il senso di quanto accade. Le competenze sono sfumate, possono essere più o meno profonde così come la partecipazione attiva che si può svolgere a più livelli e può riguardare lo sviluppo software vero e proprio, la scrittura di manuali d'uso, la traduzione, la promozione dei prodotti attraverso un proprio sito, la partecipazione a discussioni e forum, finanche il semplice utilizzo al posto di un corrispettivo prodotto software proprietario. I sacerdoti del nuke1 di sviluppo, d'altro canto, percepiscono la presenza degli adepti attraverso i log (tracciamento del numero di accessi ai siti), il numero dei download (numero di installazioni), segnalazioni di bug e via dicendo.

A questo punto possiamo riprendere l'idea di sacro di Durkheim (1962) aggiungendo un altro elemento che emerge da quanto sopra e cioè il coordinamento spontaneo dei comportamenti e la sincronizzazione che trasforma i sentimenti individuali in collettivi. Il focus attentivo è il prodotto software prodotto dalla community e ciò che viene celebrato alla fine è la community. Emergono prescrizioni implicite ed esplicite che si rifanno sullo stile della netiquette2 oppure riguardano obblighi di solidarietà, divieto di introdurre nella discussione temi non attinenti e quindi contaminanti e devianti dal focus attentivo, fino ad arrivare, come per la comunità Debian, ad un vero contratto sociale, con tanto di versione come per pacchetti software, attualmente La versione 1.1 approvata il 26 aprile 2004. Possiamo quindi riconoscere seguendo Collins (1988) gli elementi che caratterizzano il rito religioso ad eccezione della riunione fisica di un gruppo di persone che abbiamo argomentato:

1) la condivisione di un medesimo focus attentivo;

2) una tonalità emozionale comune;

3) oggetti sacri che rappresentano l'appartenenza al gruppo (il software);

4) aumento della fiducia e dell'energia emozionale degli individui che vi partecipano;

5) giusta rabbia e punizione per chi non rispetta gli oggetti sacri.

Ad eccezione delle riunione fisica attenuata, ma non del coordinamento. Di fatto la dinamica del tutto mediata riportata sopra mostra un buon grado di coordinamento tra persone che non si conoscono. Intervengono utenti «sprovveduti» che vengono aiutati da utenti più esperti che aiutano a risolvere temporaneamente il problema e poi un programmatore accreditato che annuncia la soluzione del problema omettendo per distrazione un suffisso che viene aggiunto poi. C'è inevitabilmente già qualcosa di più che non una semplice interazione mediata tra sconosciuti ma allo stesso tempo sembra mancare qualcosa. Da testimonianze dirette che vengono documentate e osservazioni etnografiche, in particolare di Gabriella Coleman che ha osservato la comunità Debian, emerge comunque l'importanza della compresenza fisica, tanto più quanto il progetto assume le caratteristiche del software libero, con le sue implicazioni etiche maggiori e codificate, come è appunto nel caso della comunità Debian.

Stefano Zacchiroli Project Leader di Debian e ricercatore alla Université Paris Diderot esprime bene questa esigenza della compresenza fisica, per cui è importante incontrarsi almeno una volta per potersi meglio coordinare. Ma è proprio il coordinamento che denota la dimensione del sacro e che permette l'effervescenza collettiva; in più lo stesso Zacchiroli sottolinea un processo di cambiamento attraverso la «prima» compartecipazione fisica:

«trovo che andarci per la prima volta senza esserci mai andato serva tantissimo a migliorare la situazione. Tornarci se ci sei già andato contribuisce ma non così tanto come la prima volta»

Rappresenta un rito di passaggio, ma rappresenta anche un evento annuale a cui è importante/necessario partecipare almeno una volta - nella vita – così come recarsi alla Mecca almeno una volta nella vita è uno dei cinque pilastri dell'Islam.

La differenza consiste nel fatto che nel caso della comunità Debian non esiste un luogo sacro, un Harim fisico verso cui dirigersi, questo è virtuale, in quanto il focus attentivo è virtuale, è il software che si trova nella rete. Pertanto non c'è bisogno di un posto fisico determinato e localizzato, ma allo stesso tempo la compresenza è necessaria. Dice Zacchiroli nell'intervista del 17 agosto:

«è risaputo che lavorare di persona in un lasso di tempo limitato con delle persone che collaborano su uno stesso progetto ha un'efficienza ed un'efficacia molto superiore che il lavoro remoto, perché si è concentrati, perché la cosiddetta banda passante della comunicazione è molto più alta»

C'è un elemento importante che emerge da queste affermazioni. Il termine di paragone con cui si misura la qualità delle relazioni è un elemento virtuale, cioè la banda, il canale comunicativo. La compresenza fisica è migliore, ma la si misura in termini virtuali. Non è un'ambiguità ma un'ambivalenza tipica della modernità radicale (Giddens, 1995), e nello specifico spiega esattamente come compresenza fisica e virtuale siano tra loro coordinate. In qualche modo pone l'accento sulla «intensificazione» degli oggetti come la velocità banda e il progetto inteso come software; della comunicazione; dei sensi attraverso la concentrazione; quindi dell'efficienza e dell'efficacia, che non sono solo la misura «economica» dell'evento ma anche e soprattutto una misura valoriale, in quanto fanno la differenza, permettono - unitamente a molto altro ovviamente - la distinzione dal software proprietario e quindi l'identificazione.

In questo contesto, seguendo l'analisi analogica di Durkheim sulla relazione tra comunità e sacro, viene celebrato il software, il sistema operativo Debian con i suoi quarantamila pacchetti, come prodotto della comunità e come tale emanazione del sacro. Questa analogia nell'analisi sociologica nulla toglie all'identità, alle peculiarità o all'esperienza dell'uno o dell'altro contesto, semplicemente evidenzia una strategia storica dell'agire umano che ha il

significato, se non il «senso», di essere funzionale alla coesione e, in ultima analisi, di risolvere il problema di «come tenere unita la società» sia essa umana o community.

È veramente difficile arrivare ad una definizione univoca del termine hacker in quanto polisemico e mutevole in funzione del contesto. Per affrontare questo compito analizziamo quattro definizione:

1. Dizionario Zingarelli della lingua italiana (2007, 2133 pagine) : Chi attraverso la rete internet è in grado di superare i sistemi di protezione per accedere ai dati contenuti nella memoria di un altro computer.

2. Wikipedia italiano (http://it.wikipedia.org/wiki/Hacker del 31.07.2010): Un hacker (termine coniato negli Stati Uniti che si può rendere in italiano con maneggino o smanettone) è una persona che si impegna nell'affrontare sfide intellettuali per aggirare o superare creativamente le limitazioni che gli vengono imposte, non limitatamente ai suoi ambiti d'interesse (che di solito comprendono l'informatica o l'ingegneria elettronica), ma in tutti gli aspetti della sua vita. Esiste un luogo comune, usato soprattutto dai mass media (a partire dagli anni ottanta), per cui il termine hacker viene associato ai criminali informatici, la cui definizione corretta è, invece, "cracker".

3. Wikipedia inglese (http://en.wikipedia.org/wiki/Hacker del 31.07.2010): Hacker (informatica), un termine controverso per indicare diversi di persone:

a. Hacker (sicurezza informatica) o cracker, chi accede ad un sistema informatico aggirando il suo sistema di sicurezza;

b. Hacker (subcultara di sviluppatori), chi condivide un approccio anti-autoritario allo sviluppo software ora associato con il movimento “software libero”;

c. Hacker (hobbista), chi fa personalizzazione innovative o combinazioni di parti elettroniche o componenti di computer;

4. Manifesto hacker (The Mentor: anonimo, 8 gennaio 1986):

Ne è stato arrestato un altro oggi, è su tutti i giornali. "Ragazzo arrestato per crimine informatico", "Hacker arrestato dopo essersi infiltrato in una banca"...

Dannati ragazzini. Sono tutti uguali. Ma avete mai, con la vostra psicologia da due soldi e il vostro tecno-cervello da anni 50, guardato dietro agli occhi del

Hacker?

Non vi siete mai chiesti cosa abbia fatto nascere la sua passione? Quale forza lo abbia creato, cosa può averlo forgiato? Io sono un hacker, entrate nel mio mondo...

Il mio è un mondo che inizia con la scuola... Sono più sveglio di molti altri ragazzi, quello che ci insegnano mi annoia... Dannato sottosviluppato. Sono tutti uguali. Io sono alle Junior High, o alla High School. Ho ascoltato gli insegnanti spiegare per quindici volte come ridurre una frazione. L'ho capito. "No, Ms. Smith, io non mostro il mio lavoro. è tutto nella mia testa..."

Dannato bambino. Probabilmente lo ha copiato. Sono tutti uguali. Ho fatto una scoperta oggi. Ho trovato un computer. Aspetta un momento, questo è incredibile! Fa esattamente quello che voglio.

Se commetto un errore, è perché io ho sbagliato, non perché io non gli piaccio... O perché si senta minacciato da me... O perché pensi che io sia un coglione... O perché non gli piace insegnare e vorrebbe essere da un'altra parte... Dannato bambino. Tutto quello che fa è giocare. Sono tutti uguali.

Poi è successa una cosa...una porta si è aperta su un mondo...correndo attraverso le linee telefoniche come l'eroina nelle vene del tossicomane, un impulso elettronico è stato spedito, un rifugio dagli incompetenti di ogni giorno è stato trovato, una tastiera è stata scoperta. "Questo è...questo è il luogo a cui appartengo..." Io conosco tutti qui...non ci siamo mai incontrati, non abbiamo mai parlato faccia a faccia, non ho mai ascoltato le loro voci...però conosco tutti.

Dannato bambino. Si è allacciato nuovamente alla linea telefonica. Sono tutti uguali. Ci potete scommettere il culo che siamo tutti uguali...noi siamo stati nutriti con cibo da bambini alla scuola mentre bramavamo una bistecca... i pezzi di cibo che ci avete dato erano già stati masticati e senza sapore. Noi siamo stati dominati da sadici o ignorati dagli indifferenti. I pochi che avevano qualcosa da insegnarci trovavano in noi volenterosi allievi, ma queste persone sono come gocce d'acqua nel deserto.

Ora è questo il nostro mondo...il mondo dell'elettrone e dello switch, la bellezza del baud. Noi facciamo uso di un servizio già esistente che non costerebbe nulla se non fosse controllato da approfittatori ingordi, e voi ci chiamate criminali. Noi esploriamo...e ci chiamate criminali. Noi cerchiamo

conoscenza...e ci chiamate criminali. Noi esistiamo senza colore di pelle, nazionalità, credi religiosi e ci chiamate criminali. Voi costruite bombe atomiche, finanziate guerre, uccidete, ingannate e mentite e cercate di farci credere che lo fate per il nostro bene, e poi siamo noi i criminali.

Si, io sono un criminale. Il mio crimine è la mia curiosità. Il mio crimine è quello che i giurati pensano e sanno non quello che guardano. Il mio crimine è quello di scovare qualche vostro segreto, qualcosa che non vi farà mai dimenticare il mio nome.

Io sono un hacker e questo è il mio manifesto. Potete anche fermare me, ma non potete fermarci tutti...dopo tutto, siamo tutti uguali.

È possibile ricavare un filo conduttore comune in tutte queste definizioni. I termini usati evocano problematicità, rivoluzione, crimine, trasgressione, anticonvenzionalità, devianza. In particolare la definizione del dizionario Zingarelli evoca un atteggiamento malevolo ma si può facilmente dimostrare che non necessariamente superare i sistemi di sicurezza è un atteggiamento criminoso. È possibile anche professionalmente, quindi legittimamente immedesimarsi in un "criminale" per penetrare un sistema protetto per poi relazionare al committente, proprietario del sistema, sui possibili punti deboli in cambio di un compenso onesto. Allo stesso modo è possibile penetrare un sistema protetto per riappropriarsene a seguito di una condizione di lock-in dovuta all'acquisizione di un software chiuso di cui nemmeno il possessore della licenza d'uso è in possesso delle chiavi. A questo punto sono necessarie delle spiegazioni. Il lock-in è una situazione in cui ci si trova nell'impossibilità di utilizzare o riparare un sistema di cui si ha solo la licenza d'uso. La licenza d'uso è quindi una sorta di concessione ad usare un software che di fatto appartiene alla software house che ne concede l'uso.

Questo accade comunemente quando si acquista un personal computer. Il sistema operativo contrattualmente non appartiene al possessore del computer ma alla software house che la ha prodotto. In termini contrattuali ciò significa che in caso di lock-in non si è in grado di rivendicare alcuna possibilità di penetrare il sistema. La maggior parte dei personal computer sono dotati di un sistema operativo che non è "personal" mentre i dati contenuti negli archivi certamente lo sono. In questo caso è perlomeno controverso il fatto che "hackerare" il proprio computer, dotato di sistema operativo "proprietario" ma non "proprio", possa costituire un crimine.

È una condizione ricorrente per motivi di lavoro dover trasferire dei dati da un sistema proprietario ad un altro sistema proprietario, di cui non si hanno gli accessi formali, attraverso tecniche di hacking, l'alternativa sarebbe quella di chiedere una consulenza a chi detiene formalmente le chiavi del sistema montato in un server del proprio datore di lavoro. Questa situazione è quantomeno controversa perché l'aspetto legale e contrattuale non risolvono in alcun modo la contraddizione di avere dei dati propri gestiti da un sistema non proprio. Riportando la stessa situazione, ad esempio nel mercato dell'auto, sarebbe come acquistare un'auto di cui non si possiedono le chiavi del cofano del vano motore. Questi punti di vista possono essere più o meno confutati, ma lo scopo è far vedere quali sono i meccanismi dialettici attraverso i quali gli hackers si percepiscono positivamente e come portatori di valori etici.

In questa situazione si può ricondurre la definizione del dizionario Zingarelli ad atteggiamento anticonvenzionale, in quanto la contraddizione dei contratti delle licenze d'uso costringe professionalmente a tanto. Allo stesso modo negli anni '80 Richard Stallman, che poi fonderà il movimento del software libero, contestava il fatto di non poter accedere ai driver compilati (chiusi) di una stampante Xerox per aggiungere la funzionalità di avviso dell'inceppamento della carta.

Questo rivela quindi un atteggiamento non necessariamente antigiuridico, se non in casi estremi, ma piuttosto non convenzionale, dal quale emerge una dinamica di tensione tra software libero e software proprietario, tra movimento del software e case produttrici di software proprietario. “Non antigiuridico” non significa “non deviante”, in particolare se prendiamo in considerazione la sua peculiarità trasgressiva rivolta essenzialmente verso le regole informali o implicite e non quelle codificate. Di fatto l'etica hacker è molto rispettosa della legge, ed anche se è difficile trovare una codifica comunemente accettata di etica hacker, lo si può evincere dalla rete. Ad esempio una discussione su www.vicenza.linux.it, come molte che si possono trovare nella rete nei forum dedicati al software libero, assumono questi toni:

Il punto nevralgico è stato samba. Il problema di samba è che se lo vuoi mostrare e dirci che fa da file-server a windows, devi avere un windows per forza di cose. Perché abbiamo provato con due macchine Gnu/Linux a fare da server e client, ma la gente giustamente chiedeva di vederlo da windows. In lugvi-cd ci sono probabilmente le discussioni in materia. Alla fine il senso del discorso è questo: il lug promuove tutto e solo il software libero per la promozione di alcuni

software (in verità solo samba, ma si è voluta fare una cosa generica) è necessario un client proprietario. Si autorizza ad usare tale client per il solo scopo di pubblicizzare il software libero. chi fa le dimostrazioni deve avere tutte le licenze necessarie (e nello specifico non venire con una copia pirata di win)

Quindi per hacking intendiamo specificatamente un atteggiamento di riappropriazione della tecnologia attraverso tecniche non previste dai produttori e distributori di tecnologia che riflette in generale una ben precisa visione del mondo volta al controllo dei processi informatici per giungere nella sua maturità a sostituire la stessa tecnologia proprietaria con tecnologia non proprietaria, aperta e intelligibile. È necessario rilevare come Stefano Zacchiroli, nell'intervista del 17 agosto 2010 prediliga il termina "geek"[1] che si contrappone al termine "nerd"[2] così come il termine hacker si contrappone al termine cracker:

[…] *ho l'impressioni che ci siano visioni di come funzionano nel mondo dei geek diverse. Diciamo una visione preliminare. Molto hacker, molto "one man show", "che belle queste comunità che non si incontrano mai di persona". Mentre adesso stiamo realizzando che molte cose non erano come le immaginavamo all'inizio* (Stefano Zacchiroli, intervista del 17 agosto 2010 a Bologna).

Geek si riferisce ad un affezionato della tecnologia ed assume una dimensione più positiva, disinteressata al conflitto, all'autoritarismo, ponendo invece l'accento su un kantiano "pubblico ragionare" scientifico che ha incidentalmente conseguenze politiche per la diversa concezione economica che sottende:

[…]*un modo fantastico di vedere i propri cambiamenti accettati da altri è quello che si chiama "show me the code", cioè si fa vedere che un cambiamento di cui si sta discutendo da secoli è possibile […] per me semplicemente è un'idea sbagliata di come si fa economia col software e che è arrivata prima di tutti noi purtroppo, è riuscita ad imporsi sul mercato come cosa giusta, cosa che io ritengo non sia giusta per niente e per questo l'obiettivo principale è liberarsi di questi blocchi.* (Stefano Zacchiroli, intervista del 17 agosto 2010 a Bologna).

Nonostante la chiara e corretta precisazione di Stefano Zacchiroli si ritiene di dover continuare ad utilizzare il termina hacker nel prosieguo in quanto questo termine evoca una precisa esperienza storica che ha inizio al MIT negli anni '50. Diversamente dal concetto di geek, il termine hacker consente una maggiore

generalizzazione e ricomprende pertanto più variabili utili alla definizione di un ideal-tipo attorno al quale articolare una proposta di ricerca.

È necessario a questo punto fare delle precisazioni importanti. Il termine hacker non ha nulla a che fare, alle sue origini, con il concetto di pirata informatico attribuitogli dalla stampa e spesso anche dalla psicologia criminale. Io ritengo ancor oggi questa "devianza" del significato del termine hacker scomoda ancorché ingiusta. Ci obbliga a distinguere attraverso epiteti come "hacker innocuo" o "hacker malevolo" con tutta una serie di complicazioni non banali in quanto la stessa definizione di "hacker malevolo" verrebbe riferita, da parte di un "aficionados" del software libero, a un programmatore che sviluppa programmi proprietari, cioè rilasciati sotto licenza d'uso, nel mercato tradizionale dei brevetti, esattamente l'opposto del significato che gli attribuisce la psicologia, la quale, a sua volta, sottintende per hacker, senza aggettivo, quello malevolo tanto da farne dei profili criminologici.

Non volendo "deviare" dal termine storico originale, nel proseguo di questa tesi, si continuerà a sottintendere per hacker senza aggettivo un "ideal-tipo sociologico" e non un "profilo patologico" eventualmente indicato con il termine "cracker". È molto complesso in ogni caso riuscire a definire con esattezza il campo di indagine di un fenomeno così vasto e articolato, e non ci riescono, a mio avviso, Raoul Chiesa e Silvio Ciappi nella pubblicazione della loro ricerca: "Profilo hacker - 2007". Già nella prefazione si legge:

«A dimostrazione che la realtà supera ancora la fantasia, e che leggere il racconto di un'intrapresa scientifica può essere coinvolgente e appassionante quanto leggere (o vedere) una fiction»

poi la definizione del campo di indagine si sviluppa in una direzione tale per cui si giustifica il significato del termine hacker come problematico poiché, ad esempio:

«programmatori più giovani iniziano a sperimentare le proprie capacità con finalità malevoli ... e il termine assume così una connotazione negativa»

adeguando la grammatica al senso comune e giornalistico si rischia di accondiscendere il fatto che pochi programmatori più giovani e malevoli si approprino di una storia più ampia che non è loro. Implicitamente si rischia di affermare, anche senza volerlo, che esistano generazioni più malevole di altre, nella fattispecie, quelle più giovani.

Gli aspetti metodologici che non posso condividere di tale saggio sono:

1. selezione dei soli aspetti malevoli;

2. descrivere, senza argomentare, un trend negativo dell'hacking collegato allo sviluppo e potenziamento della tecnologia;

3. ridurre l'etica hacker a filosofie anarchiche, o all'auto-descrizione di Robin Hood informatici.

Per contro, Himanen (2001), seleziona solo gli aspetti positivi:

Passione, Libertà, Coscienza sociale, Verità, Antifascismo, Anti-Corruzione, Lotta contro l'alienazione dell'uomo, Eguaglianza sociale, Accesso libero all'informazione (cultura libera), Valore sociale (riconoscenza tra simili), Accessibilità alla rete, Attivismo, Responsabilità, Creatività. Inoltre contrappone l'etica Hacker all'etica del capitalismo protestante studiata da Weber assimilandola a quella odierna dell'iper-modernità, della contrazione del tempo, e della "gabbia di ferro" di cui lo stesso Weber avvertiva del pericolo:

[...] *In questo senso, l'etica hacker si presenta come una nuova etica del lavoro che sfida la mentalità che ci ha resi schiavi per così tanto tempo, quell'etica del lavoro protestante analizzata nel classico di MAx Weber, L'etica protestante e lo spirito del capitalismo.* (Himanen, 2001, pg. 5)

Weber, come noto, descrive una situazione ascetica favorevole allo sviluppo del capitalismo nel suo nascere e non la sua essenza spirituale tout court, cosa che Himanen sembra non cogliere. Infatti Weber riporta le parole del teologo puritano Richard Bexter:

[...] *la cura per i beni esteriori deve avvolgere le spalle dei santi soltanto come un sottile mantello che si possa gettar via in ogni momento.* (l'etica protestante e lo spirito del capitalismo, 1997, BUR, pag. 18)

e quindi commenta (si badi dopo Bexter):

[...] *I beni esteriori di questo mondo acquistarono un potere crescente e, alla fine, ineluttabile come mai prima nella storia.* (Weber, 1997, p. 18)

Altro aspetto che sembra controverso è quello monetario in cui Himanen scrive:

[...] *Nel descrivere questa dimensione del vecchio spirito capitalistico, l'etica del denaro protestante, Weber disse: "Il 'summum bonum' di questa 'etica' ", il suo bene supremo, è "guadagnare denaro, sempre più denaro". Nell'etica protestante, sia il lavoro sia il denaro sono visti come fini a sé stanti.* (Himanen 2003, p.101)

Questa frase è decontestualizzata in realtà Weber la inserisce in un discorso di largo respiro, dopo aver descritto l'ethos di Benjamin Franklin al fine di definire l'ideal-tipo ascetico protestante e il processo di perdita della virtù etica protestante. Il "dramma" non è l'etica protestante, ma proprio il suo opposto, cioè il suo venir meno. Vale la pena, per la rilevanza centrale della questione, riportare l'intero periodo:

[...] *Non solo il carattere proprio di Benjamin Franklin, quale viene appunto in luce in quella che è la comunque rara onestà della sua autobiografia, e la circostanza che egli attribuisca lo stesso fatto di avere scoperto l'«utiltà» della virtù a una rivelazione di Dio, che voleva così indurlo alla virtù, mostrano come qui si tratti di qualcosa di diverso da un abbellimento di massime puramente egocentriche. Ma, soprattutto, il «summum bonum» di questa «etica» - guadagnare denaro, sempre più denaro, alla condizione di evitare rigorosamente ogni piacere spontaneo – è così spoglio di ogni considerazione eudemonistica o addirittura edonistica, è pensato come fine a se stesso con tanta purezza, da apparire come alcunché di totalmente trascendente, in ogni caso, è senz'altro irrazionale, di fronte alla felicità o all'utilità del singolo individuo. L'attività lucrativa non è più in funzione dell'uomo quale semplice mezzo per soddisfare i bisogni materiali della sua vita, ma, al contrario, è lo scopo della vita dell'uomo, ed egli è in sua funzione.* (Weber, 1997, p. 75-76)

insiste poi a più riprese sul senso del lucro, proprio per la preoccupazione che questa dimensione ascetica non venga compresa come già è accaduto, prima dell'«etica hacker» di Himanen, con

[...] *L'avidità del lucro, la ricerca del guadagno, del denaro, di un guadagno quanto più alto possibile, in sé e per sé non ha nulla a che fare con il capitalismo...il capitalismo può addirittura identificarsi con l'inibizione di questo impulso irrazionale.* (Weber, 1997, p. 37)

Nella postfazione della stessa edizione (storia di una controversia) Ephraim Fischoff scrive:

[...] *La seguente analisi della controversia relativa all'acuto saggio di Weber cerca di mostrare come questo lavoro sia stato ampiamente frainteso, sia da amici che da avversari, come sia stata falsata la ricerca di Weber, cauta e incompleta per sua stessa ammissione, e come non siano stati osservati i suoi avvertimenti critici necessari per capire la sua tesi.* (Weber, 1997, p. 352)

In realtà Weber è puntuale nel sottolineare il distacco dell'imprenditore dalle cose terrene, mentre le ricchezze rappresentano il riscontro della grazia divina, la "certitudo salutis". Il lavoro "fine a se stesso" è inteso dal punto di vista dell'utilità del singolo, non dal punto di vista spirituale. Il capitalismo necessità di lavoro fine a se stesso, lo spirito protestante necessita di un riscontro terreno dello stato di grazia in cui il guadagno coincide col bene della collettività. Questi due aspetti sono incidentalmente compatibili con l'agire ascetico intra-mondano. Per la tesi di Weber è fondamentale il distacco dalle ricchezze in quanto è ciò che permette il reinvestimento aggirando la tentazione di godere dei beni di questo mondo e quindi la nascita del capitalismo e nel contempo l'atteggiamento ascetico. Sarebbe una contraddizione insostenibile, che farebbe crollare seduta stante tutto l'impianto teorico Weberiano sull'ascesi intra-mondana, il fatto che Weber "vedesse" nell'etica protestante "il lavoro e il denaro come fini a se stessi" che sono invece propri del capitalismo che per dispiegarsi necessita di inibire l'avidità. Quindi questo approccio sembra non rendere giustizia all'etica hacker e nemmeno a Weber. In realtà, quello che ne emerge è proprio l'opposto, cioè una sostanziale coerenza tra etica hacker ed etica protestante che si sostanzia nel comportamento razionale e nell'impegno nel mondo, con la costruzione della "Gerusalemme in terra" attraverso la professione:

[...] *l'idea di un dovere che l'individuo deve sentire e sente nei confronti del contenuto della sua attività professionale.* (Weber, 1997, p. 77)

Nell'approccio di Weber sta l'idea che possa essere plausibile l'agire nel mondo in combinazione con la dimensione ascetica che determina suo malgrado un processo di razionalizzazione e secolarizzazione generalizzato. I soldi rappresentano per l'etica protestante semplicemente lo strumento per misurare la dimensione della propria grazia. Il rapporto tra l'ideal-tipo etico protestante e l'ideal-tipo spirituale capitalista, che Himanen non distingue, è causativo attenuato e non intenzionale. Attenuato nel senso che Weber non esclude altre concause alla

nascita del capitalismo moderno prima, e la sua degenerazione, se vogliamo, poi. Oltre a questa minore distinzione tra etica e spirito, nell'analisi è peculiare anche la distinzione di intensità, in senso quantitativo, e di peculiarità in senso qualitativo di questa nelle diverse sette protestanti e nel tempo inteso come processo di genesi del capitalismo. Nell'analisi di Weber le sfumature assumono una importanza centrale perché sono quelle che permetto il passaggio fondamentale da una ascesi extra-mondana ad una intra-mondana, così il pietismo tedesco si colloca al confine tra queste due visioni del mondo, ma nonostante anche nel pietismo sono operanti quelle forme di dedizione al lavoro coerente con l'intero sistema etico protestante che prevede l'amore del prossimo attraverso le opere. Dal punto di vista processuale il passaggio significativo si attua con il calvinismo. Nel luteranesimo la dedizione al lavoro è precettiva, un sacrificio legato al destino dell'uomo e l'agire che determina è sostenuto tradizionalmente. Nel calvinismo si assiste ad un'evoluzione, la dedizione al lavoro assume una connotazione sociale, diviene un agire razionale orientata al valore. Vi è il passaggio da una concezione di individuo come "contenitore" della grazia a "strumento" di grazia:

[...] *esso si esprime in primo luogo con l'adempimento dei compiti professionali dati dalla «lex naturae», e assume così un peculiare carattere oggettivo e impersonale: quello di un servizio reso alla configurazione razionale del cosmo sociale che ci circonda.* (Weber, 1997, p. 170)

I soldi hanno una funzione strumentale alla misurazione della grazia, funzione che nella fattispecie del software libero può essere assunta da altri valori simbolici: suo grado di diffusione, grado di approvazione da parte della community, ma non da ultimo, il numero di commesse, consulenze ed il giro di affari che si creano attorno ad un progetto open source, senza per questo entrare in contraddizione con la sua filosofia, così come l'etica protestante descritta da Weber non entra in contraddizione con la sua teologia. Questa chiave di lettura può, addirittura, essere ulteriormente rafforzato dagli aspetti contrattuali che il contesto open source rielabora o produce in chiave decisamente moderna e razionale (Maine, 1998).

Il problema sostanziale sta nel fatto che Himanen non distingue una dimensione etica e ascetica intra-mondana precedente, che è quella che descrive Weber, e che ha contribuito, suo malgrado, ad innescare la modernità successiva pur restando estranea alla stessa, culla di un capitalismo destinato divenire poi diverso, autonomo da qualsiasi etica.

L'errore di Himanen, secondo questa interpretazione, è quello di contrapporre un'«etica» che nasce dal contesto della riforma luterana e che enuclea istanze di libertà e di riscatto, non soltanto ad esempio contro la vendita delle indulgenze la cui storicità può difficilmente essere messa in dubbio - così come non possiamo mettere in dubbio oggi la vendita delle licenze d'uso software - ma in generale contro un potere costituito, esso si fine a se stesso[3] e basato sulle rendite, verso il quale la crescente borghesia puritana del XVII secolo contrappone l'idea di lucro attraverso la professione. Nel suo complesso articolato l'etica protestante non giustifica la ricchezza in sé ma il lavoro, le ricchezze che non provengono da un agire etico sono condannate. La definizione “fine a stesso” evoca una “mancanza di senso” assolutamente incompatibile con il modo di procedere dell'analisi Weberiana impegnata prioritariamente a comprendere il senso che l'attore attribuisce al suo agire, a meno che non si tratti di un agire affettivo o tradizionale ancora presente, in qualche misura, nel primo luteranesimo e che verrà superato col calvinismo. All'opposto Weber oppone l'etica protestante proprio all'agire tradizionale e irrazionale. Weber non può allo stesso tempo sostenere un senso ed una mancanza di senso dell'agire etico protestante.

Quindi l'idea di lucro attraverso la professione contrapposta alle rendite passive, ad esempio della mano morta, o della servitù della gleba, assume tutt'altro significato che non il lucro fine se stesso contrapposto ad una nascente etica hacker. I riferimenti ai contesti storici non possono essere ignorati.

L'occasione che viene perduta con questo approccio è quello di non vedere universali di disagio, di rivendicazioni di libertà di agire che nuovamente affiorano, proprio dalle criticità e dalle contraddizioni della modernità. Rispetto all'epoca della riforma a cui si riferisce Weber, nel contesto iper-moderno attuale dell'etica hacker, la razionalità assume significato opposto. Non più una razionalità burocratica e individuale che si oppone alla magia e alla superstizione, ma una razionalità incrementale, riflessiva e collettiva che si oppone alla razionalità irrazionale della modernità (Luhmann, 2003). In entrambi i casi si tratta di un agire etico-razionale, cioè un agire razionale sostenuto eticamente, in grado di produrre rotture con il passato.

Si rischia di non vedere la tensione, la rivendicazioni di spazi professionali inediti contro il monopolio, non più dei latifondi e delle indulgenze, ma del software proprietario. Questo approccio ci impedisce di cogliere la coerenza di schema di senso tra le rivendicazioni della borghesia puritana nascente e Eric Steven Raymond:

[...] *Gli hacker sono anti-autoritari per natura. Chiunque possa darti degli ordini, può fermati dal risolvere problemi dai quali sei affascinato - e, visto il modo con cui le menti autoritarie funzionano, tali ordini generalmente saranno motivati da ragioni orribilmente stupide. Così, l'atteggiamento autoritario deve essere combattuto ovunque si trovi, affinché non soffochi te e gli altri hacker.*

A questo punto si rende necessaria una precisazione metodologia: il contributo di Eric Steven Raymond a questo studio verte su analisi di secondo livello e non di primo livello. Quindi l'affermazione sopra riportata di Eric Steven Raymond non viene considerato un assunto su cui articolare un discorso di ricerca. Questa affermazione, come il manifesto hacker, mostra i registri e le parole d'ordine che sortiscono degli effetti rilevanti ed osservabili, e che quindi, come tali, divengono significativi dal punto di vista scientifico, così come potevano essere le prediche del Reverendo Bexter ai fini della ricerca di Weber sull'etica protestante. Questo chiaramente lungi dall'affermare che Eric Steven Raymond muova da una qualsiasi intenzione religiosa. Piuttosto Eric Steven Raymond riesce a farsi interprete – in modo strumentale secondo molti esponenti del software libero – di una tensione etica, e di disagio come poteva essere quello di un artigiano del medioevo costretto a muoversi negli interstizi di libertà lasciati "casualmente" liberi in un mondo dominato dalla teologia e avente come referente centrale il pontificato romano. Questo aspetto può essere meglio compreso se viene dato seguito a quanto tracciato da Weber come fa nel 1988 Pellicani con il «Saggio sulla genesi del capitalismo»:

[...] *solo grazie alla formazione di un sistema politico a struttura policentrica il capitalismo è riuscito a svilupparsi sino a imporsi come il modo di produzione dominante. Esso poté fare i suoi primi passi, proprio in quanto si erano aperti, nel mondo feudale, numerosi interstizi di libertà. In questo senso, si può senz'altro dire che la storia del capitalismo e la storia del Potere limitato sono un'unica storia o, quantomeno, si presentano sulla scena come storie strettamente intrecciate.* (Weber, 1997, p. 179)

In questo senso, la riforma ha la funzione, così come fa Eric Steven Raymond, di mettere in discussione l'autorità, ed è così tanto attinente questo aspetto, che consapevolmente o meno, Eric Steven Raymond scrive nel 2001 «The cathedral and the bazaar». La coerenza tra etica hacker ed etica protestante non sta quindi sul piano teologico, né su quello spirituale, sui quali si possono costruire infinite contrapposizioni dialettiche, ma sul piano filosofico e squisitamente sociologico e quindi sul modo di intendere l'autorità nella sua

diversa attualità storica. Innanzi tutto come limite all'auto-realizzazione e alla realizzazione di una società migliore.

Questo a dimostrazione del fatto che l'open source ed il software libero oggi rischiano la mistificazione da parte di chi, in bene o in male, ha deciso di argomentare sul piano del “dover essere” senza cogliere la portata non solo etica, ma se vogliano anche morale, di un fenomeno che è già hic et nunc. Più che mai oggi serve una precisa definizione accettata a livello interdisciplinare di cosa significhi open source, software libero, licenza d'uso e hacker se si vogliono investigare in modo avalutativo i relativi fenomeni. L'esito di questo fenomeno è tutt'altro che scontato a fronte delle criticità e dell'inadeguatezza dell'economia nel riuscire ad ottimizzare e a mobilitare le risorse che il fenomeno open source mette in evidenza e rende necessaria una riflessione quanto più precisa sia a livello macro che a livello microsociologico.

A questo proposito val la pena di riprendere l'affermazione anti-autoritaria di Eric Steven Raymond, poco sopra, ed in particolare il passo dove dice: «Chiunque possa darti degli ordini, può fermati dal risolvere problemi dai quali sei affascinato». Con poche parole vengono sollevati i due livelli del problema. «Chiunque possa dare degli ordine» si riferisce ad un idea di organizzazione che fa parte della nostra vita quotidiana: università, lavoro, famiglia quindi alle istituzioni in genere e quindi un livello macro che riguarda le logiche generali in cui è organizzata la nostra società. « Problemi dai quali sei affascinato» riguarda l'individuo ed il suo rapporto con l'ambiente o con il sistema stesso da un punto di vista costruttivistico. Nel complesso si evidenzia una ambivalenza problematica: «fermare dal risolvere problemi» è sia un problema della società nel suo complesso ma anche un problema del singolo individuo allontanato da ciò verso cui si sente “naturalmente” attratto. Naturalmente sta a significare proprio questa sinergia tra individuo e società, il fatto che l'individuo venga precluso da ciò che lo affascina si riverbera sulla società, la stessa società ne viene diminuita. Si tratta in pratica di impedire l'esercizio della vocazione da parte della società, cioè del beruf. Eric Steven Raymond esprime questo disagio e verrebbe da chiedersi da dove nasce, quali sono i contesti e le situazioni in cui avviene questo impedimento. Innanzi tutto si rivolge ad un pubblico che già sperimenta questo disagio, peraltro evidentissimo nel “manifesto hacker”, e che pertanto non ha bisogno di spiegazioni o specificazioni perché si rivolge ai suoi simili.

Si tenga presente che, come spesso accade nelle organizzazioni e nei rapporti asimmetrici di potere tra lavoratori, le ragioni che vengono sollevate perché non si

programmi non sono mai argomentate. Si ricorre spesso a codici e ci si appella a concetti che si pensa generalmente accettati e di buon senso: economia di mercato, economia di scala, industrializzazione del software e via dicendo.

Una spiegazione può essere presa da Crozier (1978), in cui l'imponderabile tecnico (informatico in questo caso) assume un carattere di minaccia. La strategia è quindi quella di estromettere la complessità delegandola al mercato anche se inadeguato, anche se ne viene esperito il fallimento e l'impossibilità di una integrale industrializzazione del software. Quello che viene garantito è il rispetto formale delle procedure in particolare se l'ambito è quello della pubblica amministrazione. Altra testimonianza può essere offerta inaspettatamente dalla letteratura autobiografica, in un settore diverso, e quindi non sospetto, che è quello della tecnologia meccanica o più specificatamente motociclistica:

[...] *Ecco perché Sylvia non diventa matta per via del rubinetto, pensai. Si rimuove sempre la rabbia momentanea verso qualcosa che si odia a fondo. Ecco perché John fa finta di niente ogni volta che il discorso cade sulla riparazione della moto, persino quando è evidente che per lui è una sofferenza. Perché si tratta di tecnologia. Ma certo è chiaro. Se John e Sylvia hanno scelto di viaggiare è soprattutto per la tecnologia, per ritrovarsi in campagna, all'aria fresca e al sole. Il fatto che li riporti proprio sul luogo e nel punto da cui credono di essere finalmente fuggiti li raggela entrambi ... Non sono d'accordo con John e Sylvia per quanto riguarda la manutenzione della moto, ma non perché non capisca i loro sentimenti verso la tecnologia. Penso solo che la fuga dalla tecnologia e l'odio nei suoi confronti portino alla sconfitta. Il Buddha, il Divino, dimora nel circuito di un calcolatore o negli ingranaggi di un cambio di una moto con lo stesso agio che in cima a una montagna o nei petali di un fiore.* (Pirsig, 1992, p. 25-27).

Quando Robert si offre di aggiustare la moto a John la reazione è ugualmente quella del ricorso al mercato:

[...] *«uno spessore. Una striscetta piatta di metallo. La inserisci attorno al manubrio sotto il colletto per aprirlo di più...». «dove si comprano?». «Ne ho giusto qui qualcuno» dissi tutto giulivo mostrandogli la latina di birra che avevo in mano. Per un momento non capì. Poi disse: «cosa con la latina?»...Gli facevo risparmiare tempo e denaro ma mi accorsi con sorpresa che lui non apprezzava affatto...il manubrio non l'avremmo aggiustato per niente...avrei dovuto sgattaiolare dietro il bancone, tagliare uno spessore di latina, togliere le scritte e*

dirgli che eravamo fortunati, era l'ultimo che avevo, importato appositamente dalla Germania.(Pirsig, 1992, p. 60-61).

E si noti la corrispondenza del racconto di Pirsig in un'altra osservazione etnografica:

Sono in un locale dove si ritrovano degli appassionati di Gnu/Linux ed uno di loro racconta:

«gli ho sistemato la macchina, era una macchina win ovviamente. Scusate ma devo campare. Poi mi ha chiesto di installargli explorer veloce come ha anche il suo collega. Vado a vedere che browser usa il suo collega. Era Firefox. Bene, lo scarico e lo installo. Gli mostro l'icona per avviarlo e mi dice di non volerlo perché non è explorer. Gli dico che è quello che ha anche il suo collega e mi risponde tagliando corto di rimettergli quello di prima ma che glielo faccia andare veloce. Alla fine gli ho tenuto Firefox e gli ho cambiato solo l'icona, così ora lancia firefox con l'icona di explorer, ed è contento». (diario, 17 febbraio 2009).

Pur consapevoli che questo racconto si svolge in una situazione conviviale e non in una Corte d'Assise sotto giuramento, testimonia comunque una inclinazione anticonvenzionale[4] da una parte e la consapevolezza di una "immotivata" fiducia nel mercato dall'altro, da parte di alcune persone. Ciò che ne risulta è la plausibilità di chi ha un rapporto critico con la tecnologia a porre una fiducia esclusiva nel mercato formale e tradizionale e questo può riguardare anche chi occupa posizione decisionali nelle imprese o nella pubblica amministrazione più attento alla correttezza formale dei procedimenti (Crozier, 1978). Questo spiega perché il rischio di essere fermati nella soluzione dei problemi per un hacker è sempre molto elevato, e come le multinazionali, non solo del software, possano ontologicamente organizzare la fiducia degli individui in competizione con gli hacker. Ma questo messaggio viene ribadito sistematicamente anche da consulenti informatici che operano nell'ambito informatico proprietario:

«Non vi avvarrete di software fatto in casa spero? magari con open source? Se non hai garanzia di continuità è un casino, ti ritrovi col culo per terra. Cosa succede se domani il programmatore ti va via? E se gli strumenti open ad un certo punto non vengono più mantenuti?». (diario 16 aprile 2010).

Come si nota i codici usati sono quelli poi ribaditi all'interno dell'organizzazione. In effetti i cosiddetti commerciali, che hanno rapporti intensi

con i decisori, forniscono, prima che i servizi software, le grammatiche, i registri e i codici necessari agli stessi decisori (capi ufficio e direttori) per argomentare con gli stake holders ad un livello e con i propri collaboratori ad un altro livello.

Da un punto di vista complessivo si potrebbe obiettare che tutto sommato la soluzione del problema avviene, che tutto sommato il mercato formale, comunque soddisfa i bisogni e che comunque qualche programmatore da qualche parte ci sarà pure. Come si cercherà di dimostrare più avanti queste dinamiche di fatto diminuiscono quantitativamente le soluzioni ai problemi, restringono di fatto il mercato con un procedimento del tutto analogo al protezionismo commerciale. Si cercherà di dimostrare che anche a livello macro le idee di industrializzazione, continuità, garanzia del mercato, economia di scala applicata al software altro non fanno che aumentare il mercato dei brevetti e restringere il mercato delle competenze con una riduzione del surplus complessivo e, riprendendo Pirsig:

[...] *il manubrio non l'avremmo aggiustato per niente.* (Pirsig, p. 60)

Alla sequenza ritenuta e alla sequenza aggiornata (Galbraith, 1968) , si potrebbe aggiungere come sottospecie di quest'ultima, la sequenza ridotta. Tutta questa dissertazione è necessaria al fine di descrivere l'ideal-tipo hacker in quanto si tratta di sistematizzare socio-economicamente schemi di senso che emergono dalla dialettica hacker che noi, in questa sede, ci accingiamo a verificare senza darli per scontati.

Pirsig, che a questo punto ha tutte le carte in regola per essere definito hacker anche se non informatico, ci offre insperatamente anche un altra categoria che ci da indirettamente la possibilità di rinforzare ulteriormente il concetto di dinamica di inibizione, annichilimento, controllo dell'hacking. È il concetto di téchne che verrà ripreso poi altrettanto coerentemente da Gian Antonio Gilli con “Origini dell'eguaglianza” e Umberto Dante con “L'utopia del vero nelle arti visive”. La cosa straordinaria è che Pirsig si occupa di meccanica applicata alla motocicletta, Gilli di ricerche sociologiche dell'antica Grecia e Dante di Arti visive, eppure tutti questi autori aiutano a definire l'ideal-tipo hacker pur senza interessarsi specificatamente di informatica:

[...] *In realtà la radice della parola «tecnologia», Téchne, in origine significava proprio «arte». Gli antichi greci non distinguevano concettualmente l'arte della manifattura, e quindi non crearono mai due parole per definirle. La bruttezza non è intrinseca nemmeno ai materiali della tecnologia moderna –*

affermazioni che si sente spesso fare di questi tempi. La plastica e i prodotti sintetici fabbricati in serie non sono brutti di per sé. Rimandano a cose brutte per associazione. La bruttezza vera non sta negli oggetti tecnologici né, secondo la metafisica di Fedro, essa dipende dai soggetti della tecnologia, cioè da chi produce o da chi la usa, ma sta nel rapporto tra chi produce la tecnologia e le cose prodotte, il quale determina poi un analogo rapporto tra chi usa la tecnologia e le cose usate. (Pirsig, 1992, p. 281).

Ugualmente Eric Steven Raymond (2002) indica che la predisposizione mentale dell'hacker non è confinata al solo campo informatico ma la si può trovare in qualsiasi campo dell'arte, della scienza e della tecnica. Chris DiBona (2000) fa notare come nello stesso tempo l'informatica si differenzi dalle altre scienze perché nasce con la peculiarità di condividere risultati, come espressione tecnologica del metodo scientifico e che, come tale, la chiusura del codice ed il mercato delle licenze d'uso sono forzature che ne impediscono il suo dispiegarsi.

[...] *Fra le molte parole greche per le quali è impossibile trovare un corrispondente nelle lingue moderne vi è certamente la parola tèchne. Tradurla con 'tecnica', o 'arte', o 'mestiere' (e tradurre con 'tecnici', 'artigiani', 'artisti' i technitai, vale a dire i portatori di téchne) significa cogliere solo una parte del tutto.* (Gilli, 1988, p. 5).

Il ricorso alla categoria di portatore di téchne non ha il solo scopo di descrivere una peculiarità più o meno romantica di questo ideal-tipo téchne/hacker, ma ha lo scopo di evidenziarne la problematicità sociale nei confronti dello stesso. Gilli (1988) parte con la definizione di téchne, per poi sviluppare una dissertazione molto articolata e giungere ad un punto interessante anche ai fini di questa indagine. Richiamando Plutarco mostra come anche nella Grecia antica vigesse questa ambivalenza tra “diletto dell'opera” e “disprezzo per l'artista”. Il disprezzo non è gratuito, per Gilli ha un preciso significato e cioè il controllo. Falea di Calcidone, menzionato nell'opera di Aristotele, propone per i “technitai” una condizione duramente subordinata di schiavi pubblici. Dante (2002) fa notare come nella Roma antica la situazione sia ben peggiore, nemmeno la letteratura romana se ne occupa se non per ribadire con Seneca la condizione di inferiorità rispetto al vero prestigio sociale. Dante (2002) nota anche qualcosa in più, cioè la discrasia tra ciò che la letteratura attribuisce al technitai e le sue condizioni economiche:

[...] *i technai, disprezzati dai letterati, vengono arricchiti e collocati dentro la storia dalle dracme e dai sesterzi dei loro clienti.* (Dante, 2002, pag. 37).

Ma Dante (2002) mette anche in evidenza come successivamente si impongano canoni di verosomiglianza che mettono in crisi la cultura estetica dell'età classica che scompare definitivamente quando si affermano modelli canonici, uniformi e convergenti idonei alla produzione in serie, una sorta quindi di industrializzazione dell'arte che passa attraverso il necessario controllo dei technitai.

Questi "canoni di verosomiglianza" nella loro accezione negativa vengono evidenziati anche da Zacchiroli durante l'intervista del 17 agosto 2010 quando dice: « *non deve essere a posteriori un privilegio del vendere copie del software che sono tutte indistinguibili l'una dall'altra e che non costa niente produrre* »

Tutto ciò è drammaticamente coerente, se riconosciamo la téchne nell'hacking e l'informatica nell'arte e in definitiva giustifica la lunghezza di questa dissertazione sulla costruzione di un ideal-tipo fecondo che apre un ampio ventaglio di possibilità di indagine, verso direzioni a tutt'oggi poco indagate nonostante il grande interesse che suscita questo fenomeno nella psicologia, nell'economia, nella sociologia e nella filosofia, soprattutto per i suoi aspetti organizzativi paradossali.

Note

1. ↑ Geek (pronuncia: ghik) è un termine di origine anglosassone, indicante una persona affascinata dalla tecnologia. Il significato di geek non coincide con quello di nerd, avendo una connotazione positiva almeno tra coloro che si fregiano del termine e amano etichettarsi in tal modo. www.wikipedia.it 12.09.2010
2. ↑ Nerd è un termine della lingua inglese con cui viene chiamato chi ha una certa predisposizione per la ricerca intellettuale (magari associata a un quoziente intellettivo superiore alla media), ed è al contempo tendenzialmente solitario e con una più o meno ridotta predisposizione per la socializzazione. www.wikipedia.it 12.09.2010
3. ↑ Himmanen imputa all'etica protestante una predisposizione all'accumulazione di ricchezze fine a stesso, quindi un comportamento senza senso. ciò che invece si argomenta è che l'etica protestante elabori schemi di

senso ben precisi in cui diviene, semmai, priva di senso (fine a se stessa) la rendita non legata alla professione.

4. ↑ un programma viene associato con l'icona del programma concorrente

In questa tesi si è consapevoli del fatto che il movimento del software libero da una sua lettura storica del fenomeno, e che questa lettura non può essere imparziale. Molto del materiale usato è prodotto in contesti di software libero. Per contro è importante far notare almeno due aspetti:

Il materiale, per così dire, a sostegno del software proprietario è molto scarso;

Questa tesi si occupa specificatamente di software libero e open source e non di software proprietario;

È in ogni caso difficile comprendere il software libero e l'open source senza descrivere la problematicità del software proprietario a partire dalla sua peculiare tendenza al monopolio, senza per questo interessarci alla dimensione morale se non come occasione di analisi di secondo o terzo livello. Non di meno si è anche consapevoli di molte contraddizioni interne all'open source e queste stesse potranno essere sottostimate, ma non per selezione premeditata, ma per la scelta del campo di indagine volto a spiegare il successo di modelli organizzativi, processuali, sistemici e strutturali, se non inediti, quantomeno inaspettati e spesso paradossali del software libero.

“Era Aprile del 2007, mi trovavo alle prese con il problema di dover sviluppare un servizio in un ambiente di sviluppo proprietario. In realtà il mio lavoro si era di molto ridotto, i sistemi venivano commissionati da software house esterne, io dovevo occuparmi solo di farli funzionare. Spesso non ci riuscivo, non sapevo più dove mettere le mani. Mi avevano acquistato degli strumenti, molto costosi, di una grande software house americana, dovevo usarli per integrare delle funzionalità ai servizi informatici acquistati all'esterno. Molti dei servizi che avevo sviluppato erano stati sostituiti con tecnologia chiusa. Mi sentivo responsabile del funzionamento degli strumenti ma ne avevo perso il controllo.

Mi ero messo a cercare su google le specifiche su come usare il server Webmap (software proprietario) che avevamo adottato. So che ho speso delle ore senza successo finché, quasi per caso, sono capitato nel sito di Mapserver (open source). Ho trovato tutte le informazione di cui avevo bisogno e molto ancora, peccato non si riferissero Webmap che era quello di cui avevamo la licenza d'uso. Se avessi avuto la stessa assistenza - forum di discussione, gente che si scambiava opinioni, contatti esempi da scaricare e testare, gli stessi software da scaricare e usare seduta stante, suggerimenti per le configurazioni – avrei fatto molto con

Webmap. La cosa più naturale da fare era passare a Mapserver, senza nemmeno il bisogno di chiedere un impegno di spesa, e da allora cerco solo di capire com'è possibile tutto questo". Ancora oggi la situazione non è cambiata ed ogni tanto ci riprovo con google, non si sa mai, la nostra licenza d'uso è ancora valida: "webmap configuration". Ma non ottengo risultati utili. (nota autobiografica)

Richard Stallman (2000) fondatore della "free Software Foundation" ricorre al concetto di autodifesa per giustificare l'utilizzo di un sistema proprietario, UNIX, al fine di creare un sistema operativo non proprietario. Questo ricorso all'autodifesa è innanzi tutto un segno evidente di tensione tra software libero e software proprietario. In realtà non è tutto conflitto aperto e nemmeno è tutto conflitto. Vi sono molti prodotti open source che vengono eseguiti su sistemi proprietari e viceversa, Recentemente Microsoft, nel suo sito ufficiale ha aperto una sezione open source in cui annuncia il sostegno a Gnu/Linux, in realtà molto sospetta per aderenti al circolo "www.vicenza.linux.it". L'analisi del fenomeno dal punto di vista conflittuale è comunque corretta perché consente una visione ideal-tipica altrimenti difficile.

Quindi l'immagine del conflitto rispecchia un preciso intento di accentuazione unilaterale allo scopo di comprendere. La posta in gioco nel conflitto, come si cerca di dimostrare, sono gli spazi professionali, la paura di perdere l'esperienza "sacra" nel creare collettivamente, la gratificazione intellettuale, la soddisfazione di bisogno di desiderabilità sociale e non da ultimo anche gli interessi economici minacciati dal monopolio dei sistemi operativi. Nessuna di queste voci è esclusiva dell'open source, la gratificazione intellettuale è possibile anche per uno sviluppatore di software proprietario, la soddisfazione della desiderabilità sociale pure. La differenza sta piuttosto nella dimensione collettiva, nella community che amplifica gli aspetti legati alla gratificazione intellettuale e comunitaria, e non ultime, la fiducia e la solidarietà. Per contro l'isolamento del programmatore di software proprietario non può che privilegiare aspetti più tipici dell'individualismo moderno e quindi il riscontro del ritorno economico.

Più sinteticamente il motivo del conflitto è quello che esprime Robert Young:

[...] Non è possibile competere con un monopolista giocando secondo le sue regole. Il monopolio ha le risorse, i canali di distribuzione, le risorse di ricerca e sviluppo; in breve, ha troppi punti forti. Col monopolio si compete cambiando le regole del gioco a favore dei nostri punti forti. (Young, 2000).

Questo mette in evidenza l'incapacità del mercato di ottimizzare la risorse software per potere di mercato in cui anche l'approccio economico ortodosso giustifica l'intervento dello stato (Mankiw, 2007). Menkiw riporta, come esempio per spiegare cos'è il potere di mercato, l'esempio di “un unico pozzo d'acqua con un'unica città in mezzo al deserto”. Quindi Menkiw descrive il potere di mercato riferendosi ad una situazione estremamente locale quasi a volerne indicare l'improbabilità (ndr), il che è proprio l'opposto di quanto accade nel monopolio informatico, si tratta di un monopolio unico nel suo genere, cioè di un monopolio globale.

Questo rende chiaro il perché del non intervento del governo. Innanzi tutto è difficile stabilire quale governo debba intervenire e se a tale governo convenga intervenire nel momento in cui un monopolio globale con una sede locale crea evidenti vantaggi nel locale: esportazioni, entrate erariali, occupazione e via dicendo. Senza contare che un governo più accondiscendente di un altro è sempre possibile trovarlo e quindi la minaccia a delocalizzare è implicita (Beck, 2001). L'esempio più calzante sarebbe quello di un unico pozzo d'acqua in tutto il mondo e non di un unico pozzo d'acqua in una città in mezzo al deserto. In questo caso il primo ad appropriarsene costruirebbe gli acquedotti che dall'unico pozzo in tutto il mondo, si diramerebbero in tutto il globo. Anche nel caso in cui si scoprisse un altro pozzo non ci sarebbe la possibilità di scalzare la posizione dominante.

Parliamo ovviamente di sistema operativo, cioè il requisito minimo per un computer per poter funzionare ed eseguire dei programmi. Le cose non sono andate esattamente così, i sistemi operativi che si sono susseguiti sono stati molti finché MS-DOS (Microsoft Digital Operative System) è riuscita ad ottenere una posizione esclusiva. Questo è stato possibile per una caratteristica immanente all'informatica, apparentemente paradossale con una logica di monopolio che è la condivisione. È quindi riprendiamo nuovamente quanto afferma Chris DiBona in voices from the revolution (2000):

[...] L'informatica, quindi, differisce in modo sostanziale da qualsiasi altra scienza. L'informatica ha il solo significato di consentire la replica dei risultati tra pari: il codice sorgente. Per dimostrare la validità di un programma a qualcuno, devi fornirgli gli strumenti per compilare ed eseguire il programma. (Chris DiBona, 2000).

e inoltre

[...] Quando cominciai a lavorare nel laboratorio di Intelligenza Artificiale del MIT nel 1971, entrai a far parte di una comunità in cui ci si scambiavano i programmi, che esisteva già da molti anni. La condivisione del software non si limitava alla nostra comunità; è un cosa vecchia quanto i computer, proprio come condividere le ricette è antico come il cucinare. Ma noi lo facevamo più di quasi chiunque altro. Richard Stallman (2000)

Quindi il problema sostanziale è che fintanto che la condivisione di soluzioni riguarda una comunità limitata di scienziati, che tra l'altro si occupano di tecnologia, la condivisione può essere integrale, si possono scambiare anche i sistemi operativi, si aggiustano le configurazioni e si fanno adattamenti tra piattaforme diverse, ma quando l'informatica diventa di massa ed entra nelle imprese, nelle case e nelle amministrazioni statali e via dicendo il livello di condivisione deve necessariamente spostarsi ad un livello macchina alto, quindi più semplice, deva essere superata la complessità del livello macchina basso e specialistico, in pratica diventa necessario un sistema operativo condiviso per poter scambiare dati e distribuire software. In pratica è la chiusura che consente la condivisione, dal punto di vista costruttivista è la chiusura auto-poietica che consente ai sistemi di comunicazione di essere tali cioè di essere sistemi sociali (Luhmann, 1990).

Dopo aver sostituito il locale con il Globale nella metafora di Mankiw, ora sostituiamo il pozzo d'acqua con il linguaggio e quindi riformuliano il concetto di monopolio applicato all'informatica: è come se esistesse un unica lingua in tutto il mondo e dovessimo pagare le royalties a chi l'ha inventata ogni volta che dovessimo dire qualcosa, a parte l'evidente vantaggio della drastica diminuzione di discorsi inutili, la situazione diverrebbe insostenibile. Eppure il linguaggio umano in fatto di complessità non ha nulla da invidiare ad un sistema operativo, eppure è una cosa che diamo per scontata, immanente alla natura umana conquistata con l'evoluzione e non ci sogneremmo mai di dover pagare delle royalties, nel caso italiano, agli eredi dei padri della nostra lingua italiana: Dante, Petrarca, Boccaccio e Ariosto.

Quello che preme far notare è che, come il linguaggio, anche la tecnologia, diciamo di base, che ci consente di eseguire copie identiche di software su computer diversi, di scambiarci i dati, come questa stessa tesi che sto scrivendo su un sistema Gnu/Linux e che potrà essere letta anche da un sistema windows, contiene la sedimentazione di soluzioni, o se vogliamo la selezione, di soluzioni create dall'avvicendarsi di migliaia di scienziati, o semplici “artigiani informatici”

nel corso di decenni, se facciamo risalire la storia dell'informatica al primo calcolatore ENIAC creato nel 1946 presso la Moore School of Electrical Engineering dell'Università di Pennsylvania da J. Presper Eckert e John Mauchly, o al 1673 se pensiamo alla prima calcolatrice funzionante di Leibniz, ma anche oltre se pensiamo all'informatica come risultato del sedimentarsi della conoscenza umana non certo estranea allo stesso linguaggio umano e nemmeno ad altri settori scientifici come la fisica o la matematica.

L'avvento ad un certo punto della storia del monopolio del sistema operativo è pertanto un paradosso, in quanto implicito nella stessa natura collaborativa dell'informatica. Da un punto di vista costruttivista il monopolio è ciò che realizza la chiusura auto-poietica necessaria nei sistemi di comunicazione e quindi in definitiva svolge un ruolo sistemico necessario senza però evitare la contraddizione interna. La sfida dell'open source è, quindi da un punto di vista sistemico, quella di realizzare la chiusura auto-poietica (Luhmann, 1990), attraverso l'apertura del codice evitando così il monopolio. La sua funzione oggi più importante, di cui possono beneficiare anche i più assidui sostenitori del software proprietario e che usano solo software proprietario, è quello di attenuare efficacemente il monopolio di Microsoft:

Così come è avvenuto l'ultima volta, la chiave del prossimo stadio dell'industria informatica è infatti nella liberazione dello stadio precedente dai vincoli proprietari. Come ha notato Bob Young di Red Hat, distributore leader di Gnu/Linux, il suo fine non è di scalzare Microsoft dal trono dei sistemi operativi, ma piuttosto di restringere il valore monetario di quel mercato. Il punto è che il software Open Source non deve proporsi di battere Microsoft facendo il suo gioco, ma invece di cambiare la natura del gioco.(O'Reilly, 2000)

Non è compito di questa tesi prevedere il futuro, resta il fatto che il grande assente, in una situazione vistoso di fallimento di mercato, è una qualsiasi autorità globale in grado di garantire lo spazio virtuale dove si possano esercitare le libertà digitali. L'arduo compito è lasciato a volontari auto-coordinati che si riconoscono nel software libero.

Questo è quanto Ulrich Beck (2001) aveva già osservato e cioè l'incapacità dei sistemi normativi locali di fornire garanzie di fronte ad un capitalismo transnazionale. Lo stesso fatto che sia il monopolio informatico a gestire lo spazio pubblico dove si negozia un medium comunicativo importante dei sistemi sociali, cioè i sistemi operativi dei calcolatori, è indice evidente di ciò che Beck (2001)

chiama globalismo, cioè l'assunzione da parte mercato transnazionale dell'azione politica, finanche esattoriale, come emergerà più avanti analizzando gli aspetti economici del software libero.

Sebbene la “licenza d'uso” è un concetto legato al software proprietario ed assuma nell'ambito dell'open source una accezione negativa, essa svolge un'importante funzione nel consentire un'oggettivazione di qualcosa che di per se è volatile. Il software fisicamente – dal punto di vista delle scienze fisiche - è un insieme di stati di tensioni all'interno di circuiti elettronici, schede di memoria o dischi ferromagnetici. Crea stati di tensioni – utili – a partire da stati di tensioni opportunamente organizzati attraverso principalmente due processi: stesura del codice attraverso un linguaggio; compilazione del codice attraverso un altro software che è il compilatore. In questo processo intervengono più strumenti software come il debuger, l'interprete, l'editor e via dicendo ma che per i nostri scopi non vale la pena approfondire. Questi software esistono materialmente solo se esiste un “supporto” che li contiene. Da un punto di vista concettuale sono una serie di istruzione che vengono eseguite da un computer in grado, per così dire, di interpretare quei diversi stati di tensione. Si tratta di soluzioni che poste in sequenza secondo un determinato senso rendono possibile la comunicazione dell'uomo con gli apparati elettronici ai fini dell'automazione di un processo. Questa ammasso di diversi stati di tensioni è difficilmente immaginabile come oggetto, o meglio, senza un'opportuna alfabetizzazione sarebbe difficile distinguere ciò che l'hardware fa in virtù della sua dotazione software. Quando allo stesso ammasso di diversi stati di tensioni viene invece attribuito un prezzo la sua reificazione e quindi oggettivazione diviene immediata. Se non si comprano dei supporti che contengano quel determinato codice non si potrà fare quella determinata cosa.

Il software libero open source non ha questa “fortuna” di avere a disposizione un medium simbolico potente come il prezzo che ne permetta l'oggettivazione. Questa si definisce a sua volta attraverso l'analogia con il software proprietario, quindi il software proprietario è un oggetto perché ha un prezzo, di conseguenza il software è un oggetto, e quindi anche il software libero è un oggetto. Il procedimento analogico porta con se però anche un rischio: Il software proprietario ha un prezzo e quindi ha un valore, il software libero non ha alcun prezzo e quindi non ha alcun valore anche se ha dei costi e spesso molto elevati. È vero che ha un valore in sé, in quanto alla sua utilità ma questo presuppone l'esperienza del suo utilizzo che non può essere fatta a-priori cioè nel momento in cui si decide di acquistare un software. In pratica l'idea di software libero sembra derivare, nel bene e nel male, dal concetto di software proprietario.

L'idea di oggetto e di valore del software sono ciò che ne consentono la sopravvivenza, quindi il software libero deve risolvere il problema di oggettivare e valorizzare autonomamente i suoi prodotti. Per questo motivo nascono le licenze open source che, paradossalmente, sfruttano un istituto già presente nei diversi ordinamenti giuridici dei paesi occidentali, e cioè il diritto d'autore, per stravolgerlo poi a livello di clausole. Ancora una volta si nota l'interessante coerenza nel modo di procedere del movimento del software libero e dell'open source in cui un elemento, sia esso un componente elettronico, un software, una lattina di birra o un istituto giuridico, viene usato in modo anticonvenzionale, improprio, efficace e lecito.

La cosa rilevante è che questa costante anticonvenzionale non sembra essere parte di un modus operandi formalizzato, è un modo di procedere implicito nella cultura hacker, che rivela non un “dover essere” funzionale a strategie identitarie pianificate o cercate, non una moda, ma uno stile di vita fondato su uno specifico capitale culturale e sistema di valori. Si tratta di razionalità anticonvenzionale rivolta alla complessità e connotata da ethos. La cultura hacker si connota, a mio vedere, come cultura della complessità dove l'anti-convenzionalismo non è cercato ma incidentale per la sua tendenza di ampliare le opzioni delle soluzioni, o le connessioni (Luhmann) se si preferisce, ed è professionale perché deriva da uno specifico habitus (Bourdieu), cioè un costante impegno sostenuto emotivamente nei confronti di problemi gratificanti da risolvere.

La licenza open source è quindi un documento legale associato alla distribuzione di un prodotto software libero. Il detentore del diritto d'autore pur rimanendo detentore di tale diritto concede ai licenziatari di usare, modificare, integrare, riprodurre, duplicare e distribuire il software anche dietro compenso. D'altro canto il licenziatario è obbligato a rendere note le modifiche apportate e questo consente un'evoluzione autonoma del software potenzialmente illimitata. Come si vedrà più avanti questa strategia, dal punto di vista neo-funzionalista ha lo scopo di complessificare il sistema ai fini di controllare l'ambiente. Come si nota il software viene caricato di potenziale dinamico che temporizza l'elemento software il quale muore e rinasce continuamente in forme diverse adattandosi ai più svariati bisogni di automazione. In questo modo un software rischia di accrescersi a tal punto di ridurre la sua efficienza facendo emergere a livello utente le complessità che contiene. Però non tutte le modifiche vengono validate dai detentori dei diritti d'autore e quindi questo dovrebbe garantire un certo controllo sulla sua crescita altrimenti incontrollata. Se alcune modifiche sono ritenute importanti da chi le fa, ma non da chi detiene i diritti sul software, è

sempre possibile quello che in gergo si chiama “fork”, cioè la creazione di un nuovo prodotto orientato a funzioni diverse.

Il problema di una possibile appropriazione di una licenza open source è controllato sempre contrattualmente. Quanto viene implementato attraverso attività di programmazione o di integrazione di altri software eredita le caratteristiche contrattuali del software libero, in caso contrario la licenza vieta la diffusione della parte open. Il problema è capire chi controlla tutto questo. Principalmente questo avviene su due livelli. Da un parte l'autorità giudiziaria, in quanto includere in un pacchetto proprietario del software libero significa correre il rischio che un giudice a seguito di una controversia dichiari tutto il pacchetto GNU-GPL. Questo è quanto è avvenuto recentemente nel caso del produttore di apparecchi televisivi Westinghouse Digital Technologies, obbligato da una sentenza di un giudice federale del distretto sud di New York, del 27 luglio 2010, a pagare una multa di 90.000 $, le spese processuali per 47.000 $, e a consegnare i televisori ancora invenduti all'organizzazione Software Freedom Conservancy per violazione della licenza GPL versione 2.0. Inoltre il forte carattere identitario della comunità dispiega forme di controllo diffuso. In genere si tratta di professionisti, studenti, ricercatori o docenti universitari che spesso hanno a che fare anche con software proprietario e sono molto attenti a quanto succede nel campo della tecnologia. La comunità monitora continuamente l'andamento delle diverse versioni GPL di un prodotto e questo ha una funzione vitale anche nel mantenimento dei pacchetti, ad esempio quando inizia a dare segni di ipertrofia i segnali di allarme aumentano. Il feedback è continuo, rapido, diffuso e abbassa l'entropia. Quindi non solo il software libero è auto-poietico ma anche auto-controllato e in questo senso acquisisce una complessità simile a quella di organismo vivente.

Altro problema potrebbe essere quello di modificare i termini della licenza GPL. Accade sempre più spesso, e di questo se ne ha riscontro anche nelle osservazioni etnografiche, che dei prodotti software vengano spacciati come software libero, in quanto sempre di più il software su alcuni ambiti specifici, in particolare sistemistici, assume significato di marchio di qualità. Il problema è quindi che pacchetti proprietari vengano venduti come pacchetti GPL. Anche qui troviamo un altro elemento tipico dello stile hacker, cioè la ricorsività: per prevenire l'uso illecito del “marchio” GPL la licenza GPL è sottoposta alla stessa licenza d'uso GPL.

Le licenze open source non riguardano solo la fomula GPL, dalla stessa GPL è stata anche creata la GNU-LGPL (GNU Lesser General Public License) meno restrittive per il software distribuito a corredo che può anche essere proprietario. Altre licenze molto usate sono la BSD (Berkeley Software Distribution) che può essere modificata senza restrizioni se non quella di riportare il nome dell'autore (Berkeley), di non introdurre pubblicità, di non usare i propri loghi sia sul codice binario (chiuso) che sul codice sorgente (aperto). Altra licenza è la MIT (Massachusetts Institute of Technology) la meno restrittiva di tutte. Esistono poi diverse versioni giudicate dalla Free Software Foundation GPL compatibili (auto-poietiche) o semplicemente libere, su cui per ragioni di complessità, e per i nostri scopi, non vale la pena addentrarci, anche perché ampiamente documentate nella rete.

Ciò che attrae molto la ricerca attorno al fenomeno software libero ed open source è il suo paradosso economico, motivazionale e in definitiva organizzativo. Il problema che “affascina” è spiegare come sia possibile l'emergere di tecnologia di buon livello, competitiva e in molti casi migliore di quella che si trova nel mercato, a tal punto da diventare una sorta di marchio di qualità e di garanzia. La sfida è spiegare come sia possibile che uno studente al secondo anno di scienze informatiche dell'università di Helsinki riesca ad aggregare persone motivate a fare cose interessanti al di fuori di una qualsiasi organizzazione e al di fuori, quindi, da un qualsiasi sistema di incentivi formalmente economici, decidendo poi di distribuire liberamente questo prodotto che diventerà il sistema operativo più diffuso nei servers di tutto il mondo. La misura euristica di questo interesse, verso le motivazioni che sostengono la produzione di software libero, la si può evincere facendo un esercizio nel WEB attraverso i motori di ricerca usando la chiave “open source organization motivation”. I blog, i forum, le rubriche, finanche le ricerche più o meno accreditate sono tantissime pur trattando un argomento molto specifico. Ne citiamo alcune selezionate in base ad una valutazione speditiva della loro significatività prediligendo quelle che hanno un qualche riferimento accademico, pubblicazioni di libri di editori importanti, citazioni in altri contesti:

- Motivation behind open source, Rahul Chaudhary, 2003 .
- The Economic Motivation of Open Source Software Dirk Riehle, SAP Research, 2007 .
- Motivation of software developers in open source projects, Hertel, G., Niedner, S. & Herrmann, 2003 .
- Toward an understanding of the motivation Open Source Software developers, Yunwen Ye, Kouichi Kishida, 2003 .
- Motivation and Open Source, Rikki KIte, 2010 .
- An empirical analysis of open source software developers' motivations and continuance intentions, Chorng-Guang Wu, James H. Gerlach, Clifford E. Young 2007 .
- Intrinsic Motivation of Open Content Contributors, Xiaoquan Zhang, Feng Zhu, 2006 .

- Exploring motivations for contributing to open source initiatives, Shaul Oreg, Oded Nov, 2008 .

- Drive - The Surprising Throught about what is motivating us, Daniel H. Pink, 2010.

Ciò che è interessante notare è che questo interesse rappresenta esso stesso un “challange mastery”, una sfida a spiegare ciò che ha tutta l'aria di essere un paradosso. È in definitiva esso stesso parte del fenomeno, conseguenza della capacità del software libero e dell'open source di contagiare anche altri campi de sapere. Mentre il software libero e l'open source vengono spiegati in maniera particolaristica questo stesso fenomeno si amplia e si generalizza ricomprendendo gli stessi attori che cercano di spiegarlo.

Un tentativo di comprendere il paradosso del software libero potrebbe essere quello di un'osservazione dell'osservazione secondo un approccio sistemico complesso e quindi chiederci cosa suscita un così grosso interesse attorno alle motivazioni dei volontari del software libero. Una direzione di indagine potrebbe portare a verificare quanto questo fenomeno possa essere posto sulla linea di continuità con la ricerca delle motivazioni nell'ambito organizzativo.

Nei primi anni del '900 è Frédérick W. Taylor è il primo ad occuparsi di motivazioni da un punto di vista estremamente burocratico, parcellizzato, strumentale agli scopi dell'organizzazione che da vita al movimento dell'organizzazione scientifica del lavoro incentrata sulle motivazioni economiche e sulle teorie del rinforzo. Negli anni venti e trenta la scuole delle relazioni Umane con Mayo, Roethlisberger e Dickons vi contrappone il bisogno morale e di socialità degli individui. Nel 1938 Chester Barnard (1970), alto dirigente della Bell Telephone Company, cerca di spiegare le "funzioni del dirigente" come mediazione tra le esigenze del lavoratore e quelle dell'organizzazione. Abrham Maslow (1973) dispone invece i bisogni in una scala di priorità da quelli fisiologici a quelli di sicurezza, stima e di auto-realizzazione e quindi i comportamenti attivati sono quelli contingenti a quella determinata condizione in cui si trova l'individuo in relazione con l'ambiente. Weiner parla di impegno e abilità come spiegazioni interne, fortuna e difficoltà come spiegazioni esterne. Spiegazioni interne ed esterne costituiscono i criteri in base ai quali decidere il proprio impegno. McClelland (1988) elabora la teoria della motivazione alla riuscita distinguendo tre diversi tipi di bisogni presenti in modo diverso nei diversi soggetti, in cui può prevalere l'uno o l'altro: riuscita potere e affiliazione. Il bisogno di successo è acquisito e non innato:

[...] è espressione del desiderio di fare le cose nel modo migliore, di conseguire risultati sempre migliori, di ricercare e misurarsi con situazioni di complessità crescente e di competere secondo un standard di eccellenza. (Maeran, 2002)

In sintesi le teorie sulle motivazioni e soddisfazioni al lavoro possono essere orientate al rinforzo piuttosto che agli assunti cognitivi, al contenuto piuttosto che al processo. Il rinforzo riguarda l'incentivo, gli assunti cognitivi riguardano la capacità di valutare in modo complesso. Il contenuto riguarda la componente oggettiva in grado di orientare i comportamenti, mentre il processo ne sottolinea il

modo (Maeran, 2002). In definitiva si tratta di spiegare, più che comprendere, il rapporto che si instaura tra soddisfazione, motivazione e scopi contestualmente a una qualche organizzazione. In questa tesi si cerca invece di spiegare un fenomeno produttivo auto-organizzato che sfugge agli strumenti di analisi nati per contesti diversi.

Ciò che si rischia di trascurare è la dimensione non necessariamente utilitaristica di questo fenomeno, rischio ancor maggiore nel momento in cui tale fenomeno ha forti ripercussioni sul mercato dell'informatica e l'informatica ha grosse ripercussioni sul mercato in genere. Rischia quindi di restare nell'ombra la dimensione culturale autonoma del software libero, con le sue peculiarità antropologiche incomprensibili senza una certa propensione a comprendere più che a spiegare. Lo spiegare può destare il sospetto di essere mossi da motivazioni a sfondo ideologico, volte ad "addomesticare" un agire umano non coerente con gli schemi di senso a cui l'economia ci ha abituato. Lo spiegare porta inevitabilmente a dotarci degli strumenti che conosciamo, mentre il comprendere cerca gli strumenti adeguati, di volta in volta diversi. Gli strumenti che l'economia ci mette a disposizione sono quelli che funzionano con le risorse scarse quindi tentiamo di spiegare il tutto attraverso questo paradigma. Se una risorsa è ritenuta abbondante da alcuni e scarsa da altri allora gli uni non riusciranno a comprendere l'agire degli altri e viceversa:

[...] Mise i piedi a terra con Cacambo al primo villaggio che gli si presentò. Alcuni ragazzi, coperti di un broccato d'oro tutto stracciato, giuocavano alle piastrelle all'entrata del borgo. I nostri due uomini dell'altro mondo s'occupavano ad osservarli; le loro piastrelle erano tonde, assai larghe, gialle, rosse, verdi, e gettavano uno splendore singolare; venne voglia ai viaggiatori di raccoglierne alcune, e videro ch'erano d'oro, di smeraldi, di rubini, la minor delle quali sarebbe stato il più grand'ornamento del trono del Mogol. - Senza dubbio, disse Candido, questi ragazzi sono i figli del re (erano popolani ndr) del paese, che giocano alle piastrelle. Apparve in quel momento il maestro del villaggio per ricondurli a scuola: - Ecco, dice Candido, il precettore della famiglia reale. Quei baroncelli abbandonaron tosto il giuoco, lasciando in terra le lor piastrelle e tutto ciò che aveva servito al lor divertimento. Candido le raccolse, corse dal precettore, e gliele presentò umilmente, facendogli intendere, a forza di cenni, che le loro altezze reali si erano dimenticate del loro oro e delle loro gemme. Il maestro del villaggio, sorridendo, le gettò per terra, guardò un momento la figura di Candido con stupore e continuò il suo cammino (Voltaire, 2010, p. 58)

Candido e Cocambo non comprendono l'agire del maestro del villaggio; il maestro del villaggio non comprende il comportamento di Candido e Cocambo. Ciò di cui ci si è resi conto cercando di indagare il fenomeno del software libero è che le risorse in gioco sono principalmente due: le competenze (skills) ed il software. È difficile individuarle in quanto si compenetrano. Le competenze sono quelle che producono software ma le competenze si esprimono attraverso il software. La loro distinzione può essere spiegata meglio ricorrendo alla teoria dei sistemi di Luhmann. Se consideriamo software e competenze come elementi dello stesso sistema è facile intuire come da un punto di vista auto-poietico il sistema nel suo complesso temporizza in modo diverso competenze e software. Il software si distrugge e ricrea di continuo in forme diverse1, le competenze vengono riprodotte in modo molto lento. Basti pensare al tempo necessario per formare un programmatore o un ingegnere informatico a livello di sistema di comunicazione, cioè sistema sociale, o al tempo che bisogna impiegare per apprendere continuamente nuove tecniche a seguito dell'auto-poiesi accelerata del software. Quindi si spiega il perché, dal punto di vista sistemico, il software è una risorsa abbondante mentre le competenze sono una risorsa scarsa.

Confondere lo scarso con l'abbondante significa tentare di vendere oro nel paese fantastico dell'Eldorado di Voltaire. Insistere sul livello micro e sui comportamenti in modo decontestualizzato significa non capire perché dei ragazzini giocano con delle piastrelle d'oro:

[...] Non capisco chi sono quei pazzi che vogliono scrivere, leggere e fare revisione di tutto quel codice senza ricevere nessuna ricompensa. (Rober L. Glass - Of Open Source, Gnu/Linux ... and Hype, 1999, p. 104)

L'ironia di Voltaire ci suggerisce questa risposta:

[...] dei ragazzi che giocano alle piastrelle (abbondanti – ndr).

Voltaire mette in evidenza i due termini del problema: L'aspetto “ludico” e l'aspetto “abbondanza/scarsità”. Entrambi questi aspetti non sono nuovi. Mariella Berra (2001), del resto ha già rilevato questa dimensione di abbondanza del software libero che, la stessa sociologa, indica come “informatica solidale”. Solidale va proprio inteso in senso sociologico, finanche meccanico per certi aspetti. In situazioni di abbondanza prevale il valore simbolico del bene che di per se non può essere scambiato se non in termini di un obbligazione reciproca. Nelle società di cacciatori raccoglitori la condivisione è il sistema migliore per

ottimizzare le risorse, se si uccide una preda e non si ha modo di conservarla, non costa nulla donarla, costa invece molto, in termini comunitari privarne i propri simili. Ma soprattutto il vantaggio più grosso e diretto è quello di consentire la sopravvivenza di un maggior numero di individui e quindi aumentare anche la proprie possibilità di sopravvivere. Cacciare costa molto in termini di energie e di competenze, ma una volta raggiunto il risultato, in mancanza di mercato, la scelta migliore è di investirlo in relazioni. Ma questo presuppone, nel caso del software libero, che il mercato del software non sia possibile o, in qualche misura, improbabile. Presuppone inoltre che il “dono” aumenti anche le proprie possibilità di “sopravvivenza professionale”.

Considerando il software libero come una peculiarità economica sostanziale anziché formale implica che non possa sussistere l'economia formale senza l'economia sostanziale (Polanyi, 1977). Il fatto che il software libero abbia grosse implicazioni dal punto di vista dell'economia formale non fa altro che confermare questa tesi. Polanyi (1977) distingue efficacemente tra economia del dono (Mauss, 2002) basata sull'abbondanza ed economia formale basata sulla scarsità.

Nella fattispecie del software libero è quindi più che plausibile il sussistere di una risorsa per sua natura abbondante utile a gestire risorse scarse. Il software è automazione di processi che veicolano informazione al fine di supportare le decisioni. Il software ha quindi la sua peculiarità nei processi decisionali, quindi trova la usa naturale utilità nei processi organizzativi e quindi anche nell'ambito dell'economia formale e non. Si pensi ad esempio quanto tempo risparmiamo facendo pagamenti on-line anziché fare la file allo sportello, quanto tempo risparmiamo spedendo una lettera con la posta elettronica anziché attraverso una busta e gli esempi sarebbero tantissimi. In particolare il software abbondante ottimizza molto bene una risorsa scarsissima che è il tempo. A questo punto ci basta solo dimostrare che il software è per sua natura abbondante mentre il fatto che il tempo sia una risorsa scarsa nelle società moderne non ha bisogno di troppe dimostrazioni.

Il software, o meglio la produzione di software ha in realtà dei costi altissimi e questo contraddice l'idea di abbondanza. Una volta prodotto non costa nulla riprodurlo e, meno che meno, distribuirlo attraverso la rete. Nello sviluppo software sono alti i costi di progettazione, se vi includiamo la stesura del codice, di fatto anche gli ambiente di sviluppo gestiscono, in termini tecnici informatici dei progetti, in generale con il termine progettazione includiamo l'attività progettuale in senso cognitivo[1]. Nell'industria automobilistica invece sono maggiori i costi di produzione e distribuzione per ovvi motivi. Inoltre la maggior parte di codice è dedicato a soluzioni personalizzate. Secondo una stima commissionata dal Ministero per l'Innovazione Tecnologica del governo italiano del 2003, dal 35% al 45% delle spese informatiche della pubblica amministrazione riguardano personalizzazioni, mentre il 10 - 20% riguarda pacchetti di base, il rimanente 35 - 55% riguarda la formazione e il mantenimento. In media le spese per la personalizzazione sono 2,9 volte quelle per le licenze. Questo significa che le licenze d'uso essendo codice replicato in quanto non personalizzato costituisca una fetta ancora minore in termini assoluti del codice scritto ad uso personalizzato. Sia i pacchetti di base che il codice personalizzato sono acquisiti

come closed source, con formule riconducibili al copyright. Inoltre la netta prevalenza di codice personalizzato è indice di scarsa efficienza in termini di scarsa riutilizzazione.

Molte soluzioni personalizzate sono necessariamente monolitiche[2], a volte solo per piccole differenze strutturali delle diverse amministrazioni pubbliche questi pacchetti non possono essere riutilizzati. Se il codice fosse aperto diverrebbe molto più semplice adattare il codice alle diverse peculiarità organizzative. Possiamo prendere ad esempio un sistema per la gestione degli automezzi (car sharing) in dotazione ad un'organizzazione sviluppato negoziando con gli utenti le sue funzionalità, come ad esempio la possibilità di forzare la prenotazione dell'auto da parte di alcuni referenti o la possibilità di utilizzare alcuni mezzi solo da parte di alcuni specifici profili di ruolo. Un software così personalizzato se chiuso, cristallizzato non potrà mai essere usato da un altra amministrazione con esigenze anche solo leggermente diverse, mentre se sviluppato con criteri open source potrà essere facilmente riusato, cambiando poche righe di codice, da una qualsiasi altra organizzazione con simili esigenze.

L'indagine commissionata del Ministero dell'Innovazione Tecnologia del 2003 conclude con il suggerimento per la pubblica amministrazione di optare per l'open source in funzione del riuso. Open source in questo senso viene inteso come caratteristica tecnica dell'open source da cui deriva indirettamente un'idea di qualità. Il riuso richiede una forte capacità del software di adattarsi ai diversi contesti organizzativi e questo avviene più facilmente attraverso l'open source. La spesa pubblica nel settore informatico è significativa per i nostri scopi in quanto rappresentativa del mercato nel suo complesso considerato che in Italia, nel 2003 la spesa totale in software per la pubblica amministrazione è stata di 45 milioni di euro. Spesso si parla di software libero e open source come assurdo economico ma a guardar bene ciò che risulta difficile spiegare è la chiusura del codice, la richiesta di brevetti, la pretesa di diritti d'autore su prodotti molto personalizzati. Si fatica ad immaginare come il software di gestione degli automezzi, di cui si è fatto l'esempio più sopra, possa essere adeguata per una qualsiasi altra organizzazione.

Non vi è garanzia che chiudendo il codice se ne possano derivare dei vantaggi in termini di dipendenza del committente da chi ha sviluppato il software, il cosiddetto lock-in. I fattori che intervengono sono molti. In prima istanza la reputazione. L'ennesima volta che il “locker”, o “pusher” nel gergo del software libero, chiederà una cifra “esagerata” per adattare il “suo” prodotto a

nuove esigenze del cliente il cliente tenterà di svincolarsi da questo ricatto continuo, e comunque tenterà di rivolgersi a qualcun altro per ulteriori esigenze. A questa minore garanzia della chiusura del software inoltre si affianca una diversa idea di fare “business”. Se il produttore di software distribuisce il suo prodotto in forma open ha maggiori possibilità che lo stesso possa essere eventualmente adattato e adeguato anche in-house (autonomamente dal cliente). Il fatto che lo stesso software abbia maggiore possibilità di essere mantenuto nel tempo gioca a favore della sua reputazione e quindi della reputazione di chi lo ha prodotto. Il fatto che sia distribuibile in fine non fa che aumentare le probabilità che chi ha prodotto quel dato software venga interpellato per adeguarlo, mantenerlo o implementarlo ulteriormente.

Note

1. ↑ diversamente da come fa invece la scuola di ingegnerizzazione del software che si rifà al modello top-down, che vedremo più avanti distingue nettamente la progettazione dalla fase di sviluppo.
2. ↑ contengono solo codice sviluppato appositamente per quella soluzione.

Ciò che resta da chiedersi è se ad un livello più generale questo modo di agire nel mercato da parte del software libero non finisca per diminuire il surplus complessivo. Se si ragiona in termini di risorse scarse ciò che l'economia classica insegna è che il prezzo si determina dall'incontro della curva di domanda con la curva di offerta:

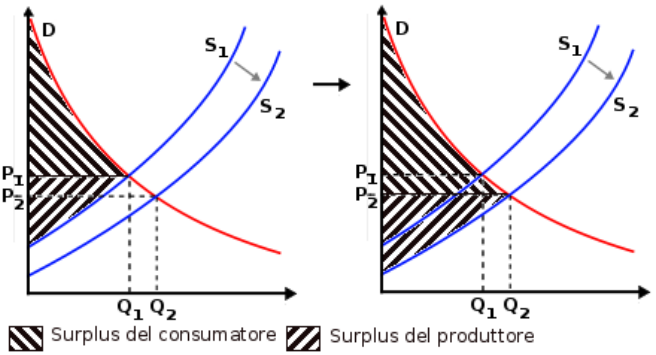

Fig. 1 Aumenta sia il surplus del consumatore che del produttore. Quindi aumenta il mercato informatico nel suo complesso

Nel primo caso, prima dello spostamento della curva di offerta verso destra, sussiste un mercato dell'informatica dominato dal software, mentre nel secondo caso la produzione di software viene delegata all'economia informale che sostiene il mercato informatico composto in maniera maggiore da competenze e hardware. Questo significa che in termini di investimenti conviene ai produttori di hardware, ed ai freelance[1] sostenere il software libero. Sinteticamente: (hw + sw) < (hw + skills)[2] Se si azzera il costo del software la curva dell'offerta di tecnologia informatica si sposta verso destra aumentando il surplus complessivo dato dalla somma tra surplus del consumatore e surplus del produttore. Questo non ha nulla di straordinario, vale per qualsiasi bene. Se si azzera il costo dello zucchero aumenta il mercato delle torte. La differenza sta essenzialmente nel fatto che il costo del software è di fatto azzerabile come dimostra l'esperienza storica del software libero, mentre l'azzeramento del costo dello zucchero non si è mai dato. Il software, ed in particolare il costo delle licenze concorre ai costi complessivi della tecnologia informatica assieme ad altri fattori, in particolare l'hardware e le prestazioni specialistiche. In pratica se si azzera il costo del software aumenta la domanda di hardware e competenze.

Questo è il motivo per cui IBM, tra i maggiori produttori di hardware, investe nel software libero; così come Oracle che detiene un notevole indotto attorno alla formazione e alla consulenza specialistica e così molti altri. Ciò che ne risulta in definitiva è che il ruolo delle licenze d'uso nel mercato informatico agisca come un'imposta sull'hardware e sulle competenze o una barriera doganale che limita il surplus complessivo con l'eccezione che la raccolta “erariale” non viene fatta da agenzie governative ma da concessionari, cioè dai rivenditori per conto di software house per lo più statunitensi.

Note

1. ↑ Liberi professionisti che programmano su commitenza.
2. ↑ Hw: Hardware; sw: software; skills: competenze.

Da questa prospettiva, che necessiterebbe di una maggiore ricerca empirica, quello che ne risulta è che non sussiste tanto un'idea di monopolio Microsoft quanto piuttosto un'idea di tassa sull'hardware e sulle competenze che riduce il surplus complessivo e quindi limita il dispiegarsi del mercato. Un esempio concreto di tassa sull'hardware trova la sua migliore espressione con il fatto che quando si acquista un computer si deve acquistare anche il sistema operativo Windows, anche se lo si vuole sostituire con un altro sistema operativo. Le procedure per chiedere il rimborso sono molto complesse e comportano istruttorie legali particolarmente lunghe. Non va dimenticato poi che comunque sussiste, ed è ampiamente documentato la reale di produzione di software in contesti di economia sostanziale coerentemente con la teoria di Polanyi secondo la quale l'economia sostanziale sostiene l'economia formale.

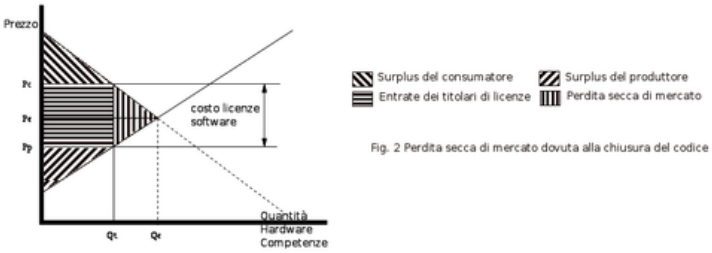

Fig. 2 Perdita secca di mercato dovuta alla chiusura del codice

Allo stesso modo l'OPEC non regala automobili in modo da aumentare i consumi di benzina. Non si tratta nella visione del software libero di un'analogia plausibile. Il software libero, diversamente dalle automobile non è una risorsa limitata perché ri-produrlo non costa nulla. Costa molto svilupparlo. La questione sta tutta qui, per svilupparlo ci vogliono competenze specialistica scarse che costano molto, per ri-produrlo e distribuirlo attraverso la rete i costi sono prossimi allo zero. Lo sviluppo del software è una risorsa scarsa, ma il software di per sé è una risorsa abbondante. Lo sviluppo del software ha costi alti solo se lo immaginiamo contestualmente ad una organizzazione con una sede, che ha costi di gestioni e che deve pagare dei programmatori, nonché coordinarli e via dicendo. Se invece si pensa che tutto questo può avvenire attraverso l'impegno sporadico di molti appassionati, studenti universitari che vogliono misurarsi e fare esperienza, programmatori che vogliono mostrare ciò che sanno fare e allungare il proprio curriculum, la situazione cambia radicalmente. Una survey del 2003 MIT Sloan School of Management[1] che ha riguardato 648 (di cui il 60% non remunarati) sviluppatori per 287 progetti FOSS (Free and Open Source Software) contrariamente a quanto si riteneva, e cioè che le motivazioni principali a

sviluppare programmi fossero dovute al miglioramento e all'avanzamento in carriera, ha invece rilevato come prevalenti le motivazioni intrinseche legate al diletto e alla gratificazione derivante dal lavoro creativo, lo stimolo dei bisogni degli utenti (altruismo), stimolazione intellettuale derivante dallo scrivere codice nonché il miglioramento delle proprie competenze. Nel 2005 Sandeep Krishnamurthy della University of Washington ha invece evidenziato come la certificazione in tecnologia open source è meglio spendibile nel mercato delle consulenze e del lavoro in genere. Questo non fa altro che confermare l'ampliarsi del mercato nel suo complesso al venir meno delle royalties sul software chiuso.

Note

1. ↑ Karim R. Lakhani, Robert G Wolf, MIT Sloan School of Management The Boston Consulting Group

Per contro nella rete si possono facilmente trovare testimonianze di tutt'altro tenore:

[...] il mio scopo è sempre stato quello di studiare soluzioni in modo artigianale, e molto mirate per un cliente, ma poi...quando raggiungo il software finale, posso rivenderlo in serie, magari con qualche ritocco qua e là. Il mio software di punta, Winshow Planning, è stato realizzato ad-hoc per un'agenzia di spettacolo a Cinisello Balsamo, ma poi è stato rivenduto negli anni seguenti a 15-20 altre agenzie sparse un po' ovunque. Questo secondo me è il punto: studiare e progettare in modo artigianale, e produrre su larga scala solo in un secondo momento. (http://blogs.ugidotnet.org/ 15 settembre 2010).

In realtà ciò che questo freelance rende evidente è che l'industrializzazione del software a codice chiuso non è così scontata, rappresenta piuttosto una sfida, un'ideale verso cui tendere che trova la sua effettiva realizzazione solo in ambiti ben precisi in particolare nei sistemi operativi, i pacchetti per ufficio che comprendono foglio di calcolo, videoscrittura, editori di presentazioni e tutti quei sistemi che rispondono ad esigenze generalizzate. Nel caso riportato si parla di un prodotto di punta rivenduto in un numero esiguo di copie, peraltro ritoccate (personalizzate) nel corso di anni e ciò conferma a maggior ragione che il software non ha un suo destino industriale così scontato. Chiaramente esistono situazioni in cui il paradigma industriale ha raccolto successi economici evidenti primo fra tutti quello Microsoft. Questa osservazione sopra riportata non ha quindi lo scopo di dimostrare l'improbabilità della produzione industriale del software, di cui peraltro molti aspetti sono operanti anche nel software libero, ma semplicemente evidenziare che tale paradigma non è generalizzabile. Nella fornitura software della pubblica amministrazione italiana il dato economico evidenzia una netta prevalenza del software personalizzato. Tale divario è senza dubbio ancora maggiore, in termini di quantità di istruzioni scritte, in quanto il software personalizzato è costituito da pezzi singoli, quello industriale da pezzi replicati. Inoltre, secondo la teoria dei sistemi di Luhmann la condizione di auto-poiesi non riguarda tanto la replica degli elementi del sistema ma la loro temporizzazione, cioè il loro succedersi in forme diverse. Queste diverse visioni del mondo operanti a livello di sviluppatori on demand come quelli che sviluppano con tecnologia open o come l'autore di “Winshow Planning”[1] che sviluppa con tecnologia closed riflettono la stessa dialettica che si dispiega a livelli scientifici specializzati e che riguardano scuole di pensiero e paradigmi:

[...] Negli ultimi anni, il settore informatico è stato caratterizzato da un notevole incremento della produzione del software[2] *nei riguardi dell'hardware, e questo si riflette sui costi di un sistema informatico. Con queste premesse risulta necessario evitare di produrre software in base alle esperienze e/o iniziative del programmatore: il processo di produzione del software non può essere un processo di tipo artigianale (negli anni '60 il programmatore usava mille trucchi per risparmiare memoria!), ma deve servirsi di metodologie e tecniche sistematiche di progettazione e programmazione con fissati parametri di qualità e in maniera standard. La software engineering (ingegneria del software) è la branca dell'ingegneria informatica che raccoglie il patrimonio di metodi e tecniche per la produzione del software. Il processo di industrializzazione del software, introduce metodi e standard, riduce i margini di creatività e personalismo e consente i seguenti requisiti: buon livello qualitativo; produttività medio-alta; impiego di personale non troppo specializzato (la specializzazione è fornita dallo standard); riduzione sensibile del costo del prodotto … Ogni modulo deve possedere un singolo e ben precisato compito … Nell'ambito di un progetto software l'analisi richiede la capacità di acquisire le informazioni necessarie alla comprensione e di strutturarle in un modello che esprima, in un linguaggio adeguato, una rappresentazione coerente e completa di cosa deve fare .. Ma tra gli aspetti o le proprietà degli algoritmi da valutare con più attenzione sicuramente il più importante è proprio il progetto che si presenta come onerosa attività intellettuale (molto più onerosa di quella di esprimere l'algoritmo con un linguaggio di programmazione) che richiede creatività ed intuito ...* (Alla scoperta dei fondamenti dell'informatica - Chianese, Moscato, Picariello, 2008, p. 114)

Questo estratto mette in evidenza un'esigenza organizzativa diffusa, come si vedrà più avanti, cioè quella di dare una collocazione sinottica e prevedibile attorno a quella che è un'attività essenziale dello sviluppo software e cioè la programmazione. Questa necessità viene ripresa ogniqualvolta si voglia affrontare il problema dell'industrializzazione del software. Diventa quindi centrale, un nodo indispensabile da risolvere:

[...] lo sviluppo software è ... lento e costoso, e genera prodotti che contengono seri problemi che causano problemi di usabilità, affidabilità, performance e sicurezza. All'inizio di questo capitolo, l'industrializzazione è stata definita come un metodo per aumentare efficienza e qualità e ridurre i costi implementando metodi standardizzati e altamente produttivi. Gli obiettivi di qualsiasi progetto software possono essere classificati in qualità, quantità, tempo e costi. Harry M. Sneed dipinge la loro interazione come il quadrilatero del

diavolo, nel quale i quattro fattori sono in una relazione antagonista. In relazione alla disponibilità produttiva di una organizzazione efficace è limitante e non può soddisfare tutte le necessità, qualcosa deve essere sacrificato. Per esempio, maggiore lavoro di alta qualità causerà più costi e maggiori tempi di sviluppo. Tuttavia, applicando i metodi industriali, qualità e complessità del prodotto possono anche essere aumentate pur riducendo costi e tempi di produzione.

Molti sforzi sono stati fatti per applicare tali metodi allo sviluppo software. In riferimento al precedente capitolo, i principi industriali chiave ora possono essere trovati nel campo dell'ingegnerizzazione del software. La specializzazione è rappresentata dal Software Product Line (SPL – sviluppo di prodotti in linea), standardizzazione e sistematico riuso può essere trovato nello Component Based Devolopment (CBD - sviluppo basato su componenti) e il Model Driven Engineering (MDE – ingegnerizzazione orientata ai modelli). Sfortunatamente, il più importante concetto, specializzazione, è stato incluso per ultimo. Sennonché semplicemente, per mancanza di scopi ben definiti, lo sviluppo basato sui componenti e la sviluppo di prodotti in linea sembrano aver fallito al loro inizio. Solo recentemente tutti i concetti sono stati posti in essere e possono essere usati per sopportare l'industrializzazione … l'ultimo e più importante concetto è quello di sviluppo di prodotti in linea … è molto difficile se non impossibili concepire come i meccanismi di riuso e automazione possano essere implementati in un contesto arbitrario … ciò richiede di separare lo sviluppo di prodotti dalla sviluppo di prodotti in linea. Il primo contiene lo sviluppo di prodotti in senso stretto, mentre l'ultimo contiene l'assetto richiesto .. concentrando gli sforzi su uno scopo chiaro e delimitato, l'assetto produttivo può essere più potente come, per esempio, il riuso di componenti, ambienti e architetture. Tuttavia la specializzazione di per se non consentirebbe alcuna diversificazione così come viene richiesto dai diversi committenti. La produzione di software in linea inoltre identifica funzionalità ricorrenti e i punti di variazione per definire piattaforme e architetture comuni nelle quali le specifiche esigenze dei clienti possono essere prese in considerazione. Questo concetto, in altri settori dell'industria, è anche conosciuto come personalizzazione di massa … durante lo sviluppo di prodotti vengono definiti nuovi assetti che ritornano come feed-back alla produzione in linea. (Minich, Muehlbauer, Wentzel, 2009)

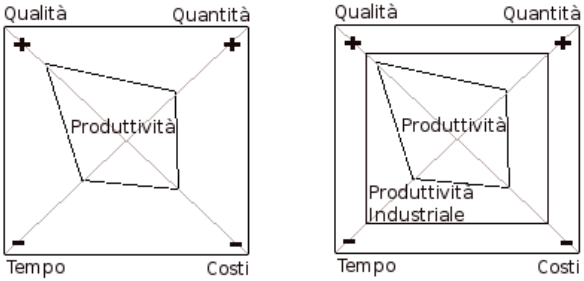

Fig. 3 Devil's Square di Sneed: compromesso dimensionale in contrapposizione alla produzione industriale

In sintesi Minich, Muehlbauer e Wentzel individuano un modello a due fasi:

- la produzione in linea che assorbe la specializzazione;
- la produzione di software in senso stretto che permette la personalizzazione.

La differenza sostanziale con il modello precedente di Chianese, Moscato, Picariello, consiste nel fatto che la creatività ed il “personalismo” non sono visti come problematici nel senso della loro esistenza, ma nel senso della loro collocazione. In questo ultimo modello il problema della non “industrializzazione della creatività” pone l'accento non sul “personalismo”, ma sulla “arbitrarietà”. Il primo è quindi un modello normativo, il secondo un modello dialettico. Quest'ultimo modello non esaurisce tutta la questione dell'arbitrarietà, semmai tenta di governarla. Non tutta la dimensione relativista viene delegata alla produzione di software in senso stretto perché la produzione in linea necessita di feed-back o, detto in altri termini, necessita di riflessività per la sua incapacità predittiva. In definitiva potremmo dire che sussistono due modelli ingegneristici del software: uno sistemico e sinottico che esclude la creatività; l'altro sistemico e autoregolato orientato a governare la creatività.

Il modello ingegneristico che si è definito dialettico evidenzia un aspetto molto importante ai fini del software libero che è la riflessività. In qualche modo tiene conto “dell'essere” in contrapposizione al “dover essere”. Minich, Muehlbauer e Wentzel rappresentano questo modello come due livelli, produzione in linea e produzione in senso stretto, cioè come due livelli che dialogano tra loro, che si scambiano informazioni in base alle quali adattare la produzione. Ed è proprio la dimensione dialettica che in definitiva caratterizza il software libero che non si limita ad una grammatica del funzionamento del prodotto “industriale”

(funziona/non funziona, gestisce/non gestisce, efficiente/non efficiente, lento/veloce, fa/non fa), ma ricomprende anche la sua essenza “intima” cioè il codice, e nella fattispecie il codice aperto. Addirittura la grammatica impiegata nella riflessività ricomprende lo stesso codice (show me the code):

[...] un modo fantastico di vedere i propri cambiamenti accettati da altri è quello che si chiama «show me the code», cioè si fa vedere che un cambiamento di cui si sta discutendo da secoli è possibile, una volta che c'è l'evidenza in termini di codice che il cambiamento è possibile è più facile che collettivamente si decida che quella è la via giusta. (Intervista Stefano Zacchiroli del 17 agosto 2010).

Nel software libero questo modello ingegneristico dell'industrializzazione del software accentua maggiormente l'aspetto dialettico-riflessivo su tre dinamiche. Una prima dinamica è l'ampliamento della grammatica che come abbiamo già visto arriva a ricomprendere ricorsivamente il codice stesso; la seconda è l'ampliamento della stratificazione in cui si dispiega la riflessività; la terza è la contrazione del tempo.

La stratificazione si riferisce ai livelli tra cui si verificano eventi riflessivi. Mentre nel modello di Minich, Muehlbauer e Wentzel si prendono in considerazione due livelli, produzione in linea e produzione in senso stretto, che dialogano tra loro, nel software libero i livelli implicati riguardano i due livelli precedenti ed inoltre il livello “utenti finali”. Inoltre questi livelli si intersecano, uno stesso attore può appartenere a più livelli, o meglio, può passare da un livello ad un altro. La produzione in linea non si riferisce alla semplice replica di software, il tutto si ridurrebbe alla semplice copiatura di programmi che è un'attività che non necessita di sofisticate strategie industriali, ma si riferisce alla produzione di oggetti specializzati con scopi generalizzati riusabili nei diversi prodotti personalizzati, costituiti dalla produzione in senso stretto e destinati a quello che si chiama utente finale.

Poniamo quindi ad esempio il caso in cui un programmatore si trovi sul livello di produzione in senso stretto e che quindi sviluppa software a partire da oggetti già costruiti dalla produzione in linea. In questo caso può facilmente accadere che un “oggetto” prodotto in linea non risponda esattamente alle esigenze di un programmatore in senso stretto per sviluppare un prodotto personalizzato. In questo caso è possibile nell'ambito del software libero, anzi frequente, che egli decida di modificare l'oggetto stesso e che quindi sottoponga

quelle modifiche stesse al core degli sviluppatori in linea per essere approvato ed incluso nelle versioni ufficiali, in questo caso l'implementazione dell'oggetto avviene già a livello di produzione in senso stretto, mentre il livello di produzione in linea assume il compito di approvare o meno tale modifica per poi nuovamente renderla disponibile a livello di produzione in senso stretto.

Ma vi è un altro elemento che questo modello non tiene in considerazione. Matthias e Harriehausen, che pure pongono il riuso del prodotto tra i criteri di industrializzazione del software, tengono conto nel loro modello solo del riuso orizzontale. È il livello della produzione in linea che si preoccupa di riutilizzare prodotti da destinare alla programmazione in senso stretto, cioè quella personalizzata, non viene contemplato il riuso orizzontale sul livello della programmazione in senso stretto come invece è possibile nel caso di sorgente aperto e libero da copyright.

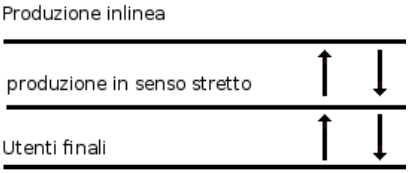

Fig. 4 Modello Matthias Minich,
B. Harriehausen-Muehlbauer e C. Wentzel

Tornando all'esempio del servizio per la gestione delle auto aziendali in un'organizzazione, sviluppato on demand (come un vestito fatto su misura), se chiuso non può essere dato ad un'altra organizzazione che magari ha, metaforicamente, taglia diversa e una diversa cultura aziendale, ma se aperto una certa quantità di codice può essere recuperato, magari quasi tutto o si potrebbe trattare di ritoccare solo qualche istruzione. A tale proposito una survey di Manuel Sojer e Joachim Henkel della Technische Universität München (TUM), che ha riguardato 686 sviluppatori open source, rivela la maggiore propensione a riusare codice già scritto da parte di chi ha un'ampia rete di conoscenze negli ambienti open source, quindi dispone di capitale sociale (Coleman, 2005) chi ha partecipato a più progetti open source. Ma questo è proprio uno degli scopi dell'industrializzazione, cioè il riuso:

[...] Le chiavi concettuali di tali metodi (industrializzazione ed integrazione del software) possono essere sintetizzate in specializzazione & standardizzazione e riuso sistematico e automazione. (Minich, Muehlbauer, Wentzel, 2009)

L'inclusione del feed-back dell'utente finale non è una prerogativa esclusiva del software libero ma la differenza sostanziale consiste nel fatto che in questo caso i canali sono molto più accessibili e fluidi. Il software proprietario non può permettersi di pubblicare la lista dei difetti del suo software per ovvi motivi di immagine. Quindi necessariamente il feed-back dell'utente finale deve essere intercettato attraverso la mediazione formale e riservata introducendo così notevoli attriti. Sempre in considerazione del fatto che il feed-back deve avvenire in modo controllato e mediato, causa una maggiore dilatazione del tempo, e per contro una maggiore contrazione del tempo nel contesto del software libero, quindi una maggiore temporizzazione e auto-poiesi come conseguenza della riflessività. Ognuno degli attori vede gli altri due. Inoltre il feed-back dell'utente verso la produzione in linea riguarda oltre la segnalazione di bugs, anche donazioni, attività di promozione nelle diverse associazioni, come ad esempio i LUG (Linux User Group) diffusi in tutto il mondo, la stesura e traduzione di manuali d'uso. Gli utenti possono essere anche organizzazioni che finanziano specifici progetti a volte anche governative come il governo austriaco che finanzia per intero il progetto Proxmox, un programma di virtualizzazione[3]. Il contributo per il software libero in questo caso è ovvio, si tratta di finanziamenti, per il governo austriaco il feed-back consiste nel risparmio dei costi per la virtualizzazione dei suoi sistemi informatici.

Fig. 5 Modello dialettico nella produzione di software libero

Questi modelli spiegano perché le argomentazioni che pone il software libero non si esauriscano semplicemente con il monopolio ma coinvolgono visioni molto più complesse e articolate che riguardano concezioni etiche, di libertà e diritti negati. C'è fondamentalmente un agire etico non solo coerente con gli assunti economici di base che qui si sono esposti ma che sostiene il settore informatico nel suo complesso. Il problema da un lato non si esaurisce con l'eliminazione del monopolio informatico e dall'altro non si esaurisce con la soluzione della questione morale che pone Richard Stallman:

[...] numero zero, la libertà di utilizzare qualunque software, a qualunque scopo. Numero uno, la libertà di studiare il funzionamento del programma, e di cambiarlo a piacimento. Ovviamente il presupposto di questa libertà è avere accesso al codice sorgente. Numero due, la libertà di ridistribuire copie di qualunque software a chi ti pare. Numero tre, la libertà di distribuire copie del tuo software modificato a chi ti pare. Il tutto in una visione altruistica, cioè per aiutare chi ti sta intorno … "

Il software libero e l'open source in genere non hanno il semplice compito di fare concorrenza a Microsoft per contenere le esternalità del monopolio; non ne hanno né lo scopo (mission), né la funzione, intesa come causa di effetti sistemici emergenti, né hanno lo scopo di calmierare i prezzi. In secondo luogo molti degli attori coinvolti non sono necessariamente moralmente impegnati, si può partecipare per scelta economica o semplicemente per divertirsi. Per contro c'è, come abbiamo visto, una concezione di modo di produrre industriale legata al software chiuso ancora per certi aspetti legata a concetti di parcellizzazione del lavoro, di controllo dei processi, di mortificazione o negazione della creatività, di alienazione e di separazione dell'uomo dal suo manufatto. C'è La pressione normativa sugli assetti organizzativi. Non di rado sono le persone e le organizzazioni a doversi adeguare ai processi di automazione e quando ciò accade è un segnale evidente del fatto che ci stiamo adeguando alle esigenze di industrializzazione del software intesa in modo parcellizzato e burocratico.

[...] La bruttezza vera non sta negli oggetti tecnologici né, secondo la metafisica di Fedro, essa dipende dai soggetti della tecnologia, cioè da chi produce o da chi la usa, ma sta nel rapporto tra chi produce la tecnologia e le cose prodotte, il quale determina poi un analogo rapporto tra chi usa la tecnologia e le cose usate. (Pirsig, 1992, p. 281)

Pirsig mette in evidenza un genere di isomoforfismo particolare, non quello tra organizzazioni che svolgono la stessa funzione, come possono essere isomorfe le industrie automobilistiche, le amministrazioni comunali, i governi. Si tratta di un isomorfismo non orizzontale ma verticale, in cui l'industria del software o chi produce il software in questo caso, determina un rapporto analogo con il prodotto da parte di chi lo utilizza. Quindi l'industria software necessita di industrializzare la produzione e pertanto necessità di utenti a loro volta industrializzati e soprattutto uguali tra loro. È facile immaginare come la cultura organizzativa si rifletta sul software prodotto. Ad esempio una software house rigidamente gerarchica, che separa nettamente la progettazione dallo sviluppo, produrrà

software strutturato gerarchicamente il cui utilizzo richiederà una profilazione[4] gerarchica degli utenti, la quale dovrà corrispondere ad un organigramma gerarchico della stessa organizzazione che lo adotta.

La creatività è ciò che consente di personalizzare, umanizzare la tecnologia di adeguare l'informatica ai bisogni dell'uomo, diversamente il dramma appare in tutta la sua evidenza. Il rischio è il “riduttivismo informatico” che delegittima le eccezioni per il semplice fatto che non riesce a gestirle in serie. La creatività viene percepita allora come una caratteristica pericolosa perché imprevedibile. Cosi Gian Antonio Gilli su Platone (1988, pg 63):

[...] se il modello della repubblica è sfavorevole ai portatori di téchne, quello proposto nelle Leggi è di gran lunga più duro. Opera insieme assorta e disincantata, frutto della tarda vecchiaia di Platone, Le leggi mostra un apprezzamento della divisione del lavoro come fonte di controllo sociale assai minore rispetto a quello mostrato nella repubblica, e una ridotta fiducia nei processi di interiorizzazione della norma da parte di ogni individuo. Il minor affidamento su queste forme di controllo potrebbe spiegare il ricorso a modalità più dure di controllo diretto sui portatori di téchne

Salvatore La Mendola (Comunicare interagendo, 2007), riprendendo da Gian Antonio Gilli spiega come il portatore di téchne non agisca secondo i principi con cui funziona la società, in modo utilitaristico, la società nasce contro i technai. Le loro capacità (creatività) devono essere assoggettate e questo avviene, nella società moderna, nelle organizzazioni e nelle aziende dove i portatori di téchne vengono affiancati a venditori perché la mediazione risolve la contraddizione tra la loro inadeguatezza sociale e la loro utilità. È ricorrente nella storia politica e organizzativa questa urgenza di controllo della creatività, oltre a questo bisogno di ribadirne la subordinazione se non l'inferiorità:

[...] è proprio il progetto che si presenta come onerosa attività intellettuale (molto più onerosa di quella di esprimere l'algoritmo con un linguaggio di programmazione) che richiede creatività ed intuito. (Chianese, Moscato, Picariello p. 114)

È la stessa urgenza che Platone attribuisce al Faraona Thaumus:

[...] O peritissimo Theuth, altro è colui che è capace di generare i ritrovati della téchne, altro è colui che è capace di giudicare quali esiti di danno o di utilità essa abbia per coloro che se ne serviranno. (Gilli, 1988, p. 45)

Per quale ragione dovrebbe essere così necessario in un manuale di ingegnerizzazione del software ribadire un tale monito? Di fatto, la legge di Platone e Alla scoperta dei fondamenti dell'informatica di Chianese, Moscato e Picariello, hanno la stessa necessità di ribadire un ordine e con esso una gerarchie di funzioni e di ruoli che non può essere confusa al fine di raggiungere un bene superiore, sia esso il governo di uno stato o il governo di un processo informatico. La minaccia a questo ordine ideale, stabilito a priori, non è data storicamente, secondo Gian Antonio Gilli, da forme di rivendicazioni dirette, corporativiste o sindacali, ma dall'energia che il portatore di téchne riesce a convogliare attraverso la sua attività in modo naturale e che minaccia un ordine sociale necessario e da realizzare. La contraddizione che il portatore di téchne porta con se è documentata dal disprezzo dei letterati e della politica nei confronti degli artigiani dell'antica Roma che nonostante questo, come dimostrano gli scavi archeologici (Dante, 2002), riescono ad acquisire uno status economico elevato nonostante il basso prestigio sociale, esprimendo ciò che Gerhard Lensky chiama squilibrio di status (Lensky, 1981).

La minaccia è la perdita di controllo che sul piano politico è dovuta alla competizione sul consenso tra l'autore dell'opera d'arte e la dimensione sacra che la stessa opere d'arte esprime e che ha la funzione di tenere unita la società (Durkheim, 1962). Sul piano informatico il problema del controllo verte sul potere informale (Crozier, 1978), cioè sulla notevole potenziale capacità dei programmatori di controllare gli spazi di incertezza in un campo che, come si vedrà più avanti, è ad alto rischio. Sempre secondo Crozier la “one best way” descritta ne L'organizzazione scientifico del lavoro da Edward W. Taylor (1974) è irrealizzabile in senso generale, porta al paradosso che l'eliminazione della discrezionalità minerebbe il senso stesso della gerarchia. Se in senso generale è difficile, se non impossibile, eliminare la discrezionalità a maggior ragione ciò diviene fortemente improbabile nel settore informatico. Il rischio che si creino situazioni di dipendenza dal technai è però molto alto. Se un'organizzazione investe su di un prodotto informatico dei capitali non può convivere con l'incertezza che il ritorno di tali investimenti possa dipendere dalla “creativita arbitraria” di un singolo programmatore. Ma nel caso informatico sussiste un altro paradosso oltre al fatto che l'eliminazione della discrezionalità porta alla perdita di senso della gerarchia (Crozier, 1978), in più vi è il fatto che è proprio il criterio della licenza d'uso che minaccia l'appropriazione del copyright da parte del programmatore e in generale di controllare gli spazzi di incertezza. Anche se le clausole contrattuali possono prevenire questa eventualità diviene estremamente costoso riuscire a controllare che gli algoritmi scritti nell'ambito

dell'organizzazione non vengano ceduti o sfruttati dal programmatore stesso in concorrenza con lo stesso datore di lavoro. Questo è vero in molte realtà produttive, ma il settore informatico è particolarmente esposto a questo rischio proprio per la possibilità di spostare facilmente prodotti software o pezzi di programmi attraverso supporti di memoria o la rete. A tale proposito Nikolai Bezroukov Fairleigb della Dickinson University (NJ) nel 1999 osservava come il 38% degli sviluppatori del kernel Gnu/Linux scrivessero codice per il progetto GNU/linux durante l'orario di lavoro.

L'affermazione che sia il progetto che si presenta come onerosa attività intellettuale e non l'esprimere l'algoritmo con un linguaggio di programmazione contiene quattro messaggi importanti:

- separa il concetto di progettazione da quello di programmazione;
- ridefinisce il programmare come l'esprimere algoritmi e di conseguenza ridefinisce il ruolo del programmatore come colui che esprime, cioè si limita a tradurre in termini informatici un progetto già ben definito;
- pone l'attività intellettuale come prerogativa della progettazione;
- ribadisce l'ordine gerarchico in cui il programmatore (espressore di algoritmi) viene subordinato al progettista.

Come quando si mette in produzione un software, la questione a questo punto diviene: funziona? Quindi: funziona questo modello organizzativo? Purtroppo la risposta a questa domanda può essere solo dialettica, la ricerca su questi aspetti organizzativi è scarsa se non nulla. Vediamo cosa succede al di fuori, quando si utilizzano i prodotti informatici, ne facciamo esperienza tutti i giorni, ma non sappiamo in che rapporto stanno questi modi di produrre software con l'utente finale in termini di ansia che si esprime come irritabilità, dolori di testa, difficoltà ad imparare ad usare il computer sino al rifiuto totale della tecnologia (Maeran, 2002, p. 278). Non sappiamo in che rapporto stanno con il tecno-stress, dove gli stressori più frequenti sono i crash della macchina, soprattutto se associati a perdita di dati, dimenticare le password, la lunghezza dei tempi di attesa di caricamento del software ed ingenerale tutti i tempi di attesa (De Carlo, 2004, vol. IV, p.86). Quindi nello specifico contesto di produzione software è davvero possibile separare così agevolmente la progettazione dalla programmazione, sapendo che altra causa di patologie organizzative è l'ambiguità dei ruoli (De Carlo, 2004, vol. IV, p.84)? Quanto è il software che non fa quanto

promette? E quanto quello prodotto con certi criteri piuttosto che con altri? O più semplicemente: Quanto il modo di produrre software influisce sul suo grado di usabilità, di ergonomia e quanto mantiene le promesse? E, non meno importante, sappiamo quanto la pubblica amministrazione spende per adottare software, ma non sappiamo quanto di questo software venga poi effettivamente messo in produzione. Non è facile rispondere a queste domande, e considerata la rilevanza socio-economica di questi argomenti non è sufficiente guardare alle patologie organizzative, allo stress, al sovraccarico, alle ambiguità di ruolo, ai giochi di potere, finanche al mobbing in maniera decontestualizzata dal modo di produrre tecnologia che necessariamente si riverbera nei contesti lavorativi dove viene applicata.

Ai nostri scopi attuali diviene importante rilevare che i modelli che si dispiegano nella produzione di software libero sono per alcuni aspetti diversi. Innanzi tutto, la differenza sostanziale, è che proprio il programmatore ha un ruolo centrale. Il caso più significativo è quello di Linus Torvalds iniziatore del sistema GNU/Linux, Andrew Tridgell iniziatore del progetto SAMBA[5], lo stesso Richard Stallman che ha programmato EMACS[6] e il debuger di Gnu/Linux, Miguel de Icaza iniziatore di Gnome[7], Rasmus Lerndorf iniziatore di PHP che è oggi il linguaggio WEB più utilizzato in assoluto. La lista potrebbe essere ovviamente molto più lunga, ma ciò che è importante notare è che si tratta di programmatori che avendo dimostrando competenze notevoli nella programmazione, ed avendo "creato" strumenti e software largamente diffusi in tutto il mondo hanno assunto ruoli importanti in molte compagnie, come google o yahoo, università, organizzazioni e via dicendo. Nell'ambito del software libero e dell'open source in genere ciò che occorre per far carriera è programmare e dimostrare cosa si riesce a fare con i linguaggi, cioè programmare in modo "creativo".

Note

1. ↑ Il suffisso "Win" prima di "show" indica che è un programma desktop per "Windows"
2. ↑ Per contro, questa tesi vuol dimostrare che il software proprietario limita il mercato complessivo.
3. ↑ Per virtualizzazione si intende la creazione di una versione virtuale di una risorsa normalmente fornita fisicamente. Ad esempio: un computer virtualizzato può essere ospitato assieme ad altri computer virtualizzati in un

altro computer fisico, così come un computer virtualizzato può essere ospitato assieme ad altri computer virtualizzati in un altro computer virtualizzato.

4. ↑ Gestione informatizzata dei ruoli dei componenti di una organizzazione.
5. ↑ Software per la condivisione di risorse tra diversi sistemi operativi
6. ↑ Interprete per l'elaborazione di dati complessi e massivi
7. ↑ Interfaccia grafica di Linux in molte distribuzioni tra cui Debian e Ubuntu

Non possiamo dire per ora quanto il tipo di modello organizzativo influisca sulla “qualità” del software, ma ci dice qualcosa di molto importante sulla “non qualità” del software in genere sia esso libero, open source o proprietario. Il presupposto da cui partano Minich, Muehlbauer e Wentze è:

[...] lo sviluppo software è … lento e costoso, e genera prodotti che contengono seri problemi che causano problemi di usabilità, affidabilità, performance e sicurezza

C'è un assunto scientificamente onesto di consapevolezza di una criticità che non può essere facilmente evitata e che riguarda il software in genere sia esso proprietario o libero. La stessa consapevolezza è insita nel software libero, tant'è che le liste dei difetti, le cosiddette bugs' list, vengono continuamente aggiornate con la scoperta di nuovi bugs e la soluzione di altri. Il software libero non ha alcun timore di pubblicare i propri difetti, ne è consapevole e rende altrettanto consapevoli gli utenti. Questo modo di pulire i panni sporchi in pubblico è possibile perché il software libero non è privato, è un bene collettivo di cui la collettività, cioè la community, si prende carico. Si possono lavare in pubblico i panni pubblici, ma non si possono lavare in pubblico i panni privati. Anzi, il bug diventa veicolo di interazioni, di discussioni, di confronto, di possibilità di azione. Il bug non viene nascosto, ma altrettanto bene è resa pubblica la soluzione. È l'occasione per aggiungere un punto al proprio curriculum, per dimostrare abilità, per farsi una reputazione o per mantenerla.

Come abbiamo già visto il focus resta sull'oggetto sacro, il bug è una minaccia nel vivo del tempio che la comunità è chiamata a sventare. Attraverso il bug la community osserva se stessa, si auto-descrive, rafforza i legami, amplia le relazioni. Un difetto di funzionamento è spesso l'occasione per avvicinarsi maggiormente al progetto, per conoscerlo meglio. È un'occasione per stabilire nuove relazione. Ciò che alla fine attrae è quindi il rapporto prodotto/utente più che il prodotto in sé, ma è pur vero che spesso si testimonia maggiore stabilità, informazioni, istruzioni d'uso rispetto ad analoghi software proprietari. In definitiva l'informatica è un fattore di rischio notevole per chi la usa e per chi la produce e necessita pertanto di fiducia. La differenza tra software proprietario e software libero sta nella modalità con cui viene gestita la fiducia.

Tutto questo avviene in uno spazio virtuale, un luogo estrapolato dalla rete, uno spazio sociologico transnazionale (Beck, 1999) ma che si definisce

community, cioè una comunità morale (Durkheim, 1962), dove non è facile determinare se prevalgono forme di solidarietà meccanica e se ciò che unisce sia l'uguaglianza, piuttosto che forme di solidarietà organica dove ciò che unisce è la divisione dei ruoli, cioè la diversità. Sono compresenti entrambi questi aspetti, Antony Giddens direbbe che è un'ambivalenza della modernità radicale, una ri-aggregazione in un mondo disgregato. Nel terzo e nel quarto principio delle libertà digitali fondamentali la solidarietà non è codificata come obbligo ma in senso positivo cioè come libertà di essere solidale, che corrispondono, in senso informatico, al principio due e tre, essendo che nella programmazione software la numerazione parte da zero e non da uno.

0. Libertà di eseguire il programma per qualsiasi scopo.

1. Libertà di studiare il programma e modificarlo.

2. Libertà di ridistribuire copie del programma in modo da aiutare il prossimo.

3. Libertà di migliorare il programma e di distribuirne pubblicamente i miglioramenti, in modo tale che tutta la comunità ne tragga beneficio.

È pertanto un fenomeno sociologicamente rilevante in termini di rischio, fiducia, solidarietà che rende più plausibile il sussistere di tali elementi all'interno delle dinamiche di globalizzazione. Smentisce, da questo punto di vista, l'assunto che la globalizzazione sia solo un prodotto ideologico del mercato. Il software libero mostra come la globalizzazione non si dispieghi solo attraverso il mercato. Nonostante la sua trans-nazionalità riesce a mantenere quegli elementi finanche della sociologia classica come la solidarietà, la razionalità, ed in generale la leggibilità fenomenologica o costruttivista. Rimane rilevante dal punto di vista organizzativo e produttivo. Questo fenomeno non è semplicemente un fenomeno globale rilevante sociologicamente perché condiziona o è condizionato dal locale, anche se questo avviene, non è solo un fenomeno glocale (Robertson 1995, 1999), ma si definisce nel locale quanto nel globale. In questo senso si può affermare che nel globale prevale l'assenza del luogo nello spazio fisico, mentre nel locale il luogo assume anche importanza in merito ad organizzazione di eventi, di compresenza fisica, di manifestazioni, di iniziative patrocinate da istituzioni come le scuole o gli enti pubblici.

Pur senza luogo la dimensione globale non rinuncia alla compresenza. La Debconf si svolge ogni anno in luglio in luoghi diversi del pianeta, quella del

2010 si è svolta a New York, Gnu/Linux Conference Australia si svolge ogni anno a Sidney, Fosdem ha dimensioni europee e si svolge a Bruxelles tutti gli anni. Ciò che è interessante, soprattutto per la Debconf, conferenza della comunità Debian, molto osservata dall'antropologa americana Gabriella Coleman, è che prevale la funzione di compresenza più che di luogo. In pratica la città dove si svolge indica l'evento non il luogo, il tempo più che lo spazio, evoca una situazione di effervescenza in compresenza ma estranea al luogo, o meglio, il luogo potrebbe essere qualsiasi luogo. New York ad esempio indica l'evento del 2010. Con ciò verrebbe meno l'idea di luogo sacro che ospita i simboli e gli oggetti sacri. In realtà nonostante la compresenza resta sempre una dimensione virtuale inevitabile che è quella del luogo sacro che contiene i simboli sacri e questa è la rete, lo spazio WEB, lo spazio FTP la memoria di massa che contiene il progetto, i dati, le firme digitali, le discussioni, la storia della comunità e i suoi guardiani:

[...] C'è un gruppo chiamato FTP masters, che sono quelli responsabili su cosa entra nell'archivio software Debian e cosa no, e che hanno un potere molto significativo, nel senso che sono quelli che decidono al momento quali sono le licenza che sono accettabili nell'archivio Debian, decidono quando un pacchetto non raggiunge i criteri minimi di qualità per entrare nell'archivio e tutte 'ste cose qua (intervista a Stefano Zacchiroli del 17 agosto 2010)

Quello che rende particolarmente interessante il fenomeno software libero è proprio la sua capacità di rimanere allo stesso tempo globale e socialmente rilevante. Diversamente dall'aspetto glocale descritto da Robertson (1995, 1999), per quanto attiene il software libero, il globale non agisce sul locale e viceversa attraverso dinamiche dialogiche, come per fenomeni di differenziazione su base locale di prodotti industriali globali, di mediazione culturale tra globale e locale. Non esiste una qualche differenziazione tra un sistema globale ed uno locale tale per cui il primo possa penetrarsi con il secondo. La dinamica del fenomeno software libero si dispiega globalmente e si addensa territorialmente. Questo aspetto può essere compreso meglio ponendo in negativo il concetto di ri-localizzazione della tradizione (Giddens, 1995). In questa prospettiva non si tratta di riappropriazione del territorio allocando il capitale simbolico nello spazio che diventa luogo, ma si tratta di riappropriazione della tecnologia attraverso l'allocazione di simboli nello spazio virtuale. La sua peculiarità sta quindi nella capacità di conquistare lo spazio globale, perché è il solo fenomeno in grado di allocare simboli nel virtuale e quindi in una dimensione altra rispetto al territorio. La densità territoriale riguarda, come si è già detto, solo la compresenza, anche se, in una certa misura, la densità territoriale entra in relazione anche con il locale in

forma dialogica con la quale si può compenetrare, ad esempio nell'organizzazione di eventi da parte dei Gnu/Linux User Group organizzati localmente:

[...] Attualmente stiamo cercando di stabilizzare la parrocchia di San Lazzaro come una sede stabile di incontri, il che ovviamente non esclude altre modalità di aggregazione sociale.

io stasera BirraCrua (CountryClub),... San Lazzaro va bene se ci sono incontri particolari o talk, per trovarsi a smanettare un po' e fare 2 chiacchiere personalmente mi trovo meglio in qualche locale, ovvio che sarebbe ottimo trovarne uno che ci permetta di coniugare anche i talk con una bruschetta/pizza (quello che era possibile fare alla Zanza una volta). Ne approfitto per chiedere consigli agli "indigeni" su quale/i locale/i di Vicenza potrebbe/ro essere adeguato/i alle nostre esigenze. (Dialogo tratto da www.vicenza.linux.it/ il 20 settembre 2010)

È importante sottolineare come in questo dialogo lo spazio venga negoziato in relazione al criterio del luogo virtuale. Questo è implicito, non viene nemmeno menzionato tanto è scontato. Devono esserci le chiacchiere, la pizza, il luogo per il talk[1], le macchine su cui “smanettare” ma questo non è il tutto necessario per costruire il luogo, ciò che veramente fa diventare questo spazio un territorio idoneo a diventare luogo è la connettività, l'accesso alla rete, lo spazio virtuale che ospita i simboli virtuali. Altri fenomeni globali legati alla tecnologia non riescono a fare altrettanto. Facebook non ospita manufatti in grado di diventare simboli come può essere un progetto software condiviso. Su YouTube si possono allocare dei manufatti come video e immagini ma non si tratta di oggetti sacri prodotti da una comunità, non contengono una storia condivisa. Attraverso i social forums è possibile coordinarsi, progettare ma non costruire qualcosa di socialmente rilevante in un luogo virtuale autonomo dalla spazio fisico. Il luogo ed i simboli sacri, così come sono stati esposti rappresentano, in questa cornice metodologica, degli ideal-tipi cioè una costruzione teorica che estremizza in senso unilaterale quei tratti che meglio spiegano il fenomeno. Quindi ciò non toglie che la sola connettività sia sufficiente, è necessaria la bruschetta/pizza, la birra. Accanto al rituale dello “smanettare” sopravvive il rituale più tradizionale del “raccontarsi storie”, cioè il talk e questo ovviamente è tipicamente locale. Il software libero quindi pone i suoi simboli sullo spazio virtuale ma non esclude il luogo di incontro tradizionale. Ma ciò che si vuole sottolineare è che è possibile il “luogo virtuale” come peculiarità del software libero.

Si è visto precedentemente un diverso stile di gestione della fiducia nel software libero e nel software proprietario. Si sono evidenziati i fattori di rischio che domandano fiducia, mentre è chiaro in questo contesto il ruolo che hanno quelli che Giddens chiama nodi di accesso ai saperi esperti che come vedremo sono anche i nodi dove diverge la gestione della fiducia del software libero rispetto al software proprietario. Il presupposto è il concetto di "modernità radicale" di Giddens in contrapposizione a quello di post-modernità.

La sua modernità è evidente in quanto si esprime con la tecnologia ed esprime tecnologia. La tecnologia informatica è lo strumento per governare i processi istituzionali e produttivi tipici della modernità. Ma è propria la sua scontatezza di modernità ad essere critica. È proprio la sua supposta modernità a fargli assumere una certa rilevanza simbolica della modernità. Ad esempio vediamo spesso come la cronaca racconti le operazioni delle forze dell'ordine impegnate nelle operazioni di lotta alla criminalità attraverso immagini dove sono presenti le gerarchie con accanto un monitor di un computer dove compare il logo del corpo di polizia con il titolo dell'operazione. Emerge, a livello di osservazioni etnografiche, che i committenti chiedono tecnologia per governare dei processi in senso generico senza specificare esattamente un bisogno. Altre volte invece gli sviluppatori lamentano richieste di automazione di processi assurde e senza senso la cui conseguenze sono la complicazione anziché l'assorbimento di complessità da parte della tecnologia. Da molte testimonianze emerge poi tutto il senso di frustrazione di programmatori ed esperti di sistemi nel vedere macchine tecnologicamente avanzate come mai prima nella storia ridotte a "simboli" di modernità più che "fattori" di modernità, a "mostrare" più che a "calcolare". Questo doppio significato può essere letto come ambivalenza della modernità: emblema simbolico e sistema esperto (Giddens, 1994). Dal punto di vista della teoria della società di Luhmann la sua dimensione moderna è data dalla conseguenza irrazionale della razionalità (Niklas Luhman, 1991, p.330).

L'informatica rimane comunque anche un sistema esperto e come tale necessita di fiducia. Come abbiamo anticipato questa viene gestita diversamente nel contesto libero rispetto al contesto proprietario. I nodi di accesso ai saperi esperti rappresentano il terreno dove questa divergenza si manifesta. I nodi di accesso ai sistemi astratti sono secondo Giddens le occasioni in cui gli attori comuni vengono in contatto con i portatori di saperi esperti. Quindi medici, hostess di aerei, avvocati, dentisti e via dicendo. Sono coloro che mediano tra le persone e la complessità della modernità. Questi nodi di accesso hanno però dei punti deboli poiché gli esperti potrebbero commettere degli errori, cioè

fraintendere o ignorare le competenze che dovrebbero avere. In qualche modo questi esperti devono sostenere il loro ruolo sulla ribalta, devono distinguere il retroscena dalla scena (Goffman, 1988).

Il software libero adotta un'altra strategia, mette in gioco un'altra risorsa per la gestione della fiducia che è la lealtà. Elimina la separazione tra scena e retroscena, (o “scienza e retroscienza” nel senso del principio di falsificabilità di Popper) e parte dal presupposto che il software ha sempre e comunque dei difetti. Il software libero essendo consapevole della complessità e delle scarsità di competenze non può far altro che ammettere i propri limiti nella speranza che sia la società, o una parte di essa, a prendersene carico ed è ciò che viene espresso dalla legge di Linus: “Dato un sufficiente numero di occhi tutti i bugs vengono a galla”. La partita della fiducia viene giocata, per così dire, a carte scoperte, la separazione tra scena e retroscena impedisce la soluzione dei problemi. Il software libero non fornisce garanzie sul prodotto ma informazione sul prodotto. Nonostante ciò resta il concetto di accesso ai sistemi esperti che sono i mantainers (mantenitori dei pacchetti), gli esperti ed in genere la comunity stessa.

Questo è proprio il caso in cui il software libero è in grado di mettere in campo competenze sociologiche. Nemmeno questa, in ogni caso, è prerogativa esclusiva del software libero:

[...] la cucina non è soggetta ad una totale introversione: al contrario diventa momento e occasione di comunicazione dello chef, grazie ad una parete di vetro totalmente trasparente che la separa dalla sala senza individuare una barriera visiva. La parete sembra piuttosto inquadrare ciò che accade dietro le quinte, ponendo i movimenti dei cuochi e dei camerieri al centro della scena. La cucina si apre così alla sala di ristorazione, si mostra nella sua valenza tecnica e funzionale, sembra voler avvicinare le persone alla ricerca sottesa ai singoli piatti che verranno serviti, portando gli ospiti a pregustarli, a immaginarli, a carpirne mentalmente i processi di lavorazione. (www.ghigos.com/wp/?p=767 26 settembre 2010).

Sarebbe però riduttivo affermare una totale trasparenza. Più che eliminazione del retroscena si tratta di una rielaborazione di questo. Spazi altri possono essere recuperati come retroscena, alcuni angoli possono comunque restare fuori dalla portata degli sguardi dei non addetti ai lavori ma, più interessante ancora, è che la ribalta può essere preparata prima dell'apertura dell'arrivo dei clienti e quindi la separazione può essere giocata sul tempo anziché sullo spazio. Nello specifico del

software libero, accade spesso che prima di intervenire nella mailing list ufficiale della comunità ci si confronti in forma privata con altri prima di decidere cosa scrivere:

[...]L'orario va benissimo, non penso di avere problemi, il titolo anche se non trovo qualcosa più ad effetto. In particolare mi piacerebbe fare del benchmarking con il sw pp[2], dipende anche da quello che riesco a mettere assieme. scusami se non ti ho dato conferme, ma in ogni caso seguo la lista e se ci sono problemi mi faccio sentire. Devo preparare una scaletta?

la scaletta non serve, quel giorno sai che hai 45 minuti e te la gestisci tu. se riesci a trovare un bel titolo che sia capibile dalla gente comune sarebbe meglio. io purtroppo non conoscendo bene la questione faccio fatica a proporne uno... "mappe aperte con il software libero"?

(diario 20 settembre 2010)

In questo modo il retroscena diviene un concetto più articolato ed elaborato, diventa pertanto più complesso da gestire. In qualche modo, anche in questa dinamica, si verifica da un punto di vista funzional-strutturalista[3], come si vedrà più avanti, una semplificazione ambientale che comporta la complessificazione interna al sistema. Ciò che esternamente può essere visto come una semplificazione attraverso l'eliminazione di barriere, comporta una complessificazione dell'assetto organizzativo interno al sistema. Questo non significa che vengono spese energie per mantenere comunque un retroscena in ogni caso indispensabile e che sostanzialmente non cambi nulla. In realtà si verifica una cosa molto importante: il sistema è in grado di spostarsi su livelli di maggiore complessità assorbendo complessità ambientale. Come si vedrà più avanti, il compito dell'informatica è proprio quello di assorbire complessità.

Note

1. ↑ Discorso su di un argomento di cui si è esperti rivolto agli altri partecipanti.
2. ↑ oftware proprietario
3. ↑ Ci si riferisce al funzionalismo di Luhmann

Uno degli aspetti che attirano l'attenzione sociologica quando si parla di software libero è il fatto che che in questo ambito si usi molto il termine community (comunità). Questo fatto obbliga ad una certa attenzione ed ad una analisi accurata sul significato che tale termine assume per chi vi partecipa, che fa parte o si sente parte della community. Fin dall'inizio si è usato il termine "community" e non "comunità" proprio per indicare quello specifico ambito senza correre il pericolo di confonderlo con il concetto sociologico di comunità (Gemeinschaft). In sociologia la Gemeinschaft (comunità) versus Gesellschaft (società) ha spesso indicato rapporti premoderni versus rapporti moderni, cioè questa dicotomia è alla base, a partire da Ferdinand Tönnies (2009), di una serie di categorie dicotomiche indicative del grado di modernità: status versus contratto, tradizione versus razionalità, ascrizione versus acquisizione, adesione libera versus appartenenza per nascita, punizione versus sanzione e via dicendo. Cercando di applicare queste variabili al fenomeno del software libero, in modo immediato senza scendere troppo in analisi meticolose, ci si rende conto che in realtà il software libero tende maggiormente alla "Gesellschaft" cioè alla società più che alla comunità. In prima istanza ciò che viene a mancare completamente in questa idea di comunità è un qualsiasi riferimento alla famiglia che invece è centrale in qualsiasi concezione di comunità da Aristotele, Sant'Agostino, San Tommaso d'Aquino, Hegel, finanche la filosofia positiva di Comte e la sociologia metodologica di Le Play.

In realtà già prima di Tönnies, Ser Henry James Samner Maine aveva rilevato il passaggio dalla comunità alla società, dalla premodernità alla modernità attraverso il passaggio dallo status al contratto. Questo è un aspetto molto interessante nel caso del software libero perché secondo Maine la modernità è una questione di evoluzione del diritto che passa attraverso tre espedienti funzionali: le finzioni legali, l'equità, e la legge. Più interessante di tutti questi strumenti, ai nostri fini, sono le finzioni legali, cioè assunzioni che dissimulano la ratio della norma facendo leva sulla plausibilità. È il caso del copyright trasformato in copyleft che verrà esaminato più avanti. Di fatto la licenza GPL (General Pubblic License), dal punto di vista sostanziale è un copyright le cui clausole ne stravolgono il senso. In questo senso più che dire superficialmente che il contesto della community è la modernità, in quanto al contratto in sé, possiamo spingerci oltre prendendo in considerazione il processo di ulteriore modernizzazione attraverso la finzione contrattuale per delineare una prospettiva di iper-modernità o se vogliano di modernità radicale (Giddens, 1994). Più che un superamento della modernità attraverso un ritorno alla "comunità" sembra piuttosto

l'anticipazione di forme inedite, non post-moderne, perché la razionalizzazione gioca ancora, come vedremo, un ruolo preminente e neppure sotto forma di materialismo storico perché non esiste in questo senso un intenzionale uso della storia per fare la storia, non viene prospettato un fine messianico o una teleologia. Sussiste come tale, è una conseguenza che si tenta di spiegare come conseguenza della modernità.

Già Nisbet (1966) aveva sottolineato che anche Durkheim, come Weber, a partire dalla seconda parte della Divisione del Lavoro (1962) si rese conto che le forme di solidarietà organica tipiche della modernità dovevano basarsi necessariamente sulla continuazione di forme di soliderietà meccanica premoderne e che la completa sostituzione delle une con le altre avrebbe portato a mostruosità sociologiche. Allo stesso modo si è già rilevato (cfr cap. 2 par. 2) come Weber avvertisse il rischio di una organizzazione sociale pervasiva a tal punto da divenire una gabbia di ferro. In qualche modo i padri della sociologia percepirono i pericoli di una razionalità e di una modernità portata agli eccessi. Ma è solo attraverso gli eventi dei totalitarismi di inizio secolo, del secondo conflitto mondiale, dell'industrializzazione della guerra e della guerra fredda che i sociologici potranno descriverne con maggiore precisione tutti i limiti. Ai nostri fini è interessante questa lucida analisi degli effetti emergenti della modernità da parte di Niklas Luhmann:

[...] l'esperienza che proprio una organizzazione che funzioni bene e che sia pienamente sviluppata secondo gli orientamenti di moda della razionalizzazione e della democratizzazione, proprio una cosi fatta organizzazione produce particolari irrazionalità. Per la crescente complessità del decidere su decisioni su decisioni su decisioni, l'auto-poiesi sviluppa strutture appropriate e una tendenza crescente a decidere di non decidere. Per il trattamento dei propri difetti, essa può applicare ancora una volta solo gli stessi strumenti che avevano prodotto quei difetti: essa cioè può applicare solo decisioni. A queste condizioni, inoltre, si atrofizza l'accoppiamento strutturale con la motivazione individuale. Poiché bisogna sempre e di nuovo decidere, manca la motivazione di rafforzarsi contro le resistenze interne ed esterne che vengono frapposte alla esecuzioni delle decisioni. (Luhmann, 2003, p. 330).

Ciò non toglie che sussistano o addirittura vengano recuperate forme comunitarie. Questi sono fenomeni che già Giddens ha spiegato come compatibili con la modernità radicale in termini di ri-aggregazione. Uno dei processi di razionalizzazione che Weber aveva identificato nel processo di modernizzazione

era, ad esempio, la separazione delle attività produttive dalla vita privata e familiare mentre, nel software libero, il tempo che i volontari impiegano per sviluppare programmi viene sottratto al tempo libero. Lo stesso fatto che gli strumenti di produzione, come la connettività e ed i computer siano anche strumenti domestici rende ancora più labile questa separazione tra intimità e professione. Nulla esclude che la prossima generazione di sviluppatori, i figli dei pionieri del software libero, continuino l'opera dei padri avendo respirato in famiglia l'ethos dei genitori, avendo accesso agli strumenti ed imitando un certo stile di utilizzo degli stessi. Si potrebbe verificare in tal caso una sorta di tradizione familiare, una trasmissione di genitori in figli della professione di freelance del software libero così come accadeva nelle ghilde e nelle comunità medioevali dove le attività familiari e economiche divengono un tutt'uno.

Sussiste anche una rielaborazione in chiave iper-moderna di cultura antropologica legata ai manufatti. Questo è il software che trova la sua collocazione nello spazio virtuale. Questi manufatti contengono i contributi di tutti, non sono semplici espressioni della cultura, ma espressione del coordinamento di tutta la collettività che partecipa alla sua costruzione. Volendo cogliere l'idea di comunità morale in senso Durkheimiano, cioè trascendente, diviene immediato attribuire, in maniera transitiva, quei manufatti all'opera degli dei. In termini analogici si potrebbe asserire che proprio per il fatto che tutti vi partecipano, quei manufatti non hanno un proprietario identificato, nessuno può rivendicarne un diritto di possesso, quei manufatti sono sottratti alla negoziazione e quindi assumono una dimensione sovraindividuale. Chiaramente chi vi partecipa sa molto bene che non può attribuirgli una valenza mogico-sacrale in senso stretto, ma la cosa sconcertante è che, in quanto inaccessibili ai singoli, svolgono efficacemente la funzione del sacro pur nella consapevolezza che non sono tali. Non di meno, lo spazio virtuale dove vengono posti è presidiato da guardiani, cioè gli FTP masters, responsabili di cosa entra ed esce dall'archivio, sono coloro che decidono quali sono i software idonei ad entrare e quali no. Questi simboli “sacri” sono ciò che in definitiva tiene unita la comunità. In altri termini, si potrebbe anche asserire che la collettività necessità di una sua visibilità sovraindividuale che anziché risolversi con il ricorso al simbolismo strettamente sacro, in termini antropologici, si risolve con la costruzione autocoordinata e collettiva di manufatti tecnologici.

Secondo Durkheim uno degli aspetti che differenziano le società moderne dalle società pre-moderna è la divisione del lavoro (1962), quindi il tipo di solidarietà che ne consegue è una solidarietà organica dove ciò che unisce non

sono più le uguaglianze ma le differenze. Se prendiamo in considerazione i principi di industrializzazione del software, uno dei fattori che spesso viene ribadito è, su questa base, la netta separazione tra progettista e programmatore. Il progettista ha il compito di interpretare le esigenze del committente, o del consumatore. Questo è ciò che riguarda la macroprogettazione a cui segue microprogettazione che viene elaborata secondo principi ingegneristici in un linguaggio formale, che tecnicamente si chiama pseudolinguaggio, che il programmatore deve essere in grado di interpretare e tradurre in linguaggio informatico. In queste diverse fasi dello sviluppo vi sono competenze codificate e quindi ruoli prestabiliti dove l'interdipendenza tra le fasi, e quindi gli attori che vi partecipano, è evidente.

Come abbiamo spiegato precedentemente questa suddivisione dei compiti viene meno nel software libero. Nella fattispecie la progettazione non è separata dalla programmazione. Sussiste quindi, soprattutto nella fase più matura, una divisione di ruoli tra i vari attori ma meno accentuata. Come osserva Giddens è difficile trovare società, per quanto moderne, dove siano assenti forme di aggregazione comunitaria, o come rileva Bagnasco (tracce di comunità, 1999) sussistono sempre forme di economia informale che entrano in rapporto di interscambio con l'economia formale. Vale però la pena rilevare come per il software libero vengano recuperate, in qualche misura, forme di solidarietà pre-moderne. Questo oltre ad essere un'ambivalenza della modernità stessa, è anche una forma di ri-aggregazione e di riappropriazione di saperi esperti (Giddens, 2005).

Infine, analizzando le variabili comunità/società della sociologia classica questa ambivalenza del software libero appare evidente. Se però si tiene conto, in senso funzional-strutturalista, del fatto che la razionalità moderna inciampa nella sua fase radicale in una serie di contraddizioni allora si può anche tentare di capovolgere i termini del problema. Se per relazioni societarie in contrapposizione alle relazioni comunitarie intendiamo il dispiegarsi del pensiero illuminista volto al disincanto, all'affermazione di un potere razionale, alla eliminazione della superstizione e alla affermazione del libero contratto tra individui, allora vediamo come oggi queste istanze vengano rivendicate, anche dal sentire comune, contro la società che soffre, per così dire, dei limiti della comunità medioevale. Più che un ritorno romantico alla comunità (Gemeinschaft) di Ferdinand Tönnies (2009) si tratta di un rinnovato spirito rivendicativo che ripropone quello che Sir Ernest barker aveva indicato come:

[...] immagine di individui naturalmente liberi che avevano razionalmente accettato uno specifico e limitato modo di associazione: l'uomo era l'elemento primario, i rapporti l'elemento secondario. (Nisbet, 1987, p. 69)

Questo stesso spirito rivendicativo si rivolge oggi contro quegli aspetti della società rimasti irrisolti. Non si tratta di un ritorno ad istanze comunitarie nel senso sociologico del termine, meno ancora romantico, come lascia intravedere Himanen, ma della riproposizione di istanze razionali e professionali tipiche delle modernità rivolte contro il persistere o il riemergere di caratteristiche premoderne all'interno della società moderna.

Il persistere o il riemergere di caratteristiche premoderne all'interno della società moderna è ciò che spiega l'ambivalenza del software libero dispiegato tra Gemeinschaft e Gesellschaft. Se da un lato con la società moderna vengono meno le caratteristiche acquisite con la nascita di fatto essere nati e socializzati in una classe sociale anziché un'altra fa una grande differenza in termini di mobilità sociale (Bagnasco, Barbagli e Cavalli, 2007). Allo stesso modo molte ricerche mostrano che il successo scolastico dipende dalle aspettative degli insegnanti sugli allievi e queste dipendono, a loro volta, dalla provenienza sociale. Pierre Bourdieu usa il concetto di capitale culturale trasmesso dalla famiglia attraverso un continuo addestramento. Ciò che viene attribuito a doti naturali è in realtà frutto della socializzazione a valori e atteggiamenti coerenti con le aspettative delle agenzie di socializzazione, cioè la scuola, il mondo del lavoro e le istituzioni.

Il metodo etnografico usato in modo pionieristico da Robert ed Helen Lynd (1937) per osservare il mutamento della modernità mostrano come negli USA del primo novecento si inneschino meccanismi tali per cui il potere tende concentrarsi su determinate famiglie. Floyd Hunter (1953) rivela come siano solo piccoli gruppi che riescono a controllare tutte le decisioni importanti. Wrigth Mills (1956) mostra come le gerarchie militari, politiche, economiche siano tra loro interscambiabili. Per Wrigth Mills il discorso sulla democrazia rimane aperto è sempre e comunque incipiente e necessita di essere continuamente riproposto.

L'acquisizione in contrapposizione all'attribuzione per nascita, l'universalismo in contrapposizione al particolarismo, la maggiore mobilità sociale, la meritocrazia e la democrazia in genere non sono garantite dalla modernità e necessitano di essere continuamente riproposti. Non esiste una struttura sociale moderna scontata, piuttosto la modernità si distingue per la pressione che è possibile esercitare su tematiche universalistiche, acquisitive e democratiche. In questa prospettiva la modernità si presenta come possibilità più che come “fatto”. Il software libero mette in campo queste istanze, ma nel contempo ne rielabora altre in chiave iper-moderna. È il caso della razionalità, della divisione del lavoro e delle forme di autorealizzazione. La razionalità che viene proposta non è più quella sinottica della moderna burocrazia, ma quella riflessiva. La divisione del lavoro assume forme più elastiche che consentano la creatività e l'autorealizzazione. Non si tratta di rievocare in senso romantico la “semplicità” della Gemeinschaft, ma al contrario si tratta, come si vedrà più avanti, di uno spostamento ad un ulteriore livello di complessità sociale

(Luhmann, 1988). Il processo di razionalizzazione mostra tutta la sua ambivalenza. Se nella prima rivoluzione industriale era razionale separare l'impresa dalla famiglia (Weber, 1997), oggi diventa razionale unificarle. Si pensi ad esempio al fatto di diminuire il traffico, l'inquinamento e guadagnare tempo attraverso il telelavoro. Lo stesso vale per il modo di lavorare di molti hacker nei propri ambiti domestici e le proposte di politica sociale e di "tempi della città" volte conciliare meglio famiglia e lavoro. Come si vede, all'opposto, diventa oggi razionale la minor separazione tra famiglia e lavoro, ma questo si deve realizzare nello spazio e nel tempo senza far venire meno la separazione funzionale di questi ambiti e ciò è indubbiamente molto complesso perché mentre nelle società premoderne non susisteva una separazione spazio/tempo tra professione e famiglia nemmeno esisteva una divisione funzionale tra questi due ambiti, le stesse gerarchie familiari corrispondevano alle gerarchie professionali. Nella modernità la separazione professione/famiglia corrisponde anche ad una divisione funzionale. Nella società iper-moderna la separazione tra professione e famiglia non è spazio/temporale ma solo funzionale. La stessa razionalità burocratica e la democrazia assumono oggi connotazioni paradossali che li negano (Luhmann, 2003) e necessitano pertanto di essere rielaborate.

Ci sono quindi aspetti del software libero che sembrano contraddire la modernità. Si tratta di atteggiamenti che non sono facilmente inquadrabili in un discorso "moderno". In particolare la divisione del lavoro e la conseguente parcellizzazione. Ciò che si verifica nell'open source è invece una riappropriazione di una visione complessiva del processo produttivo da parte di chi vi partecipa, negando così il principio di specializzazione nel cuore della tecnologia e della modernità. Inoltre il fenomeno del software si evolve attorno ad un fatto peculiare che è l'emergere del sistema operativo Gnu/Linux attorno al quale si mobilitano energie inaspettate e che mantiene a distanza di vent'anni la sua centralità. Il software libero si è diffuso sempre di più in questi ultimi anni e il trend attualmente è in piena crescita. Obbliga le multinazionali a fare i conti con questa realtà altra, che non è semplicemente rappresentata da un concorrente più agguerrito, che però fa le stesse cose, nello stesso modo, solo in maniera più efficiente. Si tratta di un concorrente polimorfo diffuso che addotta paradigmi altri. Nonostante ciò, nonostante la sua diffusione, non cambia la centralità del sistema operativo Gnu/Linux emblema, simbolo del software libero tout court. È un fatto talmente centrale che se venisse meno verrebbe meno l'open source. Ciò che tiene assieme la community è Gnu/Linux. Al paradosso visto dal punto di vista dell'azione economica e produttiva su base volontaria, ciò che colpisce è

questa convivenza tra modernità e aspetti di solidarietà comunitaria, tra tecnologia e solidarietà meccanica.

il concetto di solidarietà meccanica nel software libero è centrale. Ciò che unisce una comunità rispetto ad una società industriale è l'uguaglianza, l'allocazione di risorse secondo criteri normativi impliciti. Uno degli esempi che si riportano per spiegare questo tipo di solidarietà è quanto accade nelle comunità di cacciatori raccoglitori pre-moderne nelle quali chi uccide una preda deve condividerla con il proprio gruppo. Non vi è la possibilità di investire ciò che eccede dalla soddisfazione dei propri bisogni immediati in una assicurazione sulla vita, non esiste un mercato che permetta al cacciatore raccoglitore di scambiare l'eccedenza in modo ottimale attraverso un mezzo simbolico come il denaro, non esiste la possibilità di conservarla. C'è però la possibilità di investire l'eccedenza nelle relazioni e quindi in solidarietà. È esattamente ciò che accade nel software libero, non ha senso tenere per se una cosa che eccede il proprio bisogno e che non è materialmente scambiabile con qualcosa d'altro. Questo nel campo dell'informatica accade per due ragioni:

- Ormai la tecnologia ha sviluppato molte soluzioni che ricrearle da capo diviene assurdo. Sarebbe un continuo inseguire senza fine chi è arrivato prima e detiene il monopolio.

- Soluzioni molto personalizzate sono vendibili una sola volta, nessun altro nel mercato avrà quella stessa “taglia”.

Le soluzioni personalizzate, fatte on demand da freelance, sono già state vendute, o per meglio dire, è già stato pagato il prezzo per quel lavoro e per lo più non si trovano altri a cui può interessare la stessa soluzione. Alcune parti di un determinato programma possono però risultare utili a qualcun altro che potrebbe ricambiare con altre soluzioni. L'esempio è sempre quello del cacciatore raccoglitore che essendo stato solidale verso gli altri appartenenti al suo gruppo può sperare nella solidarietà degli altri nel momento in cui non riuscirà ad uccidere una preda, in una dinamica di reciprocità. Questa dinamiche si dispiegano come naturale propensione antropologica dell'uomo nel pieno della modernità, perché anche nella modernità radicale (Giddens, 1995), si verificano le stesse condizioni, una nuova versioni in chiave moderna di fatti sociali propri delle società di cacciatori raccoglitori.

Paradossalmente si verifica anche una situazione di uguaglianza che si spiega su tre livelli:

- Il software libero mobilita già preventivamente persone che hanno lo stesso tipo di formazione e gli stessi o simili valori. In termini relativi possono essere persone che presentano caratteristiche particolari e non frequenti. Le competenze professionali, ad esempio, sono solitamente molte elevate e relativamente scarse, ma se reclutate attraverso la rete possono diventare molto numerose in termini assoluti.

- le relazioni sulla rete rendono intrasparente i ruoli, i titoli, l'età, spesso anche la nazionalità.

- Il focus attentivo è rivolto verso la tecnologia, nella fattispecie open che rappresenta in questo schema l'elemento sacro.

Attorno al sistema operativo Gnu/Linux si può infine distinguere una tribù organizzata in clan. Innanzitutto Gnu/Linux rappresenta un'origine comune ai diversi clan. Dalla distribuzione Gnu/Linux discendono molte distribuzioni le più famose delle quali sono Debian, Ubuntu, Fedora, CentOS. Altre sono a loro volta derivazioni di queste come Red Hat derivato da CentOS e Ubuntu derivato derivato da Debian. La ramificazione è molto più estesa e complessa e va oltre gli scopi di questa ricerca.

È a partire da J.F. McLennan nell'articolo The worship of animals and plants (1869-70) che viene introdotto il concetto di totem come simbolo di un antenato mitico comune. Questo concetto fu reso più astratto da Lévi-Strauss nel 1962 che elabora una teoria sul totemismo come processo di socializzazione in cui gli oggetti simbolici vengono caricati di valore emozionale. I diversi clan si differenziano per la loro specializzazione produttiva spesso in concorrenza. Quindi le cosidette “distro” - distribuzioni Gnu/Linux - versioni in cui viene distribuito Gnu/Linux sono oggi parecchie decine ma sono tutte basate sullo stesso kernel (nucleo) Gnu/Linux. La spiegazione dell'esistenza di queste distribuzione, spesso in accesa concorrenza tra loro, può avere una spiegazione coerente con quanto accade nei clan delle società premoderne dove il numero dei membri è ottimale alla loro sopravvivenza. Quando il numero è eccessivo diventa critica la pressione sulle risorse, quando il numero è scarso diventa critica la forza del gruppo. Lo stesso accade quando un progetto cresce, non c'è spazio per tutti, anche se vi sono progetti come Debian che annoverano migliaia di volontari. Anche in questo caso ciò che si verifica è la presa in carico di “pacchetti” da parte di gruppi più piccoli. In questo modo la community Debian si pone come intermedia tra la centralità di Gnu/Linux e altri sotto-clan. Nel complesso ciò che

accade è la grande capacità del software libero di mobilitare risorse, e l'eccesso di queste risorse si riorganizza a sua volta nella periferia. Va notato che l'espressione community si riferisce tanto alla tribù dei linuxiani, quanto ai diversi progetti che vi ruotano attorno e quindi assume significato di clan o di tribù a seconda del contesto.

Si tratta come abbiamo visto, nel confronto con il “politeismo dei valori” della società moderna, di una comunità culturalmente molto omogenea. I valori che ne discendo di fatto sono quelli che questa cultura esprime. La differenza che può sussistere è che questa comunità si forma per selezione di persone che in qualche modo aderiscono a questa particolare visione del mondo. Non si appartiene per nascita e questo è quindi un elemento fortemente moderno che la caratterizza pur rimanendo coerente con una dinamica totemica. Lévi-Strauss rileva anche come nelle società totemiche il processo intellettivo sia mediato dalle analogie con le categorie della natura. In questo le diverse distribuzioni Gnu/Linux rappresentano le metafore in cui ci si riconosce. Una delle metafore più forti in cui operano veri e propri processi di identificazione si verifica ad un livello intermedio tra distribuzioni e il kernel Gnu/Linux, cioè la parte più interna e più sacra del software libero, ed è rappresentata dalle due interfacce grafiche KDE e GNOME. La prima è più complessa ma anche più sofisticata e integrata, mentre la seconda è più lineare e amichevole. La prima inoltre nasce come licenza commerciale, mentre la seconda nasce da subito come software libero nell'ambito del progetto GNU. Resta però centrale la proiezione di questa società totemica nell'antenato comune Gnu/Linux.

Altro aspetto particolarmente interessante è il fatto di rintracciare quegli elementi di pensiero totemico anche nell'iper-modernità (Bateson, 1979), nella fattispecie nel software libero. Questa è la metofora dei simboli zoomorfi come il grifone, il leone, il gallo e via dicendo che Bateson evidenzia nell'araldica delle grandi famiglie come forma secolarizzata del totemismo. Nel software libero questi sono lo Gnu (acronimo ricorsivo che significa GNU is Not Unix), il progetto della Free Software Foundation e il Pinguino Tux (Torvald's UniX) di Gnu/Linux. La metofora gnu è spesso ricondotta ad un'idea di mansuetudine ma anche di comunità in quanto gli gnu vivono in branco; il pinguino significa eleganza per via del Frac che si richiama anche dal nome come forma contratta di tuxedo che in inglese significa appunto frac. Questi due simboli costituiscono il tema centrale del software e cioè il sistema operativo GNU/Linux. Quindi i valori a cui si fa riferimento metaforicamente, esplicitamente e spesso anche implicitamente sono quelli della comunità e dell'eleganza. L'eleganza è quella delle soluzioni che si riscontrano nel codice, il concetto di comunità si riferisce alla solidarietà. Da questi valori ne discendo anche le norme. Solidarietà e libertà sono connesse in quanto quest'ultima presuppone la prima.

Di fronte all'impossibilità da parte di Stallaman di modificare il driver di una stampante Xerox per migliorarne il funzionamento non resta che uscire dall'isolamento, costituire un movimento scientifico che si oppone alle logiche della limitazione delle risorse della conoscenza destinato poi ad espandersi orizzontalmente ad altri campi del sapere umano, come ad esempio i vari progetti Wikipedia, Wikimedia e Wikisource, ed in senso verticale coinvolgendo persone che diventano utenti GNU/Linux e che in tutto il mondo fondano movimenti locali, i cosidetti LUG (Linux User Group), che si affermano in particolar modo in modo in Italia. Nascono principalmente nelle facoltà di scienze matematiche e fisiche ed ingegneria. In Italia le prima sono Padova, Bologna, Roma, Milano, ma agiscono fuori dai campus e nel territorio. Sono molto attivi, organizzano molti eventi, danno consulenza gratuita a chi decide di adottare sistemi Gnu/Linux e sono organizzazioni intermedie tra la community virtuale ed il locale, ma sono anche nodi di accesso ai saperi esperti (Giddens, 1994).

Oltre all'aspetto della solidarietà, di cui si è già accennato, un aspetto rilevante è il valore della diffusione del software libero e della conoscenza, cosa che da un punto di vista valoriale può essere letta come evangelizzazione. In questo contesto una delle regole implicite più frequenti è quella della “possibilità di solidarietà”. Se l'aiuto che viene richiesto riguarda software proprietari questo viene negato, l'appello di aiuto deve riguardare solo software libero:

- *[...]Salve a tutti, come si fa ad installare Java 6 in una Debian Lenny? Nei ripostigli (o repository se preferite) c'è soltanto Java 5, ma per far funzionare Josm ormai è necessaria la versione 6.*

- *[...]Parli di Java proprietario di Sun? Hai visto se c'è OpenJDK?*

  (www.vicenza.linux.it 15 settembre 2010)[1]

C'è un problema di coerenza importante, e cioè il software proprietario non può entrare in una dimensione solidaristica. Più che un semplice problema di non collaborazione col “nemico” si tratta di un problema di identificazione, di preservazione dei confini e di rischio di contaminazione (Mary Douglas 2003). Il software libero, i LUG, le community dei vari progetti si identificano nel software libero, intervenire su apparati proprietari significa non identificare il proprio agire in un contesto emozionale inteso sia in senso intellettivo, di stato di flusso, che in senso comunitario di appartenenza. Si può intervenire su apparati proprietari, solo in termini di scambio monetario. L'aspetto della contaminazione riguarda invece il

fatto di mescolare software proprietario e software libero. La conservazione dei confini riguarda il fatto di allontanare la minaccia che il software proprietario si appropri del software libero. Questo aspetto è il motivo di conflitti più frequente nei forum e nelle mailing-list:

- *[...]Scusa la puntualizzazione, ma a questo punto allora dovremmo rimuovere anche il bottone PayPal dal sito del LUG, se ragioniamo sito web = software proprietario. Mi piacerebbe sapere la tua opinione in merito a ciò.*

- *[...]Rispondo molto velocemente. Fosse per me lo toglierei, lo sanno anche gli altri consiglieri, l'ho detto più volte. Non perché PayPal usi software non libero, ma semplicemente perché dovendo pagare la quota associativa di persona, almeno si sarebbe "obbligati" a farsi vedere una volta l'anno. (www.vicenza.linux.it 31 agosto 2010)*

In questo scambio di battute è presente la preoccupazione per la contaminazione del software proprietario nel WEB e se sia giusto usare servizi WEB che usano software proprietario come fa PayPal. È inoltre evidente anche la preoccupazione per la scarsa compresenza nella dimensione del sacro (Victor Turner, 1972). Queste poche righe sintetizzano molto bene come la dimensione del sacro sia coerente con le dinamiche che si dispiegano nelle comunità software libero: contaminazione, compresenza, caricamento emotivo, tensione morale e regole implicite.

1. ↑ In questo esempio si chiede assistenza, e chi risponde ritiene che “java” della Sun Microsystem non risponde ai requisiti di software libero, quindi consiglia di rivolgersi ad altri prodotti.

Alcuni degli aspetti fenomenologici del software libero sono già emersi nelle trattazioni precedenti di questa tesi. È comunque necessaria una loro sistematizzazione al fine di rendere tali schemi interpretativi operativi e quindi completarne la trattazione. In particolare è necessario completare la costruzione dell'ideal-tipo, cioè dell'attore teorico che meglio riassuma l'agire causativo del fenomeno. Ci eravamo preoccupati di tenere distinto l'immagine di “hacker” da quella del “cracker”, per lo più riconducibile ad un agire utilitaristico. Successivamente poi si sono introdotti altri termini come “geek” e “nerd”. Tutti questi termini individuano separatamente quelle caratteristiche che nell'insieme entrano a far parte dell'ideal-tipo.

Il termine hacker è il principale e più documentato, spesso soggetto ad interpretazioni controverse. Letteralmente deriva dal termine inglese to hack che significa tagliare qualcosa in modo approssimativo con un grosso coltello o (in senso informatico) usare un computer per guardare a e/o modificare le informazioni che sono memorizzate in un altro computer senza permesso (libera traduzione da Oxford dictionary ed. 2006 p.ne 548). Da un punto di vista informatico il suo significato è molto controverso e la definizione che ne da l'Oxford dictionary per www.wikipedia.it corrisponde al luogo comune, mentre come definizione principale riferisce il termine hacker ad una persona che si impegna nell'affrontare sfide intellettuali per aggirare o superare creativamente le limitazioni che gli vengono imposte, non limitatamente ai suoi ambiti d'interesse (che di solito comprendono l'informatica o l'ingegneria elettronica), ma in tutti gli aspetti della sua vita (it.wikipedia.org/wiki/Hacker 27 settembre 2010).

Altra descrizione si trova nel jargon file1 che è il dizionario hacker. Si tratta di un'auto-descrizione dove il termine hacker è associato a più significati, nove in tutto, anche molto diversi e che cambiano a seconda del contesto in cui vengono usati. Quindi to hack può significare fare un buon lavoro meticoloso, dedicarsi in modo parsimonioso ad un progetto, oppure fare un lavoro in fretta ed in modo approssimativo. Questa ultima accezione in realtà collega abbastanza bene il termine letterario inglese “tagliare in modo approssimativo” con il “fare un lavoro in modo approssimativo”. Il dizionario inglese rende ancora meglio questo modo poco curato di agire nervoso e frettoloso attraverso l'immagine di un “grosso

coltello". Se si pone l'attenzione sul modo nervoso e frettoloso di procedere, più che su ciò che viene fatto, allora si possono dedurre cose diverse da ciò che può evocare l'azione di tagliare. Potrebbe evocare allora uno stato di ansia da consegna, tipico stressore dei freelance informatici, quindi lavoro frettoloso e nervoso per rispettare i tempi di consegna.

Quindi il termine hacker, nelle sue diverse accezione rappresenta un continuum che va dall'agire approssimativo e frettoloso al meticoloso ed impegnato, dal costruire al penetrare abusivamente. In questo continuum vi è un'ambivalenza moderna che riguarda due aspetti altrettanto moderni, cioè la professione ed i limiti intesi come scarsità di risorse: il tempo e l'informazione. L'agire frettoloso ed approssimativo e la meticolosità sono due facce della stessa medaglia, due fasi dello stesso processo di sviluppo. Uno in cui si dimostra al committente cosa si riesce a fare e l'altro in cui si rifinisce il lavoro, in cui, tipicamente si fa debug, cioè si risolvono i difetti di programmazione. Questa ambivalenza dei termini non è peculiare dell'hacking. Ad esempio il termine burocrazia nasce come accezione di razionalità, mentre nel senso comune assume un'accezione negativa e irrazionale; lo stesso concetto di generalità intese in senso burocratico si riferiscono, nel senso comune, a delle specificità.

Queste variazioni di significati, tra due opposti, in realtà più che indicare un concetto determinabile indicano un campo semantico, nella fattispecie un'enantiosemia. Così la burocrazia indica l'ambito della razionalità/irrazionalità, l'identità si riferisce all'ambito generalità/specificità mentre feriale, che si riferisce a giorno di lavoro, deriva dal latino feriae che significa giorno di riposo. Il termine hacker ha a che fare con il campo semantico della tecnologia, più precisamente indica la relazione con la tecnologia, e, più specificatamente ancora, indica una relazione di conoscenza con la tecnologia. Il significato di hacker può essere quindi compreso attraverso un processo filologico che passa dal generale, che è la tecnologia; al particolare, che è il rapporto con la tecnologia; di nuovo al generale che è il rapporto con la conoscenza in generale e che come tale implica una certa visione del mondo. È nel particolare che si genera la dicotomia, o meglio, l'enantiosemia del termine hacker creativo/distruttivo. Al creativo si oppone il termine distruttivo per indicare l'intrusione e la manipolazione dei sistemi altrui così come indicato nei dizionari Zingarelli ed Oxford.

Quando nuovamente si ri-generalizza, allora si perde questa ambivalenza del termine, e si riacquista un'accezione filosofica di visione del mondo. È anche intuitivo vedere come l'accezione negativa di hacker, cioè cracker, resti vincolata

al particolare e non sia in grado di assumere e riassumere un'idea nella quale sia possibile identificarsi sul piano etico. Da un punto di vista sociologico, invece, intendiamo il particolare come particolarastico ed il generale come universalistico. Quindi Hacker in senso universalistico e cracker in senso particolaristico. Questo non ci impedisce quindi di vedere il particolarismo anche come fenomeno di massa. Thepiratebay era un sito, oggi sotto sequestro, che procurava connettività peers to peers2, attraverso un sistema di indicizzazione chiamato BitTorrent che nel 2008, a suo dire, ha raggiunto i 25 milioni di connessioni univoche. In pratica ha messo in relazione 25 milioni di utenti allo scopo di scambiarsi files. Questo meccanismo avrebbe dovuto evitare conseguenze legali in quanto Thepiratebay permetteva agli utenti di scambiarsi informazioni e quindi scambiarsi file video, musicali o testuali coperti da copyright, senza memorizzarli nei propri apparati ma semplicemente mettendoli in comunicazione.

Questo sito, e molti altri, hanno costituito un vero e proprio fenomeno di massa, ma sempre e solo particolarastico, e utilitaristico, volto allo scambio di materiale digitale coperto da restrizioni sui diritto d'autore. Certo non mancano teorie e filosofie e tentativi di legittimazione di questo agire che però si scontra con una contraddizione insormontabile: il rapporto che il cracker stabilisce con l'oggetto “desiderato” è molto diverso da quello che l'hacker stabilisce con l'oggetto “sacro”. Si tratta in questo caso di un oggetto, ad esempio un file musicale, il cui autore è estraneo e contrario a questo modo di agire. Nel campo semantico hacker le accezioni sono totalmente rovesciate. Non ci si scambiano cose che appartengono a chi non condivide l'idea di condivisione, è ciò costituisce una qualità intrinseca del prodotto stesso. Ed è questo che fa la differenza sostanziale tra open source e software libero, il software libero ha quindi ha accentuato maggiormente la sua dimensione di condivisione ed ha elaborato una sua filosofia e visione del mondo coerente, cosa che non è riuscita a fare né l'open source, né il fenomeno, quand'anche di massa, della pirateria. L'accezione software libero ha quindi una dimensione etica coerente che si dispiega attraverso la professione.

Nonostante questa coerenza interna al fenomeno del software libero resta un aspetto critico non di poco conto. Definire un ideal-tipo causativo del software libero comporta preliminarmente, come si è visto, un grosso sforzo epistemologico e filologico. La difficoltà principale sta nel riuscire a trovare i termini adatti. Lo stesso termine hacker nella sua accezione positiva può anche non convincere, ma purtroppo non esiste nulla di meglio. Nella lingua inglese, il

significato letterale di to hack da cui deriva il termine hacker di per sé evoca qualcosa di distruttivo. L'espressione inglese software libero è spesso fraintesa con software gratuito. Anche questa difficoltà in fin dei conti è di per se stessa indicativa: i sistemi sociali di comunicazione non hanno ancora elaborato una grammatica adeguata a descrivere questo fenomeno.

Il problema delle definizioni non è banale nemmeno internamente alle community. Nell'intervista a Stefano Zacchiroli, leader mondiale di Debian Project del 17 agosto 2010, egli ribadisce spesso il termine geek, in particolare quando gli viene chiesto di commentare una frase di Eric Steven Raymond che conteneva il termine Hacker. Questo potrebbe essere significativo di una presa di distanza in particolare da un termine inflazionato e che, come si è visto, assume significati negativi. Zacchiroli fa delle precisazioni molto puntali sulle diverse visioni, distingue nettamente tra software libero e open source e contestualmente insiste sul termine geek. Esistono quindi correnti di pensiero, cioè quella che fa riferimento alla Free Software Foundation e quella che fa riferimento all'Open Source Definitation. La Free Software Foundation nasce nel 1985, mentre l'Open Source Definition nasce nel 1997. Mentre la FSF nasce da una tensione etica che esplicita principi di libertà, l'OSD è orientata agli aspetti tecnici e commerciali che permettono l'evolversi del progetto e come, si evince dal primo articolo, l'elemento essenziale è il codice aperto. La FSF è fondata da Richard Stallman, la OSD da Eric Steven Raymond entrambi statunitensi. Richard Stallman imputa alla OSD di evitare deliberatamente il termine libertà che è invece centrale nella FSF. Debian si inserisce nella tradizione FSF. Quindi la contrapposizione del termine geek con il termine hacker risponde ad una necessità di distinzione da OSD e Eric Steven Raymond che nei suoi scritti ha storicamente insistito molto su tale termine, la stessa grammatica del jergon file mantenuto da Eric Steven Raymond ruota attorno al concetto di hacker.

Questi due aspetti del software libero e dell'open source sono relativi a due concezioni etiche. Il termine libero indica con più precisione e forza una visione del mondo, in definitiva una più completa visione etico/professionale mentre l'aspetto etico rimane più attenuato nel caso di OSD. Entrambi queste visioni sono utili alla definizione ideal-tipica. Ciò che principalmente è importante notare è come sussista una tensione morale e questa la si evince anche da Eric Steven Raymond:

«Chiunque possa darti degli ordini, può fermati dal risolvere problemi dai quali sei affascinato»

Nell'intervista con Stefano Zacchiroli, uno dei miei scopi era di verificare questa tensione morale, ed in effetti durante la presentazione, dopo aver detto che facevo il programmatore, Zacchiroli mi chiede:

[...] e sei riuscito a lavorare sul software libero ...

Riuscire può significare essere tecnicamente in grado di, ma visto che la mission del software libero è quello di eliminare le barriere, il senso deve essere ricercato sugli aspetti organizzativi e cioè sul riuscire in qualche modo a vincere le diffidenze organizzative, nei confronti di una dimensione di valori rappresentati dal software libero e in qualche modo altri rispetto a quelli comunemente accettati. La conferma di questo avviene nel proseguo dell'intervista quando zacchiroli dice:

[...] è una visione dove la libertà del software è un valore morale come giustamente hai detto tu e denota tutto un insieme di valori che, io personalmente, ma molta gente in Debian usa per valutare se qualcosa è giusto o no

Naturalmente per quanto attiene a Richard Stallman abbiamo già riportato i suoi quattro principi sulle libertà digitali. Rimane da dire qualcosa in più rispetto ai partecipanti. Andrea 19 anni, appena iscritto ad informatica all'università, nell'intervista del 1 settembre 2010, fa trasparire spesso anche un impegno nel sociale e nel volontariato oltre all'impegno nei Linux User Groups. Questo può essere frequente in qualsiasi ragazzo di quest'età, ma ciò che Andrea ribadisce più volte è il dovere di impegnarsi, e puntualizza il venir meno di questo preciso impegno da parte di alcuni, e come siano importanti all'interno dei LUG le relazioni di aiuto con chiunque le chieda. Esiste una chiara connotazione di una dimensione professionale ed etica, il fatto di essere volontari non fa venir meno il dovere di essere competenti e professionali anche nelle relazioni di aiuto. Questo aspetto diviene particolarmente interessante volendo ora riprendere quanto già anticipato (cfr 1.2) , cioè il sussistere di un etica professionale del tutto simile a quella indagata da Weber. Un'altra affermazione di Zacchiroli, centrale a questo proposito, è la seguente:

[...] gli utenti devono essere messi nelle condizioni di vedere, toccare, modificare, redistribuire il software che usa

così come per Lutero e per la riforma protestante gli uomini devono vedere con i loro ciò che Dio ha da Dirgli. Per questo motivo vengono stampati pamphlet, per questo la gente deve saper leggere e scrivere. La stessa tensione

etica sussiste nel software libero, gli utenti devono essere messi nella condizione di vedere ciò che il Dio/community ha da dirgli. La solidarietà, l'importanza dell'aiuto tecnico a chi chiede supporto rientra quindi in uno schema semantico di evangelizzazione. Il passaggio da software proprietario a software libero viene accompagnato. Come si è già osservato più volte nel corso di questo studio, il sacro in questo caso non è riconducibile all'elaborazione di una conoscenza teologica che proviene da una visione del mondo, ma una visione del mondo che proviene dall'elaborazione di una conoscenza tecnologica. Questa disposizione all'evangelizzazione è operante anche sull'economia di mercato a conferma dell'interpretazione tra etica e professione:

[...] 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. (Michael Tiemann – Voices from the open source revolution - 2000)

La razionalità si articola su due piani, quello strettamente tecnico legato alla logica circuitale e all'intelligenza artificiale e quello socio-economico legato, come abbiamo visto, all'allargamento del mercato. Il secondo aspetto, anche se reale o quantomeno plausibile, non è così scontato. Questo ipotetico allargamento si contrappone tanto al sussistere di qualsiasi logica monopolistica, quanto ad un'idea scientifica di divisione del lavoro e di produzione in serie. In particolare ciò che è significativo dell'intervista a Zacchiroli del 17 agosto 2010, è quando afferma:

[...] Io sono particolarmente allergico ai diagrammi Gannt perché c'ho avuto a che fare per progetti di ricerca europei e trovo che siamo, come dire …. , un po' troppo constraining, cioè come dire che mettano troppi paletti per riuscire a rappresentare cosa accade nella realtà. Penso, questa è l'idea di base, poi ci sono le eccezioni, e il diagramma non è bravo per niente a rappresentare le eccezioni.

I diagrammi Gannt rappresentano in modo sintetico le fasi di produzione del software, il termine constraining è preso dal linguaggio informatico ed indica una condizione normativa, di imposizione di una regola al verificarsi di un dato evento del sistema. Partendo da questo punto Zacchiroli espone il modo di produzione del software ma soprattutto indica come sia indispensabile un certo grado di autonomia degli sviluppatori Debian. In pratica il mercato si può allargare ma resta critica l'economia di scala, cioè non si può stampare moneta. Il mercato si espande ma i vantaggi sono distribuiti (collettivi). Ciò che può tenere unita una 'si fatta community che non garantisce grosse fortune, o meglio, ciò di cui è necessario, è proprio una dimensione spirituale condivisa ed accettata. In sintesi è la dimensione etica ciò che sostiene l'economia, e pertanto si tratta di un agire etico razionale, razionale in quanto consapevolmente orientato al valore, non sinottico come quello dell'imprenditore calvinista che separa le attività domestiche dalla fabbrica, la contabilità dell'impresa da quella familiare, ma incrementale ed integrativa. Progettare e produrre software richiede una dedizione particolare, non solo in termini temporali ma anche in termini emotivi e morali. Richiede ethos, capacità di concentrazione, motivazione intrinseca, se si deve compensare la minor prospettiva di lauti guadagni. Michael Tiemann (2000) riporta lo scetticismo che riscontra nel mercato nel momento in cui decide di fondare una grande azienda basata sulla produzione Open Source:

Buon senso: Perché un cliente dovrebbe pagare per un vantaggio concorrenziale?

Economia di scala: Come può un’attività di servizi godere degli effetti dell’economia di scala?

Durata: La Cygnus sarà sempre a disposizione quando i clienti ne avranno bisogno?

Profitto: Come può l’open source essere proficuo?

Gestibilità: Come si deve gestire l’open source per assicurare sempre la qualità?

Possibilità di investimento: Come può attrarre gli investitori un’azienda che non dispone nemmeno di software IP[1]?

Nonostante sia possibile fare business, nonostante la stessa Cygnus abbia mostrato una crescita maggiore rispetto alle altre software house delle Silicon Valley impegnate sul fronte del software proprietario, resta il fatto che tale scetticismo non è fondato su pregiudizi, ma come si è visto nel terzo capitolo, sulla reale possibilità di industrializzare la produzione di software. Il modello organizzativo e di produzione buttom up del software libero occupa gli spazi lasciati liberi dal software proprietario prima di porsi come reale alternativa allo stesso software proprietario. Gli spazi ampi lasciati liberi dal software proprietario corrispondono a bisogni che l'economia di scala non è in grado di soddisfare, bisogni particolari su cui si dispiega l'open source. Di per se l'open source come tale non è sufficiente, deve essere sostenuto da una dimensione etica che ricomprenda il rapporto con gli altri e con la communty in genere. È necessaria una sorta di estensione sociale ed etica dell'Open Source, cioè il Software Libero. In altri termini sono i valori etici del software libero che rendono possibile l'open source. L'open source da solo non è autosufficiente. Le motivazioni intrinseche nascono in un contesto di software libero.

Il software proprietario lascia irrisolti tutti quei bisogni che l'industrializzazione del software non può soddisfare. È principalmente l'ethos degli hacker che se ne prende carico, in quanto rappresentano una fonte di gratificazione cognitiva intrinseca da un lato ed emotiva/morale dall'altro. Addentrandosi su questi aspetti si potrebbero evidenziare fenomeni di omologazioni operanti soprattutto a livello di governance, in cui a rischio sono soprattutto gli enti locali sub-nazionali che rischiano di vedersi uniformati, sul piano organizzativo, perdendo così le loro caratteristiche di autonomia in nome dell'innovazione tecnologica intesa in senso industriale. Esula dagli scopi di

questa ricerca misurare quanto l'informatica industriale globale eserciti una certa pressione sugli assetti organizzativi, locali ma è facilmente intuibile come ciò costituisca un rischio glocale (Robertson 1995, 1999) attuale.

Quindi per soddisfare bisogni particolari su scala globale è necessaria, ad un livello macro, una dimensione etica, dove sono operanti valori espliciti, ed, a livelo micro, ethos (Bourdieu, 2001), relativo a valori impliciti che sostengono un agire autonomo ed altro rispetto rispetto all'agire dell'agente economico razionale classico. Il termine che propone Zacchiroli, geek, esprime molto bene una parte di ethos, esprime un aspetto di questo ideal-tipo ma non lo descrive in modo esaustivo. È importante tenere assieme entrambi gli aspetti, sia macro che micro, in modo coerente. Il livello micro si riferisce ai sistemi cognitivi dotati di peculiarità che di volta in volta vengono definiti come Stato di flusso (Csikszentmihaly, 2005) dalla psicologia, funzioni frontali dalle neuroscienze, ethos (Bourdieu, 2001) dalla sociologia.

Lo stato di flusso si riferisce ad uno stato di coscienza in cui l'individuo è completamente assorto in un'attività sportiva o intellettuale particolarmente gratificante. Il grado di stress è ridotto o compensato da accentuate capacità di mastering e di copying, cioè di competenza. Lo stato di flusso viene usato per spiegare le motivazioni intrinseche in quanto l'esperienza della gratificazione è ciò che motiva ad affrontare problemi sempre più complessi.

Le funzioni frontali, collocate anatomicamente nel cervello nei lobi dell'area sovra-occipitale, sono quelle che permettono di integrare affettività e cognitività (Antonio Damasio, 1995). Pazienti che subiscono traumi ai lobi pre-frontali, così come la lobotomia praticata ai criminali violenti negli '40 e '50 del '900, hanno compromessa l'affettività, la pianificazione, la capacità di controllo delle varie situazioni ambientali, diventano inerti, anaffettivi, privi completamente di interesse per l'ambiente circostante. Conservano invece la capacità di calcolo e di ragionamento logico. Questa stessa sintomatologia è la stessa che si riscontra, in modo più o meno accentuato, nella sindrome di Asperger che appartiene alla categoria dell'autismo. Quindi i lobi pre-frontali sono in qualche modo la nostra interfaccia con l'ambiente, integrando emotività e cognitività ci permettono di essere sensibili a ciò che ci circonda e agli altri e quindi di stabilire delle relazioni. Il loro venir meno, e quindi il venir meno di competenze emotive e relazionali con l'ambiente, viene compensato con una maggiore capacità di ragionamento e memorizzazione fine a se stesso e di tipo autistico.

Le neuroscienze ci forniscono quindi i due termini del problema, del rapporto tra ciò che Luhmann chiama sistemi cognitivi e sistemi di comunicazione (sistemi sociali): da un lato l'alto grado di connessione di questi (attraverso i lobi pre-frontali), dall'altro l'isolamento del sistema cognitivo da quello comunicativo. Paradossalmente entrambi hanno l'effetto di aumentare le capacità cognitive di tipo logico e mnemonico. Quindi le capacità cognitive di tipo logico e mnemonico vengono accentuate da carenza o accentuazione delle competenze (funzionalità) emotive. Vi è una sorta di intrigante coerenza tra il paradigma di industrializzazione del software che prevede la separazione tra progettazione e sviluppo del software e la separazione tra emotività e cognizione. Diversamente nel software libero, e nel paradigma buttom up, è plausibile un maggiore grado di integrazione tra emotività e cognizione. Così viene riportato un discorso di Linus Torvalds da Glyn Moody (2002):

[...] ... scrivo codice con metodo simile a quello a quello di Richard Stallman, per lui la carta non serve. Se sto lavorando su qualcosa di complicato o devo riflettere su precisi dettagli, raramente prendo appunti. Ma in genere quello che succede è che lavoro direttamente sulla macchina. Se il progetto è complicato, prima di cominciare a elaborare il codice, per qualche settimana mi limito semplicemente a pensare a tutti i possibili aspetti e cerco di trovare la soluzione. E tutto senza scartoffie ...

Questo modo di sviluppare programmi come si vede è contrario a qualsiasi forma di separazione tra progettazione e programmazione. La carta rappresenta questa dimensione burocratica di separazione tra emotività e cognizione che impedisce la creatività. Wikipedia italiano (2 ottobre 2010) indica come la sindrome di Asperger sia spesso associata al termine geek. Si tratta di un termine non scientifico e quindi altrettanto ambiguo ed enantiosemico, quello che però evoca è una situazione di isolamento dall'ambiente e dalle relazioni che per i nostri scopi è invece fondamentale perchè spiega il fenomeno del software libero e dell'agente ideal-tipico. Tra questi due estremi, emotività e autismo, che fa la differenza da un punto di vista sociologico, oltre che la dimensione relazionale ovviamente, è l'imputazione di senso, il senso che l'attore da a ciò che fa e che ci permette di comprendere. Il senso è quindi strettamente legato alle relazioni, ai valori, alla lettura che gli attori danno della situazione e questo dipende, per così dire dal contesto, dalle cornici entro le quali si dispiega il fenomeno in cui si riconosce quella che Shutz chiama provincia finita di significato. La provincia finita di significato del software libero non è delimitata da confini fisici,

nemmeno ormai dalla lingua che si parla, la sua estensione geografica è globale, come si è visto è un fenomeno della globalizzazione.

La delimitazione di questo ambito di significato è data dalla condivisione di schemi di senso, quindi valori accettati collettivamente che ad un minore grado di astrazioni si sintetizzano in regole costitutive. L'open source, ed il software libero in particolare hanno un vantaggio notevole rispetto ad altre tipologie di relazione, cioè la verifica continua e collettiva della correttezza degli enunciati attraverso meccanismi che trascendono i singoli e che sono verificabili in modo immediato ed asettico. Chiunque può far valere le proprie ragioni al di la dei pregiudizi, delle appartenenze, dell'empatia e delle ideologie. Questa garanzia per molti aspetti unica, riscontrabile nelle comunità scientifiche, è ciò che appaga il bisogno di giustizia. Non avere altre possibilità che riconoscere i propri limiti ed errori di fronte all'evidenza è ciò consente maggiore consapevolezza. Chi decide di partecipare accetta anche i rischi, ad esempio il rischio che può correre un luminare dei sistemi operativi come Andrew S. Tanenbaum di vedersi superare da uno studente come Linus Torvalds. Per contro Torvalds riconosce di avere usato il manuale Tanenbaum, anche se a suo modo. Partecipare può essere rischioso e per farlo è necessario un accentuato senso della giustizia.

Importante è anche l'esperienza della condivisione di un sentire emotivo comune, anti-cartesiano (Damasio, 1995) orientato alla soluzione collettiva di problemi cognitivi. In questo senso il codice assume funzione di medium che nelle relazioni mediate può addirittura compensare il linguaggio non verbale. Il codice pulito ed ordinato, ben indentato[2], comprensibile e commentato[3], pronto ad essere reso pubblico e che contiene soluzioni eleganti possiede una sua valenza estetica e comunicativa che va oltre la sua utilità. Scoprirsi cognitivamente uguali è un'esperienza emotivamente forte in cui non è sufficiente integrare emotività e cognizione attraverso i lobi pre-frontali, è anche necessario integrare tra loro gli individui portatori di queste peculiarità, nel caso del software libero, attraverso il codice sorgente.

Il concetto di provincia finita di significato porta in sé una minaccia, cioè il pensiero di gruppo. Il pensiero di gruppo è ciò che inibisce proposte innovative. Zacchiroli durante l'intervista del 17 agosto 2010 sostiene che un metodo efficace per far valere le proprie ragioni è quello di far vedere il codice. Questo ovviamente implica l'apertura del codice, e quindi l'Open Source è ciò che permette tecnicamente il passaggio dal pensiero di gruppo all'intelligenza collettiva, o se vogliamo, dalla provincia finita di significato alla provincia

riflessiva di significato, dove la riflessività collettiva ha significato in sé, è una forma di garanzia che trascende i singoli. Dal punto di vista razionale si tratta di un passaggio dalla razionalità sinottica alla razionalità incrementale. La razionalità operante in questo fenomeno non è più quella sinottica e burocratica dello spirito capitalista indagato da Weber, bensì quella incrementale e riflessiva.

Mentre Weber divide nettamente tra agire razionale ed agire affettivo, oggigiorno le neuroscienze evidenziano come questi aspetti non siano così separabili. Damasio afferma che sono proprio i processi affettivi ad integrare olisticamente razionalità, emotività e creatività. Rispetto a Weber si invertono i termini del problema, non più quindi un agire religioso che determina un comportamento razionale, ma al contrario, un agire razionale/emotivo integrato che determina l'esperienza trascendentale che va oltre il singolo e consente un'etica condivisa. Ma ciò che è importante è la rilevanza sociologica, nel senso relazionale, dell'intelligenza emotiva. Tutto ciò si realizza nei sistemi comunicativi, in cui è necessario esporsi alle perturbazioni delle relazioni, i sistemi cognitivi non sono autosufficienti, l'autosufficienza cognitiva assume significato di autismo, di mancanza di senso, di parcellizzazione del lavoro e di alienazione.

È in questa provincia riflessiva di significato che agisce l'attore ideal-tipico che si è definito indicato come hacker. Tale hacker è consapevole delle sue potenzialità, fa esperienza dei suoi successi individuali e collettivi, riscontra l'evidenza che è possibile “cambiare il mondo” attraverso la professione, verifica la coerenza dei valori di libertà implicati, da molta importanza al rapporto con gli utenti, adepti o neofiti e attraverso queste relazioni verifica il suo “stato di grazia” già in questo mondo; è portato ad evangelizzare, non sempre attraverso sermoni, anzi spesso attraverso la dimensione simbolica delle sue opere. Tutto ciò si dispiega in modo universalistico e globale, senza intermediazione e senza paternalismo come forma di ascesi globale.

In conclusione è necessario un avvertimento, l'obiettivo è stato quello di costruire un ideal-tipo che vede accentuate determinate caratteristiche in modo unilaterale e che non è osservabile, nel suo stato puro, nella realtà. Questo ideal-tipo ci fornisce i termini del problema portati ai suoi estremi e ci permette di misurare in modo euristico quanto la realtà si scosti da questo termine di paragone. Inoltre queste caratteristiche non sono peculiarità del software libero, ma si esprimono nel software libero innanzi tutto perché sono peculiarità umane. Se nel 1962 fu possibile individuare gli scienziati (Watson, Crick e Wilkins, 2009)

a cui dare il premio nobel per la scoperta della struttura molecolare del DNA, altrettanto non si può fare oggi per il progetto del genoma umano, portato avanti da una community di scienziati molto vasta.

Note

1. ↑ Intellectual property – proprietà intellettuale
2. ↑ Indentazione è un modo di impaginare il codice in modo tale che le istruzione più interne al programma si trovino ad una maggiore distanza dal margine sinistro.
3. ↑ Note che vengono inserite tra le istruzioni e che vengono escluse in fase di esecuzione ma che forniscono spiegazioni utili in caso di revisione del codice.

Un algoritmo ben costruito, conciso, magari con soluzioni ricorsive eleganti, scritto in modo ordinato ha un suo fascino per chi è stato socializzato all'utilizzo di determinati mezzi di produzione e che ha un determinato capitale culturale. Lo stile di vita, il gusto sono disposizioni che si acquisiscono con l'interiorizzazione (Boudieau, 2001), attraverso la particolare forma di configurazione delle relazioni operanti nel proprio campo.

```
/*
 Given a binary tree, return true if a node
 with the target data is found in the tree. Recurs
 down the tree, chooses the left or right
 branch by comparing the target to each node.
*/
static int lookup(struct node* node, int target) {
  // 1. Base case == empty tree
  // in that case, the target is not found so return false
  if (node == NULL) {
    return(false);
  }
  else {
    // 2. see if found here
    if (target == node->data) return(true);
    else {
      // 3. otherwise recur down the correct subtree
      if (target < node->data) return(lookup(node->left, target));
      else return(lookup(node->right, target));
    }
  }
}
```

Si tratta di vedere cose che chi non ha quel determinato capitale culturale non può vedere. Lo si può apprezzare come un'opera d'arte, o semplicemente come un bel pezzo di artigianato, così come un carpentiere riconosce un mobile fatto bene, o un sommelier riconosce un buon equilibrio tra acidità e astringenza in un vino. Vedere il codice implica, per un hacker, prima ancora di eseguirlo sul computer, eseguirlo interiormente, significa capirne il senso, la sua utilità. Eseguirlo interiormente significa anticiparne il risultato, ed eventualmente anche debugarlo interiormente, cioè prevederne i difetti di funzionamento. Possedere quel determinato capitale culturale significa possedere intellettualmente i mezzi di produzione del software. Benché Weber nel capitolo di introduzione al problema del suo studio sull'etica protestante e lo spirito del capitalismo non imputi alcuna dimensione simbolica ai manufatti e quindi ai mezzi di produzione ne mostra comunque la connessione con la dimensione religiosa in un breve ma significativo passaggio dove dice:

[...] L'avversione e la persecuzione che i lavoratori metodisti subirono da parte dei loro compagni di lavoro nel secolo XVIII non furono solamente e preventivamente in rapporto con le loro eccentricità religiose (l'Inghilterra ne aveva conosciute molte, e più vistose), come suggerisce già quella distruzione dei loro strumenti di lavoro che ritorna così spesso nelle cronache del tempo, - ma era connessa con la «laboriosità» loro specifica, come si direbbe oggi. (Weber ,1997, p. 86)

L'obbiettivo quindi non è solo l'atteggiamento religioso ma il "Beruf" nel suo complesso la cui espressione simbolica sono gli attrezzi da lavoro. Attraverso gli attrezzi da lavoro gli avversari dei metodisti "individuano" i loro nemici, così come è plausibile che i metodisti si identifichino con i loro attrezzi da lavoro. Così un hacker si identifica con il codice, riconosce nel codice i suoi "fratelli", attraverso il codice si possono misurare le competenze degli altri partecipanti. Il codice aperto è un potente mezzo simbolico e comunicativo che fa tutta la differenza.

Il codice si esprime in modo testuale, non si esprime con icone. Le prime interfacce grafiche della Xerox nel 1981 montate sullo Xerox Star che costava 75.000 $ avevano lo scopo di permettere all'utente di poter far eseguire al sistema una serie di compiti senza bisogno di una conoscenza approfondita sul funzionamento del sistema. Gli hacker ovviamente non necessitano di rinunciare a questa conoscenza perché è proprio sulla base di questa conoscenza che si crea identità. Questo ovviamente comporta un concetto molto importante dal punto di vista del rapporto con i mezzi di produzione, con l'informatica nello specifico, ma anche con il trascendente. Questo rapporto può essere mediato o diretto. Il rapporto diretto comporta necessariamente la conoscenza. Nel caso religioso della riforma luterana i fedeli devono saper leggere per vedere con i loro occhi ciò Dio ha da dirgli attraverso la Bibbia. Nel caso del software libero gli utenti devono poter accedere ai sorgenti testuali. Weber (1997) parla esplicitamente di riappropriazione della salvezza attraverso la diffusione della parola rivelata. Per il software libero si tratta di riappropriazione della tecnologia attraverso la diffusione del codice aperto. La centralità della questione è il processo di riappropriazione e la rivelazione della parola.

La riforma luterana si scaglia contro la mediazione rappresentata dall'istituto della confessione attraverso il quale si vendono le indulgenze, la conoscenza deve avvenire attraverso i testi sacri e non attraverso le icone. La mediazione della GUI (Graphic User Interface) è ciò che impedisce l'accesso ai meccanismi di

funzionamento dei sistemi. Sia nella riforma protestante che nel software libero la strategia è quella di consegnare il capitale simbolico, la Bibbia ai credenti nel caso della riforma, il codice sorgente agli utenti nel caso del software libero. Con l'avvento del software libero, ciò che Weber aveva osservato nella riforma protestante riemerge come strategia per organizzare un'alternativa ad una tradizione operante attraverso strutture di potere consolidate. Si tratta, nell'uno come nell'altro caso, di far riappropriare il capitale simbolico all'uomo comune, di volta in volta fedele o utente e quindi richiamare i singoli alla responsabilità di custodirlo, di migliorarlo nel caso del software libero e di renderlo socialmente operativo. Ciò che sembra essere rilevante in questo approccio fenomenologico è l'emergere di una dinamica non più specificatamente propria della riforma protestante, ma in grado di riemergere anche in contesti del tutto diversi ed in tempi diversi. Si è più volte fatto cenno ad un agire ascetico religioso intra-mondano non per imputare una dimensione religiosa al fenomeno del software libero ma perché le categorie Weberiane spiegano ancor oggi queste dinamiche anche se non sono sistematizzate in una prospettiva generalizzabile poiché ciò contraddirebbe lo stesso approccio fenomenologico che le osserva come fenomeni unici.

I valori che Himanen (2001) riporta come appartenenti all'open source sono: passione, libertà, coscienza sociale, verità, antifascismo, anti-corruzione, lotta contro l'alienazione dell'uomo, eguaglianza sociale, accesso libero all'informazione (cultura libera), valore sociale (riconoscenza tra simili), accessibilità alla rete, attivismo, responsabilità, creatività. L'imputazione di alcuni di questi valori al fenomeno Open Source può però essere dialettico, in effetti si può notare come molti di questi siano assolutamente coerenti. La libertà e la sensibilità nei confronti dei diritti degli utenti è un punto qualificante del movimento del Software Libero e del suo leader Richard Stallman. La passione è assolutamente coerente con lo stato di flusso visto precedentemente. La verità può essere ricondotta ad un concetto di verità scientifica falsificabile in senso popperiano che si esprime attraverso l'apertura del codice. La lotta contro l'alienazione può essere ricondotta ad un principio di integrazione intellettiva collettiva contraria di conseguenza ad una visione parcellizzata ed industriale del software. Il valore sociale, come emerge più volte in questo studio, fornisce molta della motivazione. La creatività è legata alle competenze emotive, cognitive e progettuali. L'attivismo è evidente. La responsabilità rientra in una dimensione professionale. L'accesso libero all'informazione costituisce un fattore importante per limitare l'entropia, come si vedrà più avanti.

Tutto questo non entra esplicitamente nelle attività delle diverse communities impegnate nell'open source e nel software libero. Anche nei documenti costituivi della Free Software Foundation, dell'Open Source Defination, nella costituzione sociale Debian i cenni ai valori morali ed etici che riporta Himanen (2001) non sono così esaustivi. Si tratta di documenti molto brevi, lunghi una o due pagine, concisi così com'è lo stile hacker. Una apparente contraddizione sta nel fatto che nei forum di discussione questi elementi non emergono, non vengono esplicitati perché eccetto la peculiarità del valore liberale nel software libero, ribadito più volte e scritto nei documenti costitutivi, un valore implicito che costituisce una norma altrettanto implicita è quella di non mescolare questione tecniche con questioni politiche o morali. I valori che richiama Himanen non sono richiesti come condizioni per partecipare, emergono in modo puntiforme, con diverse accentuazione nelle diverse epoche e nei diversi progetti. Ciò che resta costante è invece la dimensione professionale e le competenze, anche nel caso del progetto Debian dove la sua costituzione, più di ogni altro progetto si rifà a valori democratici[1], resta centrale il possesso di competenze. Il programma elettorale di Stefano Zacchiroli eletto Project Leader Debian nel 2010 annovera, oltre alle

intenzioni strategiche, anche una serie di competenza in termini di partecipazione a diversi progetti.

La partecipazione riguarda per lo più attività di diffusione del software libero, soluzioni tecniche e organizzazione. La discussione su tematiche valoriali che riguardano argomenti con possibili connotazioni politiche rischiano di disaggregare la community, rischiano di connotare politicamente il software libero, di creare conflitti interni e di distogliere dagli obbiettivi principali. C'è quindi una norma implicita che impone di attenersi a temi circoscritti alla creazione e diffusione del software libero. Si tratta di un agire professionale e discreto, chi vuole rendersi utile lo deve fare con umiltà. Questo è ciò che consente una certa adattabilità del software libero, di mantenersi aperto alle diverse culture, confessioni e appartenenze perché il suo obbiettivo principale è la diffusione. Oltre a questa norma implicita è operante anche una regola costitutiva che riguarda il funzionamento dello strumento software attorno al quale si delinea il progetto con dinamiche bottom-up. Questa è il criterio della funzionalità, quindi una soluzione è giusta o sbagliata in base al fatto che funzioni o meno.

Se si dovesse individuare un leader di un tale movimento esteso, diffuso e policentrico, che comprenda l'open source e il software libero, questo è senz'altro Linus Torvalds. La sua capacità di leadership non nasce tanto dalle sue abilità oratorie, morali o politiche ma da ciò che è stato in grado di fare in termini di competenze tecniche, sviluppando un kernel funzionante, e organizzative aggregando attorno a questo progetto programmatori da tutto il mondo. Ha quindi dimostrato competenze tecniche e organizzative non comuni, o per meglio dire, Weberianamente, presenta quelle qualità eccezionali necessarie ad un leader perché sia tale. Il sistema operativo Gnu/Linux che nasce con questa dinamica è assolutamente centrale così come è centrale la leadership di Linus Torvalds che ne richiama il nome. Gnu/Linux è anche un pseudo-acronimo che si riferisce LINus UniX. L'identificazione di Linus Torvalds con il sistema operativo Gnu/Linux è resa ancora più evidente dal fatto che venga spesso indicato con il sopranome di Linux Torvalds. Questa centralità è inevitabile dal momento in cui non esistono, per il momento, sistemi operativi open source alternativi. Anche se val pena menzionare il progetto MINIX iniziato da Andrew S. Tanenbaum famoso per la controversia con lo stesso Linus Torvalds e che viene rilasciato con una licenza di tipo open source.

Il processo di istituzionalizzazione di un progetto che ha successo è in genere molto rapido. Nel caso del kernel Gnu/Linux dopo la diffusione nella rete delle

prime versioni nei primi anni '90 del secolo scorso, in cui chiunque riesca ad implementare delle soluzioni e a risolvere problemi può partecipare, il progetto si chiude. Rimangano i programmatori della prima ora e vengono definite delle regole operative. Ciò che si diffonde non è solo il sistema operativo, ma l'idea che ci sta dietro che viene imitata. Altri progetti nascono con la stessa dinamiche e, anche se l'implementazione del kernel resta riservata, si iniziano progetti ad un livello più alto. Si tratta delle interfacce grafiche di cui le più famose sono KDE, di derivazione commerciale poi “liberata”2, e GNOME di derivazione liberale. Su queste interfacce grafiche, che hanno l'importanza, finalmente, di costituire una reale alternativa al sistema Windows, si inseriscono poi altri progetti di implementazione che completano il sistema operativo e lo rendono usabile e completo anche per gli utenti comuni. Si tratta ad esempio di Debian, Fedora, Ubunto, Mint e molte altre. Intervengono finanziamenti di società come IBM, Sun Microsystems, Hewlett-Packard, Red Hat e Novell.

Questa frammentazione del software libero è una conseguenza del processo di istituzionalizzazione. Nel momento in cui viene mobilitata una notevole quantità di energia questa non può essere assorbita da un solo progetto e quindi è necessario che la partecipazione venga limitata e la produzione regolamentata. L'energia lasciata libera si riorganizza in altri progetti satellitari i quali a loro volta si istituzionalizzano. La differenza di questi altri progetti sta nel fatto che per quanto attiene il kernel Gnu/Linux, cioè il progetto principale, il leader resta lo stesso, mentre negli altri progetti i leader se ne vanno, occupano posizione di rilievo in altre società e cedono il posto a dei leader di funzione. Questo è quanto accade con Rasmus Lerdorf iniziatore del linguaggio PHP, Ian Murdock fondatore di Debian e autore della prima costituzione Debian e così molti altri. Il processo di istituzionalizzazione comporta quindi per il progetto principale del kernel il rafforzamento della leadership carismatica e per gli altri progetti satelliti il passaggio a leadership di funzione.

L'istituzionalizzazione dei progetti, specie se molto estesi e di successo, comportano anche la differenziazione di organi e funzioni interne riconducibili al modello AGIL di Parsons, Così Stefano Zacchiroli nell'intervista del 17 agosto 2010 descrive la struttura organizzativa Debian:

Quindi ci sono tutti gli strumenti per far funzionare la costituzione Debian:

- 1. Il project leader, ci sono i suoi delegati. Il segretario e tutti i macchinari che servono per fare andare avanti i meccanismi di voto.

- 2. Il comitato tecnico, nel caso ci siano conflitti di natura tecnica tra persone all'interno del progetto. È una specie di, come dire, tribunale a cui ci si può appellare per risolvere conflitti di natura tecnica.

- 3. Quelli che chiamiamo core teams, che hanno dei ruoli specifici e che sono in quanto su carta delegati del proprio project leader, ma che hanno dei ruoli politici abbastanza importanti anche se vuoi di divisione dei poteri.

- 4. Il cosiddetto DAM, Debian Account Manager, che sono un gruppo di persone che decide chi può diventare sviluppatore Debian e chi no.

- 5. C'è il gruppo di DSA, Debbian System Administrator, che sono quelli che mantengono tutte le macchine, tutti i server del progetto Debian in giro per il mondo, ed in generale tutta la infrastruttura tecnica.

- 6. Un gruppo chiamato FTP masters, che sono quelli responsabili su cosa entra nell'archivio software Debian e cosa no, e che hanno un potere molto significativo, nel senso che sono quelli che decidono al momento quali sono le licenza che sono accettabili nell'archivio Debian, decidono quando un pacchetto non raggiunge i criteri minimi di qualità per entrare nell'archivio.

- 7. C'è un gruppo di Keyring Manager che sono quelli che gestiscono il gruppo di chiavi e di firme digitali che servono a firmare i pacchetti che possono entrare nell'archivio.

- 8. Il release team che è un gruppo di persone responsabili di valutare il programma verso una release Debiano che è l'outcome principale di un progetto come Debian, fare una nuova release.

Gli FTP masters e i Debbian System Administrator, ad esempio svolgono una funzione economica (adaptetion), approvvigionano beni e servizi. La funziona politica (goal attainment) di garanzia della sicurezza interna ed esterna è svolta da Comitato tecnico, dai core teams e i Debian Account Manager. La funziona normativa (integration) è svolta dai Keyring Manager e Core teams. La funzione riproduttiva (latency) è svolta dal Release Team.

Il processo di istituzionalizzazione rappresenta anche per il software libero, come per tutte le organizzazioni, una fase critica, in cui l'accento si sposta da una fase emozionale collettiva ad una fase normativa. Ciò che diviene a rischio è l'identità, la positività del modello auto-rganizzato. Nell'intervista a Stefano

Zacchiroli emerge tutta questa preoccupazione di dover tenere le fila di un progetto che assume dimensioni globali ma che non vuole perdere la sua peculiarità di modello di libertà. Si è sostenuto che il sostegno morale, inteso come possibilità di senso che nasce dalla consapevolezza che si sta facendo qualcosa di “importante”, ed il sostegno etico, inteso come stile e atteggiamento professionale sono ciò che consentono al software libero di competere con il software proprietario sulla capacità di rispondere a bisogni.

[...] Anche le tattiche specifiche necessarie per spingere la strategia apparivano ormai chiare sin dall'inizio e furono infatti discusse nel primo incontro. Ecco le tematiche principali: Dimenticare la strategia "bottom-up"; puntare sulla strategia "top-down" Ci appariva ormai chiaro che la strategia storica seguita per Unix, vale a dire la diffusione dei concetti dal basso verso l'alto, partendo cioè dai tecnici per giungere poi a convincere i boss con argomentazioni razionali, si era rivelata un fallimento. Era una procedura ingenua, di gran lunga surclassata da Microsoft. Inoltre, l'innovazione di Netscape non avvenne in quella direzione, ma fu resa possibile proprio perché un importante personaggio di livello strategico, quale Jim Barksdale, aveva avuto l'intuizione e l'aveva imposta ai suoi subordinati. La conclusione inevitabile era che bisognava abbandonare la prima impostazione e passare a imporre le decisioni dall'alto, cercando quindi di coinvolgere in primo luogo i dirigenti delle alte sfere. (Eric Steven Raymond, 2000)

Forse quello a cui assistiamo è un processo che porterà ad un compromesso, oppure come accade spesso nel software, saranno nuovi “forks” - progetti collaterali più piccoli che si staccano da quello principale per dare vita a nuovi progetti – a farsi carico di istanze di libertà, mentre l'industria che vi investe sempre più capitali, e i processi di istituzionalizzazione colonizzano progressivamente l'open source. Ed anche in questo caso, non è detto che le organizzazioni industriali non si lascino sedurre dai modelli comunitari del software libero.

Note

1. ↑ Per liberazione si intende la trasposizione di un software proprietario a software GPL

La teoria neo-funzionalista di Luhmann sposta il processo conoscitivo dai sistemi cognitivi ai sistemi di comunicazioni operanti in condizioni di chiusura, dove sia possibile, quindi, riconoscere un sistema in un ambiente attraverso l'indicazione e l'osservazione complessa, cioè osservazione di osservazioni, in quanto la realtà non è data, è di per se inconoscibile e quindi non è di per sé osservabile direttamente. La conoscenza in qualche modo ha bisogno di dispiegarsi attraverso la condivisibilità operativa preclusa ai sistemi cognitivi. La percezione e la coscienza, di per sé non si possono trasferire. La conoscenza si dispiega all'interno di un sistema auto-poietico - cioè che riproduce necessariamente i suoi elementi ai fini della sua continuità – e come tale non può che essere autoreferenziale, cioè costruita.

Questo approccio alla conoscenza visto in termini generali è molto astratto, lo stesso concetto di sistema è “ancorato” a dei confini concettuali che devono essere indicati (definiti) di volta in volta. La questione diviene pertanto quella di individuare quel sistema che contenga il fenomeno “informatica aperta” ed il fenomeno “informatica chiusa” e quali gli aspetti della conoscenza che si dispiegano al loro interno, e come questi sistemi si compenetrino.

Rispetto ad un concetto generale o classico di conoscenza, la conoscenza tecnologica è più facilmente riconducibile, anche intuitivamente a qualcosa di costruito. Se gli approcci epistemologici che si sono susseguiti nella storia riguardano il dilemma di come sia possibile la conoscenza di un oggetto autonomo dai sistemi cognitivi, qui il problema non si pone: l'informatica, come la tecnologia in genere, riguarda indubbiamente una costruzione all'interno di un sistema, tanto che considerare la conoscenza tecnologica alla stregua della conoscenza problematica di una realtà altra rispetto a qualsiasi scelta selettiva soggettiva o quantomeno riconducibile ad una necessità di procedere nei sui confronti in modo pubblicamente accettato, può sembrare quantomai bizzarra se non irrispettosa.

Se utilizziamo gli strumenti che Luhmann ci mette a disposizione il dilemma non dovrebbe sussistere, poiché il presupposto dell'impossibilità della conoscenza della realtà esterna quand'anche sia plausibile la realtà esterna, mette qualsiasi conoscenza sullo stesso livello. Non può esistere in termini concettuali una

conoscenza più costruita di un altra dal momento che la realtà è inconoscibile, in ogni caso si tratta, secondo l'approccio di Luhman, di una costruzione radicale, cioè totale.

Resta il fatto che questo approccio ci sembra molto fecondo. È evidente un immanente relativismo della scienza informatica nel suo aspetto software autonomo, rispetto alla logica circuitale del livello hardware che risponde invece alle leggi della fisica riconducibile ad una conoscenza sottoposta ai processi di legittimazione, verificazione, riproducibilità, falsificazione e via dicendo. L'informatica si sottrae a questo processo dialettico di legittimazione o più esattamente definisce in modo molto più autonomo i criteri di validità. È radicalmente autoreferenziale e autopietica. Infatti nessuno può stabilire analiticamente, a-priori se il computer si debba spegnere con un procedimento piuttosto che con un altro. È una scelta arbitraria di chi costruisce il sistema, o di esperti di ergonomia che indicano al programmatore come procedere e quali metafore usare. Un semplice esempio: nei sistemi UNIX per visualizzare il contenuto di una cartella si usa il comando “ls”, nei sistemi DOS il comando “dir”. Nei sistemi UNIX la metafora è “case sensitive” cioè “LS” non funziona, nei sistemi DOS è invece “case insensitive”. Nei sistemi Gnu/Linux invece funzionano entrambe le matafore tanto “dir” quanto “ls” ma sola in modalità “case sensitive”. Queste differenze sussistono solo a livello di metafora, in sintesi lo stesso microprocessore controllato da un sistema Gnu/Linux non fa cose diverse se controllato da un sistema UNIX o DOS. Quello che diventa la nostra “cultura informatica” sono le metafore scelte da altri che solo per una questione di “buon senso”, e a volte neppure quello1, hanno deciso di chiamare un comando con un nome piuttosto che con un altro, di associarlo ad un tasto piuttosto che a un altro.

Alberto Cevolini nell'introduzione a “Conoscenza come costruzione” (Luhmann, 2007) spiega come:

[...] Heinz Von Foester radicalizza la differenza fra auto ed eteroreferenza sostenendo che per evitare contraddizioni non ci si dovrebbe limitare a dire che la mappa non è il territorio (Korzybski) ma che la mappa della mappa non è il territorio ..

dal punto di vista informatico la situazione è ben peggiore e si potrebbe dire che la “metafora della mappa della mappa non è la realtà”. In questo caso non funziona l'analogia con la lingua. Anche nelle diverse lingue esistono diversi

suoni per indicare le stesse cose, addirittura esistono suoni simili per indicare cose diverse: i cosiddetti falsi amici o interferenze linguistiche che sperimenta chiunque studi una lingua straniera. La differenza sostanziale verte sul fatto che la lingua, a differenza delle metafore informatiche, è frutto di una negoziazione collettiva sui significati lungo un arco evolutivo che si perde nella notte dei tempi. Per l'informatica non è così, una metafora è frutto di una scelta arbitraria di un singolo programmatore o di una software house fatta in modo del tutto autonomo, spesso in tempi stretti e in modo frenetico per rispettare tempi di consegna o per battere sul tempo la concorrenza.

Se vogliamo rendere operativa questa metafora della mappa della mappa dovremmo quindi supporre una “realtà informatica” cioè supporre che almeno sia possibile “la mappa della mappa della realtà informatica”. La risposta è positiva, la condizione è la negoziabilità sul codice. Innanzi tutto è necessario eliminare la ricorsione della metafora sulla metafora e questo avviene considerando le scienze informatiche operanti su di un livello più basso, più astratto e meno mediato, che non necessita di metafore o comunque non ha bisogno di considerarle come “conoscenza” ma semplicemente come una convenzione tra le tanti possibili. La realtà informatica diventa in, questa prospettiva, il paradigma scientifico (Thomas Kuhn, 1979) valido in quel determinato momento, spesso destinato ad essere falsificato e sostituito in tempi brevissimi, è il compromesso su cui si addensa consenso attraverso quel processo che Zacchiroli nell'intervista del 7 luglio 2010 chiama “show me the code” che ha come criterio di validità la conferma empirica del migliore funzionamento rispetto ad un dato precedente (versione precedente):

[...] cioè si fa vedere che un cambiamento di cui si sta discutendo da secoli è possibile, una volta che c'è l'evidenza in termini di codice che il cambiamento è possibile è più facile che collettivamente si decida che quella è la via giusta.

Questo ci permette di fare queste considerazioni: il software proprietario non può permettersi di condividere il codice, perché tale paradigma scientifico entra in contraddizione con il paradigma economico della limitazione delle risorse. Ma resta comunque la necessità della condivisione, almeno operativa, perché questo è ciò che a sua volta consente la domanda economica di software. Quindi nell'ambito dell'informatica proprietario l'idea di “cultura informatica” è spostato su un livello ancora più mediato e cioè sulla metafora della “mappa della mappa della mappa del territorio”; nell'ambito del software libero e dell'open source la cultura, pur ricomprendendo il livello “metaforico” fa suo anche il livello di condivisione più basso della “mappa della mappa territorio”. Zacchiroli

nell'intervista del 17 luglio 2010 evidenzia questi due livelli di condivisioni pur menzionandoli in un contesto diverso della sua esposizione:

[...] Nel caso di Debian è un caso diverso dai LUG (Linux Users Groups), se non altro per un motivo di scala, i LUG sono formati da gente che già si conosce, se qualcuno si approccia a Gnu/Linux si iscrive alla mailing list, si fa conoscere, il mese dopo o relativamente poco dopo partecipa alle riunioni locali, e quindi conosce subito le persone fisicamente, quindi non c'è quello che spesso è un problema in Debian ed il fatto che essendo un progetto su scala internazionale.

Quindi a livello di LUG avviene la condivisione delle “metafora” in un contesto locale, mentre nel contesto globale avviene la condivisione sul livello della “mappa della mappa”. Nei diversi LUG si può occasionalmente discutere di patch, di algoritmi e di bugs ma le discussioni più frequenti riguardano, oltre all'organizzazione di eventi, la richiesta di aiuto sul funzionamento e non sulla costruzione.

Un procedimento informatico può quindi essere progettato in molti modi diversi. Proprio l'approccio costruttivista consente di usare senza troppi problemi il termine “virtuale”, anzi il “virtuale” enfatizza ancora di più il concetto di costruito, significa già un reale in negativo. Alla fine abbiamo definito il virtuale, ma questo percorso è comunque utile per prospettare un campo di indagine.

Muoversi nella rete, spedire un messaggio di posta elettronica, navigare, usare i programmi di videoscrittura, di contabilità, gli accessori multimediali, i videogiochi; e per i più “colti” programmare, configurare, installare e via dicendo fa parte di un modo consueto di agire per ore, mesi, anni con schemi di senso che diventano parte integrante della nostra visione del mondo.

Ciò che invece è drammatico è l'autonomia con cui questa realtà viene costruita da altri in modo arbitrario, ma potenzialmente anche malevolo. La questione verte sul fatto che mentre la realtà costruita a cui si riferisce Luhmann riguarda una costruzione che sfugge al controllo dell'uomo perché sistemica, la realtà virtuale è costruita da altri uomini consapevolmente. Accanto alla conoscenza che possediamo costruita dalla coazione del sistema cognitivo con il sistema sociale, deteniamo una sempre maggiore quota di conoscenza costruita nell'ambito di processi pianificati che sfuggono alla dialettica della falsificazione. Il concetto di falsificazione introdotto da Popper va oltre il più semplice concetto di verificazione, è più attinente alla necessità che un paradigma, così come un

software, si esponga alla critica della comunità scientifica e questo è possibile solo con il codice aperto. Questo significa che il software chiuso, secondo un approccio popperiano non è scientifico perché si sottrae alla critica sul piano scientifico attraverso la chiusura, e alla competizione sul piano economico attraverso il monopolio. Non è quindi così avventata l'idea che il codice dei sistemi debba essere reso intelligibile e quindi aperto, che debba essere reso falsificabile secondo principio di Karl Popper. Ciò significa che la realtà virtuale, per il peso che occupa nella conoscenza all'interno dei sistemi sociali, deve essere resa accessibile alla critica altrimenti dovremmo rassegnarci ad accettare una conoscenza costruita arbitrariamente e sottratta alla riflessività collettiva.

Abbiamo già visto come sia il paradosso dell'informatica, che ha la peculiarità di essere orientata alla condivisione, a causarne la chiusura attraverso una situazione di monopolio. Innanzi tutto va precisato che secondo Maturana e Varela ciò che un sistema deve realizzare per diventare tale è la sua chiusura e quindi la sua distinzione dall'ambiente. Allo stesso tempo il sistema così differenziato non scambia informazioni con l'ambiente. L'informazione è per Luhmann costruita all'interno del sistema per auto-contatto. L'ambiente può solo perturbare il sistema ed è in base a questa perturbazione che il sistema verifica il suo stato attuale in base al suo stato precedente.

Importante è notare che questo chiusura può avvenire solo attraverso l'adozione di schemi di senso che riducono la complessità e soprattutto l'autoriferimento e quindi l'identità, la capacità del sistema di auto-indicarsi. Sono quindi gli schemi di senso che hanno la funzione di consentire l'identità che è legata alla razionalità della modernità e si esprime attraverso l'automazione dei processi di calcolo, di classificazione, di memorizzazione, di contrazione del tempo e in definitiva di controllo. L'automazione dei processi, tra le altre cose, connota anche la nostra identità di donne e uomini moderni.

Argomentando in negativo è più facile trovare dei riferimenti: Non ha senso spedire una lettera cartacea ad un amica che abita a migliaia di chilometri; non ha senso fare la fila allo sportello; non ha senso tenere la contabilità sul mastro giornale cartaceo; e così via. Non accettare queste istanze come “nonsensi” significa chiaramente porsi al di fuori del “sistema”. È chiaro che questi schemi di senso sono indotti da un pensare scientifico e tecnologico socialmente rilevante e di massa come mai esistito in precedenza che nasce dalla consapevolezza che la tecnologia lo consente. Sono schemi di senso che nascono dal sapere scientifico e si diffondono nella società per poi tornare ricorsivamente alla scienza sotto forma di richieste di automazione dei processi.

Man mano che questi schemi di senso diventano operanti attraverso la produzione di tecnologia il sistema elabora (o rielabora) la sua chiusura, creando nuove identità, creando quindi gap tecnologici, cioè culturali, intergenerazionali, a volte ridefinendo o mettendo in discussione i confini del mondo occidentale, o accelerando la globalizzazione, o isolando ancor di più alcune aree nel mondo più povero. L'india sta diventando un grosso esportatore di software, l'Irlanda diventa il paradiso dei brevetti informatici grazie ad una legislazione fiscale favorevole e via dicendo.

Sta di fatto che questa semplificazione operata attraverso il senso dell'automazione dei processi (ri)definisce il sistema. Questa semplificazione comporta l'assorbimento della complessità sociale da parte dei circuiti elettronici, almeno apparentemente, perché come si cercherà di dimostrare più avanti questa complessità non resta immagazzinata nei circuiti assieme ai dati ma viene ri-allocata in forme diverse. Per ora ciò che è importante notare è che questa semplificazione comporta a livello tecnologico l'integrazione tra i sistemi sociali e sistemi tecnologici, tra diversi sistemi sociali e tra diversi sistemi tecnologici. Diviene necessario ridurre la complessità della tecnologia o meglio limitarla, selezionare un solo sistema sistema operativo non su un presupposto qualitativo ma meramente quantitivo, in particolare unitario ed esclusivo. In questo modo il monopolio del sistema operativo unico e condiviso è ciò che consente, probabilmente assieme anche ad altri fattori, la chiusura auto-poietica e la definizione del sistema stesso:

[...] La riproduzione auto-poietica ha quindi bisogno di una adeguata omogeneità delle operazioni sistemiche, ed è quest'ultima che definisce l'unità di una determinata tipologia sistemica ... (Luhmann, 1990, p. 112)

Come si vedrà più avanti la complessità viene ri-allocata perché il sistema non riesce a risolvere la contraddizione del monopolio e quindi non raggiunge una soddisfacente coerenza interna. A dimostrazione del fatto che la selezione di un sistema operativo standard, globalmente accettato, ha a che fare con la sue caratteristiche intrinseche solo fino ad un certo punto, lo si evince da come MS-DOS diviene lo standard e quindi un monopolio. Nel 1979 IBM intende mettere in produzione il suo personal computer e per questo cerca nel mercato un sistema operativo idoneo. Tim Paterson, della Seattle Computer Product aveva già scritto il DOS-86 sulla base delle specifiche e degli standard di un altro DOS il CP/M. Microsoft compra i diritti di DOS-86 dalla Seattle Computer Product. IBM aveva intenzione di usare il sistema della Digital Research che deteneva il CP/M con anche una versione industriale multitasking, l'MP/M. CP/M era fino ad allora lo standard ma la digital non raggiunse mai un accordo con IBM. Quindi il PC-IBM II, quello che inizierà la diffusione di massa dei personal computer verrà equipaggiato con il DOS-86 di Microsoft che ne aveva acquisito i diritti dalla Seattle Computer Product. Solo più tardi nel 1982 Microsoft ne modificherà il nome da DOS-86 in MS-DOS. Questo episodio è esemplificativo di una storia molto più lunga e molto più intrecciata in cui esistono molti sistemi operativi di tipo DOS o di tipo UNIX ed altri ancora, quello che si afferma come standard è quello con cui PC-IBM II viene equipaggiato, e che corrisponde ad una scelta

contrattuale, commerciale che ha avuto esiti imprevisti. IBM aveva l'intenzione di conquistare il monopolio del mercato mondiale dei personal computers, ha ritenuto il sistema operativo che montavano i suoi PC un dettaglio componentistico non così significativo, lo si sarebbe potuto sostituire con qualsiasi altro DOS in qualsiasi momento. In realtà IBM contribuisce suo malgrado ad affermare uno standard di cui non ha il controllo del copyright. Di fatto IBM fornisce al mondo lo schema di senso tecnologico che consente la chiusura auto-poietica. Questo schema di senso doveva essere il PC-IBM II, mentre lo schema di senso diventa il DOS con cui viene equipaggiato.

Questo processo descrive come viene fissato lo schema di senso attraverso operazioni ricorsive. Di fatto non si assiste ad un autore individuale che scrive un sistema operativo migliore e che quindi ha il sopravvento. L'antenato del MS-DOS scritto da Tim Paterson implementa metafore già adottate in altri sistemi operativi. Fino ad una certa epoca si tratta di un equilibrio dinamico in un reticolo riflessivo di connessioni ricorsive, cioè auto-poietiche. Il fatto che le stesse interfacce utente, allora testuali, utilizzino gli stessi comandi e la stessa sintassi, e siano di fatto indistinguibili se non per la stampa a video del relativo detentore del copyright è molto significativo e segno di un isomorfismo dovuto ad un già operante, anche se in scala minore, schema di senso.

Coerentemente con la teoria dei sistemi di Luhmann il sistema operativo di Microsoft è un elemento prodotto dal sistema che ad un certo punto acquista una sua autonomia auto-poietica frenetica e ri-definisce il sistema ad un livello superiore. Lo schema di senso che viene fissato, la conoscenza che ne consegue è solo una delle tante possibili e questo trascende, anche in senso costruttivistico, il concetto di qualità.

Mentre la chiusura auto-poietica definisce in termini generali come si affermano i sistemi sociali informatizzati, l'auto-referenzialità ne spiega la continuità critica in seguito alla sua contraddizione interna. Il monopolio informatico svolge la funzione di apertura attraverso la chiusura:

[...] in quanto universale evolutivo, il senso è collegato con la tesi della chiusura delle formazioni sistemiche autoreferenziali. La chiusura dell'assetto autoreferenziale coincide con l'infinita apertura del mondo. (Luhmann, 1990, p. 150).

La criticità sta nel fatto che il sistema operativo dominante contiene in se il senso, definisce esso stesso il senso. Non esiste alcuna casa automobilistica che chiudendo i battenti metta in pericolo l'idea stessa di automobile. Nel settore informatico invece questo può accadere o poteva accader prima dell'avvento dell'open source. In termini più generali l'aspetto critico verte sul fatto che quando un sistema non ha una sufficiente coerenza interna l'unico modo per mantenere l'equilibrio dinamico autoreferenziale è quello di dividersi in sottosistemi più piccoli dotati di schemi di senso in grado di mantenere la riduzione della complessità.

Nella fattispecie della tecnologia informatica questo non è possibile in quanto l'unicità del sistema operativo è parte integrante del senso. È in pratica, da un diverso punto di vista, la contraddizione che riporta Ulrich Beck (1999) tra locale e globale. Se non esistesse il monopolio non potrebbe esistere chiusura operativa all'interno della società informatizzata. Da questo punto di vista ad una società informatizzata non resterebbe che accettare il monopolio, oppure rielaborare il senso magari accettando il “politeismo” dei sistemi operativi. Si tratta in effetti di un monopolio naturale in cui se vi fossero due concorrenti il profitto d'impresa sarebbe negativo. Questo si verifica nelle infrastrutture, nelle rete viaria, nelle ferrovie e via dicendo, solo che a differenza di questi è operante a livello globale e quindi non è possibile nemmeno nazionalizzarlo. Le proposte che il movimento FOSS (Free and Open Source Software) mette in campo è un radicale cambio delle regole del gioco tentando di andare oltre il tema della semplice contraddizione dell'economia globale non controllabile dai governi locali. In questa prospettiva il movimento esaurirebbe il senso nella contrapposizione politica.

Quindi non esistono alternative, perché come abbiamo già visto, più che l'antiautoritarismo che descrive Eric Steven Raymond, il vero schema di senso della cultura hacker è l'anticonvenzionalismo. È fondamentale questa differenza tra anticonvenzionalismo ed antiautoritarismo. Cambiare le regole in una prospettiva antiautoritaria significa di fatto un conflitto ampio sui principi di legittimazione in generale. Cercare di cambiare le regole in modo anticonvenzionale significa utilizzare gli strumenti in modo non convenzionale per fini diversi da quelli per cui sono stati "venduti" significa orientare il conflitto in modo più tematico. Nessuno può considerare il fatto di trasformare il proprio personal computer in un firewall[1] o in un server proxy[2] un comportamento antiautoritario, parimenti aggiustare la motocicletta con dei pezzi ricavati da una latina di birra o togliere l'etichetta di windows quando si acquista un computer e poi anche windows. Tutto questo non è né antiautoritario e nemmeno illegale, ma semplicemente anticonvenzionale. Nemmeno è un atteggiamento trasgressivo perché molti di questi comportamenti sono spesso privati, non si rivolgono intenzionalmente davanti ad un pubblico. Rimangono socialmente rilevanti perché culturalmente determinati.

Quello che il movimento FOSS fa è quello di rielaborare i contratti e le stesse licenze stravolgendo il senso dei diritti d'autore. La licenza di tipo GPL (General Pubblic License), con cui è rilasciato Gnu/Linux e migliaia di altri prodotti, è un esempio di hacking di un istituto giuridico straordinario, con lo scopo, visto dal lato sociologico, di proporre uno schema di senso compatibile con le esigenze di omogeneità operativa informatica. Si tratta di una licenza d'uso ricorsiva, quindi auto-poietica ed autoreferenziale.

La principale caratteristica auto-poietica della licenza GNU-GPL è quella legata al concetto copyleft. Copyleft significa che la copia è consentita, quindi a determinate condizione, è consentita la replica. In questo modo il potenziale auto-poietico risulta grandemente aumentato rispetto ad un software coperto da copyright. A questo proposito val la pena segnalare che è opinione diffusa tra molti operatori informatici il fatto che Microsoft tolleri entro certo limiti la pirateria nei confronti dei suoi prodotti, proprio per consentirne la massima diffusione possibile. Paccagnella (2010), utilizzando il rapporto della "Fondazione Luigi Einaudi" sui consumi digitali, mostra come il file sharing (scambio illegale) inibisce l'acquisto di CD e DVD di chi utilizza poco o nulla prodotti culturali, mentre chi già spende molto in dischi, film, concerti, e via dicendo il file sharing contribuisce ad aumentarne ulteriormente l'acquisto.

Se la questione si limitasse al consenso di fare delle copie si tratterebbe in realtà di semplice riproduzione senza auto-referenzialità e non in grado di tenere chiuso il sistema. In pratica il codice del software lasciato aperto ed intelligibile si disperderebbe senza mantenere la sua identità. L'auto-referenzialità opera ricorsivamente quindi la licenza GPL o GNU-GPL impone che qualora lo stesso venga ceduto ad un'altra entità giuridica, questo avvenga negli stessi termini della licenza GPL. Anche questo però non è sufficiente, poiché gli elementi del sistemi, coerentemente con la teoria dei sistemi sociali di Luhmann, non si riproducono mai in maniera perfettamente identica altrimenti non sarebbe possibile l'adattamento e dal punto di vista della produzione tecnologica, non sarebbe possibile il miglioramento.

In pratica la licenza GPL o meglio GNU-GPL, che rende meglio il concetto di ricorsività per mezzo del suffisso “GNU”, è una elaborazione contrattualistica in chiave intenzionalmente auto-poietica volta a conferire all'open source una autonoma continuità del software e del codice a cui si riferisce. C'è di più. Come abbiamo visto nelle pagine precedenti questa tecnologia costituisce anche il capitale simbolico, contiene gli schemi di senso, determina i confini e quindi consente gli elementi identitari ed, in questo modo, il sistema sociale che sottende, ne risulta particolarmente potenziato proprio dal punto di vista autoreferenziale, combinando chiusura autopietica ed apertura simbolica.

La differenza concettuale tra il copyleft della licenza GPL ed il copyright riflette e si riflette sugli stessi paradigmi di produzione del software i quali nel tempo sono diventati marchi di qualità con opposti significati nelle diverse scuole di pensiero. Vixie (2000) mette a confronto questi due paradigmi di produzione del software:

[...] Gli elementi di un processo di progettazione software vengono generalmente enumerati come segue:

Requisiti di marketing, Progetto a livello di sistema, Progetto dettagliato, Implementazione, Integrazione, Testing sul campo, Supporto

Nessuno degli elementi di questo processo dovrebbe avviarsi prima che i precedenti siano sostanzialmente completati e ogni qualvolta viene operata una modifica a qualche elemento, tutti gli elementi dipendenti dovrebbero essere revisionati o rifatti alla luce di quella modifica. È possibile che un dato modulo sia specificato e implementato prima che i moduli da questo dipendenti siano stati

completamente specificati: questa pratica è nota come sviluppo avanzato o ricerca.

...Diversamente un caso anche troppo comune di progetto Open Source è quello in cui i soggetti coinvolti si stanno divertendo e vogliono che il loro lavoro sia usato dal maggior numero possibile di persone: per questo lo danno via gratis e spesso senza porre alcuna restrizione alla ridistribuzione. Si tratta magari di gente che non ha accesso agli strumenti software cosiddetti "di livello commerciale" (analizzatori della copertura del codice, interpreti bounds-checking, verificatori dell'integrità della memoria). E le cose che più sembrano divertirli sono: programmare, confezionare ed evangelizzare: niente QA, niente MRD, di norma niente date di rilascio troppo vincolanti. (Vixie, 2000).

Anche da qui emerge tutta l'anticonvenzionalità efficace in condizioni di globalizzazione e complessità. E come si è già visto (cfr cap. 2.4), da questa divergenza ne deriva una diversa concezione di mercato.

Note

1. ↑ Computer posto tra una LAN (rete locale) e la rete internet e che ha la funzione di filtrare il traffico potenzialmente malevolo per i altri computer locali.
2. ↑ Computer che ha la funzione di procurare gli accessi alle risorse nascondendone il percorso per motivi di sicurezza.

Come abbiamo già visto lo schema di senso nella società informatizzata in generale riguarda l'affermarsi dell'idea di automazione dei processi che coinvolge la quotidianità degli individui. Quello che invece emerge a livello di produzione di tecnologia informatica è una separazione paradigmatica sul piano economico. Da una parte la convinzione che il mercato sia in grado di mobilitare al meglio le risorse e dall'altra la convinzione che il mercato sia inadeguato a gestire una risorsa che costituisce invece lo spazio dove il mercato può dispiegarsi.

Sotto questo punto di vista la risorsa che consente il dispiegarsi del mercato non può a sua volta essere oggetto di scambio perché limiterebbe in modo ricorsivo il mercato stesso. Una risorsa per poter essere scambiata nel mercato necessita di essere scarsa e se la sua natura non lo consente è necessario intervenire artificialmente. Per sua natura il software può essere facilmente copiato e quindi ceduto al di fuori del mercato formale alterando il meccanismo della domanda e dell'offerta.

Nel tempo il concetto di piazza dove si svolge il mercato si è “virtualizzato”, in epoca premoderna costituiva il luogo fisico dove avvenivano le transizioni, oggi questo è divenuto diffuso. Ma se pensiamo al concetto di piazza come spazio pubblico dove il mercato attua le transazioni diviene facilmente comprensibile che lo stesso spazio pubblico non può essere reso scarso, non può diventare a sua volta oggetto di scambio perché una tale dinamica ricorsiva finirebbe per autodistruggersi. La “piazza del mercato” per poter essere scambiata nella “piazza del marcato” dovrebbe essere scarsa per poter rientrare nel meccanismo della domanda e dell'offerta, ma allo stesso tempo se fosse limitata impedirebbe allo stesso mercato di attuarsi. È ciò che accade con le barriere doganali che di fatto limitano notevolmente il surplus generale. Non è lo scopo di questo studio giustificare o meno l'efficacia del libero mercato di allocare le risorse, ma solo quello di rilevare come anche in una visione liberista dell'economia si cade in contraddizione a tal punto che molti pensano che molte software-house siano tolleranti con la pirateria, che la considerino fisiologica entro un certo limite poiché consente paradossalmente di allargare il mercato delle licenze chiuse. In un articolo apparso su “business week” il 28 agosto del 2007, Matthias Gilke, portavoce di Quark, dice che la protezione contro la pirateria previene gli utenti “tipici” dal fare copie, ma non può nulla contro la pirateria e quindi non vale la pena investire soldi sulla protezione. Questa frase potrebbe essere interpretata da un cracker nel seguente modo: “il nostro software si può piratare”, se avete

abbastanza talento potete provarci, in caso contrario non vi resta che comprare la licenza". Paradossalmente i crackers possono rappresentare un veicolo promozionale per il prodotto stesso.

Non è lo scopo di questo studio dimostrare la veridicità di tanto, resta il fatto che è verosimile e che molti operatori del settore lo credono vero e quindi tale credenza rientra nello schema di senso di chi agisce nel marcato informatico. Diciamo che dando per vero tale presupposto la tolleranza verso la pirateria informatica, cioè del cracking, è ciò che consente l'equilibrio auto-poietico.

Per contro sembrerebbe che il software libero non consentirebbe di per se il mercato, può sussistere la domanda e l'offerta ma non il prezzo. In realtà questo problema viene risolto modificando l'idea di prodotto. Non è il prodotto software che viene scambiato nel mercato ma le competenze relative al software. In questo modo il tutto si sposta su un altro piano. Le competenze sono di per sé scarse, scarsissime se si pensa alla potenziale diffusione del software libero. In pratica più si diffonde il software e più c'è bisogno di competenze per poterlo mantenere e far funzionare.

Lo schema di senso è ciò che consente ad un sistema sociale di diminuire la complessità, di differenziarsi dall'ambiente (Luhmann, 1990), l'indentificazione e l'individuazione. Questo schema di senso riguarda quindi il diverso modo di concepire il mercato e non solo. Ma sarebbe troppo riduttivo limitare lo schema di senso ad una diversa concezione di bene scarso e non. Esiste una naturale propensione dell'uomo alla solidarietà che può anche essere vista funzionalmente con lo scopo di ottimizzare l'allocazione delle risorse in una condizione di abbondanza (Berra, 2001). In questo "senso" l'informatica, spesso simbolo della modernità pressata dalla scarsità, rientra, da un punto di vista economico nella situazione di abbondanza delle risorse che richiede diverse dinamiche re-distributive tipiche delle società primitive:

[...] Nelle società primitive, dove il problema della scarsità delle risorse non esisteva e non era così pressante per lo sviluppo, il ricorso al dono avrebbe costituito un mezzo per attenuare lo scarto fra la quantità delle risorse esistenti e la loro concentrazione in determinati gruppi sociali. (Mariella Berra, 2001, pag. 152)

Quindi mentre il software costituirebbe una risorse naturalmente abbondante e quindi la sua "allocazione" rientrerebbe in una logica di solidarietà meccanica

(Emile Durkheim, 1962) e comunitaria le competenze, come risorsa scarsa, sarebbero invece allocate mediante l'economia formale. Questo meccanismo evoca la teoria di Karl Polanyi sui tre meccanismi di riproduzione sociale: mercato, reciprocità, re-distribuzione, per cui nessuna società può esistere senza possedere una qualche economia sostanziale (Polanyi, 1977). Questo che si profila è la necessità di un'economia sostanziale del software che sostenga un'economia formale delle competenze. Ma anche questo non è esaustivo a definire lo schema di senso, in quanto non si limita funzionalmente all'informatica per le sue peculiarità di risorsa scarsa ed abbondante allo stesso momento, ma viene allargato alla conoscenza in generale. Il caso più emblematico è Wikipedia che è un progetto per la diffusione della conoscenza libera. ASAQ è un prodotto farmaceutico antimalarico non sottoposto a protezione brevettuale. Nel 2001 il governo del Sud Africa emana il medical act, una legge che consente alle case farmaceutiche sudafricane di produrre farmaci senza pagare le royalties alle multinazionali detentrici dei brevetti.

A livello microsociologico l'informatica evidenzia una peculiarità dell'uomo di esplorare i suoi limiti, di provocarsi gratificazione affrontando le sfide e soddisfacendo nel contempo il bisogno di desiderabilità sociale, coerentemente con la cultura del dono come il potlatch, una cerimonia del dono che si svolge tra alcune tribù di nativi americani. Ciò che emerge in sintesi è che:

l'idea stessa di mercato informatico è relativa, può riguardare artificialmente le licenze chiuse del software chiuso, o le naturali competenze dell'open source.

Lo stesso mercato informatico è sostenuto da una buona parte di economia sostanziale.

Lo schema di senso che ne deriva, rafforzato dal successo effettivo dell'open source, tende ad espandersi nella cultura generale.

La gratificazione intellettuale e la desiderabilità sociale svolgono una loro funzione sociale, sono storiche, fanno parte a pieno titolo l'una dei sistemi cognitivi e l'altra dei sistemi sociali e non necessariamente di profili patologici.

Rispetto alla teoria dell'azione di Talcott Parsons che prevede una struttura le cui parti sono dedite alla mobilitazione di energia ed altre alla mobilitazione dell'informazione, Luhmann attraverso l'auto-poiesi giunge ad una visione meno scontata dell'ordine e soprattutto ad un'idea di ordine, se tale si può definire, come uno dei tanti possibili. In questa concezione funzionalista l'informazione assume una specificità diversa. Non più dai livelli gerarchici più elevati a quelli più bassi ma attraverso dislivelli informativi che si verificano nel sistema senza seguire un piano prestabilito.

Il dislivello informativo è ciò che consente in ultima analisi la dinamica informativa, senza questo presupposto non ci sarebbe informazione, poiché senza differenze non è possibile la distinzione di differenze. L'informazione nasce come distinzione di differenze, quindi un sistema con poche differenza è un sistema entropico non in grado di adeguarsi. Per contro un sistema sufficientemente complesso è in grado di produrre grossi effetti a partire da piccole differenze. Un sistema sufficientemente complesso è in grado quindi, non solo di trasferire informazione, ma di costruire informazione amplificando la devianza (Luhmann, 2007, p. 72).

È interessante notare come Vixie (2000) descriva il processo di produzione software proprietario come fasi rigidamente sequenziate, tali per cui la fase successiva non può avviarsi se prima non è stata completata la precedente (modello top-down), mentre il modello di produzione open source risponde a criteri autonomi, spesso spontanei ed originali bottom-up. Il primo modello corrisponde ad un idea organizzativa di sistema come la prevede Parsons, mentre il secondo modello corrisponde ad un processo dove l'innesco delle diverse fasi di sviluppo è contingente al dispiegarsi dei flussi informativi. A questo diverso approccio corrisponde anche lo scambio di corrispondenza tra Linus Torvalds, ancora studente presso l'università di Helsinki, e Andrew Tanenbaun, docente di sistemi operativi alla libera università di Amsterdan:

Torvalds: se questo fosse l'unico criterio per la “bontà” di un kernel avresti ragione. Quello che non menzioni è che minix non fa le cose del micro-kernel molto bene e ha problemi con il vero multitasking (nel kernel). Se io avessi un sistema operativo che avesse problemi con il vero file di sistema multithreading, non condannerai altri così facilmente: infatti, farei tutto il possibile perché venga dimenticato il fiasco.

Tannenbaun: ...Un file di sistema multithread è solo uno sfizio per le prestazioni. Quando c'è solo un processo attivo, il caso tipico di un piccolo PC, non porta nulla ed aumenta la complessità del codice. Nelle macchine abbastanza potenti da sopportare la multiutenza avresti abbastanza buffer nella cache da assicurare ad ogni porzione di cache una porzione di cicli, nel qual caso ugualmente non ti fa guadagnare nulla. Hai qualche vantaggio solo con multiprocessori che fanno fisicamente I/O. Ma anche in questo caso complicare il sistema e quantomeno discutibile

Nonostante il linguaggio esoterico ciò che emerge è il diverso approccio alla complessità. Nella disciplina informatica è, a distanza di vent'anni, ancora vigente e più che operante il paradigma top-down. Per cui molte risorse vengono investite nella progettazione e quindi le informazioni vengono passate ai sistemisti e ai programmatori per lo sviluppo. Il paradigma opposto è quello per cui degli sviluppatori sperimentano dei successi su cui poi chiedono investimenti. In quest'ultimo caso ciò che si verifica non è una direzione lineare dell'informazione dal basso verso l'alto ma un emergere sistemico dell'informazione. In pratica questi due paradigmi, il top-down e il bottom-up sono i corrispettivi modelli informatici dei due diversi modelli funzionalisti, il primo Parsonsiano e l'altro Luhmanniano.

Se si nota i termini usati non sono esattamente reciproci, l'uno l'opposto dell'altro, non indicano due direzione opposte dell'informazione, ma due diverse dinamiche dell'informazione, l'una unilineare da un punto gerarchico ad un altro punto inferiore e l'altra emergente (multithread) dal basso verso l'alto. Nella prima definizione il basso viene indicato con “down”, mentre nella seconda definizione il basso viene indicato con “bottom” che da più l'idea di base. Lo stesso si può dire per “top” e “up”. La prima definizione top-down esprime una direzione dell'informazione da un punto ad un altro, nel secondo caso un emergere dell'informazione da un livello ad un altro. La cosa interessante e paradossale è che quando viene fondato la OSD (Open Source Defination) si innesca un processo di istituzionalizzazione che invoca il top-down:

[...] Dimenticare la strategia "bottom-up"; puntare sulla strategia "top-down"

Ci appariva ormai chiaro che la strategia storica seguita per Unix, vale a dire la diffusione dei concetti dal basso verso l'alto, partendo cioè dai tecnici per giungere poi a convincere i boss con argomentazioni razionali, si era rivelata un

fallimento. Era una procedura ingenua, di gran lunga surclassata da Microsoft. Inoltre, l'innovazione di Netscape non avvenne in quella direzione, ma fu resa possibile proprio perché un importante personaggio di livello strategico, quale Jim Barksdale, aveva avuto l'intuizione e l'aveva imposta ai suoi subordinati. (Eric Steven Raymond, 2000).

Linux nasce assolutamente con un processo bottom-up mentre il processo di istituzionalizzazione si orienta al top-down, legittimando questa scelta con il successo di questo paradigma nel caso Netscape. Netscape nasce in contesto industriale e solo successivamente diventa open source, ed anche la sua "liberazione" non può che essere coerente con una logica top-down. Ciò che emerge è l'idea di organizzazione come ordine, semplificazione e contenimento dell'incertezza.

Già da qui si evince che non esiste mai una scelta totalmente spostata da una o dall'altra parte. E ' però evidente una maggiore tendenza dell'organizzazione open source ad adottare processi bottom-up ed una maggiore tendenza dei sistemi industriali di produzione di software ad adottare una struttura top-down. Proprio per la sua diversa dinamica dell'informazione possiamo ipotizzare non una semplice classificazione tra questi due paradigmi, ma una sostanziale differenza della loro natura. Luhmann distingue tra strutture e processi, dove le strutture hanno una maggiore tendenza a trattenere il tempo diminuendo l'incertezza, mentre i processi segnano il carattere irreversibile del tempo. Entrambe queste strategie rispondono al problema della complessità inevitabili quando i sistemi aumentano di dimensione e assumono dimensioni planetarie. La complessità presuppone l'adozione di criteri selettivi, attraverso strutture o processi, in quanto non è possibile mantenere tutte le connessioni (possibilità) aperte.

Luhmann non ci offre una classificazione di sistemi sociali in base alla quale distinguere quelli che privilegiano più la struttura e quelli che privilegiano più il processo ma solo dei criteri. La differenza tra questi due sta nel fatto che le strutture operano le selezioni in base a criteri di validità, mentre i processi attraverso la temporalizzazione degli elementi e si rifanno al criterio dell'adattamento. L'adattamento consiste nella diversità degli elementi che si susseguono, la stabilità del sistema in generale è raggiunta grazie all'instabilità dei suoi elementi. Ciò che è importante notare è che nel modello bottom-up viene meno il bisogno di criteri di validità preordinati tipici del modello top-down. Il criterio di validità è raggiunto grazie all'accelerazione del processo selettivo.

Si pensi ad esempio alla continua interazione nei diversi progetti open source tra utenti e sviluppatori software. Questo non ha corrispondenza nel software chiuso, se non attraverso canali istituzionalizzati di assistenza definiti contrattualmente. Anche in questo caso si segnala la rapidità e spesso il rapporto diretto sviluppatore utente, tale per cui un problema rilevato da un utente viene spesso risolto seduta stante. Inoltre val la pena riprendere la tesi Gian Antonio Gilli per cui l'attività del portatore di téchne deve necessariamente essere mediata da altre figure professionali. Nel caso dell'open source il programmatore si può liberamente e direttamente relazionare con gli utenti, è di fatto fuori controllo e questo a maggior ragione rafforza l'idea di un sistema caotico a bassa entropia.

Esempi di temporalizzazione sono evidentissimi nel mondo open source. I sistemi operativi proprietari più importanti che si sono susseguiti vantano alcune versioni che si susseguono sequenzialmente. Le cosiddette “distro”, cioè distribuzioni, di Gnu/Linux, pacchetti personalizzati a seconda delle diverse esigenze, sono innumerevoli e contemporanei. A parte le principali distribuzioni da PC, con le diverse interfacce grafiche (Gnome, KDE, XFCE, LXDE) da cui si differenziano diverse interfacce utente Debian, Ubuntu, Fedora, Suse, Mandriva mantenute da altrettanti gruppi e svariati sottogruppi di specifici applicativi, le diverse versioni si susseguono freneticamente. A questo si aggiungano diverse personalizzazioni di personalizzazioni. In 5 anni Ubuntu ha raggiunto 10 versioni, Windows a partire dal 92 al 2010 annovera le versioni: 3.1, 95, 98, NT, XP, Vista, sette.

Naturalmente questo non è indice di qualità, anche se le differenze qualitative sono inevitabili, ma queste si misurano con tecniche di benchmarking e non in base al numero di versioni come spesso semplicisticamente si pensa. Piuttosto questo è indice di un diverso modello sistemico: strutturale per i sistemi operativi proprietari; processuale per i sistemi open. Il paradosso organizzativo di Gnu/Linux è uno degli argomenti più discussi e consiste nel fatto che secondo molte teorie dell'organizzazione non sarebbe possibile tenere sotto controllo una tale complessità. La questione in realtà sta a monte delle teorie dell'organizzazione che si occupano di processi di produzione, è in realtà una questione sistemica, ciò che rende possibile il controllo è proprio, in definitiva, la complessità. La semplificazione dell'ambiente può essere attuata attraverso la complessificazione interna al sistema (Luhmann, 1988). In altre parole, a parte settori molti organizzati di produzione di software open source, spesso si tratta di un emergere della tecnologia da dinamiche sociali non istituzionalizzate che non

possono essere facilmente confrontante con i sistemi industriali di produzione del software da un punto di vista meramente organizzativo orientato alla produzione.

Luhmann, nonostante la necessaria chiusura, o in quanto alla chiusura, cioè separazione dall'ambiente dei sistemi, distingue tra sistemi chiusi e sistemi aperti complessi. Nella realtà non esistono sistemi chiusi ma solo sistemi più o meno aperti. Ad una maggiore apertura corrisponde un maggior disordine, una maggiore difficoltà di prevedere il futuro in base a criteri di validazione, e quindi maggiore incertezza. Per contro un sistema maggiormente complesso e temporizzato presenta maggiore velocità, e capacità di adattamento a scapito di una maggiore incertezza espressa come bisogno di ordine e prevedibilità. È per tale motivo che i processi di istituzionalizzazione non possono che proporre il modello top-down.

In questa tesi si è cercato di leggere il fenomeno del software libero con gli strumenti che la sociologia mette a disposizione secondo la metafora della "cassetta degli attrezzi" che propone Wright Mills. La ricerca è stata articolata su due livelli del problema. Il primo ha riguardato l'analisi di materiale bibliografico che permettesse di fare il punto della situazione della ricerca in questo specifico settore che si è ritenuto essere espressione dell'iper-modernità di Giddens. In secondo luogo lo sforzo è stato rivolto a verificare nel concreto e nel locale come questo fenomeno si manifestasse e quali effetti producesse. Attraverso l'analisi della bibliografia ci si è accorti che molti degli schemi della sociologia hanno avuto scarsa applicazione in questo contesto, mentre questi schemi analitici, quand'anche parte della sociologia classica, si sono rivelati congruenti anche se non sufficienti per cui è stato necessario anche il ricorso alla sociologia più recente in particolare in riferimento a Luhmann, Giddens, Robertson e Ulrich Beck, Bourdieu.

È spesso accaduto che molti riferimenti bibliografici siano stati presi in considerazione allo scopo di fornire spiegazioni teoriche in grado di supportare il discorso, mentre nel procedere ci si è accorti che tale documentazione forniva materiale interessante per analisi di secondo livello, in grado di far comprendere come gli attori coinvolti interpretano la situazione. Questo è accaduto in particolare con Eric Steven Raymond, quand'anche autore di intuizioni di spessore sociologico importanti. Richard Stallman, leader della prima ora, ha fornito un'epistemologia feconda su cui articolare un metodo di indagine che ci permettesse di mettere alla prova la sua filosofia. Indagare il fenomeno del software libero ha significato osservare chi osserva, e spesso, osservare chi osserva chi osserva. Il software libero è in grado di mettere in campo competenze organizzative, sociologiche, economiche e legali oltre che tecnologiche. È pertanto un campo di indagine complesso e rischioso che richiede preparazione ed attenzione.

Quando contattai Stefano Zacchiroli per chiedergli la disponibilità ad essere intervistato, la sua preoccupazione fu quella di fornirmi biografie, contatti con altri antropologi e sociologi con cui aveva avuto a che fare. Molto spesso si confondono i ruoli, non si capisce chi osserva chi, come emerge chiaramente dalla

intervista a Stefano Zacchiroli, il quale mi pone delle domande preliminari. L'intervista a Stefano Zacchiroli si apre con l'intervistato che interroga l'intervistatore. Il software libero agisce in modo scientifico anche al di fuori degli ambiti tecnologici. Intervistare un hacker, o un geek, significa negoziare la definizione della situazione all'interno di un discorso scientifico. Al di la della "giusta" definizione di hacker tra le tante, questa dimensione viene colta dall'accezione che ne da wikipedia:

[...] è una persona che si impegna nell'affrontare sfide intellettuali per aggirare o superare creativamente le limitazioni che gli vengono imposte, non limitatamente ai suoi ambiti d'interesse (che di solito comprendono l'informatica o l'ingegneria elettronica), ma in tutti gli aspetti della sua vita.

Ciò che ne esce sono delle "ragioni pubbliche" (Kant) su cui vale pena investire sforzi per indagare, oltre a ciò che propone, anche i grandi temi che vanno oltre lo stesso software libero. Abbiamo scoperto, grazie al software libero, un uomo nella società per nulla scontato. Non ci si è posti il problema di capire se questo uomo sia in grado di prendere in mano il suo destino nella modernità radicale più prossima, ma di verificare la plausibilità di un uomo in grado di mettere in discussione aspetti della modernità che consideriamo spesso cinicamente e frettolosamente fattuali. Non ci si è soffermati sugli aspetti romantici, non ci si è rassegnati ad osservare dall'esterno.

Si è trattato, per lo più, di comprendere, di sporcarsi le mani, per così dire, di mettersi alla prova accettando la provocazione forte e paradossale che sia possibile un agire razionale riflessivo non utilitaristico che ha molte delle caratteristiche dell'ascesi intra-mondana. Spesso le contrapposizioni tra diverse conclusioni dialettiche sono determinate da ciò che si assume come significativo. Per me è stato significativo un agire razionale sostenuto eticamente che ha molto in comune con la genesi del capitalismo descritta da Weber: se l'accento viene posto sulla razionalità questo approccio appare come paradosso, se l'attenzione viene posta sull'aspetto etico diviene coerente, mentre diventa paradossale l'irrazionalità della razionalità moderna (Luhmann, 2003).

Ci sono anche le contraddizioni, le situazioni sfumate e l'agire utilitaristico. Molti progetti nati come open source sono stati acquistati da multinazionali e sono stati chiusi; le stesse multinazionali finanziano molti progetti; per molti "volontari" del software libero si tratta di un tirocinio, specie se studenti; per molti si tratta di allungare il proprio curriculum; per molti si tratta di ottenere

qualificazioni immediatamente spendibili nel mercato; molti volontari del software libero finiscono per essere cooptati da software house proprietarie; spesso le software finanziano l'open source per limitare gli effetti del monopolio e quindi costruire condizioni per un'economia di mercato più liberale e il software libero rappresenta solo un mezzo per raggiungere altri scopi.

Evitare di trattare questi aspetti non significa dimostrare che non esistano, ma significa concentrarsi su aspetti che mostrano la possibilità reale dell'uomo moderno di fare la storia o quantomeno di metterla in discussione. Non ci sono garanzie che l'informatica nella sua accezione libera riesca a mantenere tonico il paradosso che rappresenta nel tempo, ciò dipende da molte variabili, non ultime quelle trascurate da questa ricerca.

La sintesi della mie conclusioni può risultare riduttiva rispetto all'ampiezza delle possibilità di indagine verso direzioni poco esplorate, come ad esempio il rapporto tra modelli di produzione di software, fiducia e rischio, tecnostress e isomorfismo verticale. Ripartendo dalla domanda cognitiva iniziale (cfr. cap. Introduzione), alla luce di quanto indagato sinora, la mia risposta sintetica riguarda due momenti storici che evidenziano una intensità etica:

Il primo riguarda l'esperienza di effervescenza intellettiva, emotiva e collettiva nella comunità di hacker del MIT di Boston, alla quale segue la frustrazione per l'espropriazione per l'emergere di nuovi paradigmi industriali di chiusura del codice. Da questo specifica esperienza di effervescenza prima e mortificazione poi viene fondato, in ambito accademico, la Free Software Foundation e si avvia il progetto GNU orientato già da subito a creare un sistema operativo open source alternativo ai sistemi proprietari.

Il secondo episodio storico riguarda la creazione del vero e proprio sistema operativo operativo Linux che verrà incluso nel progetto GNU. La nascita del sistema operativo GNU/Linux nasce anch'esso in un contesto di effervescenza, Linus Torvalds attrae subito molti volontari, attiva motivazioni intrinseche sostenute da una dimensione etica.

Questi sono gli episodio storici che hanno come conseguenza il dispiegarsi dell'open source. La dimensione etica del software libero è ciò che consente e ha consentito il dispiegarsi dell'open source. L'open source di per sé, cioè estrapolato dal contesto del software libero, non è sufficiente a consentire un piano riflessivo collettivo e non è di per sé autonomo nell'attivare motivazioni intrinseche,

necessita di una dimensione etica e di un procedere scientifico in grado di rispondere al rischio elaborando modelli sociali basati sulla fiducia.

Tutto ciò ha conseguenze importanti sui paradigmi industriali di produzione di software ed in particolare diventa enigmatico il fatto che gli investimenti economici sull'open source siano sufficienti all'appropriazione delle motivazioni intrinseche da parte del mercato formale evitando le prerogative etiche del software libero. Ma questo aprirebbe un'ulteriore domanda cognitiva.

Per Richard Stallman le cose stanno così:

[...] Come risponderemo alla prossima allettante libreria non libera? Riuscirà la comunità in toto a comprendere l'importanza di evitare la trappola? Oppure molti di noi preferiranno la convenienza alla libertà, creando così ancora un grave problema? Il nostro futuro dipende dalla nostra filosofia ... (Richard Stallman, 2000).

Questo oltre ad essere un monito morale, rinforzato da termini come *trappola*, *convenienza*, *grave problema*, *futuro*, *noi*, *nostro*, evidenzia la consapevolezza che il software libero, e tutto ciò che questo rappresenta, è possibile tanto quanto è possibile una dimensione etica che lo sostenga e che renda operativa una specifica visione del mondo che Richard Stallman chiama *filosofia*.

ALLEGATI

- *Intervista aperta a Stefano Zacchiroli, Project Leader di Debian*
- *Intervista aperta ad Andrea*

Nato a Bologna, il 16 marzo 1979, Debian Project Leader, Ricercatore Laboratoire PPS, Université Paris Diderot.

Premessa:

Io e Stefano Zacchiroli avevamo già intrattenuto degli scambi via posta elettronica per concordare questa intervista. Ci eravamo già presentati. L'intervista inizia con Zacchiroli che pone a me delle domande. È necessario tenere presente che Stefano Zacchiroli è uno scienziato, un ricercatore universitario, quindi implicitamente assume un ruolo che va oltre il fatto di fornire delle informazioni. È probabilmente una sua prerogativa quella di entrare nel merito della questione e di conseguenza negoziare la definizione della situazione in un ambito scientifico. Da parte mia si è ritenuto metodologicamente corretto questo approccio per tre motivi:

Ha fornito direttamente delle notizie e delle chiavi di lettura che hanno confermato o meno le mie ipotesi di fondo e che, in ogni caso, hanno avuto la necessità di essere contestualizzate in una prospettiva sociologica.

Sono comunque emersi atteggiamenti di fondo utili ai fini dell'indagine.

In questo modo emerge in buona misura la mia personale funzione al valore relativa al campo di indagine.

Intervista:

I = IO, stefano De Boni laureando in scienze sociologiche a Padova

Z = Stefano Zacchiroli

Z: Mi scrivevi che sei stato volontario stato in Africa?

I: Sierra Leone vicino alla Liberia, Africa Occidentale. La mia intenzione era appunto di trovare un posto pubblico in modo da tornare a fare il volontario con la cooperazione internazionale, così una volta che mi fossi stancato sarei potuto tornare e quindi avere un posto. Poi ho trovato mia moglie, mi sono sposato, poi nel pubblico ho fatto carriera, carriera per modo di dire, sono entrato come impiegato amministrativo e poi sono diventato informatico attraverso corsi di formazione e esperienza lavorativa. Quindi ora sono inquadrato come analista programmatore.

Z: E sei riuscito a lavorare sul software libero a Padova?

I: io studio a Padova ma vivo e lavoro Vicenza. La cosa che mi ha fatto cambiare abbastanza idea è stato che seguendo dei progetti GIS ho provato a scaricare delle cose...

Z: Era tutto chiuso [pensa che stia parlando di software proprietario]

I: No, intendo Mapserver, Openlayer, questo tipo di cose [software libero], e ho visto che riuscivo molto meglio...

Z: ah, addiruttura qualità del software libero migliore di quello proprietario?

I: Dal punto di vista dei GIS si!

Z: So che il GIS è una cosa che ha convertito molti, ma soprattutto per via dei formati proprietari, in quanto dicevano: “ma no deve essere roba che deve essere accessibile pubblicamente”. Molte persone mi hanno detto che questo è stato il motivo iniziale per interessarsi al software libero...

I: No la causa scatenante da parte mia è stata che ... a parte che io faccio sviluppo WEB e quindi mi occupo dei GIS dal punto di vista WEB... [interruzione perché manovro con il microfono per assicurarmi che sia in funzione]...si mi occupo di sviluppo WEB, dal punto di vista sistemistico non me ne sono mai occupato più di tanto. Dal punto di vista WEB posso dire che la cosa rispetto ad Intergraph, non ho mai usato ESRI, se non i Map-objects che si usavano molti anni fa, è notevolmente migliore. Ma anche per il semplice fatto delle specifiche e la documentazione che trovi su internet. Su intergraph hai delle difficoltà proprio a ricevere le informazioni su come far funzionare le cose. Devi avere dei canali istituzionali, è tutto mediato attraverso figure commerciali e via dicendo. Se devi mettere su qualcosa in quattro e quattr'otto non c'è paragone. Poi ho cominciato ad

interessarmi a sta cosa qua e la cosa che io ho colto dal punto di vista sociologico, a parte che io mi sono occupato di sociologia anche per motivi organizzativi, perché per me nel pubblico la patologia organizzativa e diffusa...

Z: Non pensavo che la scala fosse abbastanza grande per applicarvi i metodi della sociologia...

I: Al di la degli sfoghi ho deciso di cercare di comprendere da un punto di vista più scientifico...

Z: Quindi hai preso un part-time?

I: No io lavoro a tempo pieno.

Z:Ok. Complimenti. E questa laurea qua è triennale o specialistica?

I: Triennale e comunque farò anche la specialistica.

Z: Va bene dai, dimmi tutto.

I: Allora...

Z: Ammetto che non ho fatto molti compiti...

I: Ma no, le cose migliori sono quelle che nascono spontaneamente. Le domande che ti avevo mandato erano queste (gli porgo un foglio con quattro domande:

1) racconti come È andata la debconv di NYC;

2) vorrei che comentassi questa frase di Eric Steven Raymond:

[...] Gli hacker sono anti-autoritari per natura. Chiunque possa darti degli ordini, puo' fermati dal risolvere problemi dai quali sei affascinato - e, visto il modo con cui le menti autoritarie funzionano, tali ordini generalmente saranno motivati da ragioni orribilmente stupide. Così, l'atteggiamento autoritario deve essere combattuto ovunque si trovi, affinché non soffochi te e gli altri hacker . (How To Become A Hacker, rev. 1.15, October 03, 2002)..."

3) vorrei raccontassi episodi significativi della esperienza professionale o meno legati al softaware libero

4) vorrei dessi degli spunti o contributi che ritieni utili per la tesi (se vuoi ti faccio avere il semi-elaborato prima)

Z: Si ma dimmi cosa vuoi sapere di Debconf, nel senso che la domanda è molto aperta, e come ti ho già segnalato, c'è un sacco di gente che ha già studiato come funzionano questi eventi quindi... mi dici più in specifico cosa vuoi sapere, cioè che tipo di feedback vuoi?

I: Questa volta sono io che non ho studiato...

Z: nel senso che raccontare due settimane di eventi potrebbe prendere una settimana di tempo...

I: Quello che a me interessa sapere, per restringere di più il campo ... io ti scrivevo che rispetto alla Coleman ho un punto di vista un po' diverso, più una sfumatura diversa più che un punto di vista diverso, nel senso che io sono convinto che, diversamente da quello che lei dice che la Debconf serve a costruire il “selves”, che questo “selves” in qualche modo sia costruito già prima, cioè...

Z: in rete intendi?

I: in rete sì!

Z: Nel caso di Debian è un caso diverso dai LUG (Linux Users Groups), se non altro per un motivo di scala, i LUG sono formati da gente che già si conosce, se qualcuno si approccia a Linux si iscrive alla mailing list, si fa conoscere, il mese dopo o relativamente poco dopo partecipa alle riunioni locali, e quindi conosce subito le persone fisicamente, quindi non c'è quello che spesso è un problema in Debian ed il fatto che essendo un progetto su scala internazionale, c'è gente che lavora senza mai né essersi vista, né conosciuta. Allora questo, siamo molti a pensarlo in Debian, contribuisce nel momento in cui ci sono dei conflitti tecnici a peggiorare le situazioni, nel senso che è molto più facile avere una battaglia, un flame [conflitto sulle discussione on-line] con qualcuno che non hai mai visto di persona piuttosto che con qualcuno che hai visto di persona, conosciuto di persona, con il quale hai condiviso delle esperienze emotivamente

forti, perché eventi come debconf sono indubbiamente eventi emotivamente molti forti. Quindi il motivo principale per cui molti di noi pensano che Debconf sia fondamentale in un progetto come Debian è abbattere questa barriera, abbattere la barriera del non ci siamo visti, conosciuti e questa barriera contribuisce ad esacerbare i conflitti. Quindi il motivo principale, sono d'accordo con la Coleman, che sia un po' sul ridurre questi attriti che si possono verificare nella vita normale del progetto. Noi una cosa che diciamo spesso è “vai ad un a Debconf, ti incontri con qualcuno, ti ubriachi, ci passi due settimane, ed il giorno dopo quando sarai a casa a e interagirai con questa persona remotamente, la tua interazione con lui, o con lei, sarà molto migliore e questo è a mio avviso uno dei motivi fondamentali per andare ad una debconf che è migliorare i rapporti all'interno della comunità. Secondo, è molto diverso da quello che può esserci in un LUG, perché il LUG è un'iniziativa forzatamente locale, cioè basata sul territorio. Il LUG serve proprio per raccogliere assieme tutti i linuxiani [coloro che usano Linux e si riferisce all'insieme delle subculture che fanno riferimento alle diverse versioni, le così dette distribuzioni o distro, di Linux] o quello che è di un certo territorio. Quindi è forzatamente, geograficamente molto locato e localizzato. Un progetto come Debian no, è assolutamente world wide, e quindi non ha questo tipo di garanzia di incontrarsi periodicamente. Questo è un motivo. Altre persone dicono che vanno o non vanno a Debconf per motivi diversi. C'è chi semplicemente dice che ci va per i cosiddetti sprint. Cioè per incontrarsi e lavorare assieme su un certo progetto. E li è risaputo che lavorare di persona in un lasso di tempo limitato con delle persone che collaborano su uno stesso progetto ha un'efficienza ed un'efficacia molto superiore che il lavoro remoto, perché si è concentrati perché la cosiddetta banda passante della comunicazione è molto più alta e via dicendo. Questo è un altro motivo per andarci. Ritengo per molti sviluppatori Debian non sia il motivo principale e per chi come me ci va da tanti anni è perché li hai tanti amici, si sono formate delle relazioni in 10, 11 anni di Debconf e andarci e un modo per ritrovarsi. Quindi questi sono per me i tre motivi principali per cui si va a Debconf e come Debian project leader ho invitato molta gente a partecipare sul primo motivo [i conflitti]. Spesso diciamo che abbiamo dei conflitti all'interno della comunità. Le persone che partecipano a questi conflitti se vanno alla Debconf saranno meno disposti a creare conflitti in futuro con le persone con cui hanno condiviso un'esperienza così forte. Abbiamo addirittura avuto un programma apposta in cui si invitava la gente che non era mai andata a Debconf, quindi o novellini di Debian, o gente che negli ultimi cinque anni non è mai andata a Debconf, ad andarci proprio su questa base. Se tu non ci sei mai andato, beh allora sei un target ideale per andarci perché la tua percezione sulla comunità Debian ha buone possibilità di cambiare significativamente.

I: Ma secondo te è importante andarci almeno una volta...

Z: Diciamo che cercavo il punto...

I: perché è importante? Cioè è importante andarci almeno una volta o è importante andarci frequentemente?

Z: Capisco perfettamente la domanda. Io cercavo il punto del peccato originale, nel senso che credo faccia molto di più il “andarci la prima volta senza esserci mai andato” che il “tornarvi se ci sei già stato”. Entrambe le cose sono importanti nel senso che cose come “il senso di comunità...

I: cambia il significato?

Z: cioè, io la vedo nell'ottica manageriale, quindi c'è una comunità Debian che ogni tanto ha degli attriti. Quindi: cosa possiamo fare per migliorare questa situazione? Allora trovo che andarci per la prima volta senza esserci mai andato serva tantissimo a migliorare la situazione. Tornarci se ci sei già andato contribuisce ma non così tanto come la prima volta. Questo se vogliamo dargli un'ottica più manageriale dell'utilità di partecipare a questi eventi. Poi c'è che non è che Debconf sia speciale. Cioè Debconf è l'unico evento annuale che abbiamo in cui tutti gli sviluppatori sono interessati a partecipare, che gira per il mondo così almeno periodicamente ti capiti che è abbastanza vicino a te, ma se ci fossero altri eventi in cui ci si incontra con tutta la comunità andrebbero altrettanto bene. Semplicemente l'unico evento che oggi abbiamo di questo tipo è Debconf. Ci sono altri eventi nei quali ci incontriamo. C'è Fosdem, che è la più grossa conferenza di software libero europea che si tiene in febbraio a Bruxelles tutti gli anni, ma necessariamente è una cosa più specifica per gli europei. C'è Linux Conference Australia che è un altro evento grosso dove c'è tantissima gente Debian che va. Però l'unico evento mondiale che attira tutti della comunità Debian è Debconf. Non c'è un ruolo speciale in un evento che si chiama Debconf, c'è un ruolo speciale in un evento che riesce ad attirare sviluppatori Debian da tutto il mondo per incontrarsi di persona.

I: Dal punto di vista del coordinamento...

Z: dell'organizzazione?

I: Sì lo tiro fuori perché un po' l'hai citato, ci sei passato vicino a questo aspetto, quindi al coordinamento delle attività e del progetto. Cioè tu dici che il

fatto di trovarsi fisicamente migliora poi per prosieguo del progetto il coordinamento tra le persone. Dal punto di vista del coordinamento a me interessava sapere, misurare quanto è il coordinamento pianificato da un qualche organo, da una qualche struttura formalizzata all'interno della community Debian, o piuttosto quanto è l'auto-coordinamento, o quanto viene prima l'uno e quanto viene dopo l'altro. Se è già previsto l'auto-coordinamento a livello di pianificazione, oppure se il coordinamento, la pianificazione avviene cercando di pianificare quello che è già un assetto organizzativo che si è già instaurato. Quindi riuscire a misurare queste cose e capire in che rapporto stanno tra loro.

Z: Ok, ma Debian fin dalle origine è un progetto senza troppe gerarchie, nel senso che una distribuzione linux ha nativamente una divisione di compiti attorno ai pacchetti software che si rilasciano. Inizialmente Debian non era così, Debian era proprio “tutti possono lavorare su tutto”, poi ad un certo punto nella storia del progetto si è introdotto il ruolo di mantainer di un pacchetto, quindi una persona che aveva una specifica responsabilità su un certo pacchetto, e poi in un'altra evoluzione storica del progetto questo pacchetto, che tutt'ora esiste, è stato generalizzato di fatto, nel senso che ora abbiamo molti team di persone che collettivamente si occupano di un certo numero di pacchetti. Quindi c'è una granularità implicita in un progetto come Debian che è quella del pacchetto, ok? Attorno a questo granularità si sono sviluppati nei ricorsi e concorsi storici dei gruppi di mantenimento di questi pacchetti. Quindi questa è la struttura di base che esiste in Debian possiamo dire che ad oggi abbiamo tanti gruppi di persone che lavorano assieme a mantenere gruppi di pacchetti. Questa è la struttura di base che fa la maggior parte del lavoro in Debian. Al di sopra di questo abbiamo delle strutture prescritte dalla costituzione Debian che hanno dei ruoli un po' più infrastrutturali se non politici. Quindi ci sono tutti gli strumenti per far funzionare la costituzione Debian. C'è il project leader, ci sono i suoi delegati. Il segretario e tutti i macchinari che servono per fare andare avanti i meccanismi di voto. C'è il comitato tecnico, nel caso ci siano conflitti di natura tecnica tra persone all'interno del progetto. È una specie di, come dire, “tribunale” a cui ci si può appellare per risolvere conflitti di natura tecnica. Ci sono quelli che chiamiamo core teams, che hanno dei ruoli specifici e che sono delegati del proprio project leader, ma che hanno dei ruoli politici abbastanza importanti anche se vuoi di “divisione dei poteri”. C'è il cosiddetto DAM, Debian Account Manager, che sono un gruppo di persone che decide chi può diventare sviluppatore Debian e chi no. C'è il gruppo di DSA, Debbian System Administrator, che sono quelli che mantengono tutte le macchine, tutti i server del progetto Debian in giro per il mondo, ed in generale tutta la infrastruttura tecnica. C'è un gruppo chiamato FTP masters, che sono

quelli responsabili su cosa entra nell'archivio software Debian e cosa no, e che hanno un potere molto significativo, nel senso che sono quelli che decidono al momento quali sono le licenza che sono accettabili nell'archivio Debian, decidono quando un pacchetto non raggiunge i criteri minimi di qualità per entrare nell'archivio e tutte 'ste cose. Poi c'è un gruppo di Keyring Manegers che sono quelli che gestiscono il gruppo di chiavi e di firme digitali che servono a firmare i pacchetti che possono entrare nell'archivio. Quindi questi sono i gruppi di potere principali. All'interno di questi gruppi, la vita tecnica del progetto è principalmente aut-organizzata. Cioè ogni gruppo ha una sua agenda che va avanti a seconda della disponibilità di tempo delle varie persone, e funziona tutto senza un coordinamento centrale. Quindi storicamente ruoli come il mio, di Debian Project leader, non è che abbia avuto un'agenda da seguire per organizzare la vita del progetto. Dimenticavo, c'è ovviamente il release [rilascio di una versione] Team che è un gruppo di persone responsabili di valutare il programma verso una release [rilascio di una versione] Debian che è l'outcome principale di un progetto come Debian: fare una nuova release [rilascio di una versione]. C'è questo gruppo di persone che si preoccupa di coordinare le release [rilascio di una versione]. Quindi attorno a tutti questi teams ognuno ha una sua agenda, ed in particolar modo il release [rilascio di una versione] Team ha un'agenda per capire quando faremo una prossima versione Debian e la vita del progetto gira un po' attorno a queste agende auto-organizzate. Prendiamo ad esempio un'agenda di un team che gestisce un pacchetto relativo ad un certo argomento, per esempio GNOME, l'evoluzione classica è che c'è una “shedule” [cronogramma] per fare una release [rilascio di una versione], il team GNOME decide quanto può fare all'interno di questo schedule, lavora e cerca di finalizzare il suo lavoro in modo che sia pronto nel momento in cui si decide che ci sarà una release [rilascio di una versione].

I: Queste agende possono somigliare a dei diagrammi Gannt?

Z: No! Io sono particolarmente allergico ai diagrammi Gannt perché c'ho avuto a che fare per progetti di ricerca europei e trovo che siamo, come dire …. , un po' troppo constraining, cioè come dire che mettano troppi paletti per riuscire a rappresentare cosa accade realmente nella realtà. Penso, questa è l'idea di base, poi ci sono le eccezioni, e il diagramma non è bravo per niente a rappresentare le eccezioni. Un esempio tecnico calzante è che per arrivare ad una release [rilascio di una versione] ad un certo punto decidiamo che c'è una “freeze” [congelamento], cioè il momento in cui nuove modifiche nel software non avvengono più, cioè si fissano, si mettono a posto i problemi con il software che che si ha ma non si introducono cambiamenti radicali e quindi si tende a

compattare il lavoro in vista di una release [rilascio di una versione]. A questa policy [politica] generale abbiamo le eccezioni, perché non è sempre applicabile, ci sono dei casi in cui bisogna derogare a questa policy. Quindi secondo me no, non è lo strumento che si possa usare per modellare queste agende, poi credo che nessuno ci abbia mai provato. Nessuno in Debian accetterebbe mai l'idea di essere vincolato ad un Gannt per sapere in quale direzione andare, però magari a posteriori è possibile riconoscere un'evoluzione di queste cose con uno strumento del genere.

I: Si potrebbero fare dei Gannt a posteriori?

Z: Forse si, non lo so, non so se nessuno ci ha mai provato, né tanto meno quando dico che ci sono delle agende di ogni team non so siano mai sempre formalizzate. Magari ci sono dei team dove dicono: “Ok, questa è la lista delle cose che vogliamo fare prima della prossima release [rilascio di una versione]”, e dei teams in cui il lavoro si sviluppa naturalmente senza che nessuno abbia nemmeno fatto una lista di cose da fare. Quindi c'è molta eterogeneità in Debian, ci sono dei trend generali che si possono riconoscere. Ma c'è molta eterogeneità. Per concludere la cosa sulla tua domanda quindi direi, se dovessi valutare dov'è, è molto più sull'auto-organizzazione e non sull'organizzazione centrale, con il difetto che alcuni cambiamenti, diciamo grossi, alcuni cambiamenti importanti, filosofici o politici nel progetto non hanno nessun responsabile che se ne occupi. Quindi il Debian Project Leader dovrebbe preoccuparsi di accorgersi di queste necessità di cambiamento e guidare un po' la comunità prendendo una decisione in un verso piuttosto che in un altro. E questa è una cosa che nella costituzione Debian c'è. Nella costituzione debian c'è scritto che il project leader è colui che guida le discussioni, io direi che è quello che tiene d'occhio l'agenda del progetto. Però quando dico agenda del progetto è più su meta-cambiamenti, vogliamo chiamarli, che non su la naturale evoluzione del software che funziona già benissimo così com'è in Debian.

I: Ho capito, benissimo. Poi io ti avevo proposto un'interpretazione di questa frase di Eric Steven Raymond.

Z: La leggo a me stesso: “Gli hacker sono anti-autoritari di natura, chiunque può darti degli ordini può fermarti dal risolvere problemi dai quali sei affascinato, e, visto il modo in cui le menti autoritarie funzionano, tali ordini saranno generalmente motivati da ragioni orribilmente stupide. Così l'atteggiamento autoritario deve essere combattuto ovunque si trovi affinché non soffochi te e gli

altri hacker.” Dunque si, questa è una visione forse non so, naive o un po' vecchia di come funzionano queste cose, nel senso che, insomma Eric Steven Raymond ha molto in testa la visione dell'hacker che da solo può cambiare tutto, che si applica bene quando si lavora da soli, in pratica un po' meno bene quando si lavora in gruppo. Cioè fare una release [rilascio di una versione] di una distribuzione Linux che ormai conta quasi quarantamila pacchetti ha necessariamente una dose di coordinamento al suo interno. Cioè l'anarchia totale nel fare un prodotto così grande è impensabile. Quindi anche se noi abbiamo aree di responsabilità specifiche c'è bisogno di un certo coordinamento e il coordinamento necessita di un minimo di autorità. Quindi la release [rilascio di una versione] team che c'è in Debian se vogliamo è autoritario se vogliamo nei cambiamenti che lui fa al software. Nel senso che ha un potere di dire: “basta adesso non facciamo più cambiamenti di un certo tipo ma ci occupiamo solo di risolvere i problemi che ci impediscono di fare una release [rilascio di una versione].” Quindi in questo senso, questo tipo di autorità nel progetto Debian è accettato, ma è accettata nel senso che l'anarchia totale semplicemente non funziona in questa scala così grande. Di contro direi che negli anni abbiamo costruito un pò troppi paletti alle contribuzioni. Cioè questo fatto di avere i mantainer associati ad ogni singolo pacchetto, ha fatto si che abbiamo una procedura che spiega quando un altro mantainer ha fatto un upload di un pacchetto non suo che si chiama no-mantainer upload, e negli anni abbiamo avuto la percezione che fosse qualcosa di non desiderato, come una specie di onta, ad esempio: io ho fatto l'upload di un tuo pacchetto? è un'onta! Anche se il tuo pacchetto era mantenuto male, era in pessime condizioni e cose di questo tipo. È una cosa su cui ho lavorato molto e ho trovato molta accettazione da parte della comunità ed è come dire: siamo un po più liberali su queste cose. Cioè di permettere, di invogliare, di accettare un po di più il fatto che altri possano lavorare sui tuoi pacchetti, Magari come back-up, magari perché non hai tempo. Quindi in un certo senso questo trend spinge un po più verso l'anarchia, un po meno verso la coordinazione, però, secondo me, questo tipo di visione funziona in progetti relativamente piccoli che non hanno bisogno di coordinamento.

I: ho capito.

Z: Poi è senz'altro vero che un modo fantastico di vedere i propri cambiamenti accettati da altri è quello che si chiama “show me the code”, cioè si fa vedere che un cambiamento di cui si sta discutendo da secoli è possibile, una volta che c'è l'evidenza in termini di codice che il cambiamento è possibile è più facile che collettivamente si decida che quella è la via giusta. Quindi diciamo:

coordinamento non vuol dire discussione perenne su tutto, coordinamento vuol dire che ci sono delle responsabilità, che si possono prendere delle decisioni collettivamente ma c'è un primato che dice che le cose si possono fare in un modo soltanto; che è uno strumento fortissimo per convincere di una certa posizione.

I: Ok, però senti una cosa, a me questa frase Eric Steven Raymond sembrava mettesse in evidenza il fatto che c'è qualcuno che può impedirti di risolvere dei problemi, o che può impedirti di fare delle cose. Questo io non l'avevo letto all'interno di un progetto o di una community, ma l'avevo letto come tensioni nei confronti del mondo, poi tu avevi colto l'anno abbastanza pionieristico …

Z: ma ti ho sottolineato l'anno perché ho l'impressioni che ci siano visioni di come funzionano nel mondo dei geek diverse. Diciamo una visione preliminare. Molto hacker, molto “one man show”, “che belle queste comunità che non si incontrano mai di persona”. Mentre adesso stiamo realizzando che molte cose non erano come le immaginavamo all'inizio. Quindi coordinamento, quindi andare ai meeting di persona. Ho l'impressione che le visioni stiano un po' cambiando. Probabilmente è legato al fatto che il free software si sta diffondendo, sta aumentando tantissimo la scala nella quale si sa cos'è il software libero, nella quale si sa come si sviluppa software libero e quindi bisogna anche un po' adattare il modo in cui si lavora per restare e raggiungere questa nuova scala.

I: È cambiata questa tensione nei confronti del mondo? Comunque lui vuole ostentare questa tensione tra il mondo hacker ed il resto del mondo.

Z: È che ci sono molte visioni che si mescolano nel software libero: C'è questa visione hacker, un po' anarchica, anti-autoritaria, del potere di fare quello che si vuole col codice; c'è la visione open source che ha una visione molto pragmatica in cui si dice: “ok, il software libero ci piace perché queste licenze ci danno gli strumenti per cambiare le cose, per essere di nuovo padroni delle cose che usiamo; e c'è quella del software libero, che è quella dei diritti degli utenti e quindi la possibilità di modificare il software, distribuirlo eccetera. Tradizionalmente Debian nasce dalla visione software libero. Nella visione software libero non è cambiato molto, la visione del mondo è sempre quella, la visione del mondo e: “gli utenti devono essere messi nelle condizioni di vedere, toccare, modificare, redistribuire il software che usa”, ok? Questa visione è tuttora più che moderna e non è cambiata per niente. Questa visione però non ti dice come si ottiene il risultato di portare il software libero a tutti, in tutto il mondo. Quella che i geek chiamano la “total world mination” del software libero. Questa

visione ti dice semplicemente qual è lo scopo e quali sono i motivi per cui si fa questo. I modi in cui si lavora per raggiungere quello scopo non è descritto nella visione di Eric Steven Raymond, secondo me è quello che è un po' cambiato. Cioè Eric Steven Raymond stesso, il progetto per cui è famoso è Fetchmail, credo che sia un progetto in cui c'è una sola persona che tiene le fila, certo ci sono patch, contributi da tutti quanti, però controversa. Un progetto in cui c'è una persona con un ruolo chiave. Poi altri possono "forkare" [derivare altri progetti indipendenti dall'originale] il codice e tutte e via dicendo. Ma tutti oggi siamo in una visione del software libero un po' più organizzato secondo me. Perché per stare dietro alla scala che abbiamo raggiunto adesso serve un peletto in più di organizzazione. Poi non è vero dappertutto, tieni presente che la mia visione è quella di Debian che è un progetto immenso, è uno dei più grandi che c'è in termini di numero di sviluppatori, in termini di software che viene distribuito alla gente, quindi è possibile che io sia un po' "biased" [di parte] orientato verso questo contesto, e quindi l'organizzazione serve, non c'è niente da fare. Poi all'interno delle nostre scatolette di responsabilità noi abbiamo tutta la libertà. Nel senso che uno può lavorare come vuole sulla sua parte di progetto, una delle cose fondamentali, che la costituzione Debian dice, è che ognuno è libero di prendere tutte le decisioni che vuole, tecniche o non tecniche, all'interno della sua aree di responsabilità, però c'è sempre questo limite esterno della tua responsabilità.

I: Ok, poi questa domanda …

Z: Cos'è?

I: Episodi significativi della esperienza professionale, però si potrebbe ricollegare anche a questo, cioè, almeno io l'avevo vista così, nel senso che …

Z: Aspetta, cos'è questo technitai descritto da Gian Antonio Gilli? [legge dal foglio della traccia che mi ero predisposto].

I: È un sociologo, tra l'altro quello che io sto facendo nella tesi è quello di cercare di applicare le teorie della sociologia al fenomeno open source e Gian Antonio Gilli descrive partendo dall'antica Grecia la figura del technitai che è un personaggi abbastanza diffuso, che in termini generali lo si può riconoscere nell'artista, nell'artigiano, nelle persone creative in genere e descrive come, partendo dall'antica Grecia all'età moderna ci siano delle costanti nei confronti di queste persone per cui la società ha sempre avuto il problema in qualche maniera di controllarli.

Z: Ok.

I: Quindi in qualche maniera di metterli sotto controllo. Poi Gian Antonio Gilli porta l'esempio di come nell'antica Grecia le persone che noi oggi chiameremmo “skillate” [dotate] dovevano assolutamente essere sottomesse e nella condizione di schiavi e da qui la figura dello schiavo intellettuale dell'antica Grecia. Poi lui porta tutta una serie di esempi passando per il Medio-Evo fino ad arrivare ai giorni nostri in cui evidenzia come la società abbia sempre avuto il problema di controllare, istituzionalizzare, e in qualche maniera anche tenerle isolate, controllate alla fine. La frase che sintetizza questo è: “preghiamo i nostri dei di fronte alle opere d'arte ma disprezziamo chi le ha fatte”. Sintetizza l'idea di questo technitai utile per la società ma da un altro punto di vista anche pericolosa, perché imprevedibile. Potrei dire nel caso, dell'open source, il mio problema dal punto di vista industriale è sapere che tipo di struttura dargli, che ruolo dargli all'interno della nostra società dove vige l'economia di scala, dove vige il lavoro parcellizzato, dove vige la funzione della domanda e dell'offerta, dove vige il discorso che le risorse devono essere scarse perché altrimenti non funziona più il grafico della domanda e dell'offerta. Quindi mi pone il problema di sapere cos'è 'sta cosa è e come posso controllarla.

Z: Ma guarda, i geek, come li conosco, del software libero non hanno tanti problemi di gabbie sociali nelle quali vengono messi per impedirgli di esprimersi. Le gabbie contro le quali i geek si ribellano sono le gabbie tecniche. Sono il fatto che tu mi possa dare del software proprietario sul quale io non posso fare nulla o che ho difficoltà a modificare perché non ho un codice sorgente. Quindi adesso non so bene se il technitai avesse coscienza o meno di questa gabbia sociale nel quale può essere messo. I geek come li conosco non hanno problemi di accettazione sociale, anche perché non gliene frega nulla del riconoscimento sociale. Vogliono essere messi in condizioni di possedere e usare tutta la tecnologia software o altro che gli viene messa a disposizione. Quindi la pulsione è nel “io voglio poter fare tutto ciò che voglio con questo software e arrivare fin dove le mie capacità mi permettono” e il software libero, le licenze del software libero sono l'elemento essenziale che permette e che da questa possibilità. Quindi anche la visione hacker se vuoi e del software libero è principalmente “non incatenatemi con meccanismi tecnici che mi impediscono di fare ciò che voglio”, che mi impediscono di divertirmi come piace a me, mi impediscono di mettere in uso le mie skills [capacità – competenze] al meglio. Quindi vi è più una battaglia contro queste costrizioni tecniche. La visione del software libero è un po' più politica e dice non solo gli hacker vogliono questa cosa qua, ma è un dovere

morale della società tecnica di oggi di dare a tutti gli utenti questa possibilità, anche se non se lo possono permettere, anche se non lo sanno fare, questo valore politico di libertà che devono dare a tutti gli utenti di software. Io personalmente la vedo un po' più sulle barriere tecniche che non sulle gabbie sociali. Anche perché la maggior parte del lavoro viene fatto … ma in Debian siamo tutti volontari. C'è qualcuno che è pagato per lavorare su qualche aspetto certamente, ma il progetto è un progetto su base assolutamente volontaria. Quindi qualunque gabbia sociale tu voglia mettere agli individui è gente che è libera di farlo nel proprio tempo libero, salvo che non capiti di avere firmato delle clausole assurde in qualche contratto come capita a qualche americano. La mia visione è più sulle gabbie tecniche che non su quelle sociali.

I: Comunque una rivendicazione di spazio, è comunque difficile dire dove finisce la gabbia tecnica e dove invece rivendico uno spazio che sia virtuale o fisico e dove comunque io possa esprimermi. Tu prima hai usato il termine divertirmi, però divertirmi può anche voler dire auto-realizzarmi.

Z: Hai ragione, la differenza è sul fatto che con le barriere tecniche che ci sono con il software proprietario, magari per te sono degli ostacoli sociali, per me no, per me semplicemente è un'idea sbagliata di come si fa economia col software e che è arrivata prima di tutti noi purtroppo, è riuscita ad imporsi sul mercato come cosa giusta, cosa che io ritengo non sia giusta per niente e per questo l'obiettivo principale è liberarsi di questi blocchi, poi se tu ci vedi un disegno dietro o un ruolo sociale questo è possibile chiaramente. Non so se è un disegno ma se riesci a verificarla come vincolo sociale, beh è possibilissimo.

I: Beh, adesso posso anche dirlo perché siamo alla fine di 'sto percorso. Quello che in qualche modo io sto cercando di dimostrare con questo, lo dico adesso …

Z: Così non mi hai influenzato …

I: … è di, partendo da questa rivendicazione di questo spazio, poi tu hai usato anche termini come “non è giusto” che presuppongono un giudizio morale …

Z: Di valore …

I: si, di valore che prescinde dagli aspetti tecnici, tu parli di spazi tecnici, però dici ad un certo punto non è giusto, a questo punto siamo legittimati a parlare

di rivendicazione di uno spazio, chiamalo tecnico, di auto-realizzazione o professionale. Ci sarà l'individuo che sarà più interessato a fare business e quello che sarà più interessato a divertirsi. Le motivazioni saranno miscelate diversamente all'interno degli individui. Però quello che io sto cercando di …

Z: Aspetta, giusto una chiarificazione su questo punto. Abbiamo parlato di visioni diverse del fenomeno che tu chiami open source ma che è un po più vasto. La visione del divertirsi è prettamente quella hacker e open source se vogliamo. Il giusto non giusto viene chiaramente dalla visione software libero, free software in inglese, che è una visione prettamente politica, politica in senso lato ovviamente, ed è una visione dove la libertà del software è un valore morale come giustamente hai detto tu e denota tutto un insieme di valori che, io personalmente, ma molta gente in Debian usa per valutare se qualcosa è giusto o no. Quindi in molte comunità del software libero troverai gente che ha questo tipo di valori morali, perché veramente pensiamo che il software proprietario sia un'ingiustizia sociale. Per tornare al tuo principio dell'economia del dove li metto in un contesto economico. Secondo me l'errore di fondo è che sia stato accettato che l'economia del software proprietario funziona facendo pagare il costo di una copia che al produttore non costa nulla, ok? Cioè non è come quando c'è un bene fisico, un'auto o qualsiasi altra cosa che c'è nella nostra società in cui c'è tutta una parte di ricerca e sviluppo a priori, poi c'è la costruzione di un bene fisico che costa qualcosa a chi la produce e che costa qualcosa a chi la compra e in mezzo c'è il guadagno di chi la produce. Nel software l'ottica di vendere il software copia per copia facendo pagare la singola copia è un'ingiustizia nel senso che il produttore non ha alcun costo nel produrre una copia oltre alla prima.

I: Chris DiBona usa il termine “stampare moneta”.

Z: Ecco si, non lo sapevo però esattamente. Cioè, se io di la ho una copia di Ms Windows, o qualsiasi altro software proprietari chi lo produce non ha nessun tipo di costo nel vendere una copia a me e poi vendere una copia a te. Quindi perché deve essere accettabile che ci sia un'economia che di fatto ha accomunato questa tipo di vendita alla vendita di un bene fisico. Anche l'idea che la pirateria è furto è radicata nell'idea sottostante che se io pirato [faccio una copia illegale] il software ti ho rubato il software. Ma se io ti rubo l'auto tu non hai più l'auto e io si, se io ti rubo il software nel senso che lo pirato, tu ce l'hai ancora quel software, quindi l'analogia da qualche parte non funziona. E io personalmente credo che l'ingiustizia del software proprietario venga da lì, venga da questa analogia. Sono stati bravi quelli che hanno portato l'economia del software proprietario ad

imporlo alla società, ma quello è l'errore di base secondo me. È invece giustissimo pagare per lo sviluppo di un software! Perché quello costa un sacco di soldi a chi lo fa che ha tutto un risvolto economico, ma che deve essere un costo di sviluppo del software non un prezzo. Non deve essere a posteriori un privilegio del vendere copie del software che sono tutte indistinguibili l'una dall'altra e che non costa niente produrre. Quindi personalmente penso che questa è la radice del valore morale che associato è al software libero. E quando dico è “giusto o non è giusto” lo dico spesso con questo pensiero in back ground.

I: Non l'abbiamo nominato ma sotto c'è il problema del monopolio dei sistemi operativi poi alla fine, no? O meglio, quello che meglio esprime questo problema è questo aspetto. Il fatto sostanzialmente, di questo sistema operativo che ha il monopolio, che lo ha avuto fino adesso e che forse in qualche maniera adesso è minacciato però sostanzialmente ha fornito uno schema di senso al mondo intero e il mondo intero in qualche maniera gli è andato dietro.

Z: secondo me non è solo uno schema di senso di sistemi operativi, ma è uno schema di come si fanno i soldi nel mercato del software. Quindi Microsoft è riuscita a convincere il mondo che i soldi si fanno.

I: Da parte di Microsoft, da parte di chi lo utilizza ha rappresentato uno schema di senso.

Z: Dici interfaccia, sistema di riferimento, mi sfugge il significato di “schema di senso”.

I: L'immagine che io mi faccio di una determinata realtà, i paletti che metto per identificare quella determinata realtà, quella determinata complessità o quella determinata tecnologia che mi sta entrando violentemente nel mio ufficio, nella mia vita e via dicendo. Potrei sintetizzarlo così: ha fornito, in una situazione se vuoi di Babele in cui ognuno parla la sua lingua, lui è arrivato e ha presentato questo modo per parlarsi …

Z: Questo secondo me non è un grosso problema.

I: No no, in effetti non paghiamo le royalties a Dante Alghieri ogni volta che scriviamo un libro o che comunichiamo. Cioè per dire lo schema di senso è la lingua, la scrittura cioè il medium che usi per fare delle cose, come il sistema operativo è un medium tra te e la macchina, tra te e altre persone, tra te e la rete.

Tra te e il mondo tecnologico. È un medium comunicativo monopolizzato su cui paghiamo delle royalties.

Z: Mi hai stimolato vari commenti, ma non capisco come tu veda così pervasivo questo schema di senso che ci è venuto dai sistemi Microsoft? Se lo vedi come il fatto che è difficile cambiare, non lo vedo come un problema.

I: No, no la mia è solo una analisi a posteriori di quello che è successo.

Z: loro sono quelli che lo hanno popolarizzato di più, questo è fuori di dubbio, negli anni '90 i personal computers avevano sopra windows e basta, sono quelli che li hanno resi più popolari di tutti indubbiamente, ma sia i sistemi operativi che le interfacce esistevano da ben prima, le metafore per l'interazione con le macchine esistevano da ben prima e sono sostanzialmente le stesse che ci sono tuttora oggi, oggi c'è stata un po di più evoluzione ma questo è un altro discorso. Ora non so come si definisca lo schema di senso, se si intende in termini di diffusione o se si riferisce in termini “da dove viene l'idea”, perché se il punto è l'idea, beh quella non l'hanno certo inventata loro.

I: no, in effetti, ma questo a maggior ragione rende l'idea. Sono convinto anch'io che non sia stato Dante Alghieri ad inventare la lingua italiana, caso mai l'avrà sistematizzata, forse nel caso dei sistemi operativi non è successo neanche quello. Al di la del fatto che se li sia fatti, che se li sia comprati, che se ne sia appropriato è successo e che questo abbia i giorni contati oppure no è irrilevante, io sto descrivendo quel sistema li che può finire domani mattina …

Z: può durare altri cinquant'anni …

I: La sociologia non fa previsione cerca solo di comprendere.

Commiato.

19 anni, diplomato in Informatica all'ITIS, iscritto al primo di scienze informatiche a Venezia, volontario nei LUG.

Premessa:

Si tratta di un ragazzo con cui, attraverso la community, ho interagito per alcuni anni, è stata la prima volta che l'ho incontrato di persona. L'intervista si svolge in una pizzeria dove ci eravamo dati appuntamento. I nomi di persone qui riportati sono tutti nomi di fantasia.

Intervista:

I: Io, Stefano De Boni Laureando in scienze sociologiche.

A: Andrea.

I: Lo sai che ho intervistato anche Stefano Zacchiroli? Sai chi è?

A: No!

I: È il leader mondiale di Debian Project.

A: Ah si, me l'avevi già detto, ma perché se vai ad intervistare uno così importante poi vieni ad intervistare me? C'è troppo gap!

I: Si cercano diverse tipologie di persone di verse, lui ha anche un'età diversa dalla tua. Tu hai 18, 19 anni?

A: 19!

I: Tu stai ancora facendo le superiori?

A: Ho finito, adesso vado a Venezia a fare informatica, ma faccio anche altre cose. Nell'ITIS ho curato una distribuzione specifica per le scuole, che è un progetto che ho fatto per l'esame. Per l'esame hai due possibilità negli istituti tecnici: o fai una tesina normale, come fanno ai licei; oppure puoi fare un progetto, allora il progetto che può essere un programma, oppure una configurazione particolare di un sistema, ad esempio i miei compagni hanno fatto videosorveglianza, e io ho fatto questa distribuzione Linux per le scuole. Poi la presenti alla commissione, all'esame hai dieci minuti per parlare di un argomento a piacere e quindi parli del progetto. Quindi io ho fatto una distribuzione derivata da Ubuntu che si chiama Skolelinux pensata per gli istituti tecnici. L'ho fatto assieme ad un mio compagno e adesso abbiamo deciso …

Cameriere: Buona sera avete deciso?

I: Si, io prendo una capricciosa.

A: mi da un paio di minuti?

Cameriere: Ok!

A: io sono sempre lento a decidere … e adesso vogliamo svilupparlo con un'azienda pugliese …adesso in ogni pizza mettono l'origano... qui sono specializzati nelle pizza, hanno quattro facciate solo di pizze... io sono lentissimo a scegliere le pizze e anche a mangiarle...

I: mi dicevi che lo stai sviluppando con un'azienda pugliese?

A: era partito come un progetto per l'esame, però poi io ho detto al mio compagno: “guarda Luca siccome è una distribuzione alla fine non ha senso che ce la facciamo per noi, cioè la usiamo solo noi due alla fine. Tu fai un sito web con tre pagine per spiegare quello che stai facendo, poi spargi un po' la voce. Qualcun altro potrebbe vederla”. Abbiamo fatto un sito web, ho scritto io un articolo sul giornale, passa parola e vai dicendo, alla fine questa azienda ci ha notato e quindi adesso è l'azienda ufficiale che porta avanti il progetto e quindi l'azienda si occupa di tutta la pubblicità, nel senso che va a parlare con gli enti, con le scuole, con i consigli di amministrazioni comunali, e si occupa di proporla un po' dappertutto. Abbiamo il patrocinio del comune di Vicenza, il patrocinio del comune di Modena, adesso c'è la regione Puglia che sta considerando l'offerta più

economica. Grazie ad un sponsor faremo delle copie omaggio da mandare ai LUG per il Linux Day. Adesso stiamo facendo delle modifiche alla versione che abbiamo portato all'esame, piccoli ritocchi, miglioramenti, correzioni, perché ci eravamo dimenticati qualche pacchetto. Adesso stiamo rivedendo la parte che riguarda il software libero. Cioè ci sono moltissimi software liberi, però alcuni non liberi sul DVD …

Cameriere: Allora

I: una capricciosa.

A: Salamino piccante e patatine.

Cameriere: Da bere?

I: una birra da mezzo.

A: Una coca cola da mezzo.

Cameriere: A posto così? Grazie!

I: Luca è un tuo compagno?

A: Sì, di banco.

I: Su questo DVD avete messo anche dei moduli di software proprietario, mi dicevi?

A: Sì, ad esempio Falsh, drivers ndivia, skype, cose di questo tipo, programmi comuni. Lo scopo principale era di farlo vedere alla commissione, ma anche un po' in generale, che per la scuola ci può essere qualcosa di pronto, non c'è bisogno di perder tempo a fare tante configurazioni. Si può fare qualcosa che è già pronto e che basta installarlo e funziona. Metti il live [con un sistema che si autoesegue autonomamente sul computer che lo ospita] DVD nel computer e hai già tutto quello che serve per fare informatica, scienze, biologia, chimica ma soprattuta informatica e elettronica. Questi moduli proprietari sono poca roba in realtà. Non sono nell'installazione, c'è una piccola interfaccia grafica, se l'utente vuole se li aggiunge.

I: Sceglie lui se usare software proprietario oppure no?

A: Esatto!

I: tu seguivi dei progetti, mi sembrava tre progetti, tra cui Gimp [programma di grafica].

A: Si Gimp, ITIS Linux e Tuxfeed che seguo ogni tanto. In Italia è una community che è nata nel 2004 come supporto per gli utenti, perché a quel tempo non c'era nessun sito che parlasse di Gimp. È stato il primo sito e quindi anche uno dei più conosciuti. Adesso stanno cominciando a nascere molti siti su Gimp. Però secondo me non è til massimo dividersi in tanti parti, perché molte persone stanno su quel sito o su quell'altro, per cui sono tanti gruppetti che ….

I: Si disperde energia?

A: esatto! TuxFeed invece è un aggregatore di research su Linux, è il primo aggregatore che è nato in Italia e tutti gli altri sono copie di TuxFeed alla fine. Non mi interessa dirlo perché non l'ho fatto io, mi occupo anche di altro. L'unica roba che faccio e di aiutarli ogni a gestire i nuovi utenti che si iscrivono. Perché quando i nuovi blog si iscrivono, si segnalano: “questo è il mio feed”. Ci sono dei criteri per valutare se rientrano nell'ambito di attinenza. Quindi io visito il blog, leggo gli articoli, poi dico se sono pertinenti agli argomenti, se il blog ha una certa anzianità nel senso che... abbiamo fissato come quota tre mesi, se un blog è più giovane di tre mesi non lo inseriamo per poi vederlo morire il giorno dopo, se invece vedi che è già un attimo stabilito … ma è una cosa che faccio una volta ogni tre mesi, non è molto impegnativa.

I: Come mai hai deciso di andare a Venezia a fare informatica?

A: Con il treno ci arrivi in un'ora e venti. All'inizio ho detto guarda: “c'è informatica a Trento, c'è informatica a Padova”; e quindi stavo vedendo quelle due opzioni lì. Però Padova costa il doppio di Trento, è il triplo più affollata, sei seguito di meno, però vai a Padova solo perché ha il nome, e quindi se ti laurei a Padova sei più figo. E quella era meno distante da casa mia. Trento costava molto meno, era più giovane, come modo di insegnare, come modo di lavorare, solo che era molto distante e poi è … “stata comprata da Microsoft” perché ci butta dentro un sacco di soldi. Hanno un laboratorio lì, quindi è ovvio che tutti i lavori poi se li prendono loro e loro pagano l'università e pochi soldi arrivano dal ministero. Se csei un ricercatore che è pagato coi soldi della Microsoft devi stare molto attento perché se programmi con il software libero sei obbligato a tenerlo libero. Se invece parti da zero il lavoro rimane di proprietà dell'università e l'università la

paga la Microsoft perché ci ha messo un sacco di soldi, quindi c'è il rischio di coinvolgimento. Poi c'è anche la distanza da valutare, per alcune scelte puoi decidere di rimanere a casa e fare il pendolare, per altre scelte devi trasferirti e stare lì. Venezia: ci io sono andato con la scuola in visita, abbiamo fatto un laboratorio, non mi ricordo più se sono due o tre giornate che abbiamo fatto staccate. Ci hanno fatto delle lezioni c'erano dei ricercatori o dei professori che ti facevano delle lezioni e poi si faceva laboratorio. È stato molto interessante, anche perché c'erano dei professori che ti stimolavano molto sulla ricerca sulla tecnologia informatica, anche sulla manipolazione delle immagini che è il campo che mi interessa, poi anche per il fatto che comunque è, bene o male, vicina e posso anche seguire altre cose. Banalmente, fare sport, nel momento in cui mi trasferisco a Trento non posso più andare in palestra e fare Karate, non è che lo fai con qualsiasi maestro, c'è il tuo maestro e la tu squadra anche se ce ne sono tantissimi di bravi, ma se tu hai iniziato un percorso preferisci continuare con quello ..

Cameriera: La coca …

A: Preferesci continuare con quello. Non avrei più potuto andare al LUG [Linux User Group] ad esempio, qui nella mia città, non avrei potuto più fare il volontariato che faccio adesso, tutte queste cose qua per cui …

I: Dov'è che fai volontariato?

A: Nella mia città c'è un Ufficio del comune che si chiama “Cantieri Giovani”, la fanno diverse attività, organizzazioni di concerti, eventi e in più c'è un'attività che si chiama “118 compiti” che è quello che faccio come volontariato, dove, praticamente, i ragazzi delle superiori aiutano i ragazzi delle medie o dei primi anni delle superiori a fare i compiti, che non è esattamente un doposcuola ma è un po' diverso insomma. Non sono delle lezioni, si fanno compiti. Quindi sia per quello che ho visto che fanno e sia per comodità mia ho scelto di andare a Venezia.

I: Non sapevo di discorso di Trento che era finanziato da Microsoft.

A: Hanno un laboratorio di ricerca che è tutto loro. Non l'ho visto di persona, ma l'ho letto, hanno avuto un accordo con l'università, insomma hanno il loro laboratorio. Un mio amico va lì all'università, l'ha visto e me ne ha parlato.

I: L'ultima discussione che è venuta fuori sul sito del LUG, relativamente al discorso di cosa è lecito e cosa non è lecito usare.

A: ci sono alcuni, per fortuna solo alcuni, due o tre teste calde che si fissano e fanno del pressapochismo. Ad esempio “Googlemaps non è un software libero” . Innanzitutto Googlemaps non è un software, è un sito [in realtà Googlemaps è un insieme di librerie java script, cioè sono un programma eseguito sul lato client, cioè sul computer del navigatori, cioè dal browser e non in remoto sul lato server]. Se tu mi parli che non vuoi andare sui siti web che non sono liberi, allora non apri più il browser, perché tu non conosci quei sorgenti. Quello di Google no! Quello di Yahoo no! Quello di Facebook no! Qualsiasi sito web, se tu vuoi conoscere il sorgente del mio blog … sai solo che è basato su software libero, ma allora cavolo se io cambio una riga di codice nel mio blog allora tu non visiti più il mio sito? Cioè, è un sito, un sito ti fornisce informazioni. Ho fatto un esempio banale sulla lista per cercare di farli rendere conto un po'. Puoi anche considerarlo un programma in un certo senso, tu non conosci il codice sorgente PHP che ci sta dietro per capire cosa fa. Però la Free Software Foundation ha una pagina ufficiale su Facebook, perché? Perché anche loro capiscono che non si tratta di un programma. Non è che stanno installando una licenza di software proprietario nel loro computer, stanno navigando su internet che è una cosa molto diversa. Free Software Foundation è molto ferma su i concetti di base. Quindi loro sono la fermezza ed è giusto che sia anche un punto di riferimento [Un punto di riferimento etico che sostiene l'open source nella sua accezione di software libero] che sia rigido, però tu non puoi essere più rigido di chi ha creato la definizione di software libero. Non puoi dirmi una cosa ancora più restrittiva, non ha alcun senso. Quindi sentire che Googlemaps non è un software libero ... Non è un software e basta, è un sito, grazie tante anzi... questo è volersi puntare e tirar rogne. Quando c'è da organizzare un Linux day un talk o qualcosa, non li vedi, e se li vedi son li seduti che ti guardano. Poi quando tu organizzi senza di loro quando loro non si sono mai interessati ti dicono: “a me non va bene”; e questo è l'atteggiamento. Ci sono due o tre persone che fanno così, questo ovviamente mi fa girar le palle però devi sopportarli.

I: Lo stesso se tu avessi bisogno di un certificato anagrafico, che però non è stampato con un software libero.

A: Si appunto, come quando facciamo i volantini per il Linux day allora dobbiamo dire: “no quella tipografia usa Indesign, quindi non va bene, non facciamo i volantini”; ho capito anch'io che se usano Inkscape sono iù contento, e

se usano Scribe sono più contento, ma c'è un limite a tutto. Allora non vengo a mangiare qua perché la fabbrica che crea questa tovaglia usa Windows. Bisogna anche rendersi di quale sia il limite della sensatezza secondo me. Quello che ho detto a te in modo brusco, era perché era già capitato in passato, quindi non perché ce l'avessi con te, ma perché appena uno parla di Googlemaps, ma non te, tu non valevi far nessun … appena uno tira lo fuori stai sicuro che arriva un'altro … non bisogna neanche nominarlo, questo non ha nessun senso, perché le persone civili se tu parli … come [nome proprio di persona] che mi viene a dire: “ah ah ah”; ma tu devi rispettare le altre persone, puoi dire quello che pensi ma basta che non mi ridi in faccia e non mi dici me come rispettare le altre persone: “inizia te a non fare tanto il saputello”. Dopo gli ho risposto in modo quasi scherzoso in lista. Quando tu mi dici: “Googlemaps non è un software libero”, io ti dico: “discutiamo sulla definizione di software”, non ti dico: “ah ah ah”. È una cosa diversa. Poi la lista non è … cioè quello che ho scritto io non è il caso di discuterne, ho scritto … ma in lista non puoi discutere con loro perché son baruffanti … come con [nome proprio di persona], ma quando ci siamo visti di persona ci siamo capiti su quanto stavamo dicendo anche perché usiamo quasi le stesse robe in tutta tranquillità, ma per iscritto non son capaci di discutere. Perché non è il mezzo adatto, cioè gli esperti di comunicazione, non io, dicono che le cose scritte sono fraintese tra il 40 e il 50%, la voce tra il 10 e il 15%, quindi se tu scrivi non hai l'espressione del viso, non hai il tono di voce. Tutte queste cose non le hai, ed è ovvio che è più facile fraintendere e quindi dopo la gente si arrabbia. Perché magari dici: “questo mi ha offeso, mi ha detto una cattiveria”. Ma tu magari stavi scherzando o ti riferivi ad altro.

I: su quella mailing mailing li c'è molta discussione su questioni di opinione, almeno mi sembra di notare che quando le discussioni riguardano aspetti organizzativi, oppure su come risolvere il problema a qualcuno che si trova in difficoltà eccetera, allora il grado di fraintendimento è minore.

A: ma nell'organizzazione viene fuori dopo, perché quando è già stata organizzata una cosa arriva quello che dice: “non a me non andava bene così anche se non c'ero quando è stato deciso”; “Fatti tuoi”. Abbiamo parlato per mesi di organizzare l'evento tale. “tu non ti sei fatto vivo”; se mi vieni a dire: “non mi va bene niente”; cavoli tuoi e t'arrangi. C'è qualcuno che ha detto che la mailing list non è fatta per aiutare gli altri. Ma non c'è altro scopo: o aiuti qualcuno che ti vien a chiedere una mano, o organizzi talk, il Linux day, o eventi, o serate, o quello che vuoi, se no cosa lo usi a fare? Dice [nome proprio di persona] che è uno che è socio del LUG [di una città diversa da dove vive Massimo], di solito

viene nella mia città perché è più comodo. Invece ci sono due o tre, pochi, che si credono il “dio della tastiera”: “perchè io so programmare, perché io sono più bravo qua e la”; cioè sicuramente sono delle persone che hanno delle doti straordinarie, programmatori bravissimi, personaggi che hanno fatto prodotti che adesso trovi dappertutto. Però anch'io posso dire che sono più bravo in qualcosa, però non è che devo impormi su di te perché io sono più bravo.

I: però il grado di fraintendimento quando si tratta di ad esempio di uno che ha dei problemi di funzionamento il fraintendimento è minore. Il fraintendimento riguarda discussioni che riguardano le questioni di opinione, sono quelle che provocano più flame …

A: Si però anche li sarebbe da vedere perché cos'è software cos'è non è, non è tanto un'opinione dovrebbe essere abbastanza definito. Dopo ti vengono a dire che se tu … quando tu riporti dei fatti citando anche dei link come prova, tu devi difenderti come se fossi in un tribunale …

Cameriera: capricciosa? Diavola con patatine?

I: Quante ore dedichi alla settimana nei progetti che segui?

A: Dipende, Beh dipende, perché un po' sistemo il progetto, un po' mi perdo su altre cose. Ad esempio in questi giorni sto facendo uno script in bash per scaricare le slide da Slideserve che è un sito dove tu puoi vedere delle presentazioni caricate dagli utenti. Siccome in questo sito l'utente può decidere se permettere o no lo scaricamento della sua presentazione. Tempo fa c'era un sito che aggirava 'sta cosa e ti dava direttamente il download del file, adesso hanno modificato la piattaforma e non puoi più scaricarlo, allora io mi sto facendo uno script che scarica più o meno le presentazioni. In realtà scarica delle immagini ma la presentazione funziona lo stesso.

I: Bypassi l'autorizzazione?

A: si perché scarico l'immagine video che tu vedresti nel tuo Flashplayer. Non è il file originale in Powerpoint o Pdf è convertito in immagine, e io scarico quella. Non è come avere il file vero perché non hai il testo copiabile non hai niente, vabbe', ma magari puoi studiartela, o se ti serve andare in giro con la presentazione, piuttosto che niente. Cioè magari mi metto a farmi le mie robette, i programmini, così, stupidi tra virgolette, o anche magari se mi arriva una email

dall'azienda che dice: “guarda c'è 'sta novità nel progetto, c'è da fare una modifica sul sito WEB”.

I: L'azienda, quella pugliese?

A: Magari sto facendo tre robe alla volta, faccio una cosa, sospendo, ne faccio un'altra, insomma non è che ho degli orari che uno può darmi: “ok oggi fai un'ora di progetto!”

I: Tu hai un contratto con questa azienda?

A: Sì, sostanzialmente sì.

I: Cioè sostanzialmente significa?

A: In pratica, beh non è sono dipendente loro, è un contratto che io, Luca e l'azienda, tutti i guadagni che vengono fatti da noi e da loro vengono divisi al 50% tra loro e noi. Se loro fanno 1000 euro in un anno, cosa impossibile in questo momento, io prendo 250 euro, Luca 250 e l'azienda 500. La cosa interessante è che il contratto prevede di dividere i guadagni ma noi non abbiamo spese.

I: Mi hai detto che avete il patrocinio del Comune, giusto?

A: Patrocinio significa avere un timbro se devi fare dei volantini, non è economico ma è simbolico.

I: A chi è che l'avete chiesto il patrocinio?

A: Non ho idea, se ne è occupata l'azienda. Di queste cose se ne occupano loro, noi sistemiamo il sito.

I: Questa gente lavora con open source o lavora anche software proprietario?

A: Prevalentemente open source, ma ha anche soluzioni Windows credo. Però loro principalmente puntano sulle soluzioni basate su Windows.

I: La diatriba che mi sembrava era scoppiata ancora qualche anno fa tra puristi Debian e altre distribuzioni?

A: Ti riferisci al fatto di quando è venuto Giuliano dicendo che Ubuntu è un software proprietario. Ero appena entrato in mailing list. Dopo se la sono presi

tutti con me perché ho detto: “pian coe paroe!” [Espressione dialettale che significa: “modera i termini!”] che era come dire: “dire che un Ubuntu è un software proprietario ce ne passa un po'”; solo perché c'è il pacchettino che ti serve per connetterti con wi-fi. Ok c'è un pezzo non libero, ma non significa che la distribuzione è software proprietario e che invita gli utenti a usare software proprietario mi sembra un po' esagerato.

I: Una delle cose che sostiene Zacchiroli è che rispetto ad Ubuntu, al di là delle questioni che sia proprietario o no, il disappunto sta nel fatto che Ubuntu non è riconoscente rispetto a Debian per il fatto che l'80% di quello che Ubuntu utilizza è Debian.

A: Non ha senso perché alla fine sono distribuzioni di software libero e quindi la concorrenza è fino ad un certo punto. Non è che Debian debba guadagnare soldi di più di Ubuntu.

I: No, non è una concorrenza economica, ma una concorrenza di spazio, di visibilità.

A: Comunque le patch1 e i lavori che vengono fatti da Ubuntu, da Canonical e altri, per quanto ne so io, perché non sono esperto di questi sistemi e potrei anche sbagliarmi, vengono riproposte ai pacchetti Debian. Quindi se io sono un programmatore e modifico un pacchetto per Ubuntu, poi per lo meno lo vado a segnalare sul sito di Debian o metto a disposizione il codice. Se tu guardi sul sito dei pacchetti di Ubuntu e cerchi un pacchetto, trovi il pacchetto modificato da Ubuntu tu unitamente a quello puoi scaricare il SID che è il file che indica tutte le modifiche che sono state fatte al pacchetto Debian da Ubuntu. Per cui se Debian vede che il suo pacchetto funziona peggio di quello di Ubuntu comunque può raccogliere quello che è stato fatto e applicare le stesse modifiche. Se scarichi un pacchetto Ubuntu che deriva da un pacchetto Debian è possibile vedere tutta la storia di quel pacchetto. Poi Ubuntu mantiene anche il “one paper cuts” che sono le cento cose importanti per gli utenti per migliorare l'interfaccia grafica. Queste poi vengono prese in considerazione per modificare GNOME e queste modifiche di GNOME vanno anche a vantaggio di Debian. Vabbe' uno può anche usare KDE. Il driver di connessione di connessione su Ubuntu è quello di Feadora perché l'ha sviluppato Red-Hat, se fanno modifiche a quello gli manderanno la patch e loro vedranno se accettarle. Se ognuno si lamentasse delle modifiche .... se guardi una delle quattro libertà del software libero è quella di fare delle modifiche.

I: Non è sul fatto che si possano fare delle modifiche …

A: Comunque gli autori sono sempre segnati, non è che le modifiche che fa uno sono nascoste e uno deve andare a leggere tutto il codice dall'inizio alla fine per capire dove sono, sono li, sono segnate.

I: Io penso che per riconoscimento loro intendano il fatto che nelle loro distribuzioni loro indichino esplicitamente anche per gli utilizzatori finali la parte di tecnologia Debian, che ne so, magari aggiungendo qualche nota su Debian nelle distribuzioni Ubuntu. Il discorso invece di distinguere tra open source e software libero: risponde anche questo ad un discorso tra puristi e buon senso oppure per te è qualcosa su un altro piano.

A: È come il discorso: “se lo dovremmo chiamare Linux o GNU/Linux”; ognuno ha i suoi motivi per usare un nome o l'altro. Ma è tutta energia sprecata, perché è come il tempo che spendiamo a scrivere sulla lista, perché uno dice che Googlemaps è un sito e l'altro invece dice che è un software. Se tu guardi il sito di Richard Stallman vedi che è sovversivo, lui è contro ogni cosa: è contro il libro di Henry Potter, è contro l'aeroporto di Parigi, è contro la coca-cola …

I: contro l'aeroporto di Parigi?

A: Non chiedermi il motivo, comunque so che è così, l'ho letto anche oggi. È contro i libri di Henry Potter perché una volta in Inghilterra è successo un casino per la data di rilascio del libro, i canadesi li hanno comprati qualche giorno prima, allora gli editori hanno scritto un comunicato che diceva che non si potevano leggere i libri e né prestarli, allora Richard Stallman ha detto: “non compriamo più i libri di Henry Potter”. Se vuoi si può ridire su qualsiasi cosa, tu dici una cosa e io ci trovo da ridere. Ma non è sempre così che si costruisce qualcosa. Hai avuto assolutamente ragione a ridire sul software proprietario ma fino ad un certo punto.

I: Anche Zacchiroli comunque ci teneva molto a questa distinzione tra software libero e open source.

A: se tu guardi Debian non è considerata libera dalla Free Software Foundation. C'è GNU sense, ce ne sone altre, cinque mi sembra, ma no Debian perché favorisce software proprietario non so che cavolo [in realtà FSF ha ammonito Debian e l'ha scoraggiata ad usare una libreria scritta in C# un linguaggio che non da garanzie di rimanere libero, la FSF non riporta le distribuzioni GNU/LINUX libere e non, non ha senso visto che sono GNU/Linux

e quindi ereditano le prerogative GNU]. Troverai GNU/Sense che è un Ubuntu modificato ma non ha più nulla di proprietario. Io sono il primo a dire che Debian rappresenta benissimo il software libero ma per loro non è ancora abbastanza, a volte c'è troppa esagerazione.

I: Tu hai mai letto libri di Richard Stallman o Raymand?

A: Libri no, ho letto dei testi. Di Eric Steven Raymond ho letto “il modello open source”, “la cattadrale e il bazar”. All'inizio non capivo molto di Linux e queste cose qui, quindi mi sono detto: “leggo qualcosina”.

I: su un libro di Eric Steven Raymond, su “how to become an hacker” c'è una frase che dice “Chiunque possa darti degli ordini, può fermati dal risolvere problemi dai quali sei affascinato - e, visto il modo con cui le menti autoritarie funzionano, tali ordini generalmente saranno motivati da ragioni orribilmente stupide”. Io ho cercato di capire queste parole, ma non sono riuscito a capire cosa intendesse per “chiunque può fermati dal risolvere problemi”, non riesco a capire quali sono le situazioni relativamente all'open source dove si trovano degli impedimenti. Cioè non capisco se ci sia qualche costrizione sociale, qualche impedimento?

A: L'episodio per cui è nato tutto su Richard Stallman del software libero, è la stessa cosa, lui doveva risolvere un problema con il driver, loro hanno detto: “no, perché non ti diamo il codice”; questa è un motivazione stupida, e lui sapeva benissimo come risolvere il bug. Lui ha detto: “poi vi mando le correzioni e le potete dare anche agli altri”, e loro: “no ma … non ti diamo il codice”. Loro gli hanno impedito di risolvere 'sto problema. Perché dovevano tenersi il codice del driver della stampante. In senso lato anche il fatto che ti tocca comprarti un computer con la licenza Windows ti impedisce di risolvere un problema. Quando il computer ti va lento come un ippopotamo, però il problema è che te lo compri ogni volta che ti compri un computer. Allora cos'è quello? Un deterrente per non usare altre cose. Però io voglio risolvermi il problemi e voglio usare Linux. Posso anche togliermi Windows, però intanto lo paghi. Nonostante non ci sia nessuna nazione al mondo che riconosca che Bill Gates ha una monarchia, lui comunque riesce a fare il cavolo che vuole. Purtroppo fermarlo è difficile. Ho letto una cosa veramente ridicola, perché avevo chiesto il rimborso per Windows Vista che avevo trovato installato sul portatile. Poi c'era uno donna, una ragazza, non so, che mi ha chiesto sul blog dei consigli per comprare un computer anche lei e mi ha detto :“quello che voglio comprare io ha Windows 7 e non Windows vista”. E

io gli ho detto: “penso che sia uguale”; invece mi ha mostrato il PDF scritto dalla Microsoft con le licenze di Windows 7, il paragrafo in cui si parlava di contattare il rivenditore per avere informazioni sul rimborso è diventato: [...] contattare il rivenditore per informazioni che potrebbero riguardare il rimborso dell'intero pacchetto hardware e software o potrebbero ledere i tuoi diritti … ”; io non capisco perché il rivenditore potrebbe decidere di ledere i miei diritti, io non sono un avvocato, ma penso che sia una emerita cavalata, e voglio vedere se ha un valore quella clausola lì perché tu non puoi dire: “ ah potrebbe ledere i tuoi diritti”; che senso ha? È una clausola vessatoria?

I: Ci sono vari tipi di tifosi dell'open source. Ci sono i tifosi software libero e open source: quelli che in qualche maniera dicono: “non è giusto che ci sia solo Microsoft, Microsoft costa troppo”; per dire tifo per l'open source, tanto quanto tifo per l'iMac o qualsiasi cosa che non è Microsoft. Il problema è che …

A: Il problema è che è una presa per il sedere epocale, è quello il problema.

I: Però la questione è abbastanza diversa: un conto è dire: “non è giusto che ci siano delle licenze non libere”; un conto è dire “sono contro il monopolio”.

A: Infatti sostenere l'open source non vuol dire andare contro microsoft, è quello che spesso non si capisce. Se tu ad esempio sei a favore dell'open source o del software libero vuol dire che ti sta antipatica la Microsoft. Ma se un domani Microsoft sviluppa software libero, perché dovrebbe starmi antipatica? Dipende da come si comportano. La Apple mi sta antipatica forse più della Microsoft, perché decide di fare iMac e quindi va su bsd.org, scarica tutto il sorgente, lo modifica un po', fa l'interfaccia grafica sopra e toh, fatto il sistema operativo e comunque stai facendo sofwtare proprietario. Non ha nessun senso dirmi: “tu sei conro la Microsoft”; no! Sono contrario all'idea del software proprietario. Se Microsoft cambia atteggiamento non mi sta antipatica, dipende sempre da come agisce uno. Anche Novell che per certi versi sta facendo software libero non mi sta per niente simpatica, perché fa software libero fino ad un certo punto e ogni volta che c'è da farsi dare soldi dalla Microsoft, se li fa dare. È difficile distinguere. Quello che ha creato GNOME, Miguel de Icaza, è sfegatato fan della Microsoft, infatti lavora per Novell e si occupa della compatibilità Linux/Windows e di creare Mono che è una è una macchina virtuale che rende autonomo il compilatore C# dalla piattaforma. La Novell vende il culo alla Microsoft … perché Microsoft ha detto: “Linux viola 27 brevetti nostri”; si ma quali? “27, guarda che possiamo farti causa”. Così Novell si fa corrompere da

Microsoft e così SUSE. Quando l'ho saputo mi sono girate le scatole anche se non uso SUSE, perché dice di essere un'azienda che sostiene l'open source ma poi invece …

Quindi se uno dice di essere contro Microsoft ma non contro Apple alloro non è un sostenitore del software libero. Richard Stallman crede fermente in quello che ha fatto e si adopera per farlo conosce e dice: “o software libero o niente!”; dall'oggi al domani così … : “ o software libero o niente!”. C'è invece chi come me è un po' più morbido nel senso dire vabbe', dall'oggi al domani non si può far diventare software libero tutto quello che è utilizzato, non è fattibile perché domani io mi sveglio e trovo ancora la Microsoft, trovo ancora la Apple, Allora, al posto di dire: “tutto software libero o niente”; ti dico: “bene: ti faccio trovare un sistema operativo libero, ti mostro tanti software liberi, se dopo hai bisogno di qualcosa come il driver della scheda wi-fi, per stavolta installati quello proprietario, la prossima volta stai attento e comprati il PC compatibile con i drivers open”; Perché è capitato in lista: una ragazza che conosco, con cui ho chiacchierato diverse volte …

I: Lidia?

A: No Elisa. Che ha 25 anni si è laureata in fisica. Chiacchieravano di Linux e cose del genere, mi chiedeva se avevo problemi con Ubuntu o cose del genere. Vabbe', lei praticamente è andata via da LUG perché aveva problemi …

Cameriere: Posso portare via?

A: Si si!

Cameriere: Volete un caffè? Un dolce?

I: Un caffè!

Cameriere: Liscio?

I: Liscio!

A: Per me un decaffeinato, grazie! Una volta aveva problemi con la scheda video, aveva bisogno del driver proprietario, è gli hanno detto: “t'attacchi, perché la prossima ci pensi prima di comprare un computer che non va con il driver libero”. Lei ha detto: “io Ubuntu lo conosco da due settimane, cosa faccio? Lo

butto via e ne compro un altro?” E lui ha detto: “no devi usare software libero?”. Se io ho il monitor cosa faccio? Lo uso alla cieca? È quello il fatto. Non si capisce che essendo così categorici non aiuti il software libero, allontani la gente. Prima che la gente conosca il software libero falla avvicinare, non mandarla via, sennò che cavolo, non ha alcun senso. Se io voglio convincere una persona a usare Gimp e gli dico :”no, tu devi assolutamente disinstallarti Fotoshop, passare a Linux [...] Dico: “stai calmo, perché devi venire ad aggredirmi così?” Ci vuole un minimo di passaggio graduale: “Se tu vuoi passare a Linux ok, però non potrai più navigare perché la tua scheda wi-fi non funziona!”. E lui mi dice: “bon, ciao!” È ovvio! Se io invece gli dico: “aspetta, ti metto il software libero, le applicazioni libere, la suite di office libero, il programma di grafica, il slash video libero e il driver della sceda wi-fi proprietario, ma la prossima volta valuta bene quello che compri, però intanto ti funziona il computer, lo puoi provare, puoi renderti conto che magari è un tema che ti interessa”; e quindi quando ti compri un altro computer dici. “Ok voglio provarlo con Linux”. Ci sono ragazzi più o meno della mia età a cui ho fatto conoscere Linux e Ubuntu, più altre cose. Mi è capitato spesso di trovare dei pezzi hardware che non andavano se non con driver proprietari, ma cosa faccio? Dico: “ok stai pure con Windows?” allora io a quello persona non gli ho insegnato niente? non gli fatto provare il software libero?” Se invece gli do la possibilità di provare e passa dal 100% di uso proprietario all'1% …

Cameriera: Caffe! Il deka!

A: Se sono ragionevole lo faccio passare da 100% ad 1% software proprietario, se voglio imputarmi lui rimarrà col 100% software proprietario e non inizierà mai ad usare software libero, capisci? E allora devi ragionare su quello che ha senso fare. Secondo me ha senso che le persone comincino ad avvicinarsi al software libero, no che si trovino un muro che alla prima cosa che usi non libera devi tornare a Windows … Poi le stesse persone che fanno così, poi sono quelle che usano Windows, che all'università usano Windows, e allora come fai? Capisci anche tu che fino ad un certo puoi arrivare.

I: Io al lavoro uso Windows, è il sistema che è in dotazione, anzi io programmo anche in Windows. Se c'è bisogni di un applicativo desk-top con utenti Windows, cosa fai? Io ho due macchine sulla mia scrivania: una macchina Windows e una macchina Linux. Ti pagano per far quel lavoro lì, la pagnotta devi portarla a casa per cui … Però, appunto come dici tu … però … io quest'anno ho piallato due server Windows. Ora dove lavoro ci sono due server Linux, forse l'anno prossimo ce ne sarà uno in più. Però se io dicessi: “no, pialliamo tutto!”.

A: ti prendono a schiaffi.

I: Più che altro mi licenziano.

A: Appunto! E ti ridono anche in faccia penso?

Commiato.

- Arnaldo Bagnasco (1999), Tracce di Comunità, Bologna, Il Mulino
- Arnaldo Bagnasco, Marzio Barbagli, Alessandro Cavalli (2007), Corso di sociologia, Bologna, Il Mulino
- Ulrich Beck (1999), Che cos'è la globalizzazione, Roma, Carocci
- Ulrich Beck (2001), La società globale del rischio, Trieste, Asetrios
- Ulrich Beck (2001), Libertà o capitalismo, Roma, Carocci
- Chester Barnard (1970), Le funzioni del dirigente, Novara, Utet – De Agostini
- Mariella Berra (2001), Meo Angeo R., Informatica solidale, Torino, Bollati Boringhieri
- Pierre Bourdieu (2001), La distinzione. Critica sociale del gusto, Bologna, Il Mulino
- Angelo Chianese (2008), Vincenzo Moscato, Antonio Picariello, Alla scoperta dei fondamenti dell'informatica, Napoli, Liguori
- Gabriella Coleman (2010) Hacking In-Person: The Ritual Character of Conferences and the Distillation of a Life-World. Anthropological Quarterly, Winter, New York University
- James Samuel Coleman (2005), Fondamenti di teoria sociale, Bologna, Il Mulino
- Chris DiBona (2000), Voices from the open source revolution, Sebastopol - CA, O'Reilly
- Mary Douglas (2003), Purezza e pericolo. Un'analisi dei concetti di contaminazione e tabù, Bologna, il Mulino
- Michel Crozier (1978), Il fenomeno burocratico, Milano, Etas Kompass
- Mihaly Csikszentmihalyi (2005), Fluir (Flow): Una Psicologia de La Felicidad, Barcelona (España), Editorial Kairos
- Antonio Damasio (1995), L'errore di Cartesio. Emozione, ragione e cervello umano, Milano, Adelphi Edizioni
- Umberto Dante (2002), L'utopia del vero nelle arti visive, Roma, Meltemi Editore
- Nicola Alberto De Carlo (2004), Teorie & strumenti per lo psicologo del lavoro e delle organizzazioni, Milano, Franco Angeli
- Emile Durkheim (1962), La divisione del lavoro sociale, Milano, Comunità
- Emile Durkheim (1963), Le forme elementari di vita religiosa, Milano, Comunità
- John Kenneth Galbraith (1968), Il nuovo stato industriale, Torino, Einaudi
- Antony Giddens (1994), Le conseguenze della modernità, Bologna, Il Mulino
- Antony Giddens (1995), La trasformazione dell'intimità, Bologna, Il Mulino
- Gian Antonio Gilli (1995), Origini dell'eguaglianza, Torino, Einaudi

- Rober L. Glass, Of Open Source (1999), Linux ... and Hype, Los Alamitos – California, IEEE Computer Society Press
- Erving Goffman (1988), Il rituale dell'interazione, Bologna, Il Mulino
- Pekka Himanen (2001), L'etica hacker e lo spirito dell'età dell'informazione, MIlano, Fltrinelli
- Floyd Hunter (1953), Community Power Structure, Chappel Hill, University of North Corolina Press;
- Thomas Kuhn (1979), La struttura delle rivoluzioni scientifiche, Torino, Einaudi Editore
- Salvatore La mendola (2007), Comunicare interagendo, Novara, Utet – De Agostini
- Gerhard Lensky (1981), La cristallizzazione di status, Napoli, Liguori
- Claude Lévi-Strauss (1991), Il totemismo oggi, Milano, Feltrinelli
- Niklas Luhmann (1990), Sistemi Sociali, Bologna, Il Mulino
- Niklas Luhmann (2007), Conoscenza come costruzione, Roma, Armando Editore
- Niklas Luhmann (1988), Teoria dei sistemi, Bologna, Il Mulino
- Niklas Luhmann (1991), Teoria della società, Milano, Franco Angeli
- Robert & Helen Lynd (1974), Middletown. A study in contemporary America Culture, Milano, Comunità
- Roberta Maeran (2002), Psicologia e mondo del Lavoro, Milano, LED
- N. Gregory Mankiw (2007), L'essenziale di economia, Bologna, Zanichelli
- Abraham Maslow (1973), Motivazione e personalità, Roma, Armando Editore
- Marcel Mauss (2002), Saggio sul dono. Forma e motivo dello scambio nelle società arcaiche, Torino, Einaudi
- Davide McClelland (1988), The achiving society, Princeton - NJ, Van Nostrand
- Wright Mills (1956), The Power Élite, Oxford, Oxford University Press
- Matthias Minich (2009), B. Harriehausen-Muehlbauer, and C. Wentzel - Software Industrialization in Systems Integration, World Academy of Science, Engineering and Technology, www.waset.org
- Glyn Moody (2002), Codice Ribelle, Milano, Hops Libri
- Robert A. Nisbet (1987), La tradizione sociologica, Firenza, Nuova Italia
- Tim O'Reilly (2000), Voices from the open source revolution, Sebastopol - CA, O'Reilly
- Luciano Paccagnella (2010), Open Access, Bologna, Il Mulino
- Luciano Pellicani (1988), La genesi del capitalismo, Lungro di Cosenza, Marco Editore
- Robert M. Pirsig (1992), Lo Zen e l'arte della manutenzione della motocicletta, Milano, Adelphi Edizioni

- Karl Polanyi (1977), La sussistenza dell'uomo. Il ruolo dell'economia nelle società antiche, Torino, Einaudi
- Roland Robertson (1995), Globalization: Time-Space and omogenety-heterogenety, London, Sage
- Roland Robertson (1999), Globalizzazione, teoria sociale e cultura globale, Trieste, Asterios
- Richard Stallman (2000), Voices from the open source revolution, Sebastopol - CA, O'Reilly
- Eric Steven Raymond (2001), Come diventare hacker, www.scribd.com/
- Eric Steven Raymond (1997), The Cathedral and the Bazaar, Sebastopol - CA, O'Reilly
- Eric Steven Raymond (2000), Voices from the open source revolution, Sebastopol - CA, O'Reilly
- Henry Sumner Maine (1988), Diritto antico, Milano, Giuffrè Editore
- Frederick W. Taylor (1974), La divisione scientifica del lavoro, Milano, Franco Angeli Editore
- Michael Tiemann (2000), Voices from the open source revolution, Sebastopol - CA, O'Reilly
- Ferdinand Tönnies (2009), Comunità e società, Calimera - LE, Kurumuny
- Victor Turner (1972), Il processo rituale. Struttura e anti-struttura, Brescia, Morcelliana
- Paul Vixie (2000), Voices from the open source revolution, Sebastopol - CA, O'Reilly
- François-Marie Arouet (Voltaire) (2010), Candido ovvero l'ottimismo, Collana Bacheca Ebook Stampa digitale
- Max Weber (1997), L'etica protestante e lo spirito del capitalismo, Milano, RCS Libri
- Robert Young (2000), Voices from the open source revolution, Sebastopol - CA, O'Reilly

Informazioni su questa edizione elettronica:

Questo ebook proviene da Wikisource in lingua italiana[1]. Wikisource è una biblioteca digitale libera, multilingue, interamente gestita da volontari, ed ha l'obiettivo di mettere a disposizione di tutti il maggior numero possibile di libri e testi. Accogliamo romanzi, poesie, riviste, lettere, saggi.

Il nostro scopo è offrire al lettore *gratuitamente* testi liberi da diritti d'autore. Potete fare quel che volete con i nostri ebook: copiarli, distribuirli, persino modificarli o venderli, a patto che rispettiate le clausole della licenza Creative Commons Attribuzione - Condividi allo stesso modo 3.0 Unported[2].

Ma la cosa veramente speciale di Wikisource è che **anche tu** puoi partecipare.
Wikisource è costruita amorevolmente curata da lettori come te. Non esitare a unirti a noi.

Nonostante l'attenzione dei volontari, un errore può essere sfuggito durante la trascrizione o rilettura del testo. Puoi segnalarci un errore a questo indirizzo: http://it.wikisource.org/wiki/Segnala_errori

I seguenti contributori hanno permesso la realizzazione di questo libro:

- Candalua
- Eumolpo
- Stefano De Boni
- Accurimbono
- Barbaforcuta
- Napy65
- Hazard-SJ
- Zhuyifei1999
- Alphax
- Siebrand
- Maat
- Skalman
- Ftiercel
- Marc

Il modo migliore di ringraziarli è diventare uno di noi :-)

A presto.

---

1. ↑ http://it.wikisource.org
2. ↑ http://www.creativecommons.org/licenses/by-sa/3.0/deed.it