www.copyleft-italia.it

p r e s e n t a

AUTORE: Marco Zoppelletto

TITOLO: Il modello open source vs il modello proprietario nel mercato del software

DATA DI RILASCIO AL PUBBLICO: gennaio 2005

TIPO DI OPERA: Tesi di laurea - Facoltà di economia

LICENZA UTILIZZATA: GNU Free documentation license

UNIVERSITA' DEGLI STUDI DI VERONA
Facoltà di Economia
Corso di laurea in Economia del Commercio Internazionale

Tesi di laurea
IL MODELLO OPEN SOURCE vs IL MODELLO PROPRIETARIO NEL MERCATO DEL SOFTWARE

Relatore:
Chiar.mo Prof. Angelo Zago

Laureando:
MARCO ZOPPELLETTO

Anno Accademico 2003 – 2004

INDICE

RINGRAZIAMENTI: prima di passare al testo della tesi vorrei ricordare tutti coloro che più o meno indirettamente hanno contribuito e mi sono stati d'aiuto, anche solo attraverso consigli o idee, nella realizzazione dello stesso. Oltre al Prof. Angelo Zago per l'assidua disponibilità dimostratami, vorrei ringraziare soprattutto Enri Sperotto per la correzione delle spiegazioni tecniche, i consigli e le segnalazioni durante tutto il periodo di stesura nonché Simone Aliprandi, autore del saggio "L'altra faccia del copyright" e responsabile del sito www.copyleft-italia.it , per i consigli e le informazioni sull'ambito giuridico. Fra coloro che hanno contribuito attraverso segnalazioni ricordo con gratitudine David Welton di Dedasys, Davide Dozza, mantainer del Progetto Linguistico Italiano OpenOffice.org (PLIO) , Massimiliano Gambardella e Claudio Erba di SpaghettiLearning.

Per eventuali comunicazioni l'autore può essere contattato tramite e-mail all'indirizzo zop80@yahoo.it .

INTRODUZIONE

"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 (Stallman 1999)."

Lo scopo di questo elaborato è di analizzare il modello economico su cui si basano il *Free Software* e l'*Open Source* (dall'inglese *sorgente aperto*). I termini *Open Source* e *Free Software* fanno riferimento ad un modello rivoluzionario di creazione del software; rivoluzionario poiché si contrappone al modello tradizionale che viene in mente quando si pensa all'industria del software, in genere definito *chiuso* o *proprietario* (si pensi al software prodotto da Microsoft). Il software Open Source (che deriva dal Free Software) ha la particolarità di essere creato e sviluppato da una comunità composta sia da programmatori informatici che spesso vengono definiti *hacker*[1] sia da semplici utenti e di essere rilasciato pubblicamente assieme al proprio *codice sorgente*. Per comprendere la sostanziale differenza fra i due modelli, relativa proprio alla disponibilità o meno del codice sorgente del software e ad una serie di libertà fondamentali sulla gestione degli stessi, occorre spiegare come avviene il processo di creazione del software e il concetto di codice sorgente.

Qualsiasi tipo di software (un sistema operativo[2] come Windows, un editor di testi come Word, un browser come Explorer per navigare in Internet) altro non è che un insieme di istruzioni e di dati affidati ad un calcolatore elettronico, la cui interpretazione ed esecuzione genera un risultato (si va dalla possibilità di compiere calcoli matematici fino alle più banali operazioni che solitamente si compiono con un computer come ad esempio scrivere un' e-mail o leggere un file). Il calcolatore legge dalla memoria dei comandi (sotto forma di liste di numeri) che indicano alla macchina cosa è richiesto e quali sono i dati su cui operare; la macchina li interpreta ed invia alla memoria il risultato delle operazioni effettuate. Questi comandi vengono impostati dal programmatore attraverso uno speciale linguaggio che consente di indicare in modo intelligibile quali operazioni compiere, raggruppando in una sintassi comprensibile all'uomo le sequenze di numeri che sono richieste dalla macchina. Ad esempio il calcolatore interpreta i dati esadecimali b8b8790f 8ed8 b409 ricavando quale operazione compiere ma il programmatore scrive un comando del tipo INC numero_clienti per intendere che il numero di clienti già acquisiti va incrementato di una unità.

Le sequenze di numeri sono raggruppate nel *programma eseguibile* vero e proprio (ad esempio il file calc.exe è la calcolatrice disponibile in Microsoft Windows e si presenta come una lista di 59.392 byte[3]) mentre i comandi scritti dal programmatore compongono il cosiddetto *codice sorgente* (in inglese *source code*). Si può pensare al codice sorgente come alle note che il musicista scrive sullo spartito mentre il codice eseguibile può essere pensato come la melodia, la musica stessa che si crea suonando le note con lo strumento. Alterando il codice sorgente è possibile far compiere alla macchina operazioni nuove, ad esempio si può aggiungere alla calcolatrice di Windows un tasto per la conversione lira/euro. Lo stesso risultato, in teoria, sarebbe ottenibile modificando direttamente il programma eseguibile ma in pratica operare su una lista di migliaia di numeri è impossibile[4].

La conversione del codice sorgente in programma eseguibile, quindi di un comando umano in un linguaggio comprensibile alla macchina, avviene attraverso un software chiamato compilatore che si differenzia a seconda del linguaggio di programmazione utilizzato. E' importante sottolineare come questo processo sia di fatto irreversibile in quanto il codice eseguibile non viene solo tradotto ma viene anche ottimizzato per migliorarne il funzionamento di

---

1 Un hacker è una persona che si impegna nell'affrontare sfide intellettuali per aggirare o superare creativamente le limitazioni che le vengono imposte, in primo luogo nei suoi ambienti di interesse, che solitamente comprendono l'informatica o l'ingegneria elettronica. Il termine hacker viene usato in genere in modo improprio per indicare un criminale informatico. Nel glossario alla fine dell'elaborato verrà approfondita la spiegazione del reale significato

2 In informatica, un sistema operativo (SO) è il programma responsabile del diretto controllo e gestione dell'hardware che costituisce un computer e delle operazioni di base. Si occupa anche di controllare gli accessi degli utenti e dei processi che vengono eseguiti.

3 Unità di misura informatica relativa alla dimensione di un certo file.

4 Inguaggiato (1999)

conseguenza, pur esistendo degli appositi programmi di decompilazione che effettuano il *reverse engineering*[5] cioè il processo opposto, è assolutamente impossibile giungere al codice sorgente originario ma solo ad uno che ricompilato porta ad un codice sorgente funzionalmente uguale a quello iniziale. Nel caso del software Free ed Open Source, oltre al rilascio del codice eseguibile (ossia del software, così come siamo abituati a conoscerlo) si ha il rilascio del codice sorgente creato dai programmatori, nonché la garanzia di alcune libertà "fondamentali" riguardanti la possibilità di modificare il codice sorgente e tutelate da innovative forme di licenza. In sostanza il rilascio del codice sorgente assieme al software, venduto o regalato, può essere visto come la cessione di un prodotto e del relativo progetto.

Al contrario, nel caso del software proprietario il codice sorgente non è disponibile liberamente e solo raramente viene concesso a pochissimi partner della software-house[6]: in tal senso è esemplare la decisione di Microsoft di rilasciare brevi porzioni di codice dei propri sistemi operativi (pur con notevoli limitazioni) ad alcune Pubbliche Amministrazioni (è il caso anche dell'Italia) e ad altre società creatrici di software. L'avvento dell'Open Source che, si badi bene, non prevede in assoluto la gratuità del software, sta ovviamente mettendo in discussione tutto il modello economico fin qui adottato dall'industria del software di conseguenza si cercherà di analizzare innanzitutto il modello finora usato e la sua storia, per poi cercare di determinare quali variazioni esso stia portando e dovrebbe portare in futuro. Si cercherà poi di capire se il software libero possa essere una valida alternativa a quello proprietario considerando aspetti di convenienza tecnica ed economica.

[5] Per una breve spiegazione si rimanda al paragrafo 2.8.

[6] Società operanti nell'ambito informatico nella produzione di software.

CAPITOLO 1
STORIA DELL'OPEN SOURCE

1.1 La cultura hacker

Contrariamente a quanto molti oggi potrebbero pensare, il concetto di Software proprietario è abbastanza recente; in principio infatti tutto il software era libero. Le origini di quello che solo da pochi anni viene chiamato Open Source (e prima Free Software) risalgono agli albori dell'informatica o meglio ai tempi nei quali per la prima volta fu possibile, ed aveva una sua utilità, condividere il codice sorgente tra persone senza che esistesse alcun legame contrattuale tra di loro. Sin dalla metà degli anni Quaranta quando entrarono in funzione i primi computer (i cervelli elettronici come erano chiamati a quel tempo) e fino agli anni Settanta, era possibile riutilizzare lo stesso codice anche se in un modo oggi ritenuto piuttosto "artigianale", attraverso nastri e schede perforate[7].

Negli anni Quaranta e Cinquanta ciò non succedeva spesso in quanto esistevano pochi esemplari di computer uguali e le poche organizzazioni che li utilizzavano avevano problemi non standard. Inoltre le conoscenze di programmazione erano più simili alle conoscenze scientifiche (e pertanto liberamente condivisibili) che a conoscenze tecnologiche delle quali appropriarsi per trarre profitto. La pratica di condividere il codice sorgente diventò evidente soprattutto quando si affermò il vantaggio di usare una stessa porzione di codice, il che presupponeva di avere macchine uguali e problemi simili. Inoltre fino alla fine degli anni Settanta, anche se in misura decrescente, fu l'hardware a costituire la quasi totalità del costo di una struttura informatica pur restando inutile senza il software.

Da ciò la decisione dei principali produttori di hardware, fra cui IBM, di vendere il loro prodotto accompagnato da più software possibile e di facilitarne la diffusione, fenomeno che rendeva le loro macchine più utili e dunque più concorrenziali. Il software tra l'altro non poteva avvantaggiare la concorrenza in quanto funzionava solo su un preciso tipo di computer e non su altri, a volte addirittura neanche fra macchine dello stesso produttore. IBM possedeva i diritti sul software sviluppato dai propri dipendenti tuttavia lo rilasciava assieme al relativo codice sorgente, in modo tale da incoraggiare i clienti e gli utilizzatori (che in quegli anni erano soprattutto università ed enti militari) a modificarlo e a migliorarlo, ciascuno in base alle proprie esigenze, con l'obiettivo di condividere i miglioramenti poi con gli altri clienti/sviluppatori.

Non esistendo in quegli anni un'infrastruttura di comunicazione come l'attuale rete Internet ed essendo grandi le distanze da coprire, il sistema di distribuzione del software tendeva a centralizzarne lo sviluppo quindi le modifiche che i clienti apportavano ai programmi erano condivise innanzitutto con IBM. Qui le nuove versioni erano esaminate e i cambiamenti migliori erano inseriti nelle versioni distribuite agli altri clienti: il miglior software disponibile al mondo era quindi distribuito gratuitamente e completo dei codici sorgente. Pur esistendo in teoria un *copyright*[8] sul software che ne impediva la modifica e la successiva distribuzione, in pratica esso era irrilevante quando non anche indesiderabile visto che la strada migliore per lo sviluppo era proprio la cooperazione aperta con gli utenti. Per un produttore di hardware questo sistema era il modo migliore per garantirsi, grazie alla maggiore qualità del software, la possibilità di diffondere l'uso del computer nelle aziende e quindi espandere la propria quota di mercato.

Come si è già detto, il numero di utenti che avevano accesso ai computer era limitato anche perché erano i pochi che dovendoci lavorare avevano le conoscenze in termini di programmazione del software. Questi "pionieri", in genere ricercatori di alcune università americane quali il MIT[9] di Boston e l'università di Berkeley in California, ebbero in tal senso l'indubbio merito di essere i primi a concepire le possibili applicazioni informatiche di concetti quali *interattività*[10] e *rete*[11]. Fu con loro che si crearono le basi della *cultura hacker*, movimento al cui interno si è

---

7 Gli unici supporti di memoria di quel tempo, antenati degli attuali dischetti, cd-rom e dvd

8 Forma di diritto d'autore in uso nel mondo anglosassone, equivalente al diritto d'autore in Italia.

9 Massachussets Institute of Technology.

10 In via generale, una comunicazione interattiva è una comunicazione in cui il destinatario della comunicazione può mandare dei messaggi a chi ha dato origine alla stessa. Nell'ambito informatico spesso il termine si riferisce all'interazione di utente ed interfaccia di computer.

11 Il concetto di rete è abbastanza vasto e abbraccia vari campi fra loro molto diversi come l'economia e l'informatica appunto. La rete è definibile come una serie di sistemi di componenti interconnesse. Si rimanda al glossario alla fine dell'elaborato

sviluppata appunto la filosofia del software libero. Non è un caso che il primo luogo in cui si iniziò a parlare di hacker fu appunto il MIT di Boston: l'origine della cultura hacker, come oggi la conosciamo, si può fare risalire al 1961, anno in cui il MIT acquistò il primo PDP-1[12]. Il comitato Signals and Power del "Club Tech Model Railroad" (TMRC) del MIT, adottò la macchina quale prediletto giocattolo tecnologico creando strumenti di programmazione, linguaggi e quell'intera cultura che ancora oggi ci appartiene in modo inequivocabile[13] (la cultura informatica del MIT sembra essere stata la prima ad adottare il termine hacker).

Gli hacker della TMRC divennero il nucleo dell'Artificial Intelligenge Laboratory (Laboratorio di Intelligenza Artificiale) del MIT, il principale centro di ricerca AI (Intelligenza Artificiale) su scala mondiale nei primi anni Ottanta e la loro influenza si protrasse ben oltre il 1969, il primo anno di ARPAnet[14]. ARPAnet è stata la prima rete transcontinentale di computer ad alta velocità. Ideata e realizzata dal Ministero della Difesa statunitense come esperimento nelle comunicazioni digitali, crebbe fino a diventare un collegamento tra centinaia di università, esponenti della difesa e laboratori di ricerca. Essa permise a tutti i ricercatori, ovunque essi si trovassero, di scambiarsi informazioni con velocità e flessibilità senza precedenti, dando un forte impulso allo sviluppo del lavoro di collaborazione e accelerando enormemente il ritmo e l'intensità del progresso tecnologico.

Ma ARPAnet fece anche qualcos'altro: le sue "autostrade elettroniche" misero in contatto gli hacker di tutti gli Stati Uniti e questi, fino a quel momento isolati in sparuti gruppi, ognuno con la propria effimera cultura, si riscoprirono (o si reinventarono) nelle vesti di vera a propria tribù di rete. Ben presto i ricercatori cominciarono a condividere un certo senso di appartenenza ad una comunità e ad una cultura comune, sentendo il bisogno di divulgare ciò che scoprivano e avendone in cambio altre informazioni importanti per il proprio lavoro. Il software prodotto era scambiato liberamente o in cambio di somme irrisorie a titolo di contributo per le spese e spesso chi lo riceveva operava innovazioni e cambiamenti che poi rendeva disponibili a sua volta.

Le prime intenzionali azioni di hackeraggio, i primi linguaggi caratteristici, le prime satire, i primi dibattiti autocoscienti sull'etica hacker, tutto questo si propagò su ARPAnet nei suoi primi anni di vita. La cultura hacker mosse i primi passi nelle università connesse alla Rete, in particolare modo (ma non esclusivamente) nei loro dipartimenti di scienza informatica. Dal punto di vista culturale l'AI Lab (Laboratorio di Intelligenza Artificiale) del MIT è da considerarsi il primo tra laboratori di pari natura a partire dai tardi anni Sessanta, anche se istituti come il Laboratorio di Intelligenza Artificiale dell'Università di Stanford (SAIL) e più tardi l'Università Carnegie-Mellon (CMU) divennero in seguito quasi altrettanto importanti.

Tutti costituivano fiorenti centri di scienza dell'informazione e ricerca sull'intelligenza artificiale, tutti attiravano individui brillanti che contribuirono al grande sviluppo del mondo degli hacker, sia dal punto di vista tecnico che folcloristico. Fin dai giorni del PDP-1 le sorti dell'hacking si intrecciarono alla serie di minicomputer PDP della Digital Equipment Corporation (DEC). La flessibilità, la potenza e la relativa economicità di queste macchine portarono molte università al loro acquisto. Ciò costituì l'habitat ideale per lo sviluppo della cultura hacker e anche ARPAnet fu costituita, per la maggior parte della sua durata, da una rete di macchine DEC. In occasione però dell'uscita del modello PDP-10, nel 1967, il MIT si rifiutò di utilizzare il software DEC del computer, scegliendo di creare un proprio software chiamato ITS: l'obiettivo infatti era quello di agire autonomamente.

1.2 La nascita di Unix

Nel frattempo, comunque, in New Jersey qualcos'altro era stato messo in cantiere fin dal 1969; qualcosa che avrebbe inevitabilmente adombrato la tradizione del PDP-10. L'anno di nascita di ARPAnet fu anche l'anno in cui un hacker dei Laboratori Bell di AT&T, Ken Thompson, scrisse il sistema operativo Unix. Thompson si trovò coinvolto nella fase di sviluppo di un sistema operativo chiamato Multics che divideva la propria discendenza con ITS. Multics costituì un importante banco di prova su come la complessità di un sistema operativo potesse essere celata fino a essere resa invisibile all'utente e perfino alla maggioranza dei programmatori. L'idea fu quella di rendere l'uso di

per una più ampia trattazione.

[12] Il PDP-1 era un computer fabbricato dalla Digital Equipment Corporation (DEC) a partire dagli inizi degli anni Sessanta.

[13] Levy (1984)

[14] ARPAnet sta per "Advanced Research Projects Agency Network". Fu l'antenata dell'odierna rete Internet. In particolare, in essa era presente il protocollo di comunicazione TCP/IP, tuttora colonna portante delle comunicazioni sul web.

Multics molto più semplice e programmabile, in modo da permettere di operare anche dall'esterno. Un altro hacker, di nome Dennis Ritchie, inventò un nuovo linguaggio, chiamato C[15], da usare con una versione Unix di Thompson ancora allo stato embrionale. Come Unix il linguaggio C fu progettato per essere piacevole e facile da usare oltre che flessibile.

L'interesse per questi strumenti non tardò a crescere nell'ambito dei Laboratori Bell ma Thompson e Ritchie avevano in mente qualcosa di ben più ambizioso. Per tradizione, i sistemi operativi erano stati fino ad allora scritti su misura per uno specifico hardware al fine di raggiungere la massima efficienza e questo, come si è già visto, determinava l'inesistenza di software standardardizzati fra i produttori di hardware. Thompson e Ritchie furono tra i primi a capire che la tecnologia dell'hardware e del linguaggio C aveva ormai raggiunto un livello di maturità tale da poter scrivere in C un intero sistema operativo: nel 1974 l'intero ambiente operativo era già regolarmente installato su numerose macchine di diversa tipologia. Si trattò di un evento senza precedenti e le implicazioni che ne derivarono furono enormi. Se davvero Unix poteva presentare la stessa interfaccia e le stesse funzionalità su macchine di diverso tipo, era sicuramente anche in grado di fungere da ambiente software comune per tutte. Gli utenti non avrebbero mai più dovuto pagare per nuovi software appositamente progettati ogni volta che una macchina diventava obsoleta.

Gli hacker erano in grado di trasferire gli stessi strumenti software da una macchina all'altra anziché dover reinventare l'equivalente di un certo comando ogni volta che una macchina diventava obsoleta e doveva essere sostituita con una più moderna. Sia Unix che C avevano delle caratteristiche rivoluzionarie per l'epoca poiché si basavano su un concetto di semplicità estraneo al modo di programmare un po' barocco ed involuto comune a quel tempo. L'intera struttura logica di C poteva essere, più o meno facilmente, memorizzata dal programmatore (al contrario della maggior parte degli altri linguaggi) limitando il ricorso ai manuali mentre Unix era a sua volta strutturato come un pacchetto flessibile di semplici programmi/moduli pensati per combinarsi in vari modi a seconda delle esigenze, superando il concetto imperante di un unico blocco software disegnato specificatamente e su misura per una determinata macchina.

La diffusione di Unix in AT&T fu estremamente rapida a dispetto della mancanza di programmi e di supporto formale; queste caratteristiche fecero si che Unix fosse presto adottato dalla maggior parte delle università, dei laboratori di ricerca informatica ma soprattutto da una moltitudine di hacker. Esso infatti aveva integrata al suo interno una specifica *task*[16] di networking, una caratteristica grazie alla quale due macchine Unix potevano comunicare scambiandosi dati attraverso una normale linea telefonica; nacque così quella che ancora oggi è chiamata *Usenet*, un network[17] di utenti Unix che diede un grosso stimolo alla diffusione e alla condivisione del software (andava a raccogliere i vari gruppi di discussione, oggi noti come *newsgroup*).

Anche se questo metodo di trasmissione risultava lento, consentiva comunque a qualsiasi utilizzatore di questo sistema operativo di comunicare senza per forza dover accedere ad ARPAnet, appannaggio esclusivo al tempo di pochi soggetti istituzionali ben determinati. Il fatto che AT&T, detentrice attraverso i Laboratori Bell del progetto Unix, negli anni Cinquanta fosse stata costretta dalle autorità antitrust statunitensi a non intraprendere qualsiasi attività commerciale al di fuori dei servizi telefonici e di quelli telegrafici, fece sì che Unix stesso non potesse essere fornito con un servizio di assistenza e correzione dei bug[18]. Pertanto la comunità di utenti/sviluppatori si trovò in un certo senso costretta a sviluppare autonomamente il codice sorgente, apportando modifiche e miglioramenti che ovviamente finivano per essere liberamente condivisi.

---

[15] Il linguaggio di programmazione C viene tuttora ampiamente usato per la sua "portabilità" (dal termine inglese *portability*) cioè per la sua capacità di scrivere istruzioni, indipendentemente dall'architettura del calcolatore su cui le istruzioni stesse saranno eseguite. Questo fa si che lo stesso software possa funzionare su computer totalmente diversi senza dover essere di volta in volta riadattato.

[16] Il termine informatico task indica un processo eseguito dall'elaboratore in un certo momento. Per esempio il programma usato per scrivere questo elaborato è un processo.

[17] Si rimanda alla nota 8 per una breve definizione, al capitolo 4 e al glossario alla fine dell'elaborato per una più ampia trattazione.

[18] Dall'inglese *bug*, cioè baco. Nel software sta ad indicare un difetto, un errore di programmazione presente nel codice sorgente. Spesso vengono sfruttati da esperti informatici malintenzionati per accedere ad un software e infettare un computer con un virus. Ovviamente solo avendo a disposizione il codice sorgente è possibile rilevare e correggere l'errore, per quanto la cosa richieda una certa conoscenza tecnica.

1.3 La frammentazione di Unix

Nel 1979, AT&T si rese conto della diffusione e dell'alto livello qualitativo raggiunti da Unix (giunto ormai alla sesta versione AT&T), grazie soprattutto allo sviluppo operato dagli utenti/programmatori, e percepì che il mancato sfruttamento commerciale avvenuto fino a quel momento costituiva un danno economico (fino ad allora aveva concesso il codice sorgente a università e centri di ricerca attraverso licenze a basso costo). Per questo motivo sfruttando una scissione in 26 società, chiamate BabyBell, che permetteva di aggirare le severi normative antitrust, AT&T iniziò prima a limitare lo studio del codice sorgente, impedendo di fatto sia ai centri universitari che agli stessi Laboratori Bell di continuarne lo sviluppo, e poi negli anni successivi iniziò ad esercitare il copyright che deteneva sullo stesso.

Tuttavia lo sfruttamento commerciale di Unix si rivelò presto un fallimento provocando, al contrario di quello che sperava AT&T (cioè il conseguimento del monopolio del mercato), la creazione di una babele di versioni diverse parzialmente incompatibili fra loro. Molti degli ex sviluppatori infatti, non accettando di pagare i diritti ad AT&T per l'uso del codice sorgente di Unix, preferirono passare a versioni alternative. In particolare si diffuse la BSD (Berkeley Software Distribution, dell'omonima università), sorta già nel 1977 come versione alternativa allo Unix ufficiale e che successivamente sarebbe stata ripresa dalla Sun Microsystem, una società fondata da alcuni hacker dell'università californiana. La BSD ottenne un grande successo conquistando buona parte del mercato, soprattutto perché si abbinava a soluzioni hardware innovative come la possibilità di collegare fra loro più terminali (in pratica una LAN[19]) grazie alle interfacce Ethernet[20].

Uno dei grandi meriti però che ebbe la BSD fu quello di aver per prima incorporato nei propri sistemi e supportato il rivoluzionario protocollo di rete TCP/IP che offriva una soluzione al problema delle reti favorendo un'ulteriore crescita di Internet. Si scatenò in quegli anni una "guerra degli Unix" che sarebbe durata per tutto il decennio, spesso con lotte di natura legale, a causa del copyright sul codice: da una parte vi erano la BSD della Sun e tutta una serie di altre versioni, comunque proprietarie, che consentivano di risparmiare i costi della licenza AT&T, dall'altra la versione ufficiale (AT&T).

1.4 Il software proprietario e il progetto GNU

Negli anni Ottanta però non accadde solo questo, nacquero infatti i *personal computer* cioè macchine che a prezzi tutto sommato accessibili fornivano una notevole potenza di calcolo. I computer, che fino a poco tempo prima erano destinati ad una élite informatica, divennero presto delle *commodity* cioè beni di massa. Tra i cambiamenti determinati dalla nuova configurazione hardware vi fu sicuramente la trasformazione della collegata industria del software: il sistema operativo divenne il programma più importante. Fu in quegli anni che Microsoft iniziò a stabilire le basi del suo successo. La qualità del software non era più un fattore di differenziazione tra i costruttori dell'hardware per cui in questo contesto escludere gli altri dall'accesso ai sorgente era essenziale per il dominio del mercato delle applicazioni software. Si diffuse così sempre più la pratica di creare software chiuso.

In questo contesto fu radicale il passaggio dei migliori programmatori dalle strutture di ricerca accademiche verso il laboratori delle imprese. La fuoriuscita dei ricercatori (anche dal MIT) evidenziò la crescita di un mercato del software caratterizzato dalla protezione della proprietà intellettuale. Coloro che erano rimasti nelle università furono obbligati a negoziare tali diritti con le imprese per ottenere copie eseguibili di alcuni importanti software; il loro lavoro che fino a poco tempo prima, attraverso la condivisione e la libera distribuzione dei codici sorgente, aveva contribuito notevolmente allo sviluppo di questi prodotti venne considerato alla stregua della pirateria informatica[21].

---

[19] LAN sta per *local area network*; indica una rete di computer limitata ad un certo ambiente. Un semplice esempio di LAN sono i computer di uno o più uffici, collegati fra loro in modo da poter scambiarsi dati senza ricorrere a supporti di memoria esterni come cd o floppy. Lo standard usato per il collegamento è in genere la connessione Ethernet.

[20] Ethernet è il nome del protocollo informatico più usato per le reti locali (LAN).

[21] In informatica il termine pirateria viene usato spregevolmente per indicare soprattutto l'abuso di opere protette dal diritto d'autore, come nel caso della ridistribuzione di software o nell'acquisizione dello stesso senza il pagamento della relativa licenza d'uso. A volte gli hacker vengono definiti erroneamente pirati informatici, in realtà molti di essi (in particolare Stallman) rifiutano l'idea che condividere copie di un qualche prodotto informatico, attraverso il computer, sia sbagliato o immorale.

La qualità dei prodotti software iniziò a peggiorare velocemente: la mancata disponibilità dei sorgente non consentiva la comprensione e la correzione dei difetti, per non parlare del suo sviluppo. Ma vi fu anche chi tentò di reagire a questo stato di cose.

Richard Stallman, una figura chiave del Laboratorio di Intelligenza Artificiale del MIT e accanito oppositore della commercializzazione della tecnologia del laboratorio, si impegnò per la ricostruzione della comunità di sviluppo accademica. Senza saperlo avrebbe dato vita al movimento Free Software dal quale sarebbe poi sorto l'Open Source. Stallman desiderava cambiare il sistema socioeconomico e legale che in quel contesto limitava la condivisione del software. Il primo impegno era quindi quello di contrastare l'industria del software e le regole da essa applicate per la difesa della proprietà intellettuale. A tal proposito, dopo aver abbandonato i laboratori del MIT, in quanto contrario alla prospettiva che i frutti del suo lavoro venissero commercializzati e non più resi liberamente disponibili alla comunità, avviò nel 1984 il progetto GNU (dall'acronimo ricorsivo GNU's not Unix, cioè GNU non è Unix)[22] per la realizzazione di un sistema operativo liberamente accessibile, basato su Unix e compatibile con lo stesso nonché completo di applicazioni e strumenti di sviluppo.

"*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* (Stallman, 1999)."

Nel 1985, a sostegno del progetto GNU venne fondata la Free Software Foundation (FSF) con l'obiettivo di ricostruire un insieme di prodotti basati sul concetto di software libero e di raccogliere fondi attraverso sia le donazioni che la vendita di programmi (sia GNU che non GNU) e della relativa manualistica. Nello stesso anno, per sollecitare la partecipazione e il sostegno al progetto, Stallman scrisse il documento contenente le linee guida del progetto: il *Manifesto GNU*[23]. Nel vocabolario inglese il termine free assume il significato di "libero" tuttavia viene spesso inteso anche come "gratis"; Stallman riferendosi al free software ovviamente non si riferiva alla gratuità dello stesso quanto alla sua libertà, da cui la celebre frase *"free as in free speech, not as in free beer"*, cioè *"libero come la libertà di parola e non come la birra gratis"*[24]. Per la FSF l'espressione "software libero" si riferiva (e si riferisce tuttora) alla libertà dell'utente di eseguire, copiare, distribuire, studiare, cambiare e migliorare il software. Più precisamente, esso si basa su quattro tipi di libertà per gli utenti del software:

- Libertà di eseguire il programma, per qualsiasi scopo.
- Libertà di studiare come funziona il programma e adattarlo alle proprie necessità. L'accesso al codice sorgente ne è un prerequisito.
- Libertà di ridistribuire copie in modo da aiutare il prossimo.
- Libertà di migliorare il programma e distribuirne pubblicamente i miglioramenti, in modo tale che tutta la comunità ne tragga beneficio. L'accesso al codice sorgente ne è un prerequisito.

Queste quattro libertà (ancora valide) portarono a delle conseguenze molto importanti. La prima è che un programma libero quale il sistema operativo GNU non necessita di alcun permesso aggiuntivo per essere eseguito in diversi ambienti; non vi sono limitazioni di sorta all'uso in scuole, aziende, enti pubblici, secondo le finalità determinate da ciascun contesto. La seconda conseguenza è la possibilità di modificare il software in ogni sua parte, nella maniera che più si adatta alle esigenze personali senza alcun obbligo di notifica. Tuttavia, chi voglia rendere disponibili i programmi originari (terza libertà) o le proprie modifiche (quarta libertà) è libero di farlo, se non addirittura incoraggiato, nel nome di quella collaborazione che il software proprietario invece nega. La distribuzione può avvenire sia in forma eseguibile che sorgente (nel primo caso il codice sorgente deve comunque essere sempre facilmente reperibile).

Un altro aspetto molto importante, ribadito da Stallman, è che il termine "software libero" non significa "non commerciale" e che quindi la redistribuzione del software non debba essere necessariamente gratuita: chiunque può

[22] Stallman (1999)
[23] Stallman (1985)
[24] Free Software Foundation (2002)

richiedere il pagamento per il supporto fisico[25] attraverso cui avviene la distribuzione, per l'eventuale assistenza fornita al prodotto o per qualsiasi altro motivo. Lo stesso Stallman si espresse chiaramente in questi termini: "*The term free software has nothing to do with price. It is about freedom*". Va fatto notare che in realtà nel software libero c'era e c'è tuttora qualcosa di gratuito, cioè la licenza d'uso che invece nel software proprietario viene fatta pagare. La Free Software Foundation aveva poi il compito di proteggere il software libero da possibili tentativi di appropriazione e di sfruttamento commerciale: senza alcuna protezione legale infatti chiunque avrebbe potuto inserire il codice sorgente del software GNU all'interno di soluzioni proprietarie. In tal senso il primo problema che si dovette affrontare fu l'inadeguatezza delle licenze che a quel tempo accompagnavano il software il cui scopo era la descrizione dettagliata dei comportamenti ammessi e di quelli considerati illeciti da parte di programmatori e utilizzatori del software.

Ricorrendo all'aiuto di esperti legali, Stallman creò la General Public License (GPL) e fu introdotto il concetto di *copyleft* che si contrappone a quello di copyright: invece di proteggere i diritti di proprietà su un prodotto ne protegge la libertà di essere copiato, diffuso, analizzato e modificato (nel prossimo capitolo se ne parlerà più ampiamente). La GPL finalmente offriva regole certe per i progetti sviluppati dalla FSF, fu così possibile costruire una comunità di sviluppo volontaria formata non solo da ricercatori e universitari e basata su un modello alternativo di creazione del software che più avanti sarebbe stato chiamato Bazar in netto contrasto con il modello commerciale denominato Cattedrale[26]. Nel corso degli anni Ottanta il progetto GNU progredì senza però raggiungere la completa autonomia da Unix: alla fine del decennio infatti mancava ancora una componente fondamentale, il kernel[27], cioè il cuore del sistema operativo. Stallman si era dedicato ad esso attraverso il progetto HURD senza però arrivare ad un risultato soddisfacente dal punto di vista tecnico.

1.5 Internet e il kernel Linux

Nel 1991, proprio quando le cose sembravano andare per il peggio, accaddero tuttavia una serie di eventi che avrebbero segnato una svolta nella vita del movimento. Linus Torvalds, uno studente dell'università di Helsinki, iniziò a studiare a fondo il sistema operativo Unix concentrando le sue attenzioni sul relativo kernel. L'obiettivo era di sviluppare una versione compatibile con l'architettura hardware che nel frattempo si era diffusa e avrebbe poi dominato a partire dagli anni Novanta (fino ad oggi) grazie al basso costo e alle elevate prestazioni: il processore Intel 386. La scelta di Unix e non di altri sistemi operativi come il DOS (Microsoft) fu forzata in quanto esso era l'unico che implementava il *multitasking*[28], oggetto dei suoi studi. L'eccessivo costo della licenza d'uso e la carenza di computer equipaggiati con Unix a disposizione degli studenti costrinsero però Torvalds ad impegnarsi nello sviluppo di un proprio sistema operativo di tipo Unix, partendo da *Minix*, un suo clone, sviluppato a fini didattici da Andrew Tanenbaum presso l'università di Amsterdam.

Indubbiamente egli non possedeva le risorse, il tempo e le capacità necessarie, tuttavia diede inizio alla sfida: in tal senso la sua inesperienza nel valutare l'impegno richiesto dal progetto fu decisiva per la continuazione dello stesso. Lui infatti affermò che se avesse valutato correttamente l'impegno richiesto non lo avrebbe mai lanciato. Sfruttando le applicazioni software create dalla FSF creò una prima versione, per quanto incompleta, nell'autunno dello stesso anno sotto il nome di *Linux*. Ovviamente non poteva sapere che aveva posto le basi per il più importante e famoso progetto Open Source. Seguendo i principi del software libero, Torvalds mise in Internet il suo lavoro: non restava infatti che richiedere alla comunità in rete l'indispensabile collaborazione per far evolvere il progetto verso una forma più completa. Per tale motivo egli diede l'annuncio del proprio progetto in uno dei forum

---

[25] Supporto di memoria, tipo cd o dvd.

[26] Raymond (1998a).

[27] Il kernel è la parte fondamentale di un sistema operativo. Si tratta di un software con il compito di fornire ai programmi in esecuzione sul computer, un accesso sicuro e controllato all'hardware. Siccome possono esserci più programmi in esecuzione simultanea e l'accesso all'hardware è limitato, il kernel ha anche la responsabilità di assegnare una porzione di tempo macchina e di accesso all'hardware a ciascun programma, funzione detta multiplexing.

[28] Come si è detto l'elaboratore, o meglio il processore (CPU) lavora su ciascun processo. Il multitasking consente di eseguire più processi contemporaneamente: ai giorni nostri è una cosa normale poter usare più applicazioni nello stesso momento ma, a quel tempo questa funzionalità era agli inizi e rara.

dell'università e con esso chiese esplicitamente aiuto per lo sviluppo del kernel[29].
La risposta della rete fu immediata e straordinaria, attorno al progetto Linux si costituì una numerosa comunità di sviluppo. In tal senso, va ricordato il grande contributo, fondamentale, dato dalla rete Internet. Benché avesse visto la luce già negli anni Settanta, fu soltanto agli inizi degli anni Novanta, con la diffusione del protocollo *http*[30] e la nascita dei primi browser, che Internet cominciò ad essere diffusa prima in ambito accademico e poi in modo sempre più capillare anche tra i semplici privati. La diffusione dei codici sorgente in una Rete che stava vivendo un boom di accessi e la contemporanea convergenza di interessi nei suoi confronti da parte della comunità hacker, che ancora non aveva un kernel per il proprio sistema operativo (GNU), furono senz'altro i due fattori che decretarono un rapido sviluppo del kernel Linux.

La crescita esponenziale di Internet mise in contatto sviluppatori volontari da tutto il mondo, favorendo così l'espansione della comunità. La capacità produttiva aumentò sia in quantità, come numero di progetti attivabili, che come qualità. Dalla collaborazione tra la comunità Linux e la Free Software Foundation venne quindi sviluppato GNU/Linux ovvero un sistema operativo completo, non commerciale, rilasciato sotto licenza GPL (General Public License) e destinato a diventare il simbolo dell'Open Source. Il sogno di tutti gli hacker si era finalmente realizzato: un sistema operativo che era stato progettato "da un hacker, per gli hacker" come lo stesso Torvalds lo definì. Nel giro di pochi anni, la crescita del sistema operativo fu esponenziale; le linee di codice sorgente che componevano Linux (inteso come kernel) sarebbero passate dalle 10.000 scritte da Torvalds alle circa 1,5 milioni di righe nel 1998.

---

[29] Gli annunci che Torvalds pubblicò sul newgroup dell'università dedicato a Minix, al fine di rendere pubblico il suo progetto e trovare collaboratori nella comunità (Torvalds 1999).

From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki

"Hello everybody out there using minix - I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix (...) I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them."

From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: Free minix-like kernel sources for 386-AT
Message-ID: <1991Oct5.054106.4647@klaava.Helsinki.FI>
Date: 5 Oct 91 05:41:06 GMT
Organization: University of Helsinki

"As I mentioned a month(?) ago, I'm working on a free version of a minix-lookalike for AT-386 computers. It has finally reached the stage where it's even usable (though may not be depending on what you want), and I am willing to put out the sources for wider distribution (...). This is a program for hackers by a hacker. I've enjouyed doing it, and somebody might enjoy looking at it and even modifying it for their own needs. It is still small enough to understand, use and modify, and I'm looking forward to any comments you might have. I'm also interested in hearing from anybody who has written any of the utilities/library functions for minix. If your efforts are freely distributable (under copyright or even public domain), I'd like to hear from you, so I can add them to the system."

[30] Http è l'acronimo di HyperText Transfer Protocol (protocollo di trasferimento di un ipertesto). Viene usato come principale sistema per la trasmissione di informazioni sul web.

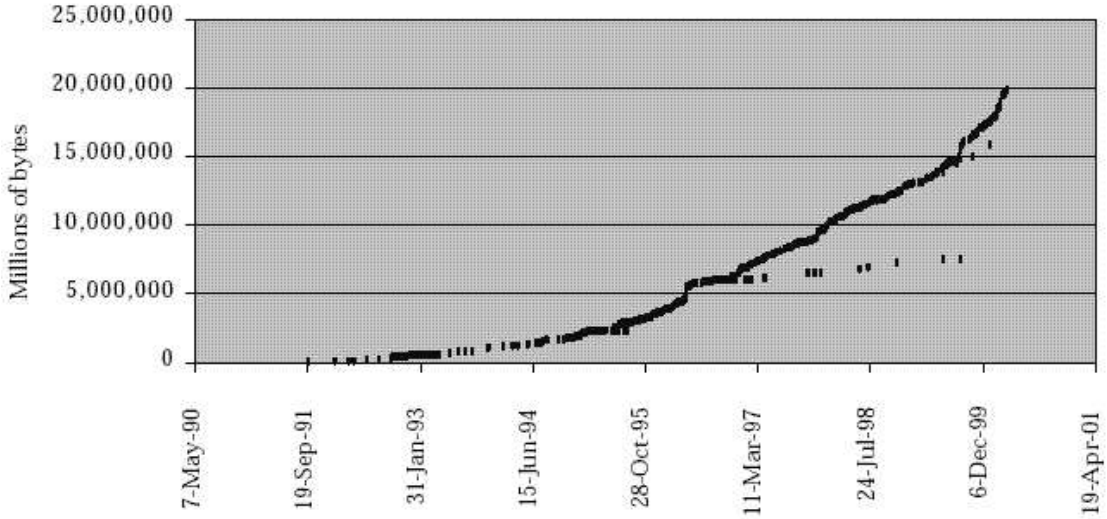


Figura 1.1 Dimensioni del kernel Linux (codice compresso).
Fonte: Weber (2000).

Eric Raymond, che sarà uno dei personaggi di spicco del movimento Open Source, descrisse così quegli anni:

"*Incontrare Linux fu per me uno shock. Nonostante fossi da anni attivo nella cultura hacker, recavo con me il pregiudizio, peraltro non provato, secondo cui gli hacker amatoriali, per quanto dotati potessero essere, non fossero in grado di chiamare a raccolta le risorse e le abilità necessarie per produrre un sistema operativo multitasking effettivamente utilizzabile. Proprio qui gli sviluppatori HURD avevano accumulato fallimenti su fallimenti per almeno un decennio. Ma, dove loro non erano riusciti, avevano trionfato Linus Torvalds e i suoi compagni* (...) *Vedere tutto questo stupendo codice muoversi di fronte a me come un vero e proprio sistema funzionante fu per me un'esperienza entusiasmante* (...) *Fu come se, per anni, avessi cercato e selezionato ammassi di componenti automobilistici sconnessi per poi, improvvisamente, trovarmi di fronte agli stessi particolari assemblati in una Ferrari rosso fuoco, con la portiera aperta, la chiave penzolante dal cruscotto e il motore rombante quasi a cantare un inno alla potenza..*(Raymond, 1999a)".

Sotto l'aspetto più tecnico, di programmazione informatica, la crescita della comunità hacker a tali livelli e con tali risultati fu una sorpresa tanto da contraddire alcune delle leggi base dell'informatica. Tutta la tradizione della progettazione software è dominata dalla Legge di Brooks, secondo cui, aumentando il numero N di programmatori, il lavoro eseguito aumenta in modo direttamente proporzionale a N, ma la complessità e la vulnerabilità ai bug aumenta di N al quadrato, dove N al quadrato è il numero di percorsi di comunicazione (e potenziali interfacce del codice) tra le basi di codice degli sviluppatori. In pratica, secondo la Legge di Brooks, un progetto a cui avessero contribuito migliaia di sviluppatori non poteva che essere un'accozzaglia di codice instabile e traballante. In qualche modo la comunità di Linux aveva sconfitto questo effetto "N al quadrato" ed era riuscita a produrre un sistema operativo di sorprendente qualità[31]. L'eccezione rappresentata dal fenomeno GNU/Linux sarebbe stata spiegata, seppur in modo un po' irriverente, attraverso la cosiddetta "Legge di Linus" (dal nome di Torvalds) secondo la quale "se avete abbastanza paia di occhi vi mangerete tutti i bachi".

L'enorme numero di sviluppatori che da tutto il mondo partecipavano al progetto di sviluppo di Linux rendeva molto probabile che da qualche parte potesse esserci qualcuno che osservando una parte dei sorgente trovasse un errore sfuggito a tutti gli altri programmatori. E probabilmente da qualche altra parte c'era qualcuno capace di sistemare con facilità questo baco. In sintesi la comunità aveva creato il più efficiente ed efficace metodo di sviluppo software mai visto e non se ne era resa conto. Di questo si trattava: diverse consuetudini si erano evolute diventando una serie di pratiche, trasmesse per imitazione ed esempi, senza alcun linguaggio né alcuna teoria che potesse spiegarne il funzionamento e la riuscita[32]. In effetti il rilascio della versione 1.0 che nella progettazione di un

[31] Raymond (1999a)
[32] Raymond (1999a)

software sancisce di fatto una certa stabilità e affidabilità del prodotto, avvenne già nel 1994. Va comunque ricordato che sebbene Linux stesse raggiungendo grandi traguardi, sia in termini tecnici che di semplice popolarità, non era l'unico progetto software di quel genere e in tal senso la comunità GNU/Linux non era sola.

Infatti nel 1989 era stata rilasciata una prima versione del sistema operativo FreeBSD, anch'esso nato come clone di Unix dalla BSD della Sun e anch'esso coperto da una licenza che tutelava la libertà del software (pur con una minore forza della GPL), la BSD License. Va poi menzionato il progetto Apache, operante sui server[33], che verrà trattato più avanti. Il fenomeno Linux crebbe proporzionalmente all'interesse che suscitava nella comunità hacker dando vita ad un singolare "assurdo economico", come poi sarà ribattezzato[34]: risultava infatti incomprensibile agli analisti comprendere come, in un mercato che si era ormai totalmente piegato alle impostazioni neo-fordiste dettate dal monopolista di fatto, Microsoft, un nuovo potenziale concorrente si potesse affacciare e riscuotere un tale successo senza un solido modello economico a guidarlo[35]. In tal senso occorre precisare che la GPL, e più in generale la filosofia del software libero, non vietasse, ma anzi incentivasse, la possibilità di ottenere un profitto dalla vendita di software facente parte del sistema GNU; l'unica condizione che si doveva sempre soddisfare era il rilascio, assieme al software, del relativo codice sorgente, rispettando quindi le libertà imposte dalla GPL. In sostanza il prezzo non si riferiva tanto al software ceduto quanto ai servizi "accessori", forniti dal distributore, come il supporto di memoria in cui veniva registrato e l'eventuale servizio di assistenza. A tal proposito Stallman si espresse così:

"*Molta gente crede che lo spirito del progetto GNU sia che non si debba far pagare per distribuire copie del software o che si debba far pagare il meno possibile - solo il minimo per coprire le spese. In realtà noi incoraggiamo chi ridistribuisce il software libero a far pagare quanto vuole o può* (..) *Il termine "free" ha due legittimi significati comuni; può riferirsi sia alla libertà che al prezzo. Quando parliamo di "free software" parliamo di libertà, non di prezzo. I programmi liberi sono talvolta distribuiti gratuitamente e talvolta ad un prezzo consistente. Spesso lo stesso programma è disponibile in entrambe le modalità in posti diversi. Il programma è libero indipendentemente dal prezzo perché gli utenti sono liberi di utilizzarlo. Programmi Non-Liberi vengono di solito venduti ad un alto prezzo, ma talvolta un negozio vi darà una copia senza farvela pagare. Questo non rende comunque il software libero. Prezzo o non prezzo, il programma non è libero perché gli utenti non hanno libertà* (Stallman, 2000)".

Sarebbe iniziata in quegli anni una diatriba su quale fosse il modello economico adottato dal software libero, se uno nuovo, non più fatto di scelte imposte dall'alto verso il basso (top-down), ma dalle scelte degli utenti (bottom-up), al tempo stesso anche potenziali ideatori, produttori e distributori o in realtà un adattamento, limitato all'industria del software, del modello preesistente. Tuttora le opinioni a riguardo sono discordanti, si cercherà di approfondire l'argomento nel terzo capitolo, dove verranno trattati gli aspetti tecnici e organizzativi, e nel quarto dove si passerà a quelli economici.

1.6 Le distribuzioni

Dal punto di vista operativo, la distribuzione di Linux prese presto due strade distinte che avrebbero segnato inevitabilmente i rapporti all'interno della comunità: da un lato la creazione di alcune versioni curate e seguite dalla comunità hacker e che si sarebbero mantenute nel tempo più in linea con le origini del movimento, dall'altro la distribuzione commerciale curata da alcune aziende come Red Hat, che venne fondata nel 1994. Al primo filone sono riconducibili distribuzioni[36] come Debian e Slackware che soprattutto nella prima conservano l'idea di un sistema operativo "per gli hacker", più tecnico. Nel secondo invece rientrano distribuzioni quali Red Hat appunto e Suse, curate dalle omonime società informatiche: uno degli obiettivi perseguito da queste aziende, che inizialmente credevano l'intero affare Linux "un fuoco di paglia", come avrebbe dichiarato Robert Young fondatore di Red Hat, fu quello di risolvere le principali lacune di Linux riassumibili nell'assenza di un'interfaccia grafica agevole e nella

[33] Per una breve spiegazione si rimanda al paragrafo 1.8

[34] Bonaccorsi et. al. (2001)

[35] Young (1999)

[36] In merito a GNU/Linux il termine distribuzione viene usato per indicare un insieme di pacchetti software assemblati dalla relativa comunità e/o software-house (Debian o Slackware o Red Hat....) e che vanno a formare il sistema operativo, completo di alcune applicazioni.

scarsità di programmi applicativi, come fogli di calcolo o word processor, compatibili con il nuovo sistema.

La loro attività però si sarebbe incentrata soprattutto sulla fornitura di assistenza tecnica, documentazione e formazione nonché ovviamente sulla distribuzione del sistema Linux presso gli utenti meno esperti. Adottando un modello di business completamente focalizzato sulla distribuzione di prodotti copyleft e dei servizi di contorno richiesti vennero chiamate *pure player*: infatti, solo una minima parte dei profitti proveniva dalla vendita del software il cui valore era legato non ai pacchetti che lo componevano ma, come già detto, dal contributo in termini di integrazione e confezionamento[37]. La condivisione dei costi di sviluppo con la comunità e la distribuzione attraverso la licenza GPL permetteva a queste aziende di offrire delle distribuzioni a prezzi molto inferiori alle alternative software costituite da prodotti proprietari.

1.7 L'Open Source

Nonostante i punti di forza che il software libero dimostrava di avere e la diffusione di Linux molte aziende del settore informatico mostravano una certa diffidenza nei confronti del fenomeno. Vi era infatti ancora molta confusione sul concetto di software libero ed era comune associare l'espressione "free software" al concetto di gratuito. Un gruppo di sviluppatori interni alla comunità hacker, molti dei quali avevano costruito la loro credibilità all'interno della Free Software Foundation, iniziò perciò a riunirsi per trovare una strategia che aiutasse a rendere il free software più attraente per le imprese: sebbene la FSF sostenesse il motto "*free speech*, *not free beer*", l'errata interpretazione data proprio dal termine free restava il punto debole. Tutto ciò era sufficiente perché molte software house decidessero di non interessarsi e implicarsi al fenomeno, temendo di perdere gran parte del proprio capitale intellettuale costituito dalla proprietà esclusiva dei codici sorgente.

Il punto di svolta avvenne nel 1998: Eric Raymond, un personaggio che aveva in passato partecipato attivamente al lavoro della comunità hacker, conscio dell'innovazione che la stessa stava portando nel mondo dell'informatica ma anche dei suoi limiti, divulgò un libro che avrebbe "sdoganato" la comunità hacker agli occhi del mercato. Il libro si intitolava "The Cathedral and the Bazaar" e altro non era che il racconto della sua esperienza, testimone ma anche attore di un progetto Open Source, dove la collaborazione fra utenti/sviluppatori era incoraggiata dalla disponibilità del codice sorgente. Raymond distingueva il modello di sviluppo libero usato dalla comunità, "il Bazar", da quello commerciale usato dalle software-house, "la Cattedrale", e ne illustrava il funzionamento facendone risaltare le differenze.

"*Mi decisi quindi a scrivere ciò che avevo visto, impostando il tutto in uno stile narrativo, con vivaci metafore che descrivessero la logica affiorante da tali consuetudini ed esercizi. CatB*[38]*, in realtà, non fu una scoperta. Nessuno dei metodi descritti nell'opera fu da me inventato. La novità non furono i fatti descritti, ma le metafore utilizzate e lo stile narrativo adottato: una storia semplice e interessante che spingeva il lettore a vedere i fatti in modo completamente nuovo* (Raymond, 1999a)."

Il saggio ebbe un grande successo attirando non solo l'attenzione della comunità hacker ma anche quella di una società, la Netscape Communications. La Netscape, società all'avanguardia nella tecnologia Internet e titolo in rapida ascesa alla borsa di Wall Street, era diventata il bersaglio delle armi offensive messe in campo da Microsoft. Microsoft, giustamente, temeva che i formati aperti[39] presenti nel browser Navigator di Netscape potessero erodere il proprio lucrativo monopolio nel mondo dei PC. Tutto il peso dei miliardi di Microsoft e una tattica ambigua che avrebbe poi dato vita a una causa antitrust furono impiegati contro il browser di Netscape. Alla Netscape fervevano le discussioni sulla migliore controffensiva da adottare per sconfiggere questa minaccia.

---

37 Young (1999)

38 CatB è l'abbreviazione di "The Cathedral and the Bazaar".

39 In informatica il formato indica generalmente le modalità con cui i dati vengono rappresentati elettronicamente in modo che i programmi possano elaborarli; per elaborare un file, un programma deve sapere come i dati vi sono "memorizzati" cioè in che forma, sono rappresentati. Il formato specifica la corrispondenza fra la rappresentazione binaria e i dati rappresentati (testo, immagini statiche o dinamiche, suono, ecc.). Esempi di formati sono BITMAP, GIF, JPEG, ecc. I formati aperti così come il codice sorgente aperto sono resi pubblici assieme alla relativa documentazione di conseguenza sono liberamente utilizzabili e spesso divengono degli standard.

Una delle prime idee fu quella di aprire il sorgente del browser di Netscape ma era difficile dimostrare che, così facendo, si sarebbe evitato il predominio di Internet Explorer, il browser di Microsoft. Allora Raymond non se ne rese conto, ma CatB assunse un ruolo importante in questa azione. Nell'inverno del 1997, fu preparata la scena che doveva consentire a Netscape di spezzare le regole del gioco commerciale e offrire alla comunità hacker un'opportunità senza precedenti. Il 22 gennaio 1998 la Netscape annunciò che avrebbe ceduto a Internet i sorgente della linea *client*[40] di Netscape. Poco dopo che la notizia gli fu giunta, il giorno seguente, Raymond apprese che il CEO[41] Jim Barksdale, parlando ai reporter dei mass media nazionali, aveva definito la sua opera come "l'ispirazione fondamentale" da cui era scaturita questa decisione.

"Per la prima volta nella storia della cultura hacker una società inclusa fra le 500 di Fortune, quotata a Wall Street, aveva puntato il proprio futuro sulla convinzione che la nostra strada fosse quella giusta e, più specificamente, sull'analisi che io avevo formulato e secondo cui la nostra via era giusta (Raymond, 1999a)."

Non ci volle molto per dedurre che, a questo punto, la comunità hacker non poteva far altro che assistere la Netscape nella sua battaglia. Se la società avesse perso, sarebbe stata una sconfitta pesante anche per la comunità che per vent'anni aveva visto tutte le idee brillanti e le nuove tecnologie nate al suo interno venire sconfitte dalle scaltre strategie di marketing delle società commerciali, le sole vincitrici reali[42]. Raymond si mise subito a collaborare con la società per redigere una licenza adatta agli scopi del progetto, denominato Mozilla, e definire una strategia da seguire. La licenza avrebbe dovuto incentivare i contributi volontari della comunità e nello stesso tempo proteggere gli interessi commerciali di Netscape; in tal senso nessuna delle licenze esistenti (nemmeno la GPL) sembravano corrispondere a tali esigenze[43]. Il compromesso fu raggiunto sul rilascio dei codici sorgente di Navigator che avvenne sotto una doppia licenza in modo da tutelare, come appena detto, i diversi interessi della Netscape e delle altre società che possedevano porzioni del codice sorgente originario nonché quelli della comunità che voleva mantenere libere le proprie future modifiche.

La data del 31 marzo 1998 rappresentò uno spartiacque temporale: sul codice sorgente creato fino a quel momento Netscape e soci avrebbero mantenuto dei diritti privilegiati mentre sulle innovazioni e i miglioramenti introdotti dal rapporto di collaborazione con la comunità avrebbe vigilato la Mozilla Public License. Tuttavia l'aspetto su cui sembravano sorgere i problemi maggiori era la necessità di ricorrere ad una campagna pubblicitaria che richiedeva, per essere efficace, la costruzione di un'immagine e il cambiamento del marchio. Fu proposto l'utilizzo del termine "Open Source" come migliore sostituto del termine "free" che negli anni sembrava aver fatto più danni che portato benefici, vista l'ambiguità del termine e la presunta ostilità del free software nei confronti dei diritti di proprietà intellettuale (quando non anche ad accuse di comunismo), sebbene questa "novità" non incontrasse il favore di Stallman. All'interno della comunità infatti, nonostante lo sviluppo ed il miglioramento della applicazioni esistenti fosse virtualmente aperto a chiunque, si era riscontrata una certa tendenza accentratrice da parte della stessa FSF che impediva, o nel migliore dei casi limitava, i contributi provenienti da fonti esterne ad essa. L'esempio calzante è fornito dal caso sviluppatosi attorno al kernel HURD, su cui Stallman impedì di lavorare a gran parte della comunità hacker, e trasformatosi presto in un progetto semi-fallimentare.

Una parte consistente della comunità hacker non si era mai riconosciuta pienamente nelle idee "totalitaristiche" propugnate da Stallman. La nuova corrente "moderata", ormai in via di istituzionalizzazione e a capo della quale stavano ponendosi nuovi leader quali lo stesso Raymond e Bruce Perens, non poteva quindi che incontrare numerose simpatie. Era allora ed è tuttora inutile spiegare che la Free Software Foundation non è ostile alla proprietà intellettuale e che la sua posizione non è esattamente ciò che risponde ai principi del comunismo. Gli

---

[40] Il client è una sorta di alter ego del server. Si tratta sempre di un computer collegato ad un server al quale richiede uno o più servizi. Molti software si dividono in una parte client e in una server, per indicare la loro diversa sede: il termine client indica quindi anche il software usato sull'omonimo computer per accedere alle funzionalità offerte dal server. Il browser è appunto un esempio di software client.

[41] CEO è l'acronimo dell'espressione inglese *Chief Executive Officer*, usata per indicare la persona che ha la responsabilità più alta all'interno di una società.

[42] Raymond (1999a)

[43] La licenza GPL era ancora considerata improponibile dalla maggior parte delle imprese software perché in contrasto con la difesa dei diritti di proprietà intellettuale. Nel prossimo capitolo verrà presa in esame in maniera approfondita.

hacker lo sapevano ma, sotto la pressione della release[44] di Netscape, capirono che non contava tanto la posizione effettiva della FSF, quanto piuttosto il mancato successo riscosso dalle sue intenzioni originarie e la spiacevole ma effettiva associazione tra il termine "free software" e questi stereotipi negativi nella stampa commerciale e nel mondo aziendale. Il successo dipendeva quindi dalla capacità di ribaltare questi stereotipi negativi e di associare la FSF a concetti positivi, che suonassero dolci agli orecchi di dirigenti e investitori, che fossero legati a idee di maggiore affidabilità, bassi costi e caratteristiche migliori.

Espresso nel gergo del marketing, si trattava di rinnovare marchio al prodotto, di costruire per il prodotto una reputazione tale da attirare gli interessi delle aziende. Linus Torvalds abbracciò questo progetto sin dal giorno successivo al primo incontro fra la Netscape e Raymond. Il primo passo concreto fu la fondazione dell'Open Source Initiative (OSI), un'organizzazione destinata ad elaborare un documento che aiutasse a definire il software Open Source superando così i pregiudizi sul concetto di free software. Il documento, che venne chiamato *Open Source Definition* (OSD), derivava, su proposta di Perens, dalle *Debian Free Software Guidelines* (Guida Debian al Free software), a loro volta figlie della definizione di software libero data da Stallman. La OSD si caratterizzò per il fatto di non fornire tanto una precisa definizione del software Open Source quanto i dieci requisiti che la licenza a tutela dello stesso avrebbe dovuto soddisfare per essere considerata tale.

Debian, una delle prime distribuzioni Linux, tuttora popolare, fu costruita interamente con free software. Tuttavia, dal momento che c'erano altre licenze oltre al copyleft che comportavano la gratuità, essa ebbe qualche problema nel definire che cosa fosse gratis e i produttori non resero mai chiara la loro politica di free software al resto del mondo. All'epoca, trovandosi a capo del progetto (Debian), Perens affrontò questi problemi proponendo un Contratto Sociale Debian e la Guida Debian del Free Software nel luglio del 1997. Molti sviluppatori inviarono critiche e miglioramenti che furono incorporati nei documenti. Il Contratto Sociale documentava l'intenzione di Debian di costituire il proprio sistema interamente con software libero e la Guida rendeva facilmente possibile la classificazione del software come tale o meno, confrontando la licenza software con la guida stessa[45]. Le linee guida della nuova strategia, sarebbero state quindi le seguenti[46].

● *Dimenticare la strategia "bottom-up"; puntare sulla strategia "top-down"*

Appariva ormai chiaro che la strategia storica seguita per Unix, vale a dire la diffusione dei concetti dal basso verso l'alto (bottom-up), 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.

● *Linux è il caso più rappresentativo*

● *Catturare le società "Fortune 500"*

Bisognava sfruttare la portata di Linux, sia dal punto di vista tecnico, vista la capacità produttiva della sua comunità, che da quello mediatico, per attirare le simpatie dei potenziali investitori, in particolare di quelli più semplici da raggiungere e con maggiori capitali, cioè le 500 più importanti società quotate a Wall Street.

● *Cooptare i mass media di prestigio che si rivolgono alle società Fortune 500*

● *Istruire gli hacker in tattiche di guerriglia marketing*

La scelta di puntare a queste società richiedeva di attirare l'attenzione dei mezzi di comunicazione che formano le opinioni dei top manager e degli investitori, specificamente New York Times, Wall Street Journal, Economist, Forbes e Barron's Magazine. Era necessario però che lo sforzo in tal senso avvenisse non solo dagli ambasciatori come Raymond, Perens, Torvalds ma da parte di tutta la comunità hacker.

● *Utilizzare il marchio di certificazione Open Source come garanzia di genuinità*

---

44 Rilascio di software. In genere la release sta indicare il primo rilascio, la prima distribuzione che avviene su Internet.

45 Perens (1999).

46 Per un'analisi un po' più approfondita della Open Source Definition si rimanda al prossimo capitolo.

Per evitare che il termine Open Source venisse "abbracciato ed esteso"[47] da Microsoft o da altri grandi produttori, che lo avrebbero corrotto e ne avrebbero stravolto il messaggio, occorreva ricorrere a forme di tutela legale. Per questo Raymond e Perens decisero di registralo come marchio di certificazione e di legarlo alla Open Source Definition. In tal modo sarebbe stato possibile intimorire eventuali prevaricatori con la minaccia dell'azione legale[48].

Figure 1.2 e 1.3 I marchi adottati dalla Open Source Initiative (OSI).
Fonte: Open Source Initiative (1999)[49].

1.8 Il business dell'Open Source

Ovviamente la notizia non passò inosservata e venne amplificata dalla stampa destando due reazioni: da un lato un certo scalpore ma anche piacere nella comunità che veniva in tal modo legittimata come realtà nel mercato del software, dall'altro un'ulteriore spinta al definitivo coinvolgimento delle aziende. In tal senso il 1998 può essere considerato l'inizio di una nuova era per il software libero. Dal 1999 in poi il numero di società che parteciparono allo sviluppo e alla diffusione di software Open Source crebbe notevolmente anche perché Linux non era più, già da qualche anno, l'unico progetto di un certo rilievo: nel 1995 nacque il progetto Apache, mentre nel 2000 avrebbe avuto inizio il progetto OpenOffice.

Apache è il nome dato ad una piattaforma *web server*, un web server è il programma (e, per estensione, il computer) che si occupa di fornire su richiesta del browser una pagina web. L'insieme di web server presenti su Internet forma il WWW ossia il World Wide Web, indubbiamente uno dei servizi più sfruttati in rete. Nel 1995 il server web più diffuso era l'HTTPD sviluppato da Rob McCool al NCSA (National Center for Superconputing Application), presso l'Università dell'Illinois; l'abbandono del proprio incarico da parte di McCool determinò la fine (apparente) del progetto. In realtà, partendo da una versione dell'HTTPD, alcuni sviluppatori privati diedero vita al progetto Apache, con l'obiettivo di creare un server web Open Source, basato quindi sulla disponibilità del codice sorgente e su uno sviluppo "comunitario" come avveniva per Linux.

Anche qui il modello di sviluppo adottato incontrò terreno fertile e, nel giro di pochi anni Apache ha conquistato il maggior numero di server web grazie alla sua superiorità tecnica in termini di stabilità, sicurezza e facilità di amministrazione, tutti aspetti fondamentali in un server, costretto a funzionare 24 ore su 24. Nel 1999 fu creata l'omonima fondazione, allo scopo di attribuire una personalità e una tutela legale al progetto. In questo momento Apache è il server web più diffuso, con una quota di mercato pari a circa il 67% e un trend di crescita basso ma costante: la conquista di una quota così ampia del mercato va ricercata non solo nei suoi punti di forza, appena citati, ma anche nella preferenza che grandi società come IBM hanno deciso di accordare a tale soluzione per i propri prodotti. Al contrario,

---

47 Nel 1998 trapelarono da Microsoft alcuni memorandum confidenziali, presto denominati *Halloween Documents*, a proposito delle strategie relative al software Open Source e a GNU/Linux. Contrassegnati come "Microsoft confidential", identificavano il software Open Source e in particolare il sistema operativo GNU/Linux come la maggiore minaccia al dominio di Microsoft dell'industria del software e suggerivano modi per distruggerne il progresso. I documenti ammettevano la superiorità del software Open Source rispetto ad alcuni prodotti Microsoft e impostavano una strategia per combatterli; furono molto imbarazzanti, perché contraddicevano le dichiarazioni pubbliche fatte a questo proposito. Inoltre, le strategie suggerite furono giudicate molto negativamente: invece di migliorare il prodotto offerto fino a superare la concorrenza, secondo i documenti la strategia migliore era di impedire ai prodotti concorrenti di funzionare con quelli Microsoft, non per cause tecniche ma in modo artificioso. Una strategia che è stata descritta da alcuni come "Embrace, extend and extinguish" (abbraccia, estendi ed estingui): si proclama ad alta voce di abbracciare uno standard esistente per i propri prodotti, si prosegue poi estendendo tale standard con nuove funzioni non previste (in un certo senso, non rispettando lo standard che si diceva di rispettare) in modo che i prodotti concorrenti non funzionino più col proprio, e si utilizza questa differenza per far chiudere i concorrenti per mancanza di clienti.

48 Raymond (1999a)

49 http://opensource.org/trademarks/

come risulta peraltro chiaramente dalla figura 1.4, le varie alternative proprietarie hanno visto ridursi di molto la loro quota o nella migliore delle ipotesi, come è accaduto per Microsoft, mantenere un ruolo di secondo piano.

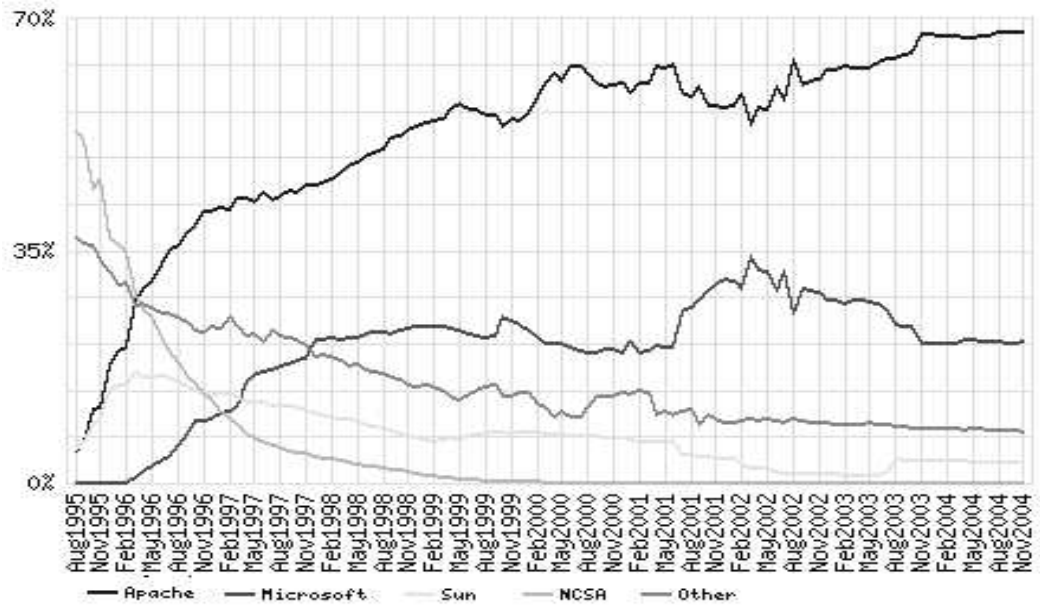


Figura 1.4 Quota di mercato dei principali web server in Internet.
Fonte: Netcraft (2004)[50]

Accanto al fenomeno Apache va poi ricordato un altro dei progetti più importanti: OpenOffice, una suite di applicazioni per l'ufficio sorta come variante aperta del codice sorgente di StarOffice 5.2, dopo che questa fu, a sua volta, acquisito e trasformata in prodotto Open Source da Sun Microsystems, nel 2000. L'obiettivo che si posero i fondatori del progetto, similmente a quanto accaduto nella vicenda Mozilla/Internet Explorer, fu la creazione di una serie di applicazioni per l'ufficio (editor di testi, foglio di calcolo, programma per le presentazioni, disegni) che potesse funzionare su tutte le piattaforme più importanti (Windows, Linux, Mac OS) costituendo una valida alternativa libera a Microsoft Office, già da tempo lo standard sul mercato.

I punti di forza su cui fare leva per la diffusione del prodotto sarebbero stati la compatibilità con i formati dei file di Office e il ricorso al formato aperto XML per il salvataggio dei propri documenti, caratterizzato oltre che dalla sua diffusione, da un'estrema velocità e un'estensibilità a vari formati fra loro diversi. La disponibilità del codice sorgente ha favorito, come era già accaduto in passato in situazioni analoghe, l'instaurazione di un rapporto di stretta collaborazione fra la comunità e la società, riuscita in questo modo ad accedere ad uno sviluppo rapido con costi più contenuti. Così facendo Sun ha potuto rilasciare una doppia versione del software per l'ufficio: la prima, OpenOffice appunto, gratuita ma con un'assistenza limitata all'operato della comunità; la seconda denominata StarOffice, a pagamento e comunque frutto del rapporto collaborativo, dove anche qui come nel caso di alcune distribuzioni Linux (vedi RedHat), il prezzo corrisposto riguarda non tanto il software in sé, quanto il servizio di supporto e assistenza forniti dalla società.

A distanza di quattro anni dalla nascita del progetto, OpenOffice è divenuto ormai una delle alternative più credibili a Microsoft Office, non solo per la sua gratuità, ma anche per alcune caratteristiche tecniche innovative, basti pensare alla possibilità di salvare i documenti in PDF, uno dei formati più popolari per la loro distribuzione in Rete. In questo modo il documento diviene leggibile da chiunque, a prescindere dal sistema operativo. Di sicuro gli aspetti tecnici riguardanti il formato dei documenti presentano notevoli implicazioni circa la possibilità di ridurre il fenomeno del *lock-in*, tipico del mercato dell'informazione, che verrà analizzato più avanti: molto probabilmente non è azzardato affermare che la stessa Microsoft non introduce funzionalità di questo tipo al solo scopo di impedire la migrazione dei propri clienti verso soluzioni alternative, non necessariamente Open Source ma pur sempre compatibili.

[50] http://news.netcraft.com/

CAPITOLO 2
LA TUTELA LEGALE DEL SOFTWARE

2.1 Licenze e copyright

Il software è un codice e come tale è protetto dalle leggi sul diritto d'autore, così come avviene per il più ampio gruppo delle opere artistiche e letterarie. Esso presenta delle caratteristiche che lo accomunano al bene informazione ma lo distinguono dagli altri: la prima riguarda la sua natura, ossia il fatto di essere composto da istruzioni digitali e di aver perciò bisogno di un supporto fisico in cui essere memorizzato, la seconda come conseguenza della tecnologia digitale su cui si basa, è che può essere non solo facilmente ma anche economicamente (per non dire a costo zero) copiato e ridistribuito. La tutela giuridica del software incontra due problemi di fondo, uno legato proprio alla sua riproducibilità e alla sua trasmissibilità a costo prossimo allo zero in un contesto globalmente interconnesso (grazie alla Rete), l'altro alla disomogeneità delle normative nazionali ed internazionali di tutela.

A livello mondiale tale tutela risulta in alcuni casi contraddittoria o inapplicabile con alti costi di misurazione e l'impossibilità di ricorrere all'esecuzione coattiva[51]. Il problema nasce più che altro dall'incapacità degli enti preposti di comprendere l'innovazione che l'informazione digitalizzata impone, non solo a livello economico ma anche nei vari aspetti della vita comune, e di adeguare gli strumenti finora esistenti alle esigenze attuali. Secondo alcuni la soluzione migliore per arginare il problema è quella di ricorrere all'arma del brevetto anche nel software, già applicato diffusamente negli Stati Uniti e in fase di adozione anche in Europa. Tuttavia questa scelta solleva molti dubbi sull'effettiva necessità di questa azione e l'adeguatezza di questo strumento, nato certamente con scopi giusti ma probabilmente più adatto ad altre situazioni o meglio a beni con caratteristiche diverse; una breve riflessione sulla questione verrà riportata più avanti.

Finora i due strumenti usati a tutela dei diritti sul software sono stati il Non-Disclosure Agreement (NDA) e la licenza, che regola il diritto d'autore (copyright). Il primo è un accordo di non divulgazione: in pratica si ottiene l'accesso a informazioni o dati riservati, impegnandosi a non divulgarli e a non utilizzarli al di fuori dell'incarico per cui è stato sottoscritto lo stesso. La fattispecie più frequente è quella del rilascio di codice sorgente a partner industriali, con il vincolo di mantenerne la segretezza; negli ultimi anni la stessa Microsoft è ricorsa ad accordi di questo tipo concedendo ai clienti governativi (se ne è parlato anche recentemente pure per l'Italia[52]) la possibilità di consultare il codice sorgente di Windows. Il software è poi protetto nella maggior parte dei paesi del mondo dalle varie leggi sul diritto d'autore: in sintesi esso garantisce all'autore del codice un certo numero di diritti che in Italia si dividono in diritti morali inalienabili (paternità dell'opera) e diritti patrimoniali (o di sfruttamento economico dell'opera) temporanei.

La licenza invece altro non è che un contratto stipulato fra il detentore del copyright e l'utente, regolante i termini di utilizzo del software da parte di quest'ultimo. In questo modo è possibile porre delle condizioni più dettagliate rispetto a quelle imposte per legge. L'uso del software può essere concesso gratuitamente o a pagamento nei limiti stabiliti dalla licenza stessa; in quest'ultimo caso il prezzo pagato si riferisce all'acquisizione della facoltà di utilizzo e non del relativo diritto d'autore. Teoricamente l'autore del software può creare per lo stesso una licenza su misura e questo è il motivo per cui esiste una pluralità di licenze diverse. La diretta conseguenza è che il nome della licenza usata viene usato, per estensione, ad indicare il software tutelato, per cui sono distinguibili diverse tipologie di software. In tal senso è importante fare una precisazione: in precedenza, nel testo, si è spesso fatto ricorso a termini come "software proprietario" o "software non proprietario". Considerando che (quasi[53]) tutto il software prodotto viene sottoposto al copyright per legge, non sarebbe corretto usare questo termine poiché tutti i programmi (coperti da copyright) sono proprietà di chi detiene il copyright anche quando sono sottoposti a licenze libere quindi l'uso del termine proprietario avviene in questo caso impropriamente per indicare il software non libero.

---

[51] Didonè (2001)

[52] http://www.microsoft.com/italy/stampa/comunicati_stampa/nov04/1011_giust.mspx

[53] Si escluda il software di pubblico dominio di cui si accennerà brevemente nel relativo paragrafo.

2.2 Tipologie di software

Seguendo le licenze più diffuse è possibile dividere il software in tre gruppi:
- software proprietario o chiuso
- software di pubblico dominio
- software libero o aperto

Lo scopo di questa classificazione, seppur sintetica, è quello di fare un po' di ordine e semplificare la comprensione del tema trattato. E' comunque estremamente facile appurare, anche solo navigando in Rete, come ci sia molta confusione sull'argomento e non esista una classificazione ufficiale ma alcune più comuni fra cui questa.

2.3 Software proprietario o chiuso

Ricordando che l'uso del termine proprietario non è del tutto corretto, con esso si intende comunque il software più comune, più noto (o ad esempio Microsoft Windows). Si caratterizza per il fatto di essere rilasciato solo in forma eseguibile, ossia senza codice sorgente, e per essere sottoposto a licenze restrittive circa il suo utilizzo, la modifica e la distribuzione. I diritti che abitualmente si trovano nel software libero sono negati, solo raramente concessi e con delle nette limitazioni (per esempio nel caso di un Non-Disclosure Agreement). Sebbene in genere la cessione del diritto d'uso avvenga a titolo oneroso, sono riscontrabili all'interno di questo gruppo un paio di licenze relative a software ceduti a titolo gratuito e quindi molto diffusi: è il caso del software *shareware* e di quello *freeware*. Il primo è una versione gratuita, liberamente copiabile e distribuibile ma limitata nelle funzionalità e nel periodo d'uso (per esempio un mese), di un certo software. Concedendo questo periodo di prova si cerca di stimolare gli utenti all'acquisto della versione completa e senza limiti temporali d'uso, ma a pagamento.

Il freeware è invece quello che più si avvicina al software libero, non per la comunanza di idee alla base ma per il fatto di essere totalmente gratuito nonché liberamente usabile e copiabile come molti programmi liberi. Proprio la grande notorietà dello stesso fra gli utenti e il ricorso al termine "free" nella sua denominazione per indicarne la gratuità[54], sono alcuni dei motivi per cui si crea una certa confusione sul significato di free software: non software gratuito bensì libero. A titolo esemplificativo una delle applicazioni freeware più comuni è il visualizzatore di documenti PDF Adobe Acrobat Reader. Al di là della coincidenza circa la gratuità del prodotto va però detto che la licenza freeware non prevede il rilascio del codice sorgente e spesso comprende una clausola che impedisce l'uso dell'applicazione a fini di lucro, permettendo solo quello personale. Venendo a mancare molte delle libertà fondamentali del software libero, esso è da considerarsi comunque "chiuso". In alcuni casi il software shareware o freeware viene anche definito software semilibero a causa della sua gratuità e dei diritti concessi sullo stesso.

2.4 Software di pubblico dominio

Dal punto di vista internazionale, il pubblico dominio è quell'insieme di opere d'ingegno e altre conoscenze (opere d'arte, musica, scienze, invenzioni, software ecc.) nelle quali nessuna persona o organizzazione ha un interesse proprietario. Tali opere e invenzioni sono considerate parte dell'eredità culturale pubblica e chiunque può utilizzarle o modificarle senza restrizioni. Mentre il diritto d'autore venne creato per difendere l'incentivo finanziario di coloro che svolgono un lavoro creativo e come mezzo per incoraggiare ulteriore lavoro creativo, le opere di pubblico dominio esistono in quanto tali e il pubblico ha il diritto di usare e riutilizzare il lavoro creativo di altri senza dover pagare un prezzo economico o sociale. Le opere d'ingegno, fra cui il software, rientrano nel pubblico dominio quando non esiste nessuna legge che stabilisca dei diritti di proprietà o quando l'oggetto in questione sia specificatamente escluso da tali diritti dalle leggi vigenti.

Ad esempio, la maggior parte delle formule matematiche non è soggetta a diritti d'autore o brevetti nella larga parte dei casi anche se la loro applicazione in forma di software per computer può essere brevettata[55]. E' però possibile che l'opera ricada nel pubblico dominio anche in altre due fattispecie: più comunemente per la rinuncia

[54] Si ricorda che in inglese il termine *free* assume il doppio significato di libero ma anche gratuito.
[55] Wikipedia (2004)

esplicita, da parte dell'autore, dei diritti patrimoniali sulla stessa oppure per la scadenza del relativo brevetto o diritto d'autore (per il primo il termine è in genere di vent'anni mentre per il secondo ci si rifà alla legislazione del paese di riferimento, spesso legata alla convenzione di Berna sul diritto d'autore). Va infine precisato che questa tipologia di software, a prescindere dalla disponibilità o meno del codice sorgente, non è considerabile come software libero in quanto pur concedendo molte libertà non prevede alcuna protezione legale sul prodotto (in entrambi i sensi).

2.5 Software libero o aperto

I concetti di software libero e di software proprietario si escludono a vicenda di conseguenza la prima denominazione comprende ogni prodotto non classificabile come software chiuso. Le sue peculiarità sono innanzitutto la disponibilità del codice sorgente nonché le quattro libertà sancite dalla Free Software Foundation e già illustrate precedentemente. In breve:

- Libertà di eseguire il programma, per qualsiasi scopo.
- Libertà di studiare come funziona il programma e adattarlo alle proprie necessità.
- Libertà di ridistribuire copie in modo da aiutare il prossimo.
- Libertà di migliorare il programma e distribuirne pubblicamente i miglioramenti, in modo tale che tutta la comunità ne tragga beneficio.

Come già detto, che la distribuzione del software sia o meno gratuita, non è rilevante. E' invece interessante analizzare e approfondire, seppur sinteticamente, quali siano gli obiettivi che in via generale si posero i creatori della prima (se non altro per notorietà e importanza) licenza libera, cioè la General Public License (meglio nota come GPL e traducibile come Licenza Pubblica Generale). Le licenze che fanno parte di questa tipologia sono varie di conseguenza verrà presa a modello, vista la sua importanza e diffusione, soprattutto la GPL e il relativo aspetto del copyleft per poi passare alla Open Source Definition (che come si vedrà in realtà non è una licenza). Innanzitutto è opportuno sottolineare come nella sua storia, relativamente breve, l'industria del software abbia sempre considerato e usato lo strumento della licenza a senso unico: certamente allo scopo generale di tutelare l'autore ma questo stesso fine è sempre stato identificato unicamente nel diritto di sfruttare economicamente il software e di imporre nette limitazioni al suo uso da parte degli utilizzatori.

Con lo sviluppo del concetto di software libero, così come fu pensato da Stallman con il progetto GNU, e il consolidamento della pratica di condividere liberamente i codici sorgente, si posero di fatto due grossi pericoli per il movimento, con implicazioni prettamente legali: fino a quel momento lo scarso impatto del software libero sul mercato aveva mantenuto freddo l'interesse da parte delle software house per il modello ma, il fatto che iniziasse a circolare in Rete del codice libero, in alcuni casi di qualità addirittura superiore a quello proprietario, senza alcuna tutela e in genere gratuito, poneva il rischio che le stesse cercassero di accaparrarsi questo lavoro senza alcun pericolo di ripercussioni legali. Il secondo punto riguardava l'assenza appunto di una licenza: secondo la convenzione di Berna[56] a tutela del diritto d'autore, il software non accompagnato da alcuna licenza viene considerato come se fosse sottoposto ad una licenza proprietaria di conseguenza non può essere distribuito o modificato senza l'esplicito permesso dei titolari dei diritti di sfruttamento economico.

Fu a questo punto che Stallman ebbe una intuizione, rivelatasi poi vitale per il software libero: capì che l'arma più efficace per difendersi dalle maglie troppo strette del copyright stava nel copyright stesso. In pratica il programma doveva sì essere accompagnato da una licenza (ovviamente ribattezzata "libera") nella quale però fossero tutelate le libertà fondamentali e questo anche nel caso in cui il sorgente fosse stato rilasciato, gratuitamente o meno. In questo modo lo strumento della licenza iniziò ad essere usato in modo nuovo, alternativo, per tutelare non il diritto dell'autore di limitare l'uso dell'opera bensì la sua decisione di rendere la stessa liberamente disponibile per chiunque ne fosse interessato e nella sua forma più accessibile, cioè come codice sorgente: il copyright quindi si applicava all'apertura del codice. Lo scopo primario della licenza libera non era la gratuità del software ma la sua sopravvivenza ovvero la certezza che vi fosse la possibilità per chiunque e in qualunque momento, anche futuro, di apportare miglioramenti al programma e di installarlo senza alcuna limitazione[57]. Si otteneva quindi una protezione legale al pari delle più conosciute licenze "proprietarie" con la possibilità di perseguire potenziali abusi o appropriazioni.

[56] http://www.interlex.it/testi/convberna.htm e http://www.ompi.int/treaties/en/ip/berne/index.html
[57] Wikipedia (2004)

2.5.1 GNU General Public License (GPL) e copyleft

La licenza che di fatto ha meglio rappresentato e protetto i principi alla base del software libero è indubbiamente la General Public License introdotta nel 1989 dalla Free Software Foundation (ad opera di Stallman in collaborazione con Eben Moglen[58]) per far sì che le copie del progetto GNU, comprese quelle modificate, fossero accompagnate dal codice sorgente o che fosse comunque facile procurarsi quest'ultimo. Nel giro di breve tempo la GPL è divenuta la licenza libera più comune anche in altri progetti per tutta una serie di motivi: innanzitutto scrivere ex novo una licenza di copyright anche per un legale specializzato in questa materia, che fra l'altro è al centro di continue dispute giurisprudenziali, è molto difficile; inoltre, il suo uso consolidato, la provata validità sul campo e la scrittura da parte di legali esperti ha avvantaggiato sia gli utilizzatori del software che conoscendola già ne hanno saputo valutare le implicazioni sia lo scambio di codice fra progetti liberi diversi. E' innegabile che a distanza di anni dalla sua stesura la GPL rimanga una delle licenze più efficaci per mettere al sicuro il lavoro di un programmatore.

Una delle novità introdotte non solo dal punto di vista legale ma anche sociale dalla stessa resta il concetto di *copyleft*, cioè il "permesso d'autore" che è già stato nominato in precedenza ma che ora verrà definito chiaramente. Indubbiamente esso ha costituito il punto di forza su cui questa licenza si è sviluppata e per il quale è stata ampiamente apprezzata. L'espressione inglese copyleft, gioco di parole sul termine legale copyright per rimarcare la relazione fra i due, indica il metodo generale per realizzare un programma di software libero e richiedere che anche tutte le versioni modificate e ampliate dello stesso rientrino sotto il software libero. In pratica si tratta di una clausola di reciprocità che impone delle restrizioni sul rilascio di opere derivate da un codice sorgente in modo tale che queste si mantengano sempre libere, generalmente sotto la stessa licenza dell'opera originale.

Gli obblighi di reciprocità non sussistono qualora il software sia solo utilizzato ma quando questo, se modificato, viene anche distribuito. Esattamente come per il copyright, in questo caso la distribuzione senza codice sorgente o la violazione della licenza rendono il soggetto perseguibile legalmente da parte dell'autore. La regola alla base del copyleft è identificabile nel "quid pro quo": chi trae benefici dal software è obbligato a condividere con altri i vantaggi che derivano da modifiche e contributi. Ancora una volta una chiara spiegazione sulla prassi seguita viene fornita dallo stesso Stallman nelle sue opere.

"La maniera più semplice per rendere libero un programma è quella di farlo diventare di pubblico dominio, senza copyright. Ciò consente a chiunque di condividere tale programma e i relativi perfezionamenti, se questa è l'intenzione dell'autore. Ma così facendo qualcuno poco incline alla cooperazione potrebbe trasformarlo in software proprietario. Potrebbe apportarvi delle modifiche, poche o tante che siano, e distribuirne il risultato come software proprietario. Coloro che lo ricevono in questa versione modificata non hanno la stessa libertà riconosciuta loro dall'autore originale; è stato l'intermediario a strappargliela. L'obiettivo del progetto GNU è quello di offrire a tutti gli utenti la libertà di ridistribuire e modificare il software GNU. Se l'intermediario potesse strappar via la libertà, potremmo vantare un gran numero di utenti, ma privati della libertà. Di conseguenza, anziché rendere il software GNU di pubblico dominio, lo trasformiamo in copyleft.

Questo specifica che chiunque ridistribuisca il software, con o senza modifiche, debba passare oltre anche la libertà di poterlo copiare e modificare ulteriormente. Il copyleft garantisce che ogni utente conservi queste libertà. Il copyleft fornisce inoltre ad altri programmatori l'incentivo ad aggiungere propri contributi al software libero. Per trasformare un programma in copyleft, prima lo dichiariamo sotto copyright, poi aggiungiamo i termini di distribuzione, strumento legale onde garantire a chiunque il diritto all'utilizzo, alla modifica e alla redistribuzione del codice di quel programma o di qualsiasi altro da esso derivato, ma soltanto nel caso in cui i termini della distribuzione rimangano inalterati. Così il codice e le libertà diventano inseparabili a livello legale. Gli sviluppatori di software proprietario ricorrono al copyright per rubare agli utenti la propria libertà; noi usiamo il copyright per tutelare quella libertà. Ecco perché abbiamo scelto il nome opposto, modificando copyright in copyleft (Stallman 1996)".

Ben presto però ci si rese conto di un problema a livello pratico, una sorta di effetto collaterale posto dalla GPL, a causa del copyleft, che è tuttora controverso e visto con sospetto da molti: la sua *viralità* o *contaminazione* ossia la sua capacità di assegnare ad ogni utente di software copyleft determinati diritti ma anche determinati obblighi e

[58] Eben Moglen (1956) è un professore di legge e storia legale presso la Columbia Law School di New York. Si occupa di software libero.

responsabilità. In pratica, come risulta dal secondo articolo della licenza, la GPL estende i suoi termini a qualsiasi componente che venga incluso o distribuito insieme ad un programma tutelato dalla stessa, non anche però alle eventuali opere che siano state create tramite tale software; quest'ultima ipotesi risulta abbastanza chiara nel caso in cui il software libero, e quindi licenziato sotto GPL, sia un sistema operativo (quale è per esempio Linux): qualora l'utente apporti modifiche al codice sorgente di questo software, ebbene, se licenziate, queste modifiche saranno sottoposte alle previsioni della GPL (saranno quindi a loro volta modificabili, liberamente copiabili e ne dovrà essere reso disponibile il codice sorgente).

Se però tramite questo sistema operativo l'utente riproduce o crea nuovi software, ebbene questi quando saranno ceduti non dovranno rispettare le condizioni della GPL: sarà allora ben possibile per l'utente creare software proprietari avvalendosi di software liberi. All'interno della GPL non appare infatti rinvenibile alcun elemento che vieti questa condotta. Fu quindi proprio l'aspetto appena visto la ragione per cui essa fu scartata al momento del rilascio del codice sorgente di Netscape Communicator: era semplicemente impensabile, infatti, che tutte le parti di codice di terze parti incluse potessero essere rilasciate sotto GPL. Se la vicenda del browser fu una delle prime a sollevare la questione, le difficoltà maggiori sorsero con le "librerie", cioè con quei file che per loro stessa natura contengono funzioni richiamabili e utilizzabili da più applicazioni, senza per questo divenire parte del loro codice sorgente: occorreva correre ai ripari, ma come?

La soluzione più comoda fu la stesura di una nuova licenza, sempre ad opera del movimento ma fatta su misura per risolvere il problema delle librerie appunto e il loro vasto impiego, da parte di un po' tutto il software. Fu chiamata Lesser General Public License o GNU LGPL, traducibile come Licenza Generica Pubblica Attenuata: quest'ultima verrà descritta nel prossimo paragrafo. Per completare questa breve trattazione riguardo la GPL vanno accennati altri due aspetti della licenza che meritano di essere menzionati: il primo è che la licenza non fornisce alcuna garanzia sul prodotto e sul suo funzionamento, tant'è che l'uso dello stesso è espressamente dichiarato a rischio dell'utente. Il secondo aspetto è che la FSF detiene i diritti di copyright sul testo della licenza: a differenza dei software con essa distribuiti, la stessa non è liberamente modificabile quindi copiarla e distribuirla è permesso, ma modificarla è vietato. All'appendice A dell'elaborato, è possibile consultare la traduzione non ufficiale del testo, mentre per la versione originale in lingua inglese, si rimanda al sito internet[59] della Free Software Foundation.

2.5.2 Lesser GNU General Public License (LGPL)

Come si è visto in precedenza la licenza GPL si è dimostrata troppo rigida in alcune situazioni, in particolare nel caso assai frequente in cui si abbia la necessità di usare o richiamare delle librerie, sottoposte ad essa, all'interno di altri prodotti non tutelati allo stesso modo. La licenza originale prevederebbe l'estensione dei propri termini a qualsiasi applicazione, libera o non, che faccia uso delle librerie libere sfavorendone però in questo modo l'uso, lo sviluppo e quindi indirettamente la diffusione sul mercato. Per ovviare al problema si è quindi deciso di creare una versione attenuata della GPL che ne mantenesse il contenuto anche per quanto riguarda il copyleft ma che nello stesso tempo consentisse di collegare alle librerie qualsiasi software senza l'obbligo di rilasciare il nuovo codice ottenuto, appunto sotto GPL: in sintesi è concessa la creazione di software, anche proprietario tramite librerie libere (l'esempio calzante è quello della libreria di sistema *glibc*, presente in Linux, che essendo sottoposta a LGPL può essere collegata a qualsiasi software funzionante in esso per crearne di nuovo).

Questa licenza si rivela essere una buona soluzione quando si desidera creare un software Open Source garantendosi la possibilità di consentire a software forniti di licenza proprietaria di utilizzare le proprie tecnologie[60]. Ovviamente è richiesto che il codice sorgente della parte LGPL sia reso disponibile e che la stessa parte LGPL sia aggiornabile indipendentemente dal resto del software a cui è collegata: è importante precisare che le eventuali modifiche a questa parte ricadono nei termini della omonima licenza. Tuttavia va sottolineato come la maggior elasticità offerta, se da un lato l'ha resa la più appetibile e di conseguenza la più diffusa, dall'altro ha favorito una certa confusione a livello pratico nello stabilire in definitiva a quale licenza sottoporre il codice o le parti di esso, dato che la loro distinzione è spesso difficile. Proprio questi pro e contro della LGPL e la sua difformità o comunque minore attinenza ai principi del software libero hanno prodotto un cambiamento al significato attribuito all'acronimo

[59] http://www.gnu.org/licenses/gpl.html
[60] Fink (2003).

LGPL: non più *library* GPL com'era in origine ma *lesser* GPL; la FSF e lo stesso Stallman ne sconsigliano l'uso[61] pur essendone gli autori, suggerendo di usare al suo posto comunque la GPL assieme ad una clausola speciale:

"*Come eccezione, se si collega questa libreria ad altri file per produrre un eseguibile, quest'ultimo non viene coperto dai termini della GNU General Public License per il legame che sussiste con tale libreria. Questa eccezione non invalida ogni altra ragione per cui il programma eseguibile dovrebbe essere coperto da licenza GNU General Public License* (Fink 2003)[62]."

2.5.3 Licenze doppie e GNU Free Documentation License (FDL)

Dopo aver analizzato la General Public License e la sua variante per i file libreria è opportuno soffermarsi, seppur brevemente, su qualche altra licenza che merita di essere menzionata in funzione dell'argomento trattato in questo elaborato. Come si è accennato in precedenza, in alcune situazioni, per esempio nel caso del browser Netscape Communicator o in quello di OpenOffice, gli sviluppatori si sono trovati di fronte all'impossibilità di ricorrere esclusivamente alla GPL, o per ragioni puramente legali oppure (e questo è il caso più comune) per obiettivi diversi; tuttavia, si è sentito il bisogno di venire incontro alla comunità di utenti/sviluppatori garantendo quindi un certo grado di libertà. Per aggirare il problema si è ricorsi alla pratica, ormai diffusa, di applicare una doppia o addirittura tripla licenza su alcuni software: la cosiddetta *dual licensing* o *dual system*. In sostanza la classica GPL è stata affiancata da varie licenze create *ad-hoc* per ciascun progetto: l'esempio classico resta Mozilla (nato dalle ceneri di Netscape), nel quale è stata redatta la Netscape Public License (NPL) per attribuire alla società proprietaria del vecchio codice, da cui il progetto è partito, alcuni privilegi peraltro non visti di buon occhio da tutta la comunità.

In un secondo momento si è passati alla stesura anche della Mozilla Public License (MPL) riferita invece al codice nuovo. Restando a questo caso concreto, lo staff che si occupa del progetto lavora quindi per rilasciare il codice sorgente sotto tre licenze: la GPL, la LGPL e la MPL. Similmente sono nate varie licenze su misura, una delle più note è la software Apache License legata al progetto omonimo di web server. Va comunque precisato che nonostante questa pratica si sia ormai trasformata da eccezione a regola, in genere le licenze libere non sono reciprocamente compatibili. Anche in altri casi si è deciso di ricorrere alla doppia licenza ma lo si è fatto come strategia di mercato: in pratica per garantire all'utente la possibilità di evitare le clausole di reciprocità e distribuire il codice modificato senza i relativi obblighi (alcuni di essi erano infatti disposti a pagare pur di evitarle).

Si è quindi iniziato a rilasciare il software non solo al di sotto di una licenza reciproca libera (come la GPL) e spesso gratuita ma anche attraverso le classiche licenze proprietarie che prevedono però il pagamento dei diritti; a titolo esemplificativo si pensi a Sun e alla doppia distribuzione del suo pacchetto per ufficio. Resta infine da analizzare un'ultima licenza, all'interno del vasto numero esistente, che si differenzia rispetto a quelle considerate finora per l'oggetto della tutela: non il software bensì la documentazione che normalmente lo accompagna. Da tempo la maggioranza degli autori ha preso l'abitudine di estendere la licenza per il software alla relativa documentazione; benché sia in linea di principio scorretto, visto che la licenza per il software libero fa esplicito riferimento al software, quest'uso è ormai comune e non ha dato adito a problemi legali. Usare la stessa licenza software per la documentazione è quindi una prassi priva di rischi.

La licenzia oggetto di questa breve analisi è la terza ed ultima a soddisfare i termini del copyleft: si tratta della GNU Free Documentation License (FDL), creata dalla FSF per distribuire la documentazione di software e materiale didattico tutelando nello specifico i contenuti liberi del progetto GNU. Sintetizzando, stabilisce che ogni copia del materiale, anche se modificata, debba essere distribuita con la stessa licenza, funzionando quindi in maniera del tutto simile alla GPL. Tali copie possono essere vendute e, se riprodotte in gran quantità, devono essere rese disponibili anche in un formato che faciliti successive modifiche. L'applicazione della licenza ad un documento è estremamente semplice in quanto è sufficiente inserire oltre al testo della stessa un breve avviso

[61] http://www.gnu.org/philosophy/why-not-lgpl.html e http://internet.cybermesa.com/~berny/lgpl.html

[62] In pratica la LGPL rimane compatibile con la GPL e questa è un'importante caratteristica della licenza poiché diviene possibile prendere una parte di codice LGPL, incorporarlo in un programma GPL e distribuire comunque il tutto sotto la GPL.

disponibile su internet[63], dopo il titolo. Wikipedia[64], l'enciclopedia digitale libera è la più grande raccolta di documentazione sottoposta a questa licenza.

2.6 Open Source Definition (OSD)

Dalla breve storia del software libero riportata nel primo capitolo appare chiaro che l'Open Source è sorto come una "costola" del movimento del software libero per reagire allo stato di cose che si stava creando, evitare nel contempo l'ambiguità che il termine free poneva e ritagliarsi un'immagine più gradita al mercato. Il risultato di questa scissione è stata la stesura di un proprio manifesto, simile anche se modificato rispetto allo *GNU Manifesto* di Stallman da cui discende, cioè l'Open Source Definition, con l'obiettivo di ribadire non solo i principi di riferimento ma fornire anche dei criteri chiari con cui stabilire se una licenza e, per estensione, il relativo software siano o meno Open Source. La particolarità di questo documento è che lo stesso non è propriamente una licenza e non è inteso per avere valore legale. Affronta tuttavia una delle questioni più spinose relative alla realtà Open Source, cioè la licenza, a cui si aggiunge come specifica e non come sostituta. Prima di riportare il testo della OSD è opportuno riportare alcune considerazioni di Bruce Perens, autore della Guida Debian da cui poi è stata tratta la OSD, sull'utilità di questo strumento:

"*La Open Source Definition è una carta dei diritti dell'utente di computer. Definisce certi diritti che una licenza software deve garantire per poter essere certificata come Open Source. I produttori che non rendono Open Source i loro programmi trovano difficile competere con chi lo fa, dal momento che gli utenti imparano ad apprezzare quei diritti che avrebbero dovuto sempre essere loro* (...) *I volontari che hanno reso possibili prodotti come Linux ci sono, e le aziende possono cooperare, solo grazie ai diritti che vengono con l'Open Source. Il programmatore medio si sentirebbe stupido nel riversare tanto lavoro in un programma per vedere poi le sue migliorie vendute dal proprietario senza che lui ne riceva alcun ritorno. Quegli stessi programmatori contribuiscono volentieri all'Open Source perché esso assicura loro questi diritti* (Perens 1999)."

La seguente traduzione non ufficiale della Open Source Definition, suddivisa in nove punti, è tratta da Wikipedia[65]; per la versione ufficiale si rimanda al sito internet della Open Source Initiative[66]. Le sezioni non in corsivo sotto ciascun punto appaiono come note della definizione di Open Source (OSD) e non sono una parte della stessa; la loro utilità consiste nell'esplicitare le possibili fattispecie previste dalla definizione. Open Oource (sorgente aperto) non significa semplicemente accesso al codice sorgente. La distribuzione in termini di programmi Open Source deve soddisfare i seguenti criteri:

1. ***Ridistribuzione Libera*** *- Le licenze non potranno limitare alcuno dal vendere o donare i programmi come componenti di una distribuzione aggregata di software contenenti programmi di varia origine. La licenza non potrà richiedere royalties o altri pagamenti per tali vendite.* **Motivo**: imponendo la licenza a richiedere una redistribuzione libera, eliminiamo la tentazione di preferire un guadagno di breve periodo, generato dalla vendita, rispetto ai guadagni di lungo periodo. Se non lo facessimo ci sarebbe una forte pressione all'abbandono sui collaboratori.

2. ***Codice Sorgente -*** *Il programma deve includere il codice sorgente e deve permetterne la distribuzione così come per la forma compilata. Dove alcune forme di un prodotto non sono distribuite con codice sorgente ci deve essere un modo ben pubblicizzato di ottenerne il codice sorgente per niente più di una ragionevole riproduzione; preferibilmente, per via dei costi, scaricandolo da Internet gratis. Il codice sorgente deve essere la forma preferita in cui un programmatore modificherebbe il programma. Codice sorgente deliberatamente obnubilato non è permesso. Forme intermedie come l'output di un preprocessore o traduttore non sono permesse.* **Motivo**: si richiede un facile e rapido accesso al codice sorgente perché in assenza di questi requisiti non saremmo in grado di modificare i

63 http://it.wikipedia.org/wiki/GNU_Free_Documentation_License#Usare_la_GNU_FDL
64 http://it.wikipedia.org/wiki/Pagina_principale
65 http://it.wikipedia.org/wiki/Open_Source_Definition
66 http://mirror.openia.com/mirrors/www.opensource.org/docs/definition_plain.php

programmi e quindi migliorarli. Il nostro obiettivo è facilitare l'evoluzione dei progetti, pertanto richiediamo che anche lo sviluppo delle modifiche sia facilitato.

3. *Prodotti derivati -* *La licenza deve permettere modifiche e prodotti derivati e deve permettere loro di essere distribuiti sotto le stesse condizioni della licenza del software originale.* **Motivo**: la sola possibilità di leggere il codice non è abbastanza per sostenere un confronto paritario e una selezione evolutiva rapida. Per far avvenire questa rapida evoluzione, gli utenti e i programmatori devono poter sperimentare le modifiche e ridistribuirle.

4. *Integrità del codice sorgente dell'autore -* *La licenza potrà impedire il codice sorgente dall'essere ridistribuito in forma modificata solo se la licenza consentirà la distribuzione di pezze ("patch files") con il codice sorgente al fine di modificare il programma all'installazione. La licenza deve esplicitamente permettere la distribuzione del software costruito da un diverso codice sorgente. La licenza può richiedere che i lavori derivati abbiano un nome diverso o versione diversa dal software originale.* **Motivo**: incoraggiare il miglioramento è bene ma gli utenti hanno diritto di sapere chi è responsabile del software che stanno usando. Gli autori e i tecnici hanno diritto reciproco di sapere cosa è loro chiesto di supportare e di proteggere la loro reputazione. Perciò una licenza Open Source deve garantire che il codice sorgente sia prontamente disponibile ma può richiedere che esso sia distribuito come sorgente originale di base più le pezze. In questo modo, i cambiamenti "non ufficiali" possono essere disponibili ma prontamente distinti dal codice di base.

5. *Nessuna discriminazione contro persone o gruppi -* *La licenza non deve discriminare alcuna persona o gruppo di persone.* **Motivo**: per ottenere il massimo beneficio dal processo, la massima diversità di persone e gruppi deve avere eguale possibilità a contribuire ai codici sorgente. Quindi si proibisce qualsiasi licenza Open Source dall'escludere chiunque dal processo. Alcuni paesi, inclusi gli Stati Uniti, hanno restrizioni alle esportazioni per certi tipi di software. Una licenza conforme all'Open Source Definition può avvertire i licenziati di restrizioni applicabili e ricordare loro che sono obbligati a rispettare la legge; comunque, non potrà incorporare tali restrizioni essa stessa.

6. *Nessuna discriminazione verso i campi di applicazione -* *La licenza non deve impedire l'uso del programma in un qualsiasi ambito. Non potrà quindi impedirne l'uso nelle attività di un'impresa, o nell'ambito della ricerca scientifica.* **Motivo**: la prima intenzione di questa clausola è di proibire trappole nella licenza che impediscano all'Open Source di essere usato commercialmente. Vogliamo che gli utenti commerciali si uniscano alla comunità, non che se ne sentano esclusi.

7. *Distribuzione della licenza -* *I diritti allegati a un programma devono valere per tutti coloro cui il programma è ridistribuito senza necessità dell'emissione di una addizionale licenza da parte dei licenziatari.* **Motivo**: questa clausola intende proibire la chiusura del software per mezzi indiretti come una richiesta di un accordo di non diffusione.

8. *La Licenza non deve essere specifica di un prodotto -* *I diritti allegati al programma non devono dipendere dall'essere il programma parte di una particolare distribuzione di software. Se il programma è estratto da quella distribuzione e usato o distribuito all'interno dei termini delle licenze del programma, tutte le parti cui il programma è ridistribuito dovranno avere gli stessi diritti che sono garantiti nel caso della distribuzione di software originale.* **Motivo**: questa clausola impedisce ancora un'altra classe di licenze-trappola. Viene impedito che un prodotto identificato come Open Source possa essere distribuito con licenze non Open Source se esso viene incluso in una particolare distribuzione non Open Source. In altri termini il prodotto deve rimanere Open Source anche se viene distribuito separatamente dalla distribuzione software da cui proviene.

9. *La licenza non deve porre vincoli su altro software -* *La licenza non deve porre restrizioni su altro software che è distribuito insieme al software licenziato. Per esempio, la licenza non dovrà insistere che tutti gli altri programmi distribuiti sugli stessi supporti siano software Open Source.* **Motivo**: I distributori di software Open Source hanno il diritto di fare le loro scelte riguardo al loro software. Questa clausola riguarda solo l'aggregazione

cioè la semplice inclusione di due o più software sullo stesso supporto di memoria. La GPL è conforme a questa clausola; il software collegato con librerie accompagnate da licenza GPL eredita la stessa licenza solo se esso forma un unico prodotto.

Dopo aver esaminato i vari punti cardine che qualificano una licenza come Open Source, assicurando al relativo software il marchio omonimo, è utile ricorrere ad una tabella che riassuma alcune peculiarità delle licenze esaminate. I criteri presi in considerazione sono: la gratuità del software, la distribuzione libera, l'utilizzo libero, la disponibilità del codice sorgente e il diritto di modificarlo, la presenza del copyleft, la possibilità di combinare in unico software codice proprietario e codice libero (mix) e infine l'assenza di privilegi speciali (si rimanda al caso Netscape Communicator). Dalla stessa dovrebbe inoltre apparire chiaro che sebbene l'origine sia la stessa, ormai Free Software e Open Source sono due cose diverse: per l'analisi di questo aspetto si rimanda al paragrafo successivo.

Licenze Software	***Open Source***			***Non Open Source***				
	GPL	LGPL	MPL	Pubblico Dominio	NPL	Freeware	Shareware	Proprietaria
Gratuità del software	√	√	√	√	√	√	A tempo	No
Distribuzione libera	√	√	√	√	√	√	√	No
Utilizzo libero	√	√	√	√	√	No	No	No
Codice sorgente	√	√	√	√	√	No	No	No
Modificabile	√	√	√	√	√	No	No	No
Copyleft	√	√	√	No	√	No	No	No
Possibile mix	No	√	√	√	√	No	No	No
Assenza di privilegi	√	√	√	√	No	No	No	No

Tabella 2.1 Differenze fra licenze Open Source e licenze Non Open Source

2.7 Free Software o Open Source?

La nascita del movimento Open Source non fu, ovviamente, vista di buon occhio dai sostenitori più accaniti del free software, primo fra tutti Richard Stallman, che vedeva il loro duro lavoro di anni e anni ignorato. In pratica se da una parte costituì la svolta di cui il software libero aveva bisogno per essere seriamente preso in considerazione come una realtà dal relativo mercato, dall'altra determinò anche la definitiva scissione del movimento nelle sue due correnti. Ben presto i mezzi di comunicazione e persino parte degli addetti ai lavori presero l'abitudine di contrapporre fisicamente Raymond e Stallman come i rappresentati di due movimenti sostanzialmente contrapposti fra di loro. La cosa singolare è che in realtà esse condividevano e condividono tuttora molti principi di base relativi al software, alla sua condivisione e alle licenze da impiegare per la sua tutela: il punto di incontro è che il software debba essere rilasciato completo del proprio codice sorgente in modo da consentire eventuali modifiche o miglioramenti successivi. Entrambi sostengono che arricchire l'utente di questa possibilità possa avere un effetto positivo sullo sviluppo del software in via generale.

Lo scontro, ammesso che sia così rilevante (come spesso invece hanno cercato di far credere i mass media), va ricercato più che altro nei diversi obiettivi dei due movimenti. A distanza di anni ormai dalla nascita della disputa, è sempre più facile constatare come ci sia una gran confusione sulla questione e come spesso i due termini vengano usati impropriamente come sinonimi: è rilevabile anche nel materiale, usato per questo elaborato la presenza di alcune discordanze su questi concetti. Il movimento di Stallman (FSF) pone l'accento sulla libertà del software come problema sociale, come tema irrinunciabile, anche se questo dovesse significare perdere i favori delle aziende. L'idea base è che l'informazione debba essere condivisa e che il concetto di copyright sia intrinsecamente ingiusto, specie se relazionato al software, considerato un prodotto della conoscenza e alle tecnologie digitali che facilitano notevolmente la copia e la modifica. Un contratto che impedisce la condivisione

dell'informazione obbliga ad un comportamento ostile nei confronti delle altre persone e, in definitiva, impoverisce la società.

Inoltre le persone non possono essere davvero amiche se la legge vuole che conservino il segreto sulle informazioni. Le cose utili, ed il software rappresenta bene il concetto, sono come imprigionate dal copyright. Si crea una società ingiusta e sperequata tra chi ha denaro e chi non può accedere legittimamente alle medesime informazioni. Questa visione, di così ampio respiro, è la base dell'azione di Stallman per sostituire, nel campo del software, il copyright con il copyleft. Questa opzione di licenza annichilisce le ambizioni del monopolista anche senza proibire la vendita del software posto sotto copyleft perché impone che i progetti software che hanno tratto un vantaggio sul piano tecnico dalla Free Software Foundation non possano mai essere fatti oggetto di diritti sulla proprietà intellettuale[67].

L'Open Source, così come si può leggere sul sito Internet[68] dell'organizzazione (OSI), è in realtà più corretto definirlo come una strategia, un programma di marketing mirato a coinvolgere il mondo aziendale nello sviluppo e nella diffusione del software libero. Le motivazioni a capo dell'OSI sono prevalentemente di natura tecnica, ovvero riguardano l'efficienza dei processi di produzione e la qualità dei prodotti software[69]. Difatti l'aspetto su cui si fa leva è che la disponibilità del codice sorgente e lo sviluppo collaborativo aperto a tutti consentano la scrittura di software migliore e più economico rispetto a quello proprietario: l'elevato numero di sviluppatori che "ispezionano" il codice alla ricerca di errori (bug) in tempi molto rapidi grazie alla Rete migliora il livello qualitativo del software e riduce i costi. Le conseguenze pratiche si estendono quindi anche all'ambito economico puntando a favorire la competizione in un mercato spesso caratterizzato da forti monopoli e inefficienze. Per il movimento Open Source, il fatto che il software debba essere Open Source o meno è un problema pratico, non un problema etico: il software non libero è una soluzione non ottimale. Per il movimento del Software Libero, il software non libero è un problema sociale e il software libero è la soluzione[70].

Inoltre secondo la FSF quando si parla di software Open Source si corre il rischio di incorrere in errate interpretazioni, per esempio facendo ricadere in questa definizione qualsiasi software purché completo del proprio codice sorgente: è chiaro che le cose non stanno del tutto così ma in effetti alcune aziende hanno cercato e cercano tuttora di far passare del normale software proprietario come tale, per sfruttarne la notorietà e la simpatia dimostrata dagli utenti a proprio vantaggio. Il termine software libero sicuramente appare meno attraente per le aziende (sempre che al movimento ciò interessi) per cui si rinuncia ad una rapida diffusione dello stesso al di fuori di certi ambiti e si presta di più a fraintendimenti sulla gratuità ma se non altro si rifà ad un concetto difficilmente manipolabile. In sintesi va sottolineato come nonostante il disaccordo sugli obiettivi e l'esigenza di rimarcare, da parte di ciascuna corrente, il proprio ruolo avuto nella storia del software libero, sia comunque riscontrabile una vicinanza di idee sugli aspetti pratici che determina spesso lo sviluppo congiunto di alcuni progetti, per esempio Linux. I concetti appena esposti sono riassumibili attraverso le due tabelle seguenti.

---

67 Inguaggiato (1999)
68 http://www.opensource.org/advocacy/faq.php
69 Muffatto (2003)
70 Free Software Foundation (2002, 2004).

PRODOTTI GCC/C++ Linux Gnome Gimp	**OBIETTIVO:** Gnu's not Unix, creazione di un ambiente software completamente libero soprattutto per quel che riguarda i sistemi operativi **CREDO:**L'informazione e il software devono essere liberi

PRINCIPI GUIDA	**CONNOTAZIONI NEGATIVE**
● Non ci può essere compromesso tra libertà e software proprietario ● Libertà di: eseguire e modificare il programma, visionare e distribuire il codice sorgente ● Free speech not free beer ● Nessuno ha il diritto di fermare la condivisione ● Nessuno si può definire proprietario ● Uso della licenza Copyleft	● Non è sostenibile ● Adattabile solo alla comunità hacker e non alle aziende ● Prodotti difficili da usare ● Difficilmente viene generato profitto

Tabella 2.2 Aspetti principali della Free Software Foundation

PRODOTTI Linux Apache Mozilla Sendmail Perl	**OBIETTIVO:** Organizzazione e gestione della collaborazione tra industria e comunità. Ottenere software migliore, limitare i costi e favorire la competizione **CREDO:** Si può creare profitto anche donando il codice

PRINCIPI GUIDA
● La Open Source Definition e licenze più appetibili per il mondo commerciale ● Libertà di: eseguire e modificare il programma, visionare e distribuire il codice sorgente ● Rilasciare presto e spesso ● Documentare e tenere traccia del percorso di sviluppo del prodotto ● Trattare l'utenza come co-sviluppatori ● Condivisione dei rischi e dei costi ● Uso di supporti tecnologici che permettono trasparenza, riconoscimento, collaborazione e internazionalità dei contributi ● Flessibilità e adattabilità all'innovazione e ai nuovi mercati, no eccessiva pianificazione a priori ● Uso del peer review per assicurare affidabilità e qualità dei risultati

Tabella 2.3 Aspetti principali dell'Open Source Initiative
Fonte: Muffatto (2003)

2.8 Brevi considerazioni sulla brevettabilità del software

Da anni il software proprietario (prevalentemente statunitense) è costretto a fare i conti con una forte crisi in termini di innovazione tecnologica e una pressante insidia sia da parte della pirateria informatica che del software libero, di conseguenza le società del settore sono alla ricerca di strumenti di tutela sempre più efficaci, non solo nell'ambito tecnico ma soprattutto in quello legale: l'ultimo strumento su cui sembra abbiano fatto affidamento è il brevetto, in aggiunta all'oramai noto copyright. Negli Stati Uniti il brevetto nel software è già realtà (va comunque detto che il brevetto sul software sarebbe proibito come lo è stato finora in Europa ma la pratica legale, che in quel sistema ha un peso normativo rilevante, ha rivoltato questa norma) mentre in Europa è ancora oggetto di dibattito al Parlamento Europeo nonché al centro di ampie perplessità e critiche sollevate dal fronte contrario; quest'ultimo è costituito in primis dal movimento del software libero ma paradossalmente anche da molte piccole e medie aziende: l'applicazione del brevetto, al di là della sua pertinenza, costituisce un vero pericolo poiché determinerebbe notevoli ostacoli all'attività di programmazione stessa, per entrambi i modelli.
Nel presente paragrafo si cercherà di spiegare cosa sia il brevetto, si faranno alcune considerazioni sulla sua applicabilità al software o in via più generale alle idee astratte e si prenderanno in esame le motivazioni contrarie al

suo utilizzo, senza per questo dilungarsi troppo su un argomento attinente al tema dell'elaborato ma troppo complesso e vasto per essere trattato in questa sede. Il brevetto è lo strumento giuridico che conferisce all'autore di un'invenzione il monopolio temporaneo di sfruttamento dell'invenzione stessa, ossia il diritto di escludere terzi dall'attuare l'invenzione e dal trarne profitto: rappresenta pertanto un monopolio legale, seppur limitato territorialmente e temporalmente (in genere la sua durata è di vent'anni). Alla base del sistema brevettuale sussiste una forma di scambio: da una parte il titolare del brevetto riceve protezione per la propria invenzione dall'altra però è obbligato a svelare e a descrivere la stessa; offrendo protezione in cambio della divulgazione, si creano incentivi ad investire nella ricerca e nello sviluppo garantendo alla società l'acquisizione immediata delle idee innovative.

I motivi per cui si è introdotto il brevetto vanno ricercati proprio nella sua capacità di stimolare il progresso scientifico, nella possibilità attraverso il monopolio di coprire i costi di ricerca e gli investimenti che non necessariamente portano sempre ad un prodotto commerciabile e remunerativo, infine nella maggiore tutela assicurata alle invenzioni industriali (l'oggetto più comune di brevetto) rispetto al segreto industriale: quest'ultimo infatti, di fronte alle moderne tecniche di *reverse engineering*[71] che consentono la facile riproduzione delle stesse, si è rivelato ormai inadeguato. Al fine di stimolare lo sviluppo, un punto cardine della normativa brevettuale in tutte le legislazioni nazionali sta nella rivelazione dell'*insegnamento inventivo*[72], ossia si deve realizzare l'oggetto del brevetto perché lo stato dell'arte possa proseguire. Inteso in questo senso il brevetto si rivela utile in molti settori industriali ed è per questo che molte società informatiche ne richiedono l'introduzione.

Diversa però è la situazione per quanto riguarda le *idee*, comprendendo in questo termine le invenzioni astratte, per esempio un procedimento matematico o un programma per elaboratore; l'inapplicabilità del brevetto alle stesse è motivabile su tre fronti. Innanzitutto la convenzione europea dei brevetti vieta la brevettabilità dei metodi commerciali, delle teorie matematiche, dei programmi per elaboratore (software) e altre categorie di invenzioni astratte[73], divieto presente anche nella normativa italiana[74]. Essendo il software assimilabile ad un prodotto della logica e del pensiero, ovvero a un qualcosa di astratto come appunto un procedimento matematico (si pensi al teorema di Pitagora), esso non dovrebbe essere brevettabile. In secondo luogo va ricordato che i programmi per elaboratore ricadono già sotto la tutela del diritto d'autore come ratificato da tutti i maggiori trattati internazionali[75], nonché del segreto industriale, quindi su queste basi legali qualsiasi argomentazione sulla necessità di proteggere ulteriormente il software si rivela infondata.

Oltretutto il brevetto rappresenterebbe un terzo livello di protezione, in palese contraddizione però con il segreto industriale: quest'ultimo tutela l'invenzione impedendo che venga rivelata mentre il brevetto concede la protezione solo se la stessa viene rivelata, così come affermato dal principio di rivelazione dell'insegnamento inventivo. Nel caso specifico del software va fatto notare che il brevetto, impedendo la rivelazione del codice sorgente, non consentirebbe comunque alcuna rivelazione delle conoscenze contenute in esso inoltre la sua durata temporale non si adatta ad un settore dove il ciclo di vita del prodotto è una questione di pochi anni, se non di mesi. Per quanto riguarda l'aspetto economico, la necessità di coprire gli investimenti iniziali non ha alcun riscontro nel campo delle idee astratte poiché non esistono costi di ricerca concreti a fronte dello sviluppo di idee; di conseguenza non è necessario concedere l'esclusiva sull'utilizzo della presunta invenzione perché l'idea viene realizzata in ogni caso. Se anche all'idea dovesse aggiungersi il lavoro, per esempio la scrittura di un codice sorgente, gli investimenti sostenuti per realizzarlo sarebbero già tutelati dal copyright.

Nonostante la breve riflessione fatta finora, il software è già da tempo oggetto di brevetto, in genere ad opera di grandi società americane che lo sfruttano, com'è ormai chiaro, in modo improprio al solo scopo di riscuotere

---

[71] Non esiste una traduzione all'espressione, comunque essa sta ad indicare il processo inverso a quello che genera un prodotto, un risultato. In questo caso si parte dal risultato appunto per determinarne il processo di produzione, che può consistere per esempio in una formula o un'invenzione.

[72] Articolo 83 della European Patent Convention http://www.european-patent-office.org/legal/epc/e/ar83.html

[73] Articolo 52 dela European Patent Convention http://www.european-patent-office.org/legal/epc/e/ar52.html

[74] Decreto 1127/1939 e successive modificazioni, articolo 12, comma 2: «Non sono considerate come invenzioni ai sensi del precedente comma in particolare: [...] b) i piani, i principi ed i metodi per attività intellettuali, per gioco o per attività commerciali e i programmi per elaboratori»

[75] Convenzione di Berna http://www.law.cornell.edu/treaties/berne/overview.html, Articolo 4 del Trattato sul Copyright WIPO http://www.wipo.org/eng/diplconf/distrib/94dc.htm

le licenze d'uso. La spinta a brevettare non giunge dai programmatori, che anzi vengono ulteriormente penalizzati, ma dai manager e dagli avvocati: i primi vedono nel brevetto il mezzo con cui trionfare sui propri concorrenti senza doversi prendere la briga di sviluppare prodotti di qualità superiore o abbassare i prezzi, i secondi ne intuiscono il potenziale con cui espandere i propri affari. La cosa preoccupante è che mentre la normativa sul diritto d'autore tutela qualsiasi forma di espressione di un programma per elaboratore avente caratteri di originalità, senza offrire però alcun tipo di monopolio quanto alle idee che sono alla base del programma stesso, il brevetto preclude ad ogni soggetto diverso dal titolare la possibilità di esprimere le stesse idee e i medesimi principi con altri programmi, anche quando il codice sorgente del nuovo software sia originale. Gli effetti pratici, di questo abuso vengono chiaramente espressi ancora una volta da Stallman:

"*Adesso si stanno registrando brevetti software ad una velocità allarmante, stando ad alcuni conteggi, ne vengono emessi più di un migliaio ogni anno. Sfortunatamente, la maggior parte delle invenzioni brevettate hanno in sé pressappoco tanta bravura e originalità quanta se ne trova nella ricetta del riso bollito, semplice di per sé ma parte essenziale di piatti complicati. Molti riguardano algoritmi e tecniche davvero elementari e specifiche che vengono usate in un'ampia varietà di programmi. Frequentemente le "invenzioni" menzionate in una applicazione brevettata sono già in uso da altri programmatori al momento in cui viene presentata la domanda per il brevetto. Quando l'Ufficio Brevetti concede un brevetto su un algoritmo o una tecnica, sta dicendo ai programmatori che essi non devono usare un particolare metodo per risolvere un problema senza il permesso del proprietario dell'idea.*

Per i programmatori, brevettare un algoritmo o una tecnica è come brevettare una serie di note musicali o un giro di accordi, forzando di conseguenza i compositori ad acquistare una "licenza di sequenza musicale" (...) *In pratica una volta al mese, le reti di computer della nazione sono scosse dalla notizia di un altro brevetto basato su un concetto fondamentale che è largamente usato. Sebbene l'Ufficio Brevetti non sia preposto a registrare le idee, questo è essenzialmente quello che sta facendo con i brevetti software, facendo a pezzi il domino intellettuale della scienza informatica e consegnandone in pratica i piccoli pezzi ad ogni compagnia che presenta una domanda di brevetto. E questa pratica sta devastando l'industria software americana* (Garfinkel, 1991)".

Da questo estratto appare evidente che non essendo sempre possibile riscrivere una procedura o un algoritmo brevettato, si rinuncia alla implementazione della relativa funzione all'interno del programma, ostacolando così l'adozione di nuove tecnologie e il progresso, contrariamente a quanto sostenuto dai fautori del brevetto. Le piccole società e i programmatori indipendenti, spesso rientranti nel software libero e Open Source, che rappresentano il motore dell'innovazione e dello sviluppo non hanno e non avranno mai la forza economica per pagare le licenze d'uso, tanto meno per affrontare cause civili di milioni di dollari: si ritrovano così alla mercé delle grandi società che senza alcuno sforzo, consolidano la loro posizione a scapito dell'effettiva concorrenza.

Fra i casi più eclatanti vanno citati il tentativo (per il momento non riuscito) da parte di British Telecom, di brevettare il "link" cioè il classico collegamento fra pagine web, e i brevetti già concessi sul formato grafico GIF, sull'acquisto in Rete con un singolo click del mouse[76] (Amazon) oppure sulla barra di avanzamento[77] (la classica barra grafica che indica lo stato di avanzamento di un processo, come il trasferimento dei file): quale società è in grado di sostituire funzioni così banali ma anche così comuni per qualsiasi utente? Alcuni studi economici[78] dimostrano quanto appena detto, in particolare come l'estensione del regime di brevettabilità sia nocivo per lo sviluppo del mercato del software e della relativa tecnologia.

L'avversione al brevetto quindi non è solo una questione di simpatia verso il software libero. Probabilmente non è un caso se lo sviluppo e l'introduzione di nuove tecniche informatiche siano ormai un'esclusiva delle piccole e diffuse società europee, non oppresse, almeno per il momento, dalla scure del brevetto. Secondo tali studi infatti, la sua applicazione anche in Europa (situazione ormai, quasi reale) non aiuterà il mercato e la concorrenza ma stroncherà le esili possibilità di dar vita ad un polo produttivo autonomo e libero dalle imposizioni americane.

L'industria statunitense, che si è sempre posta come punto di riferimento a livello internazionale e riunisce i

[76] http://www.apogeonline.com/webzine/2003/09/10/01/200309100101

[77] http://l2.espacenet.com/espacenet/viewer?PN=EP0394160&CY=ep&LG=en&DB=EPD

[78] http://swpat.ffii.org/vreji/minra/sisku.en.html e http://www.ipmall.fplc.edu/hosted_resources/jepson/unit1/aneconom.htm

"giganti" del settore, ha attraversato negli ultimi anni una forte crisi in termini di ricerca e innovazione mentre le società europee e quelle asiatiche, che assumono dimensioni limitate e detengono una quota minima del mercato, hanno dimostrato queste capacità, spesso attraverso progetti Open Source. Sempre secondo questi studi, il motivo di questo avvicendamento andrebbe cercato nei danni che il brevetto stesso ha prodotto Oltreoceano: le piccole software house, non potendo sopportare l'onere di una rigorosa indagine brevettuale su ogni minimo aspetto del proprio progetto (per vedere se per caso qualcuno lo ha già brevettato), sono fallite e le grandi società non si sono più dovute preoccupate di produrre e vendere software nuovo, o quanto meno migliore, ma semplicemente di incassare i diritti legali.

In Europa invece la scarsa diffusione del brevetto ha portato ad uno sviluppo del settore che solo in questi ultimi anni sembrava aver raggiunto una certa indipendenza ed una buona produttività. L'opinione più diffusa è che le forti pressioni esercitate dalle società statunitensi per l'introduzione del brevetto anche in Europa non siano motivate dalla volontà di tutelare chi sviluppa il software ma dalla intenzione di mantenere il controllo sul mercato internazionale e quindi imporre le proprie strategie e i propri prodotti, evitando la sana competizione.

CAPITOLO 3
MODELLO OPEN SOURCE: ASPETTI TECNICI E ORGANIZZATIVI

3.1 Il Bazar

Finora si sono visti gli aspetti storici e legali dell'Open Source, tuttavia non sono stati affrontati quelli di natura organizzativa e tecnica, per esempio non si è visto come nasce un progetto Open Source, quali sono le conseguenze tecniche della disponibilità del codice sorgente, come avviene lo sviluppo nel Bazar. Già, il Bazar, questa è la denominazione che Eric Raymond diede al metodo di sviluppo adottato dalla comunità del software libero (e Open Source) per contraddistinguerlo da quello adottato dalle imprese. La comunità si presentava infatti come un immenso "calderone ribollente di idee"[79], continuamente alimentato dal contributo di soggetti, di estrazione sociale e preparazione tecnica profondamente diverse, che interagivano tra di loro spinti da interessi e obiettivi diversi. Negli anni Raymond aveva contribuito in prima persona a vari progetti Open Source, ciò nonostante, era convinto che esistesse un punto critico di complessità al di sopra del quale si rendesse necessario un approccio centralizzato e a priori. In altre parole era convinto che certi software, come un sistema operativo, andassero realizzati come le cattedrali (da qui la denominazione dell'altro metodo di sviluppo), cioè attentamente lavorati a mano da singoli geni o piccole bande di maghi che lavoravano in uno splendido isolamento[80].

Lo sviluppo avrebbe quindi dovuto essere portato avanti da pochi ed esigui gruppi di programmatori, attraverso un dettagliato piano che copriva ogni singolo aspetto, dalle finalità iniziali e le integrazioni, alle verifiche; di conseguenza i rilasci del codice sorgente sarebbero stati pochi e il feedback esterno sarebbe stato ricercato solamente durante i cicli di verifica alpha e beta[81]. Raymond rimase alquanto stupito quando si rese conto che il metodo adottato da Torvalds nel progetto Linux, per quanto assomigliasse ad un grande e confusionario bazar pullulante di progetti e approcci tra loro diversi (efficacemente simbolizzati dai siti contenenti l'archivio di Linux dove apparivano materiali prodotti da chiunque) e dal quale soltanto una serie di miracoli avrebbe potuto far emergere un sistema stabile e coerente, funzionava e pure bene.

Egli stesso ebbe modo in prima persona di accorgersi della validità di questo modello quando partecipò al progetto Fetchmail, al punto che proprio da questa esperienza trasse l'ispirazione per CatB[82]. Non essendo possibile definire in modo univoco gli innumerevoli progetti Open Source a causa delle profonde differenze presenti in essi, ad esempio nel numero e nel tipo di utenti coinvolti o nelle modalità organizzative, Raymond compì un'opera di sintesi enfatizzando solo gli aspetti più comuni: nelle prossime righe verrà sinteticamente descritta la storia del progetto Fetchmail ed essi verranno segnalati in corsivo. In genere *ogni progetto di realizzazione di un software nasce dalla frenesia personale di uno sviluppatore* cioè da una sua necessità. Ciò potrebbe sembrare ovvio ma troppo spesso gli sviluppatori lavorano su programmi di cui non hanno alcun bisogno e che non apprezzano (ciò non accade con l'Open Source, il che spiega l'alta qualità del software originato dalla comunità).

Nel caso specifico, Fetchmail nacque perché Raymond aveva bisogno di un programma che trasferisse la sua posta elettronica, dai server remoti in cui veniva salvata, al computer di casa. All'inizio egli si trovò a dover decidere se scrivere da zero il codice sorgente o riutilizzare quello altrui come base di partenza, risparmiando così del tempo prezioso. Seguendo l'esempio di Torvalds[83] decise di partire da uno dei tanti progetti liberi già esistenti (PopClient) scegliendo, in accordo con l'autore originario, di apportare alcune correzioni e di integrare le funzionalità di cui aveva bisogno. Secondo Raymond *i bravi programmatori sanno cosa scrivere ma i migliori sanno cosa*

---

79 Raymond (1998a)

80 Raymond (1998a)

81 Quando si realizza un software, sia libero che proprietario, prima di rilasciarlo si eseguono una serie di controlli e verifiche per stabilire la qualità del codice. Inizialmente si esegue il *testing* cioè la ricerca di errori (bugs) nel codice attraverso la sua esecuzione poi si passa al *debugging* cioè alla loro correzione in modo da garantire che il funzionamento del software non possa ad esempio compromettere quello del sistema o la sicurezza di quest'ultimo. In genere la fase di testing si divide a sua volta nelle due fasi *alpha* e *beta:* in entrambe il software viene provato prima del rilascio da utenti reali, tuttavia nella fase alpha l'ambiente di prova è l'ambiente di produzione, nella beta, pur con delle limitazioni, l'ambiente reale. Ovviamente il software libero in genere non limita l'accesso alle sole versioni alpha e beta e ad un certo numero o tipo di utenti, anzi, ne incoraggia l'analisi pur sottolineando i possibili rischi.

82 CatB è l'abbreviazione di "The Cathedral and the Bazaar".

83 Per il kernel Linux Torvalds analizzò il codice di Minix che era un clone di Unix.

riscrivere e riusare[84], questo per ricordare come spesso si ottenga il meglio non per le energie impiegate ma per il risultato raggiunto e come sia più facile fare ciò iniziando da una buona soluzione parziale piuttosto che dal nulla assoluto. D'altronde la disponibilità dei codici sorgente ha sempre favorito il loro riutilizzo, riducendo in parte il lavoro di creazione ad un mero lavoro di ricerca e modifica dei vari terabyte[85] di codice disponibili nella comunità.

Resosi presto conto che l'autore originale, Carl Harris, aveva ormai perso interesse nel programma, egli non solo continuò ad apportare modifiche ma, sempre in accordo con quest'ultimo, assunse l'intera gestione del progetto. In una cultura del software che incoraggia la condivisione del codice, non si trattava d'altro che della naturale evoluzione di un progetto dato che *quando si perde interesse in un programma, l'ultimo dovere è di passarlo a un successore competente*. Raymond non ereditò solamente il progetto ma con esso anche la sua comunità di utenti; quest'ultima però, a causa dell'evidente disinteresse di Harris, si era atrofizzata e non forniva più buoni spunti. Per stimolare una ripresa di interesse egli decise di ispirarsi nuovamente a Linux, cercando di replicare quelli che riteneva i punti di forza di quel modello di sviluppo. Proprio la presenza di una comunità viva[86] è uno dei cardini di qualsiasi progetto Open Source in quanto gli utenti abbandonano il ruolo passivo di semplici utilizzatori per assumere quello di co-sviluppatori.

La disponibilità del codice rende infatti gli utenti dei potenziali hacker, peraltro molto efficaci: con un po' di incoraggiamento, ognuno di essi è in grado di rilevare i difetti, suggerire le soluzioni, contribuire al miglioramento del codice (debugging) in maniera impensabile per una persona sola. Si può quindi affermare che "*trattare gli utenti come co-sviluppatori è la strada migliore per ottenere rapidi miglioramenti del codice e un debugging efficace*". Dove lo sviluppo "a cattedrale" aveva permesso un'evoluzione limitata del software, lo sfruttamento degli utenti alla Linux, permise di riscrivere tre o quattro volte il codice prima di raggiungere una forma stabile e definitiva. La sola collaborazione degli utenti non era però sufficiente, occorreva anche *distribuire presto e spesso* il codice creato. Prima di Linux molti sviluppatori, aderendo al vecchio modello, consideravano negativa questa pratica poiché le versioni iniziali sono piene di bug, quasi per definizione, e non pareva il caso di far spazientire inutilmente gli utenti: se l'obiettivo finale era quello di far vedere meno errori possibili agli utenti (a prescindere dall'effettiva qualità del codice) allora conveniva distribuire una nuova release a distanza di mesi e lavorare duramente sul debugging tra una release e l'altra.

Ma in questo caso l'obiettivo era quello di scrivere del codice di qualità e il modo migliore per farlo era di velocizzare lo sviluppo attraverso continui rilasci delle versioni modificate, a distanza di pochi giorni se non addirittura di ore; in questo modo gli utenti erano costantemente stimolati dalla propria soddisfazione per aver preso parte all'azione e ricompensati dai frequenti miglioramenti costanti ottenuti nel loro lavoro. Si doveva in pratica cercare di massimizzare il numero di "ore per utente" destinate allo sviluppo e al debugging in modo da attuare la "legge di Linus", vista nel primo capitolo, per cui il numero di utenti che ispezionano il codice influisce positivamente sulla probabilità che qualcuno scopra un bug e qualcun altro lo corregga: tanto maggiore è questo numero, tanto è più facile che si giunga ad una rapida correzione e questo è dimostrabile anche in progetti complessi come il kernel Linux.

Per far crescere la base di utenti di Fetchmail, Raymond seguì alcune delle regole che, secondo la sua opinione, avevano costituito la fortuna del progetto Linux. Innanzitutto creò una *Beta List* contenente i nominativi non solo di coloro che testavano le versioni beta ma di chiunque lo contattasse per avere informazioni riguardo il programma (nel giro di pochi mesi avrebbe raggiunto i 300 contatti). Il passo successivo fu sollecitare la partecipazione attraverso un'apposita *mailing list*[87] dove gli sviluppatori potessero confrontare le proprie idee e dove venissero tempestivamente annunciate le nuove versioni[88]. La comunicazione non sarebbe però avvenuta a senso

---

84 Raymond (1998a).

85 Il terabyte è un'unità di misura utilizzata in ambito informatico. Colloquialmente quando si parla di terabyte si intende una capacità di memoria di mille miliardi di byte. In questo caso è stato usato come eufemismo per indicare l'enorme numero di progetti Open Source esistenti.

86 Qualsiasi progetto Open Source deve raggiungere una dimensione di massa critica in termini di utenti per poter svilpparsi secondo il modello a Bazar.

87 Una mailing-list (letteralmente, *lista per corrispondenza*, dall'inglese) è un servizio Internet che permette ad ogni iscritto di inviare e-mail su un determinato argomento a tutti gli altri iscritti e di ricevere quelle inviate da ogni altro iscritto.

88 Ogni progetto Open Source è caratterizzato da un'evoluzione continua del software che lo mantiene sempre all'avanguardia tuttavia ciò determina anche una certa instabilità e inaffidabilità dei codici sorgente di conseguenza si deve ricorrere ad un compromesso fra l'esplorazione delle nuove soluzioni e lo sfruttamento di quelle esistenti ricorrendo a due percorsi di

unico, Raymond puntava infatti a ricevere preziosi consigli e idee sullo sviluppo dei pacchetti e a gratificare il loro operato menzionandone i contributi al progetto[89]. A distanza di pochi mesi il codice raggiunse una stabilità tale da non richiedere più grossi interventi ma una semplice manutenzione, tant'è che molti utenti gli chiesero di essere rimossi perché il programma funzionava così bene che non c'era più alcun motivo di seguire il traffico della lista.

Ciò fece comprendere a Raymond che anche un progetto Open Source, portato avanti con lo sviluppo "a Bazar", aveva un suo ciclo vitale ben definito ed una sua maturità. Inoltre proprio l'esperienza fatta con il progetto Fetchmail gli consentì di trarre alcuni insegnamenti su quelle che possono essere considerate le premesse (intendendo sia le qualifiche del leader del progetto sia lo stato del codice al momento della diffusione pubblica e della nascita della possibile comunità di co-sviluppatori) necessarie a sviluppare, secondo il modello a Bazar, il progetto di un software. Innanzitutto esso richiede la presenza di un codice sorgente iniziale poiché la nascente comunità di sviluppatori può sì testare e cercare i bug ma deve avere un codice di partenza su cui lavorare. Esso può anche essere crudo, pieno di bug, incompleto e scarsamente documentato ma non deve però mancare di convincere i potenziali co-sviluppatori che possa evolversi in qualcosa di veramente ben fatto in futuro.

In secondo luogo, per quanto possa avere un certo peso nel progetto, non è essenziale un eccezionale talento nel design da parte del coordinatore, semmai è fondamentale che sappia riconoscere le buone idee progettuali degli altri sviluppatori. D'altronde, nella comunità la reputazione ha un importanza tale da spingere i soggetti a non avventurarsi in un progetto se non ne hanno le competenze ma a cercare l'aiuto altrui. Occorre infine che il coordinatore sia in grado di comunicare efficacemente con gli altri e di lavorare in gruppo: non è infatti sufficiente la sola esperienza tecnica per attirare la simpatia degli utenti e gratificarne l'operato. Non è certo una coincidenza che Linus Torvalds sia un tipo simpatico, capace di piacere alla gente e di farsi aiutare.

3.2 I protagonisti

"*Ogni gruppo di persone che sviluppa software e rende disponibili i propri risultati al pubblico sotto una licenza open source, costituisce un progetto open source* (Evers 2000)". Questa è la definizione più ricorrente di progetto Open Source ma probabilmente è anche la più accettabile viste le diversità che possono esistere fra i molteplici progetti esistenti. In essa non viene menzionato un particolare gruppo di agenti poiché anche in questo ambito i progetti possono distinguersi, tuttavia è abbastanza semplice indicare le categorie di soggetti che di solito partecipano o contribuiscono all'attività di sviluppo, ossia quali sono i "protagonisti":

● **Istituzioni**: comprendono governi e pubbliche amministrazioni interessate all'uso (e quindi allo sviluppo) del software Open Source. L'informatica riveste ormai un ruolo chiave anche nelle strutture organizzative e l'Open Source può essere lo strumento adatto per affrancarsi dalle società, spesso straniere, che forniscono software proprietario, per tutelarsi dal punto di vista della sicurezza e della flessibilità delle applicazioni nonché per ridurre l'incidenza dei costi, alla quale il settore pubblico risulta particolarmente sensibile. In genere le strutture pubbliche più coinvolte sono le Università che producono molto software sia per uso interno che di ricerca; in questo senso occorre ricordare come il rilascio del software sotto licenze Open Source ben si adatti alla tradizione scientifica di rendere pubblici i risultati delle proprie ricerche. Non mancano comunque esempi di sponsorizzazioni a progetti Open Source come nel caso del progetto KDE, finanziato in parte anche dalla pubblica amministrazione tedesca.

● **Imprese**: Le imprese interagiscono con il mondo Open Source in quanto utilizzano il software prodotto e contribuiscono più o meno direttamente allo sviluppo. Alcune si limitano a finanziare alcuni progetti, altre partecipano allo sviluppo congiunto con i programmatori indipendenti della comunità per cui non è raro che dipendenti delle stesse

sviluppo. Il primo è quello *stabile* che assicura un codice stabile e compatibile con le future versioni ma un po' più vecchio (per modo di dire), il secondo è quello *instabile* che assicura il codice più recente e tutte le ultime funzionalità ma anche una maggiore probabilità di incorrere in bug. Anche Raymond adottò questa metodologia di sviluppo.

[89] Il contributo che un utente può portare alla comunità può essere di vario genere e consistere anche solo nella semplice scoperta di un bug, tuttavia occorre precisare che a certi livelli di conoscenza tecnica, la comunità stessa è nota per avere pochissima tolleranza nei confronti di contributi di poco conto. In pratica chi non fornisce contributi sostanziosi prima o poi verrà allontanato: per quanto nessuno lo abbia mai ammesso, in termini aziendali, questo atteggiamento è definibile come gestione dei talenti interni.

collaborino alla stesura di nuovo codice con la stessa. Altre ancora basano la loro attività interamente su di esso, è il caso delle società che curano e forniscono le varie distribuzioni Linux esistenti come alternativa ai sistemi operativi proprietari[90].

● **Organizzazioni No-profit**: E' il caso della Free Software Foundation (FSF), della Open Source Initiative (OSI) o di altre organizzazioni che si occupano di promuovere e sviluppare singoli progetti come KDE e GNOME[91].

● **Comunità**, intesa come unione di tre gruppi, l'utenza, gli sviluppatori indipendenti e i gruppo dei leader.

Utenza. Una buona parte della comunità è costituita da coloro che usano il software Open Source ma non possiedono né la capacità né il tempo e le risorse per partecipare direttamente allo sviluppo. Questo non significa che non contribuiscano, seppur indirettamente, allo sviluppo; la loro presenza è infatti fondamentale poiché permette di provare, quasi contemporaneamente, il software in migliaia di condizioni diverse (hardware e software) che rappresentano il migliore banco di prova per un codice sorgente. Il frequente rilascio dello stesso attraverso Internet consente agli sviluppatori di ottenere un feedback tempestivo, a volte anche a distanza di poche ore dal rilascio, sulla presenza di eventuali bug nel codice riducendo i tempi totali di sviluppo (non è un caso se il software proprietario una volta rilasciato presenti tutta una serie di errori, dopotutto la base utilizzata per il testing non è paragonabile a quella della comunità Open Source). Il contributo spesso avviene anche in termini di idee, consigli e design.

Sviluppatori indipendenti. Sono i programmatori che partecipano volontariamente e attivamente allo sviluppo del software attraverso la scrittura di nuovo codice, la correzione degli errori, la proposta e l'introduzione di nuove funzionalità. Possiedono delle capacità e delle conoscenze tecniche di un certo livello anche se non necessariamente svolgono il lavoro di programmatore. Spesso infatti si tratta di studenti, professionisti o semplici appassionati, di estrazione sociale, provenienza geografica[92] ed esigenze professionali molto diverse che per pura passione si cimentano in questa attività (in questo caso è corretto parlare di hacker). Un aspetto che merita di essere sottolineato è quello delle motivazioni che li spingono a collaborare, dedicando risorse e tempo, senza per questo avere la certezza di una diretta ricompensa economica. Esso verrà trattato nei prossimi paragrafi.

Leader: Ovviamente anche questo sottogruppo è composto da programmatori, da hacker, tuttavia questo gruppo rappresenta una sorta di élite selezionata dalla comunità su basi meritocratiche e alla quale vengono riconosciute l'autorità e la responsabilità di gestire il processo di sviluppo; ne fanno parte oltre ai soggetti che hanno dato via al progetto in genere coloro che da più anni collaborano allo sviluppo. In generale essi si occupano della progettazione, della gestione e del controllo dei processi di sviluppo, coordinano in pratica la comunità di sviluppo e i vari contributi anche se questo spesso significa dover trascurare l'attività di programmazione. Se a prima vista si potrebbe pensare che una comunità formata da soggetti così diversi (sotto vari punti di vista) sia proprio come un Bazar, cioè caotica e disorganizzata, in realtà proprio coloro che fanno parte di questa categoria assumono dei ruoli ben precisi e tipici di qualsiasi progetto Open Source; occorre infatti distinguere la figura del *Project Manager* da quelle del *Mantainer* e dell'*Amministratore.*

Il Project Manager è colui che più si avvicina alla figura del capo, infatti coordina i vari gruppi di lavoro nelle varie fasi dello sviluppo, stila le scadenze del progetto (anche se in genere il software Open Source non è sottoposto a termini di rilascio tassativi come nel caso del software proprietario) e prende le decisioni di maggiore importanza. Non sempre la figura del project manager esiste, o meglio, spesso accade che essa si sovrapponga con quella del *Mantainer* (ad esempio Raymond per Fetchmail o Torvalds per Linux). Quest'ultimo è di solito colui che ha concepito l'idea iniziale alla base del software nonché il responsabile del progetto, tuttavia può accadere che vi siano più mantainer di conseguenza ognuno si occupa di una determinata componente del progetto, quale un'applicazione o una particolare funzione e la sua

---

90 Si rimanda al primo capitolo.

91 L'interazione dell'utente con un computer può avvenire in modalità testuale o grafica. Nel primo caso occorre che l'utente abbia una conoscenza molto profonda del software e del sistema su cui lavora nonché una notevole capacità di manipolazione dei comandi e delle regole sintattiche tipiche dei linguaggi di programmazione. L'uso di un'interfaccia grafica, come quella introdotta da Apple (e poi adottata anche da Microsoft in Windows) semplifica di molto la gestione di un software riducendo spesso l'interazione all'uso del mouse. Nel mondo Linux i due principali progetti per la creazione di un ambiente grafico sono appunto KDE e GNOME.

92 La maggiore concentrazione di sviluppatori si ha in Europa e negli Stati Uniti. A livello mondiale l'Italia detiene il quarto posto per il numero di sviluppatori.

importanza è proporzionata a quella del componente seguito. Al mantainer spetta il compito di rilasciare il software inoltre riceve, valuta e applica le correzioni al codice sorgente. Ovviamente incide sull'assegnazione dei singoli incarichi anche se, come si è visto nel precedente paragrafo, ciò dipende in larga misura dai riconoscimenti e dai meriti che vengono attribuiti ai soggetti scelti dal resto della comunità; è quindi scontato che il mantainer debba possedere una carisma capace di attrarre i contribuenti e di compattarli, evitando il possibile *forking*[93] e quindi lo spreco di risorse.

L'ultima figura è quello dell'amministratore che in parte assomiglia a quella del mantainer, occupandosi però ad un livello inferiore della gestione delle risorse: i suoi compiti sono la conservazione, intesa sia come archiviazione e catalogazione, sia come protezione da accessi errati o indiscreti, e la diffusione all'interno del processo in modo che ogni oggetto, prodotto effettivo dello sviluppo o semplice informazione, sia effettivamente disponibile a chi ne ha bisogno. In pratica mette in condizione i singoli programmatori di ciascun sotto-progetto di lavorare assegnando a loro gli strumenti sia hardware che software necessari.

● **Team Virtuale**: L'esistenza di un'infrastruttura come Internet permette alla comunità e agli altri soggetti visti finora di "incontrarsi" in rete, dando così vita ad un gruppo di lavoro virtuale, caratterizzato da continue uscite e ingressi dei singoli agenti che compongono i gruppi citati precedentemente. In genere i "punti di incontro" sono rappresentati dalle mailing list, già citate per la vicenda Fetchmail, e dai newsgroup cioè i gruppi di discussione o dai siti Web.

Figura 3.1 Struttura di una comunità Open Source
Fonte: Muffatto (2003)

3.3 Cultura hacker ed economia del dono: motivazioni sociali e tecniche

[93] Il termine forking indica il possibile "biforcamento" di un progetto in due o più versioni concorrenti. Il forking può essere in certi casi positivo, se serve a creare una proficua competizione tra due approcci e prodotti radicalmente diversi, generalmente tuttavia esso si rivela negativo, ed è pertanto visto, nel mondo Open Source, come una delle minacce più gravi. Le forze economiche che spingono contro di esso sono: la perdita di economie di scala dovute alla creazione di comunità più piccole, addirittura non in grado di raggiungere da sole una massa critica; le difficoltà generate in segmenti adiacenti dal dover scegliere quale versione adottare o nel dover prevedere come integrare due versioni; infine il forking pesa sull'immagine, sulla credibilità di un progetto e della sua comunità. Le motivazioni che possono portare un progetto a dividersi sono essenzialmente riconducibili al prevalere di interessi particolari e privati sul progetto da parte di un gruppo di programmatori o di una azienda oltre che a un conflitto insanabile all'interno della comunità di sviluppo.

Da quanto visto risulta evidente che una buona parte dei soggetti coinvolti in un progetto partecipano ad un'attività non necessariamente remunerativa e che, il più delle volte, non si traduce affatto in un guadagno. Ma cosa spinge questi soggetti e gli altri ad impegnare le proprie risorse? Si è visto che la base di partenza è in pratica la volontà di uno o più individui di risolvere un certo problema o adattare un certo software alle proprie esigenze, tuttavia ciò non basta a giustificare le dimensioni, raggiunte in termini sia di utenti che di codice scritto, da progetti come Linux. Negli anni sono stati compiuti degli studi sulla questione dai quali risulta che le motivazioni sono fondamentalmente tre, cioè non solo tecniche ma anche sociali e perché no pure economiche[94]. La comprensione risulta facilitata se si compie una suddivisione dei partecipanti in agenti individuali e organizzazioni, comprendendo in questo secondo gruppo le singole imprese e le istituzioni.

Da un punto di vista sociale occorre sottolineare come l'agente individuale contribuisca allo sviluppo del codice innanzitutto per un senso di appartenenza ad una comunità libera da interessi economici e priva di rapporti gerarchici nella quale aiuto reciproco e solidarietà sono valori indiscutibili. L'origine di questi valori va sicuramente cercata nella *cultura hacker* dalla quale storicamente è sorto il movimento del software libero e della quale va ricordato il diritto alla cooperazione per cui, in un rapporto di scambio fra pari, ogni individuo ha la possibilità di accrescere la conoscenza collettiva oltre che la propria cultura. In essa esistono poi altri "usi e costumi", quali una spiccata attenzione alla reputazione e un particolare senso di proprietà che condizionano in maniera incisiva la produzione di software libero e ne garantiscono la sopravvivenza e il successo.

Si potrebbe pensare che a causa delle forti libertà (soprattutto di copia e di ridistribuzione) garantite dalle licenze libere sia frequente il forking dei progetti, in realtà esso è piuttosto raro poiché all'interno della comunità vigono delle tacite norme che condizionano fortemente la produzione del software: innanzitutto vi è una forte pressione sociale contro i progetti divaricanti se non giustificati pubblicamente; in secondo luogo è disapprovata dalla intera comunità la distribuzione delle modifiche apportate a un progetto senza la cooperazione dei moderatori. Inoltre è pratica diffusa non rimuovere mai il nome di uno sviluppatore dalla cronologia di un progetto se non dopo il suo esplicito consenso.

Si potrebbe poi obiettare che il concetto di proprietà per quanto riguarda il prodotto sia, nel caso del software Open Source, estremamente labile. In effetti è lecito chiedersi anche cosa si intenda per proprietà quando questa sia replicabile all'infinito e non sia soggetta a logiche economiche basate non sulla scarsità di beni, quanto piuttosto sull'ampia disponibilità degli stessi. Dubbi leciti che le licenze sembrano non spiegare in maniera abbastanza esaustiva. Teoricamente, secondo le stesse, tutte le entità coinvolte rivestono il medesimo ruolo nel gioco evolutivo, di fatto però esiste una distinzione molto ben riconosciuta tra gli aggiustamenti "ufficiali", approvati e integrati nel software in evoluzione da parte di quanti, pubblicamente riconosciuti, si occupano della sua gestione, e quelli "volanti" realizzati da terze parti. In genere questi ultimi sono rari e non riscuotono molta fiducia. Questa prassi mantiene le risorse (anche umane) prevalentemente all'interno della comunità che diviene il centro vitale per l'evoluzione del progetto non solo a livello di proprietà ma anche a livello di reputazione, la quale riveste un ruolo altrettanto importante nella psicologia collettiva che tiene legate le comunità Open Source.

Infatti, non necessariamente l'hacker programma per pura soddisfazione artistica o per necessità personale ma lo fa anche (e soprattutto) per guadagnarsi un certo tipo di fama all'interno del gruppo. Quest'idea viene ribadita anche da un'altra corrente di pensiero secondo la quale questo modello di sviluppo presenta parecchie analogie con quello adottato nel campo della ricerca scientifica accademica al punto da considerare la scrittura di codice una forma di produzione artistica come la musica o la pittura: per un programmatore il fatto di sapere che il proprio codice non solo funziona bene ma è anche stato utilizzato ed apprezzato dagli altri utenti della comunità costituisce un'enorme soddisfazione, pari a quella di un musicista che improvvisamente raggiunge il successo. Come quest'ultimo acquisisce la notorietà, analogamente il programmatore si crea una reputazione all'interno della comunità che lo distingue e gli assicura sì il rispetto e la gratitudine degli altri utenti ma soprattutto uno *status sociale* superiore, che è l'aspetto di maggiore rilievo. Raymond in una delle sue opere ne sottolinea e ne spiega l'importanza:

"G*li esseri umani sono dotati di una predisposizione innata alla competizione per il miglioramento del proprio status*

[94] Non è del tutto corretto ritenere che un soggetto sia spinto solo da una motivazione sociale o tecnica o economica, probabilmente è più giusto dire che in ogni caso vi è sia una componente sociale che una tecnica ed una economica solo che esse assumono pesi diversi.

sociale; qualcosa che è parte intrinseca della nostra evoluzione. Per il 90% della storia precedente all'invenzione dell'agricoltura, i nostri antenati vivevano in piccoli gruppi nomadi dediti alla caccia e alla raccolta. Agli individui con lo status sociale più alto spettavano i partner più sani e il cibo migliore. Una competizione che si esprime quindi in modi differenti a seconda del livello di scarsità delle merci necessarie alla sopravvivenza. La gran parte delle modalità secondo cui gli esseri umani si organizzano sono adattamenti alla scarsità e alla volontà. Ogni modo porta con sé maniere differenti di ottenere il miglioramento del proprio status sociale. La gran parte delle modalità secondo cui gli esseri umani si organizzano sono adattamenti alla scarsità e alla volontà. Ogni modo porta con sé maniere differenti di ottenere il miglioramento del proprio status sociale.

La nostra società è per lo più un'economia di scambio dove lo status sociale viene determinato essenzialmente dalla possibilità (o meno) di avere il controllo delle cose, non necessariamente delle cose materiali, da usare o da commerciare. Esiste tuttavia un terzo modello, radicalmente diverso e generalmente non riconosciuto se non dagli antropologi: la cultura del dono. Le culture del dono sono adattamenti non alla scarsità bensì all'abbondanza e si sviluppano all'interno di popolazioni senza problemi per quanto concerne la penuria di merci necessarie alla sopravvivenza. Possiamo osservare le culture del dono in azione tra gli aborigeni che vivono in ecozone dal clima mite e cibo abbondante oppure in certi strati della nostra società, soprattutto nel mondo dello spettacolo e tra persone molto ricche. L'abbondanza rende difficili le relazioni di comando e inutili le relazioni di scambio.

Nelle culture del dono, lo status sociale viene determinato non da quel che si controlla ma da quel che si regala. È così che avviene per le gesta filantropiche dei multimiliardari, in genere elaborate e pubbliche ed è così che avviene per le lunghe ore di lavoro che occorrono all'hacker per produrre software open source di alta qualità perché è ovvio che sotto quest'ottica, la società degli hacker open source non è altro che una cultura del dono. Al suo interno non esiste alcuna seria scarsità di "materiale di sopravvivenza", spazio su disco, ampiezza di banda di rete, macchine potenti. Il software è liberamente condiviso. Un'abbondanza che crea situazioni dove l'unico ambito disponibile per la competizione sociale è la reputazione tra i colleghi (Raymond, 1998b)."

A questo punto occorre sottolineare anche come la buona reputazione che ci si crea all'interno della comunità possa avere molte facce e non costituisca quindi solo una semplice ricompensa. Da un lato essa diventa anche un ottimo mezzo con cui attirare l'attenzione e la cooperazione altrui (d'altronde tanto migliore è la reputazione di una persona tanto è più facile convincere gli altri che potranno trarre giovamento dall'associarsi con tale persona), dall'altro, dato che il valore del software donato non è per nulla così ovvio come quello dei doni materiali (o come il denaro alla base dell'economia di scambio) ma anzi è fortemente soggettivo, la reputazione tra colleghi può servire come "merce di scambio" e di valutazione del bene prodotto. Per quanto riguarda le motivazioni di natura tecnica che portano un soggetto a partecipare allo sviluppo, esse sono senza dubbio le più evidenti e immediate in quanto in un contesto come la comunità è possibile riutilizzare liberamente il codice e i contributi altrui per risolvere le proprie esigenze tecniche. Così facendo si ha accesso ad un codice di qualità da cui poter partire con un proprio progetto o con un nuovo personale contributo (in questo gli esempi di Linux o di Fetchmail rendono bene l'idea) ma non solo.

Questo modello infatti, regala anche al programmatore la possibilità di "riassaporare" il *piacere della creatività*[95] che va inesorabilmente perdendosi nel modello proprietario, dove l'incubo dei "tempi di consegna" fa apparire l'attività di programmazione come una sorta di catena di montaggio[96], inoltre gli permette di soddisfare la domanda di funzioni o tecnologie specifiche in assenza dell'offerta corrispondente, come ad esempio spesso accade per i driver che fanno funzionare i componenti hardware[97].

Lo "sfruttamento" tecnico della comunità non si limita però solo a quanto appena detto o ad esempio, alla possibilità di "scaricare" ad altri l'onere del testing e del debugging; esso si esplica anche nell'utilizzo del *peer-review*, termine inglese traducibile come "revisione da pari a pari". Com'è già stato ribadito la reputazione conta molto all'interno della comunità ed essa dipende direttamente dalla qualità dei contributi offerti: con questo

[95] Bonaccorsi et. al. (2001).

[96] Questa situazione è ben descritta nel libro "*Microservi*", opera di alcuni giovani programmatori fuoriusciti da Microsoft.

[97] Non è raro che i produttori di componenti hardware come modem o scanner, rilascino i driver per una sola piattaforma, ad esempio Windows, impedendone di fatto l'uso su quelle alternative, come Linux. In casi come questo l'unica soluzione (anche se assai ardua) è scrivere il codice del driver con l'aiuto della comunità.

meccanismo il contributo offerto è sottoposto all'esame di corrispondenti sviluppatori, dotati di un pari livello di conoscenze (quindi i più adatti a giudicare) che sono in grado di correggere direttamente i difetti trovati e di suggerire migliorie alle strutture del programma. Quanto appena detto viene spesso associato alla cosiddetta "legge di Linus"[98].

3.4 Motivazioni economiche

Accade spesso che i detrattori o più semplicemente coloro che sono increduli nei confronti di questo modello, vedano la disponibilità del codice sorgente (con le conseguenze che ne derivano) come un palese controsenso rispetto al processo di sviluppo di un software, adducendo l'assenza di incentivi economici tali da indurre l'utente comune a contribuire. In sostanza l'idea prevalente è che in una situazione del genere si configuri il fenomeno economico del *free riding*[99] per cui lo stesso utente è portato a tenere un comportamento opportunistico utilizzando a proprio vantaggio il codice e i contributi altrui senza in cambio partecipare attivamente allo sviluppo. Questa osservazione non è del tutto errata, in effetti l'esistenza di free riders nel campo dell'Open Source è una diretta conseguenza della disponibilità stessa del codice sorgente, tuttavia occorre sottolineare come in questo contesto si possa e si debba compiere una distinzione fra una forma attiva ed una passiva di free riding. La prima si configura quando l'utente cerca deliberatamente di appropriarsi del risultato del lavoro altrui (vantando poi eventuali diritti sullo stesso), questa fattispecie però è piuttosto rara poiché la comunità la previene adottando strumenti legali come la licenza GPL.

La seconda è invece quella più comune e si ha quando egli si limita a ricevere i contributi altrui senza mai partecipare al lavoro collettivo; obiettivamente in questo caso la comunità non può fare nulla per impedirla. Ricollegandosi a quanto detto finora, è facile riscontrare l'esistenza di alcune tesi secondo le quali esisterebbe una relazione inversa tra l'ampiezza del gruppo e la probabilità che un'azione collettiva sia condotta con successo[100], in pratica all'aumento della dimensione del gruppo diminuirebbe la probabilità che un comportamento di defezione possa essere individuato e aumenterebbe in corrispondenza l'incentivo a godere del bene senza contribuire a produrlo. Ovviamente le cose non devono stare così viste le dimensioni assunte dal fenomeno e un primo motivo è la presenza di una notevole eterogeneità di soggetti partecipanti, per cui un piccolo gruppo di soggetti fortemente motivati (nel caso specifico gli hacker) e dotati delle risorse necessarie (il know-how informatico), è spesso in grado di superare le fasi iniziali del processo di fornitura del bene pubblico, determinando l'innescarsi di un "circolo virtuoso" che permette al fenomeno di autosostenersi[101] (il concetto di massa critica).

In secondo luogo l'utilizzo del software non ne diminuisce il valore, anzi, l'uso massiccio di software Open Source tende a farlo aumentare, con la possibilità da parte degli utenti di aggiungere correzioni e funzionalità a loro piacimento. Il passo successivo è che essendo pressoché impossibile stimare il valore di mercato di una propria *patch*[102] (per non parlare poi di come riscuotere gli eventuali pagamenti senza costi aggiuntivi) un soggetto si trova a dover scegliere fra il tenere per sé la patch o regalare la stessa alla comunità. Nel primo caso egli non otterrebbe alcun guadagno e in realtà nemmeno nel secondo ciò accadrebbe, tuttavia quest'ultima scelta potrebbe anche incoraggiare la distribuzione a catena ad altri utenti che, in futuro, si rivolgerebbero a lui per eventuali problemi (è un po' quello che accade con le distribuzioni Linux); questo comportamento è quindi solo apparentemente altruistico. Dopo aver compiuto questa breve riflessione sul problema del free riding all'interno del mondo Open Source, bisogna portare l'analisi ad un livello più materiale cercando di capire se ci siano (e se si quali siano) altri incentivi,

98 Si rimanda al paragrafo 1.5.

99 Nel linguaggio economico, questo termine indica l'accesso a un servizio che consente l'ottenimento di un beneficio senza oneri per sé. Il fenomeno del free riding è tipico dei beni pubblici cioè di quei beni che si caratterizzano sia per una indivisibilità dei benefici che per una non escludibilità dai benefici: la prima si verifica quando il consumo di un'unità del bene da parte di un individuo non influisce sulla possibilità di consumo della stessa unità da parte degli altri individui, la seconda invece quando l'offerta del bene ad un individuo non influisce sulla possibilità di offrire lo stesso anche a tutti gli altri individui. Nessuno sarebbe disposto ad assumersi l'onere di costruire una strada, tuttavia se essa esistesse ognuno la percorrerebbe: i beni pubblici (a cui il software libero viene da molti accostato) sono una delle cause di fallimento del mercato.

100 Olson (1965).

101 Bonaccorsi et al. (2001).

102 Traducibile come "pezza" indica in genere una correzione apportata ad un codice sorgente.

ancora più diretti, di natura economica.

Non si può negare il fatto che nel lungo periodo le motivazioni tecniche e sociali viste sinora non siano sufficientemente esaurienti per poter giustificare l'entità e la frequenza dei contributi dalla comunità inoltre, nel caso specifico dello sviluppatore cioè colui che partecipa alla scrittura del codice libero, è ovvio che questa attività comporti per esso dei costi opportunità[103], ad esempio il tempo libero o eventuali altre ricompense economiche a cui egli ha dovuto rinunciare. Da queste osservazioni emerge che devono esistere delle motivazioni anche di natura economica, per lo meno per coloro che partecipano assiduamente allo sviluppo: in tal senso gli stimoli più comuni provengono senza dubbio dalle opportunità di apprendimento e di miglioramento della propria reputazione professionale e quindi un aumento del proprio "capitale professionale". La maggioranza di questi soggetti infatti svolge l'attività di programmazione anche nella vita reale come lavoro.

Le competenze e le conoscenze acquisite in questo contesto (skills) vanno ad arricchire il bagaglio tecnico del soggetto, la cui professionalità riscuoterà di conseguenza un maggior apprezzamento sul mercato del lavoro: lo scambio tra lavoro intellettuale e reputazione può quindi essere convertito in denaro[104] (si crea un legame parallelo tra lo status sociale all'interno della comunità e quello nel mondo reale). Nel caso invece meno frequente dello sviluppatore che non svolge l'attività di programmazione nell'ambito lavorativo, gli incentivi a partecipare e a sopportare gli inevitabili costi opportunità sono diversi ma riconducibili sostanzialmente alla possibilità di ricevere molto più di quanto si è dato, anche se il vantaggio economico che deriva dall'utilizzo del software non è in genere direttamente valutabile in denaro. A mio avviso una valida riflessione in tal senso è stata compiuta da Inguaggiato:

"*Trattando con beni diffusi tramite la rete Internet si ha che lo sforzo necessario per produrre e diffondere una sola copia di un programma è identico a quello occorrente per dar via un milione di copie. E' quindi possibile regalare milioni di copie in cambio di una sola copia di qualcos'altro che ci è utile. La ricompensa cercata è parametrata all'unica copia che ha richiesto uno sforzo per essere prodotta e non rispetto alla distribuzione dell'informazione reso possibile dalla tecnologia di Internet* (..) *Questo schema è spesso rappresentato come una pentola in cui gettare quello che viene prodotto. La pentola restituisce quindi tutto il suo contenuto a chiunque lo voglia* (..) *Se un numero anche modesto di persone getta qualcosa nella pentola i beni saranno disponibili per tutti nella loro completezza e quindi ognuno riceverà molto di più di quanto ha dato. Nonostante l'importanza che ha la reputazione nello scambio di beni gratuiti, questa non costituisce un'economia così come non lo farebbe il solo denaro senza beni concreti di cui rappresentare il valore. La reputazione, la stima, l'autorevolezza sono l'indicazione di valore data dalla comunità ad un contributo gettato nella pentola* (..) *La partecipazione ad una siffatta "economia" non è data* (solo) *dall'intenzione di accrescere la propria reputazione, anche se questa è una componente degli scambi, ma dal ricevere molto più di quanto si è messo nella pentola* (Inguaggiato, 1999)".

Finora si sono analizzate le motivazioni che spingono l'agente individuale a partecipare ad un progetto: da una serie di indagini è stato possibile stabilirne l'importanza agli occhi degli sviluppatori e i risultati sono rappresentati dal grafico 3.2. Restano tuttavia ancora da chiarire le motivazioni legate all'intervento delle aziende e della Pubblica Amministrazione; dato che per questi soggetti gli incentivi sono quasi esclusivamente economici la loro analisi verrà compiuta nel quarto capitolo.

[103] Il costo opportunità di usare una qualunque risorsa (tempo, denaro..) per un determinato scopo è il beneficio che si sarebbe potuto trarre dall'impiego di quella risorsa nel miglior uso possibile alternativo.

[104] L'esempio più banale da farsi è quello di Linux, quale tipo di pagamenti in denaro lo ha determinato? Linus Torvalds è considerato una sorta di divinità in tutti gli ambienti in cui potrebbe beneficiarne la sua carriera per cui alla crescita di reputazione è corrisposta una rara "assicurazione" a vita contro la disoccupazione.

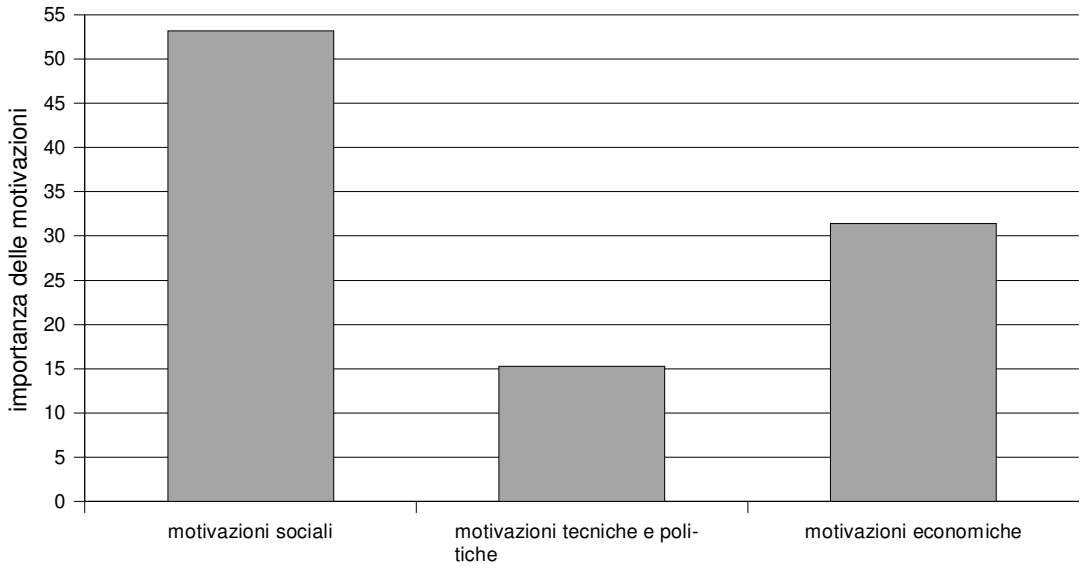


Grafico 3.2 Indagine sull'importanza attribuita alle singole motivazioni
Fonte: FLOSS (2004)[105].

3.5 Qualità e caratteristiche tecniche del software Open Source

L'Open Source costituisce indubbiamente un importante fattore di cambiamento nella storia del software e del suo mercato tuttavia il più delle volte vengono trascurate le dirette conseguenze di questo modello sul piano tecnico e sulla qualità del software. L'obiettivo di questo paragrafo è quello di fornire una serie di informazioni, anche di natura tecnica, che possano contribuire ad estendere il confronto con il software proprietario ad un maggior numero di aspetti, integrando così quelli che in genere appaiono più interessanti dal punto di vista economico ma che per questo non sono più importanti. In termini generali, la qualità di un prodotto o di un servizio può essere definita come la capacità di soddisfare requisiti esplicitati e non. Essendo tali requisiti riferiti alle esigenze di soggetti diversi, esistono più modi di valutare la qualità; in particolare si possono distinguere alcuni approcci di valutazione in funzione dei diversi punti di vista, ad esempio quelli dell'utilizzatore, del progettista, della produzione e del valore, quest'ultimo in relazione al rapporto qualità/costo del prodotto[106].

Ad esempio, se per gli utilizzatori la qualità di un prodotto dipende sostanzialmente da aspetti quali gli attributi, l'affidabilità, la presenza di difetti, la praticità, l'assistenza, nel caso dei produttori invece, essa viene di solito intesa come la conformità alle specifiche tecniche. Bisogna poi ricordare che all'interno di queste "categorie" si possono riscontrare a loro volta delle differenze sulla percezione della qualità, dovute alla diversa importanza che ciascun soggetto assegna agli aspetti appena visti. La classica difficoltà a stabilire dei parametri qualitativi oggettivi risulta accentuata nel caso del software a causa di alcune sue peculiarità: innanzitutto la natura digitale che determina l'assenza di fisicità (e i relativi vincoli) e non consente l'uso delle metriche tradizionali; in secondo luogo esso non si consuma, non si modifica né si usura con l'utilizzo o con il passare del tempo. Infine esso va visto come uno strumento, di conseguenza la sua qualità va espressa riferendosi non solo ad esso come prodotto ma anche in base al servizio reso. Fatta questa doverosa premessa è ora possibile cercare di fornire alcuni spunti di riflessione sulla qualità del software Open Source, una questione su cui da anni esiste un acceso dibattito.

Il software Open Source è migliore di quello proprietario? Probabilmente non è sempre così, tuttavia in termini generali è evidente che i fatti stanno smentendo l'idea che non possa offrire alcuna garanzia di qualità il frutto di un lavoro collettivo, svolto con modalità apparentemente caotiche da una pluralità di sviluppatori disseminati ovunque. Tralasciando per il momento l'aspetto finanziario (nel quale fra l'altro l'Open Source di solito presenta dei consistenti vantaggi) è infatti possibile constatare come in certe situazioni il software Open Source sia, se non superiore, per lo

[105] http://flossproject.org/papers.htm
[106] Garvin (1984).

meno valido quanto quello proprietario. Nell'analizzare gli aspetti che più comunemente vengono ritenuti indicativi della qualità di un software occorre ricordare ancora una volta come quello Open Source sia caratterizzato dalla disponibilità del codice sorgente e da una serie di libertà legate ad esso[107]; inoltre si utilizzeranno come esempi studi numerici aventi ad oggetto prevalentemente il sistema operativo Linux e il web server Apache.

Il primo di questi aspetti ad essere preso in considerazione è l'*affidabilità* ovvero la capacità di un software di mantenere le proprie prestazioni nel tempo e nelle diverse situazioni che si possono presentare. In questo ambito sono stati condotti diversi test da società autonome[108] su prodotti Open Source di vario tipo che sono stati confrontati con le alternative proprietarie. Ad esempio lo scorso dicembre la Reasoning Inc.[109], società che studia soprattutto le debolezze strutturali del codice e cerca di determinare gli eventuali punti critici che potrebbero portare difetti e malfunzionamenti, ha deciso di comparare i risultati di una ricerca sul database[110] Open Source MySql e sui concorrenti database proprietari. La versione presa in esame è la MySQL 4.0.16, composta da oltre 235.000 linee di codice, distribuite in 517 file. Il risultato è stato un totale di 21 bug che corrisponde a 0.09 errori per linea di codice; quello dei software commerciali è stato invece di 0.57, molto più alto e basato sull'analisi di oltre 35 milioni di righe di codice.

Altro dato da rimarcare è il fatto che il team di sviluppo di MySQL ha risolto 13 dei 21 difetti appena la Reasoning li ha comunicato loro, una reattività sicuramente imputabile all'apertura dei codici sorgente e all'elevato numero di soggetti coinvolti nelle fasi di testing e debugging, tutte contromisure inattuabili nel modello proprietario (analoghi studi[111] condotti su altre applicazioni Open Source da altre società, compresa IBM, hanno dato esiti simili). Il fatto stesso che Linux sia utilizzato in importanti siti come Google, ne testimonia la buona reputazione in termini di affidabilità: in una recente indagine condotta dal FLOSS[112] su 1452 aziende e organizzazioni circa l'83% degli intervistati ha dichiarato che la sua "superiore stabilità" era un ottimo motivo per adottare software Open Source.

Dall'affidabilità si passa all'analisi delle *performance* cioè le prestazioni del software con particolare riferimento alla velocità e alle funzionalità implementate in esso. Anche in questo caso sono stati condotti, e lo sono tuttora, molteplici test per confrontare i due modelli tuttavia la loro attendibilità risulta spesso alquanto controversa poiché accade frequentemente o che le configurazioni hardware e software utilizzate non siano le stesse oppure che i parametri di riferimento (i cosiddetti *performance benchmarks*) siano particolarmente sensibili agli ambienti di prova e alle ipotesi fatte. Se è vero che è difficile trovare dei parametri rigorosi è anche vero però che nella maggioranza dei test effettuati il prodotto Open Source ha sempre dimostrato un'elevata competitività. Nel 2001 Lo SPEC Consortium[113], una delle organizzazioni più dedite allo sviluppo di benchmark imparziali, ha eseguito un confronto fra il sistema operativo Linux e il sistema Windows/IIS di Microsoft che, pur avendo rilevato la qualità del primo, non ha permesso, per i motivi citati poco fa, di affermare con certezza il netto predominio tecnico di un prodotto sull'atro.

Il terzo aspetto che viene in genere ritenuto indicativo della qualità di un software è la sua *sicurezza*, ovvero la capacità di difendersi da possibili violazioni dell'integrità o da usi non appropriati. A volte i sostenitori del software proprietario affermano che i loro prodotti possono contare su livelli di sicurezza più elevati e in particolare su una migliore crittografia[114] rispetto ai prodotti aperti che adottano algoritmi noti a tutti in quanto il loro codice può essere

---

[107] Si rimanda al primo capitolo.

[108] Non è raro che i produttori di software proprietario (uno in particolare, Microsoft) affidino a delle società di consulenza il compito di testare i loro prodotti e di stabilirne la qualità in relazione a quelli dei concorrenti (specie se questi sono Open Source), tuttavia è lecito pensare che il risultato della ricerca compiuta sia in qualche modo condizionato proprio dal fatto che il software testato sia quello del committente. I test migliori sono quelli condotti da società terze.

[109] Http:/www.reasoning.com

[110] Termine che indica un insieme di archivi riguardanti uno stesso argomento, o più argomenti correlati tra loro, formando una base di dati (appunto, un database). La base di dati, oltre ai dati veri e propri, deve contenere anche gli elementi necessari alla loro rappresentazione, quali ad esempio le informazioni sulla loro struttura e sulle relazioni che li legano. Un requisito importante di una buona base dati consiste nel non duplicare inutilmente le informazioni in essa contenute: questo è reso possibile dai gestori di database relazionali, che consentono di salvare i dati in tabelle che possono essere collegate. Il linguaggio più utilizzato dai database è l'SQL di cui sono stati pubblicati diversi standard.

[111] http://www.dwheeler.com/oss_fs_why.html

[112] http://floss.infonomics.nl/

[113] http://www.spec.org

[114] La crittografia è un processo di conversione da un formato facilmente comprensibile a tutti, ad un formato accessibile a pochi, allo scopo di rendere certe informazioni segrete. L'operazione di conversione delle informazioni è chiamata cifratura

letto e studiato a fondo. Sebbene anche in questo ambito vi sia una certa divergenza di opinioni, ciò non è del tutto vero infatti quasi tutti i sistemi crittografici spesso presentano punti deboli al loro interno che rendono prima o poi il sistema facilmente attaccabile. L'unico modo che sembra garantire una ragionevole tranquillità è quello di sottoporre procedure e algoritmi a severe verifiche da parte di studiosi di alto livello, possibilmente appartenenti ad ambienti diversi. Per sapere se una serratura è sicura occorre affidarla a uno o più esperti che la aprano e la studino a fondo per verificare che non possa essere aperta con un idoneo chiavistello: lo stesso accade con la crittografia nella sicurezza informatica dove ovviamente l'ispezione da parte di più soggetti (nel caso specifico gli hacker/sviluppatori) richiede l'apertura del codice sorgente[115].

I sistemi Open Source non sono infallibili e inattaccabili (probabilmente non esiterà mai un software con queste caratteristiche) tuttavia, se i parametri più rilevanti nella valutazione della sicurezza sono di solito il numero e la gravità dei danni causati dai virus informatici, essi vantano una buona solidità e resistenza. Secondo una ricerca del CERT (Center for Internet Security Expertise della Carnegie Mellon University) dal 1998 il numero di computer attaccati dai virus raddoppia ogni anno, ciò nonostante, per quanto i sistemi Linux detengano ancora una piccola quota del mercato desktop[116] essi sono relativamente immuni a questo problema[117] (anche ipotizzando che nel codice siano presenti dei bug sfruttabili ai fini della sicurezza i tempi di correzione sono comunque molto più ridotti nel caso del software OS). A conferma della loro bontà, Okemos, un'agenzia specializzata nei prodotti assicurativi per i sistemi informatici, ha aumentato del 5-15% i premi per i clienti che non utilizzano sistemi Unix o GNU/Linux nelle loro infrastrutture di rete[118]. Oltre che per affidabilità, performance e sicurezza, il software Open Source si distingue, grazie all'apertura del codice, anche per altre caratteristiche che di seguito verranno brevemente riportate:

● *Portabilità*, cioè la compatibilità del codice con sistemi hardware e altre architetture software; essa in genere rende possibile il riutilizzo di componenti vecchi (la cui obsolescenza è spesso creata artificialmente dai responsabili commerciali) con un impatto benefico sui costi. Il motivo va anche cercato nella frequente *modularità* del software Open Source cioè nel fatto di essere composto da singoli moduli di codice che possono essere "montati e smontati" a seconda delle specifiche esigenze, permettendo così di avere programmi più snelli e affidabili.

● *Flessibilità*, ovvero la capacità del software di adattarsi a varie e mutevoli esigenze degli utenti. Teoricamente ogni utente può, se ne ha le capacità e le conoscenze tecniche, partendo dalla versione rilasciata dalla comunità crearsene una propria acquisendo quindi anche una certa indipendenza (dovrà rivolgersi ad altri solo per l'assistenza). Questa possibilità è ovviamente impensabile nel modello proprietario dove lo sviluppo di una differente versione deve essere giustificata da validi ritorni economici. Un esempio di come la flessibilità possa attuarsi a livello pratico è fornito dal sistema Linux di cui possono essere tolte ampie parti, riducendolo a dimensioni tali da poter essere memorizzato in un floppy e utilizzato per il recupero di dati sensibili oppure per la gestione di apparecchi elettronici quali telefoni cellulari o palmari.

● *Interoperabilità*, che può essere intesa come la capacità di due o più sistemi (o applicazioni) di scambiarsi informazioni e di usare mutuamente l'informazione che è stata scambiata oppure come la capacità dell'hardware e

---

oppure crittazione, ed è effettuata tramite un apposito algoritmo cioè una sequenza ordinata e finita di istruzioni. La sicurezza informatica è fortemente basata sulla crittografia.

[115] I produttori di software proprietario sostengono che la chiusura del codice permette di preservare un certo grado di sicurezza tuttavia considerando che quest'ultima si concretizza quanto più il codice può essere controllato è evidente che l'approccio Open Source si rivela superiore: nel 1996 quando la società Borland decise di rilasciare i codici sorgente del suo famoso database "Interbase" si scoprì che esso era affetto da un pericoloso bug presente da svariati anni. Per inciso, tale programma era, ed è, utilizzato da organismi importanti come ad esempio la Nasa e altri enti governativi.

[116] Il mercato del software viene di solito diviso in due parti, il segmento desktop (o client) costituito dai computer utilizzati a casa e in ufficio e il segmento dei web server, dove Linux e Apache stanno dimostrando la loro forza. Finora il software Open Source non è riuscito ad imporsi come valida alternativa a Windows di Microsoft che rappresenta lo standard nel segmento desktop.

[117] Evans Data Corporation (2002) http://www.businesswire.com/cgi-bin/f_headline.cgi?bw.040802/220982285

[118] http://news.com.com/2100-1001-258392.html?legacy=cnet oppure http://www.dwheeler.com/oss_fs_why.html

del software di macchine diverse, fornite da costruttori diversi, di comunicare in maniera comprensibile. Questa caratteristica è uno dei maggiori punti di forza dei prodotti a codice sorgente aperto tuttavia richiede anche l'utilizzo di formati e interfacce "aperti" per la gestione dei dati[119]: è palese che la possibilità di far interagire macchine e programmi diversi attraverso dei formati aperti, spesso resi degli standard, determini dei benefici sia per gli utilizzatori che per i produttori, basti pensare all'HTML divenuto ormai uno standard per la diffusione di documenti in Internet.

In questo paragrafo i prodotti Open Source sono stati in genere valutati positivamente con riferimento ad un certo numero di parametri. Non è possibile (almeno per ora) dimostrare l'esistenza di una stretta correlazione tra i livelli di qualità e la loro natura né sembra possibile affermare che essi siano migliori dei software proprietari tuttavia le analisi compiute e le informazioni facilmente reperibili in Internet permettono di considerarli come una valida alternativa a questi ultimi.

[119] Si rimanda al prossimo capitolo.

CAPITOLO 4
L'OPEN SOURCE NEL MERCATO DEL SOFTWARE

4.1 Economia dell'Informazione: la tecnologia cambia, le leggi dell'economia no

L'obiettivo principale che ci si pone con questo elaborato non è tanto quello di presentare gli aspetti più generali dell'Open Source quanto di comprendere quali siano gli effetti dello stesso sul mercato del software, da anni assuefatto alle logiche del modello proprietario. La cosa non è semplice né immediata e questo sostanzialmente per due motivi: innanzitutto occorre prendere atto che questo mercato, tutto sommato, è ancora relativamente giovane per cui i dati a disposizione e la conoscenza dello stesso sono parziali; in secondo luogo presenta delle caratteristiche particolari e pone dei problemi nuovi, difficilmente riscontrabili nei mercati tradizionali. Riguardo questo secondo punto va sottolineato come molte siano state le divergenze d'opinione fra gli economisti, riscontrabili in due correnti di pensiero: per alcuni la *new economy* o *information economy*, a cui appartiene il mercato del software, rappresenta una novità, un caso a parte, troppe sono le differenze presenti con il mercato tradizionale di conseguenza non è possibile rifarsi alle leggi "classiche" e "obsolete" dell'economia ma occorre definirne di nuove.

Altri invece ritengono che "l'economia dell'informazione" non sia poi così diversa da quella tradizionale, essa è nuova solo per chi non ha studiato a sufficienza quella vecchia[120]. Si tratta solo di adattare le vecchie leggi al nuovo contesto prendendo in considerazione da un lato le caratteristiche uniche che possiede il bene tipico di questa economia, cioè l'*informazione,* dall'altro la tecnologia che incide non solo sull'economia e i suoi principi ma, ormai, anche sulla vita comune. In effetti una situazione del tutto analoga si presentò all'inizio del secolo scorso, quando le allora avveniristiche reti elettriche e di telecomunicazioni fecero credere che stesse avvenendo una "rivoluzione economica" e fossero necessarie nuove regole di base. Niente di più falso, come si è dimostrato.

Sebbene negli ultimi anni abbia finito per prevalere la seconda tesi, alla prima va comunque riconosciuto il merito di aver fatto luce su alcuni aspetti insoliti, ma non per questo "rivoluzionari", tipici dell'economia dell'informazione, in particolare sul valore che i beni gratuiti assumono in Rete anticipando il concetto di *economia* o *cultura del dono*, relativa al comportamento degli operatori e riscontrabile anche nel fenomeno del software libero. Fatta questa premessa occorre analizzare le caratteristiche che il software commerciale, in quanto informazione, presenta nonché gli effetti che le stesse producono sulla struttura del mercato. Occorre in tal senso precisare che in questo contesto il termine "informazione" verrà usato con un'accezione molto ampia: fondamentalmente, tutto ciò che può essere digitalizzato, ossia essere rappresentato come una sequenza di bit, è informazione; rientrano in questa definizione quindi la musica, le riviste, i libri, i film, i punteggi delle partite di calcio, il software.

4.2 La struttura dei costi

Il bene informazione presenta una caratteristica unica che lo distingue da tutti gli altri: produrre informazione costa, riprodurla poco o nulla, specie se ci si basa sulla tecnologia digitale. Un film (ma l'esempio si potrebbe fare con i libri piuttosto che con il software), per la cui realizzazione vengono spesi milioni di euro, è facilmente riproducibile per pochi euro, il costo di un dvd. Risulta dunque facile capire perché qualsiasi produzione in questo settore presenti una struttura dei costi sbilanciata che a loro volta presentano delle peculiarità: chiunque produca questo bene si trova a dover sostenere soprattutto degli elevati costi fissi iniziali, irrecuperabili o comunque difficilmente recuperabili. Se si investe nella costruzione di un edificio e in secondo momento esso si rivela inutile è comunque possibile recuperare una parte dell'errato investimento attraverso la vendita dell'immobile, la stessa cosa è invece impraticabile se l'oggetto dell'investimento iniziale è un film o un cd musicale di scarso successo.

A questi costi fissi spesso vanno ad aggiungersi costi di marketing e di promozione: l'esagerata abbondanza di informazioni, specie in Internet, dovuta al loro basso costo di distribuzione e alla loro ubiquità genera una *povertà di attenzione* di conseguenza le aziende non hanno tanto la necessità di raggiungere i potenziali clienti quanto di attirare la loro attenzione[121]. Al contrario il costo variabile, sostenuto per la riproduzione e la diffusione

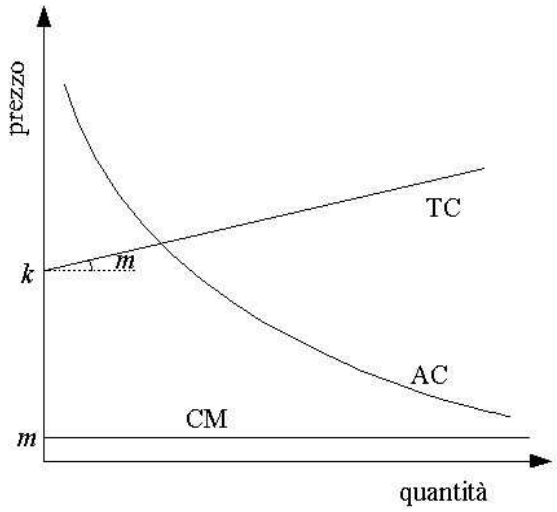
[120] Shapiro et. al. (1999)

[121] Spesso riferendosi all'economia dell'informazione viene usata la denominazione "economia dell'attenzione".

dell'informazione, è nullo o quasi; analogamente il costo marginale, inteso come costo relativo all'unità addizionale prodotta, è da considerarsi prossimo allo zero. A differenza di quanto accade per la Fiat, Microsoft non è sottoposta a consistenti vincoli di capacità produttiva: non ci sono limiti produttivi per cui, che ne produca una, centomila o un milione di copie del suo sistema operativo, il costo unitario di riproduzione non cambia. Elevati costi fissi e bassi costi marginali implicano forti economie di scala[122], peraltro riscontrabili anche in altri settori, in genere quelli relativi alla produzione di beni ad alta tecnologia.

Ipotizzando che i bassi costi variabili siano imputabili ai processi di copiatura, di confezionamento e alla distribuzione delle innumerevoli copie, nel breve periodo si può ritenere che la funzione di costo di un'impresa produttrice di software sia una funzione lineare del tipo $TC = k + mx$ dove *k* rappresenta i costi fissi e *mx* il costo variabile proporzionale al numero *x* di copie prodotte. Nella figura 4.1 la presenza di economie di scala appare chiara dall'andamento decrescente del costo medio (*AC*) mentre il costo marginale (*MC*), ottenuto come derivata della funzione del costo totale, risulta costante e pari a *m*.

Figura 4.1 Struttura dei costi



Sebbene la Rete costituisca l'esempio più noto non si deve ritenere che il ridotto costo marginale del bene informazione sia una novità introdotta da Internet. Anche la tecnologia di stampa, ad esempio, determina un costo marginale per la riproduzione di un libro molto ridotto. Occorre però sottolineare come le tecnologie digitali, su cui si basa anche Internet, siano state le prime a influire sia sul costo di riproduzione (annullato) che su quello di distribuzione (quasi nullo) *contemporaneamente* mentre quelle precedenti influivano solo su uno dei due alla volta; questo è il motivo per cui i costi marginali si possono ritenere in alcuni casi virtualmente uguali a zero. Dall'asimmetria nei costi derivano tre conseguenze: innanzitutto, essendo il costo marginale prossimo allo zero, essa costringe i produttori a fissare il prezzo dei propri beni non in funzione del costo di produzione unitario, che non consentirebbe loro di coprire gli elevati costi fissi, bensì in funzione del valore che viene percepito dai consumatori, quindi dalla loro *disponibilità a pagare*[123]; si ha in pratica una distorsione nella determinazione del prezzo. Dato che i consumatori possono percepire dei valori diversi per lo stesso prodotto, le aziende operanti in questo mercato spesso forniscono i propri prodotti a prezzi diversi, ossia attuano una discriminazione di prezzo[124].

Un secondo effetto è che una volta coperti i costi irrecuperabili, non esiste più, almeno teoricamente, un limite minimo al prezzo dell'informazione generica poiché ogni produttore sarà costretto ad abbassare i prezzi non appena un concorrente farà lo stesso. Per semplificare il concetto è possibile ricorrere ad un esempio banale: si supponga che nel mercato delle enciclopedie digitali, l'azienda X e l'azienda Y offrano entrambe a 100 euro la propria versione, di cui entrambe hanno già coperto i costi irrecuperabili di produzione; inoltre si ipotizzi che i due prodotti siano omogenei in quanto si differenziano nell'origine ma presentano la stessa funzionalità, lo stesso numero di voci e la stessa interfaccia grafica di consultazione. E' ovvio che a parità di condizioni se X riduce il prezzo del suo prodotto a 99 euro, i consumatori sceglieranno la versione con il prezzo minore: come conseguenza Y abbasserà il prezzo a 98 euro, riconquistando il mercato; a sua volta X sarà costretta a ridurre nuovamente il prezzo a cui seguirà un taglio di prezzo da parte Y e così via.

[122] Si ha un'economia di scala quando il costo medio di produzione diminuisce all'aumentare della produzione; nell'economia dell'informazione, ma non solo, la possibilità di "spalmare" il costo totale su un numero anche elevato di beni, determina un'incidenza decrescente dello stesso sul singolo bene.

[123] Varian (1996)

[124] La discriminazione di prezzo è definibile come la pratica di fissare prezzi diversi per lo stesso bene o servizio in funzione di un qualcosa, ad esempio della quantità acquistata o dell'acquirente; a seconda del parametro preso in considerazione sono distinguibili tre gradi di discriminazione di prezzo. Nell'economia dell'informazione, la discriminazione di prezzo più comune è quella di terzo grado, cioè quella basata sulla suddivisione del mercato in diversi gruppi di persone caratterizzati da curve di domanda diverse (a causa della diversa disponibilità a pagare), ad esempio studenti e anziani, meglio nota come segmentazione del mercato. Nel caso specifico del software la discriminazione si può attuare attraverso una tariffa a due stadi, che offre maggiori garanzie di coprire i costi fissi.

In questo modo, anche nel caso in cui le aziende siano più di due, si ottiene una riduzione del prezzo fino al punto in cui esso coincide con il costo marginale; questo aiuta a capire perché ci sia un'enorme quantità di informazioni generiche gratuite in Rete: in realtà esse hanno un prezzo (cioè zero) solo che questo è uguale al costo marginale sostenuto per produrle. Il terzo effetto (e il più rilevante) riguarda la struttura che il mercato dell'informazione assume e che è riconducibile sostanzialmente a due tipi:

- una concorrenza oligopolistica, anche se non mancano situazioni di duopolio o addirittura di monopolio, dove a prescindere dall'effettiva qualità del prodotto offerto le poche imprese dominanti sfruttano una *leadership di costo* rispetto ai potenziali concorrenti dovuta alle dimensioni delle stesse, alle forti economie di scala nella produzione e alla presenza delle esternalità di rete (oggetto del prossimo paragrafo); è la struttura più frequente dell'industria del software. In questo contesto l'assenza di concorrenza rende impossibile il verificarsi dell'effetto descritto poco fa, secondo cui una volta coperti i costi fissi il prezzo si riduce fino ad eguagliare il costo marginale;
- una concorrenza fra un numero elevato imprese che offrono *prodotti differenziati*, cioè lo stesso tipo di bene informazione ma con diverse varietà; adottano questa struttura in genere l'industria cinematografica e quella dell'editoria.

4.3 Le economie di rete: esternalità e feedback positivo

Nelle industrie ad alta tecnologia e più ampiamente nell'economia dell'informazione oltre alla struttura dei costi è riscontrabile un'ulteriore elemento di distinzione dall'economia tradizionale: il concetto di *rete* e la presenza non tanto di economie di scala, bensì di *economie di rete.* In genere il termine rete viene utilizzato in senso fisico per indicare la rete telefonica, quella autostradale oppure Internet, tuttavia in questo contesto esso assume un significato più ampio, comprendente anche quello di *rete virtuale*: nella rete reale i collegamenti fra i nodi della stessa, rappresentati ad esempio dai computer, sono assicurati da connessioni fisiche come i cavi telefonici; in quella virtuale i collegamenti fra i nodi sono invisibili ma non per questo meno importanti per le dinamiche del mercato e le strategie degli operatori del mercato[125]. I nodi di una rete virtuale possono essere costituti da individui, ad esempio nell'ambito del software gli utenti appartengono ad una stessa rete se utilizzano lo stesso software e possono di conseguenza scambiarsi i dati. Sono dunque identificabili una rete virtuale composta dagli utenti di OpenOffice oppure una composta da quelli di Microsoft Office.

Sia nelle reti fisiche che in quelle virtuali sono riscontrabili due aspetti tipici di questa economia ossia le *esternalità* o *effetti di rete* nonché il *feedback positivo di rete*. Un'esternalità si crea ogni qualvolta che un individuo, partecipante alle attività di mercato, influenza gli altri individui senza che per questo ci sia alcuna corresponsione di denaro: in questo contesto la sola appartenenza ad una rete può variare l'utilità degli altri soggetti presenti in essa. Risulta quindi che il valore assegnato da ciascun utente alla connessione alla rete dipende dal numero di altri utenti che già vi fanno parte; generalmente, a parità di altre condizioni, è meglio essere connessi ad una rete di grandi dimensioni piuttosto che ad una rete piccola. Occorre precisare che sono distinguibili effetti di rete diretti ed indiretti. I primi si hanno quando il numero di utenti che adotta il bene influisce sull'utilità dell'individuo per il fatto che egli può interagire con più persone. Il fax, il telefono, l'e-mail, i software applicativi vedono crescere il proprio valore al crescere della rete di utenti.

Si parla invece di effetti di rete indiretti (o anche di *complementarietà*) quando l'utilità dell'utente non dipende tanto dal poter interagire con un elevato numero di altri utenti bensì dalla maggiore attenzione che il proprio prodotto riceve da parte delle aziende produttrici di beni complementari, di software, di aggiornamenti. E' probabile che un utente dovendo acquistare un componente hardware scelga, a parità di altre condizioni, quello più diffuso perché così facendo ottiene dei maggiori benefici, identificabili nella garanzia di una maggiore compatibilità con gli altri componenti o software presenti sul mercato e lo sviluppo nel tempo dei relativi driver[126] da parte della casa produttrice. Per lo stesso motivo il valore di un sistema operativo dipende fortemente dalla disponibilità e dalla varietà di applicazioni compatibili. Queste osservazioni del tutto naturali costituiscono la base di una regola pratica,

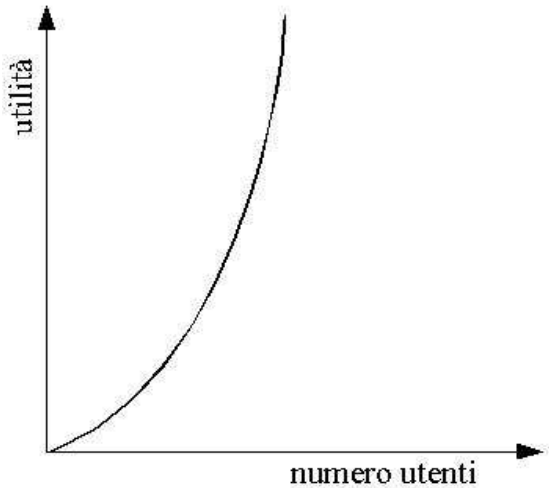
[125] Shapiro et. al. (1999)

[126] Il driver è il software che permette ad un sistema operativo di pilotare un dispositivo hardware, ad esempio una stampante o un modem.

nota come legge di Metcalfe[127], secondo la quale *il valore di una rete cresce in maniera esponenziale al crescere del numero di utenti che ne fanno parte.*

Se si ipotizza che la rete sia composta da un numero *n* di individui e che ciascuno di essi le assegni un valore, proporzionale al numero di altri utenti appartenenti alla stessa, allora il valore totale della rete è proporzionale a $n(n-1) = n^2-n$. L'esempio più banale resta quello del telefono: se solo tre persone nel mondo avessero il telefono, si sarebbe restii ad installarne uno in casa perché l'utilità derivante dal potersi mettere in contatto con altri sarebbe limitata, al contrario essa cresce (e analogamente il valore della rete telefonica) tanto maggiore è il numero di individui con cui è possibile comunicare. Dagli esempi fatti si evince quindi che il desiderio di standardizzazione è la causa principale degli effetti di rete, illustrati chiaramente nel grafico 4.2. Nell'economia dell'informazione è facile osservare anche come le tecnologie o i prodotti caratterizzati da consistenti esternalità di rete, tendano ad esibire dei periodi di introduzione molto lunghi seguiti poi da una crescita esplosiva; al crescere del numero di utenti che si sono già dotati di quella tecnologia o di quel prodotto un numero sempre maggiore di altri utenti vengono da questa attratti.

Grafico 4.2 Legge di Metcalfe



Fonte: Shapiro et. al. (1999)

Quando si assiste alla crescita esplosiva di cui si è accennato poco fa, si dice che il prodotto o la tecnologia hanno conquistato la cosiddetta *massa critica*, cioè una quota di mercato in termini di utenti, oltre la quale, per il singolo utente, il non appartenere alla rete di quel prodotto o di quella tecnologia costituisce uno svantaggio rilevante. Ciò si verifica per l'innescarsi della legge di Metcalfe dalla quale, come si è già visto, si evince che il vantaggio relativo dell'utente marginale oltre una determinata soglia è molto più ampio del vantaggio relativo dei primi utenti. Oggigiorno la presenza del fax in un ufficio viene considerata ovvia ma questo prodotto, la cui tecnologia fu brevettata già nel 1843[128], è rimasto fino ai primi anni Ottanta uno strumento di nicchia: prima del 1982 praticamente nessuno lo possedeva, nel 1987, quindi a distanza di soli cinque anni la maggior parte delle attività commerciali si era dotata di almeno un apparecchio[129]. Il fenomeno appena descritto, che assieme alle esternalità di rete caratterizza fortemente quest'economia, è denominato feedback positivo di rete; sintetizzando si può dire che il feedback positivo rende le grandi reti ancora più grandi.

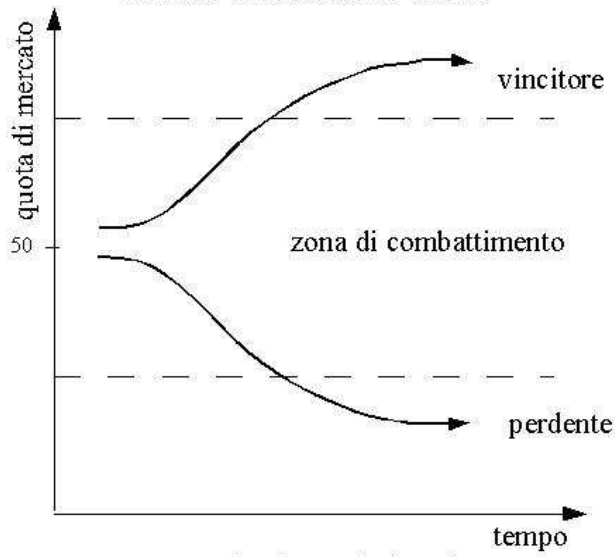
Analogamente esso rende l'impresa più forte ancora più forte e indebolisce quella debole, determinando spesso uno spostamento del mercato verso situazioni di monopolio o oligopolio (al contrario il feedback negativo porta il mercato verso una situazione mediana in cui gli operatori si trovano alla pari). In passato spesso si è assistito a "guerre" fra due o più tecnologie, per il controllo del mercato, basti pensare allo scontro che si ebbe sempre negli anni Ottanta, fra i due formati di videoregistrazione Vhs e Betamax, con la vittoria della prima. Il grafico 4.3 illustra proprio quanto appena detto, cioè come spesso la situazione iniziale di equilibrio possa evolversi verso quella più estrema in cui una sola tecnologia si impone sul mercato.

[127] La legge deve il suo nome a Bob Metcalfe, l'inventore di Ethernet.

[128] Occorre precisare che la tecnologia necessaria a compiere la trasmissione fotografica per via telefonica nacque però solo intorno agli anni Venti.

[129] Shapiro et. al (1999).

Grafico 4.3 Feedback Positivo



Fonte: Shapiro et al. (1999)

Il feedback positivo rappresenta la principale forza di mercato e incide notevolmente sul successo di una tecnologia o un prodotto rispetto ad altri concorrenti, tuttavia, esso non va confuso con la semplice crescita che questi, magari per la novità che rappresentano, possono avere sul mercato; indubbiamente può accentuarla ma non la determina. Un ruolo fondamentale per il successo e l'adozione di un prodotto viene infatti rivestito anche dalle relative *aspettative dei consumatori*[130], a volte a prescindere dalla sua qualità, per cui se l'opinione prevalente è che esso si imporrà come lo standard di riferimento è probabile che si abbia un'ulteriore adozione e viceversa nel caso di un diffuso giudizio negativo. In tal senso il successo o l'insuccesso può dipendere anche dalla capacità dell'impresa di riuscire ad innescare il feedback positivo a proprio favore, per esempio adottando politiche di prezzo graduali ai livelli della domanda.

Detto questo occorre però fare tre precisazioni: innanzitutto non si deve ritenere che gli effetti del feedback si producano su tempi brevi, anzi, essi possono prodursi anche su più anni. In secondo luogo il fatto di detenere almeno inizialmente una quota di mercato superiore a quella dei concorrenti, proprio per il ruolo giocato dalle aspettative che si creano, non garantisce la vittoria; in questo il pacchetto Office di Microsoft che partì da una quota di mercato inferiore a quella della concorrenza, è un ottimo esempio[131]. Infine non necessariamente il feedback positivo determina una situazione in cui il vincitore "piglia tutto": la conquista di una larga parte del mercato da parte di Office non significa che le alternative, seppur "perdenti", siano sparite, al contrario, quindi bisogna prestare attenzione a non interpretare il grafico 4.3 in modo troppo rigido.

Finora il feedback positivo di rete è stato considerato come un aspetto tipico dell'economia dell'informazione, in realtà in qualsiasi tipo di industria le economie di scala, che attraverso la riduzione del costo unitario di produzione stimolano ulteriormente le vendite, rappresentano una forma di feedback positivo; in questo contesto è più corretto definirle *economie di scala dal lato dell'offerta*[132] (*supply-side economies of scale*). Detto questo è anche vero però che in genere esse si esauriscono entro un certo livello produttivo, ben al di sotto di quello necessario a conquistare il mercato, difatti la maggioranza dei settori industriali mantiene posizioni concorrenziali o oligopolistiche, raramente monopolistiche; si può quindi affermare che nell'economia tradizionale il feedback positivo risulta limitato ad un certo livello, oltre il quale si corre il rischio di incorrere in feedback negativi.

Nell'economia dell'informazione invece la situazione è diversa tant'è che i monopoli sono più frequenti. La causa di ciò va cercata non tanto nella capacità di conseguire maggiori economie di scala dal lato dell'offerta, quanto nella presenza di forti *economie di scala dal lato della domanda* (*demand-side economies of scale*) ossia degli effetti di rete diretti, visti poco fa. Questi ultimi non si esauriscono al crescere delle dimensioni del mercato poiché maggiore è la popolarità di un prodotto o di una tecnologia maggiore è il valore che ne viene percepito. Il risultato è un *effetto incrociato* per cui l'aumento delle vendite riduce i costi produttivi ma non solo, rende anche il prodotto più attraente agli occhi dei consumatori, accelerando ancor di più la crescita della domanda: in pratica in questo contesto il feedback positivo si ripresenta sotto una veste nuova e più aggressiva[133].

Il grafico 4.4 illustra come facilmente in questa situazione, il successo si autoalimenti e generi altro successo mentre le imprese che non riescono a reagire velocemente, magari proponendo un nuovo prodotto con funzionalità

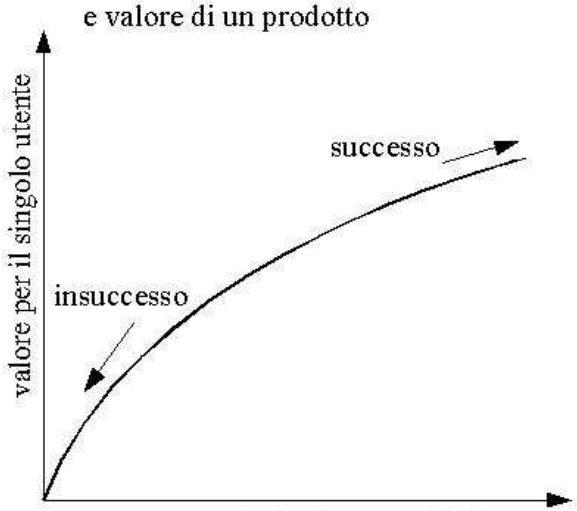
[130] La rilevanza assunta dagli effetti di rete in questi contesti suggerisce come in molti casi sia interessante oltre che utile analizzare le diverse strategie usate sia dai consumatori che dai produttori attraverso la teoria dei giochi e la determinazione dei relativi payoff.

[131] A titolo puramente informativo, una situazione analoga si ebbe anche nella caso disputa Vhs/Betamax: quando la prima venne lanciata sul mercato erano già stati venduti 100.000 videoregistratori in formato Betamax. La presenza di caratteristiche tecniche superiori e l'adozione da parte di più imprese del formato Vhs permise nell'arco di circa otto anni non solo di recuperare il gap esistente ma di vincere anche la guerra per divenire lo standard.

[132] Shapiro et al. (1999).

[133] Shapiro et al. (1999).

Grafico 4.4 Rapporto fra popolarità e valore di un prodotto



Fonte: Shapiro et al. (1999)

innovative in grado di attirare l'interesse degli utenti, siano destinate ad un inesorabile declino. Come nel grafico 4.3 anche qui si può ipotizzare che i prodotti o le tecnologie si trovino inizialmente nel punto intermedio della curva e che poi le aspettative del mercato influiscano in modo sempre maggiore sul successo o l'insuccesso degli stessi.

In questo caso è frequente il ricorso all'espressione "circolo virtuoso" per indicare che un prodotto molto diffuso acquista ancora più valore al crescere del numero di consumatori che ne fanno uso. Al contrario si parla di "circolo vizioso" per indicare la spirale negativa in cui il prodotto perde sempre più valore quanto più viene abbandonato dagli utenti della rete. Da quanto visto risulta quindi fondamentale riuscire ad innescare il feedback positivo, tuttavia per riuscirci spesso l'impresa deve effettuare alcune scelte di base che potranno portare a risultati molto differenti a seconda del contesto competitivo. Innanzitutto, supponendo che l'oggetto della questione sia una tecnologia, l'impresa dovrà trovare un compromesso tra compatibilità e performance ovvero dovrà scegliere se seguire o meno una traiettoria evolutiva proponendone una compatibile con quelle esistenti, che tuttavia difficilmente porterà l'impresa al dominio completo del mercato. Inoltre la scelta di compatibilità potrebbe trovare ostacoli insormontabili nell'impossibilità tecnica di realizzare la compatibilità stessa oppure in limiti legali a servirsi della tecnologia già esistente.

Questa strategia può portare al completo dominio del mercato tuttavia implica notevoli rischi poiché non è sufficiente possedere una tecnologia superiore per ottenere il successo ma è necessario anche convincere i potenziali utenti che il beneficio derivante dal miglioramento delle prestazioni superi i notevoli costi di transizione. In sostanza si presenta all'azienda la difficile scelta tra una competizione "per il mercato", ovvero per ottenere il dominio tecnologico e perciò possibili profitti monopolistici, e una competizione "nel mercato", dove l'adozione di uno standard comune permette di allentare la tensione competitiva assottigliando tuttavia i potenziali margini di profitto[134].

4.4 Lock-in e costi di transizione (Switching Cost)

I *costi di transizione* (switching cost) e il *lock-in* sono come due facce della stessa medaglia e senza dubbio rivestono un ruolo di primo piano nell'economia dell'informazione, in aggiunta agli altri aspetti (asimmetria nei costi e reti) visti nei precedenti paragrafi; per spiegare di cosa si tratta è anche qui utile nonché più pratico, ricorrere ad alcuni esempi. Dopo molti anni è normale voler sostituire la vecchia auto con una nuova: all'acquisto sarà pressoché indifferente decidere se restare su un prodotto della stessa marca o passare invece ad altri. D'altronde, tutte le auto soddisfano fondamentalmente il bisogno di movimento che ogni individuo ha, non richiedono molto tempo per prendere confidenza con i nuovi comandi o con il cruscotto, non comportano costi accessori rilevanti In linea di massima quindi rivolgersi o meno al vecchio fornitore è irrilevante.

Il discorso appena fatto non è però estendibile a tutti i prodotti, basti pensare ai computer. Se per anni si è utilizzato un computer Apple sarà più difficile decidere di passare ad un PC o ad una macchina Unix: il motivo è che probabilmente nel corso del tempo si sono acquistati beni complementari a quel sistema (ad esempio una stampante o del software) oppure si è utilizzato uno specifico formato per il salvataggio dei dati, tutti difficilmente riutilizzabili con le due alternative perché difficilmente compatibili. Con tutta probabilità anche la familiarità raggiunta con il vecchio computer diverrebbe inutile nell'uso di nuova macchina. Con il passaggio ad un nuovo prodotto con standard, hardware e software diversi da quelli usati finora, si devono quindi sopportare, oltre al costo d'acquisto della nuova macchina, dei costi non completamente ammortizzati relativi all'investimento in beni durevoli sostenuti per il precedente computer nonché tutta una serie di costi fissi irrecuperabili come ad esempio il tempo impiegato

[134] Bensen et al. (1994)

per apprendere il funzionamento del nuovo computer (il cosiddetto wetware). E' ovvio che una persona razionale difficilmente opterà per il passaggio ad un computer di marca diversa, è più facile invece che la sua scelta, forzata dalle osservazioni fatte, ricada sulla vecchia marca.

I costi sostenuti per cambiare auto o passare ad una tecnologia informatica diversa sono in entrambi i casi dei costi di transizione (switching cost) tuttavia mentre nella prima fattispecie essi sono limitati al costo d'acquisto nella seconda appare chiaro come essi siano più estesi e vadano ad incidere sulle decisioni d'acquisto. Quando gli switching cost sono rilevanti, e questo accade spesso nel passaggio da una tecnologia informatica ad un'altra, si dice che gli utenti subiscono il fenomeno del lock-in[135]. Se ci si pensa un attimo non è affatto raro imbattersi nel lock-in; che ci si voglia credere o meno ogni utente del sistema operativo Windows, subisce questo fenomeno, basti pensare alla dipendenza che si ha verso Microsoft in termini di assistenza, di aggiornamenti oppure di compatibilità con software di altre aziende, per non parlare dell'addestramento che si dovrebbe seguire passando ad un altro sistema, quale Linux o Os X di Macintosh.

Qualche anno fa proprio Microsoft decise di interrompere ogni iniziativa commerciale e promozionale relativa al pacchetto Office non solo aumentando i costi delle licenze ma applicando anche delle incompatibilità forzate con le vecchie versioni. Nonostante questa strategia potesse sembrare azzardata le vendite non calarono perché la diffusione del pacchetto e dei relativi formati per il salvataggio dei dati era così ampia che chiunque avrebbe preferito rinnovare la licenza anziché sopportare i costi di una eventuale riconversione verso altri software[136]: i consumatori si trovarono in pratica alla mercè del fornitore. Si può quindi affermare che i costi di transizione misurano fino a che punto un consumatore dipende da uno specifico fornitore. Occorre sottolineare però che anche i fornitori concorrenti sostengono una parte dei costi di transizione totali quando acquisiscono nuovi clienti, alcuni ininfluenti come l'immissione dei loro dati in un database, altri rilevanti come la necessaria creazione di un servizio di supporto alla clientela[137]. Nella tabella seguente sono riportate le situazioni di lock-in più comuni e i relativi switching cost.

Tipo di Lock-in	***Switching cost associati***
Impegni contrattuali	Oneri di liquidazione o di compensazione
Acquisti di beni durevoli	Sostituzione delle attrezzature (computer e beni complementari). Tendono a diminuire con il passare del tempo a causa dell'ammortamento.
Addestramento specifico	Apprendimento di un nuovo sistema informativo, costi diretti e in termini di perdita di produttività. L'introduzione di modelli dalle caratteristiche superiori, il livello di familiarità con il vecchio software e il passare del tempo incidono sulla crescita dei costi
Informazioni database	Conversione dei dati ai nuovi formati. Tendono ad aumentare con la mole di dati
Fornitori di beni specifici	Finanziamento di nuovi fornitori. Possono crescere col tempo se le caratteristiche dei beni richiesti sono difficili da trovare o mantenere
Costi di ricerca	Costi di ricerca complessivi (dei clienti e del fornitore). Comprendono la necessità di raccogliere informazioni sulla qualità delle alternative
Programmi di fidelizzazione	Ogni beneficio perduto dai vecchi fornitori, più la possibile necessità di ricostituire i crediti accumulati.

Tabella 4.5 Tipi di Lock-in e Switching Cost associati
Fonte: Shapiro et al. (1999)

4.4.1 L'arma del lock-in

[135] Il lock-in può essere dovuto anche ad un'altra causa: l'utente infatti può decidere di non passare a una nuova tecnologia superiore perché quella vecchia, avendo una maggiore base d'utenza, genera maggiori benefici dovuti alle esternalità di rete. Gli switching cost elevati restano comunque la causa principale.

[136] Nel caso del software ad accentuare il lock-in spesso contribuisce il *bundling,* cioè una forma particolare di versioning che prevede l'offerta congiunta di due o più prodotti distinti in un unico pacchetto, caratterizzato da un prezzo complessivo inferiore alla somma dei prezzi dei singoli programmi.

[137] I costi associati alla transizione del generico consumatore C, dal fornitore A al fornitore B, sono il costo che deve essere sostenuto da C e da B, collettivamente, per porre il consumatore in una posizione, nei riguardi del fornitore B, confrontabile con quella che il consumatore C ha correntemente nei confronti del fornitore A; solo se il fornitore B ha già dei margini di profitto elevati c'è la possibilità di attenuare l'effetto negativo dei costi di transizione totali.

Se è vero che il lock-in costituisce un problema frequente per i consumatori o comunque per chi usufruisce di una certa tecnologia, è altrettanto vero però che esso rappresenta un formidabile strumento di potere nelle mani delle imprese fornitrici. Riflettendoci un attimo, per evitare che i clienti possano decidere di "migrare" a prodotti o tecnologie concorrenti quale soluzione migliore se non quella di promuovere il lock-in? Tutto sommato, si può dire che esso sia parzialmente "fisiologico" tuttavia è facile constatare come spesso le incompatibilità fra prodotti o tecnologie concorrenti vengano accentuate, se non addirittura create ad hoc dalle imprese, per influire negativamente sui costi di transizione dei propri clienti che, in definitiva, risultano superiori ai potenziali benefici di una transizione.

Accade spesso che passando da una compagnia telefonica ad un'altra si debbano sostenere, fra i costi di transizione, anche i cosiddetti *costi di interruzione*: i tempi tecnici necessari a cambiare operatore sono abbastanza limitati, tuttavia gli operatori hanno la "buona abitudine" di allungarli volutamente, da un lato a svantaggio dei consumatori per scoraggiarne il cambiamento, dall'altro come barriera all'entrata per i concorrenti. Questa situazione è particolarmente frequente nel passaggio dal fornitore che detiene il monopolio dell'infrastruttura di comunicazione, com'è ad esempio Telecom Italia, ad una compagnia più piccola che fornisce solamente il servizio telefonico; difficilmente l'attuale fornitore del servizio (Telecom) sarà disposto a concedere una transizione veloce ed economica verso i propri concorrenti; se lo facesse rinuncerebbe infatti ad uno dei numerosi vantaggi che gli derivano dal semplice fatto di detenere una posizione di monopolio o quasi.

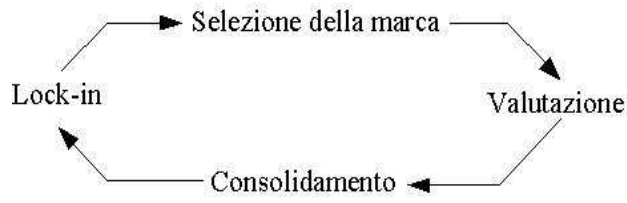
L'esempio dei costi di interconnessione è uno dei migliori, tuttavia casi simili sono facilmente riscontrabili anche nell'ambito del software dove il lock-in viene ottenuto prevalentemente con l'adozione di tecnologie proprietarie che scoraggino da un lato gli stessi consumatori e dall'altro le aziende concorrenti che si troverebbero magari a pagare delle pesanti royalties per l'utilizzo delle stesse. In sostanza, dal lato dei produttori concorrenti gli switching cost rappresentano sotto vari punti di vista delle ottime barriere all'entrata cosa ancora più accentuata nei mercati di massa. In genere tanto più un'azienda detiene una quota di mercato rilevante (e questa è la configurazione tipica del mercato del software) tanto più avrà l'interesse a promuovere e sostenere in modo artificioso il lock-in poiché, così facendo, potrà conservare o aumentare più facilmente le proprie quote e i propri profitti (in genere a discapito della qualità del prodotto).

4.4.2 Ciclo di lock-in

Da quanto si è visto si può affermare che il lock-in è un concetto fondamentalmente dinamico che ha origine dagli investimenti fatti in passato e dalle necessità che si manifestano nel corso del tempo. Il processo che lo determina è ciclico ed è riassumibile in quattro fasi:

- *selezione della marca*: il ciclo di lock-in inizia nel momento in cui il consumatore deve decidere a quale marca di un certo prodotto affidarsi; alla prima scelta egli non sarà influenzato da alcuna considerazione sui probabili costi di transizione e lock-in futuri, questi ultimi infatti si manifesteranno solo in seguito ad essa.

Figura 4.6 Ciclo di Lock-in



Fonte:Shapiro et al. (1999)

- v*alutazione del prodotto.*

- *consolidamento*: in questa fase il consumatore si abitua al prodotto, sviluppa una preferenza verso lo stesso e verso gli investimenti complementari. I produttori cercano di prolungare più possibile questa fase sperando che gli switching cost del cliente aumentino sempre più.

- *Lock-in*: la fase di consolidamento, appena descritta, sfocia nel lock-in, per cui gli switching cost divengono insostenibili. Il ciclo si chiude e ricomincia tuttavia, dopo la prima chiusura. A questo punto il consumatore si trova a

dover prendere in considerazione gli eventuali costi di transizione prima di compiere una nuova scelta.

Dalla breve analisi compiuta appare chiaro come il lock-in sia inevitabile tuttavia in alcuni in alcuni testi[138] relativi all'economia dell'informazione sono reperibili dei consigli su come gestirlo e cercare di prevenirlo. Nel caso specifico del software questi consigli sono sintetizzabili in tre punti:

- valutare le possibili difficoltà future nella compatibilità dei dati con altri formati
- valutare la possibilità che in futuro ci si leghi ad un unico produttore
- cercare di stimare i probabili costi di transizione futuri.

Inoltre, siccome al momento della scelta del prodotto il lock-in non si è ancora realizzato, l'utente gode di fatto di una maggiore forza contrattuale sulla quale dovrebbe far leva per negoziare condizioni più favorevoli, volte a limitare i futuri costi di transizione. Indubbiamente queste osservazioni possono essere utili se l'utente è un'azienda, oltretutto di certe dimensioni, tuttavia è difficile pensare che il signor Rossi detenga una forza contrattuale tale da poter negoziare a proprio favore condizioni migliori di fronte a società che detengono ampie quote di mercato (è il caso di Microsoft o IBM). Questa situazione di forte dipendenza dei consumatori nei confronti delle società produttrici di software dura da anni ora però una soluzione valida esiste: l'Open Source.

4.5 Effetti economici dell'Open Source

L'impatto che il software Open Source può avere sull'industria del software sia in relazione all'ambito pubblico che a quello privato è notevole in quanto agisce su più fronti. Gli effetti più importanti si hanno sulla struttura del mercato, sulla qualità dei prodotti offerti in esso, sui vecchi modelli di business usati finora, infine sui costi. Per cercare di compiere un'analisi accurata e per comodità si divideranno le relative analisi su più paragrafi.

4.5.1 Lock-in e struttura del mercato

Nei paragrafi precedenti si è visto come il mercato del software commerciale sia caratterizzato da dei fenomeni (asimmetria dei costi ed esternalità di rete) che ne favoriscono la concentrazione e quindi l'assunzione di forme oligopolistiche o monopolistiche; esse accentuano il problema del lock-in e determinano tutta una serie di ovvie inefficienze, sia economiche che tecniche. I produttori, disponendo del codice sorgente e adottando formati proprietari per la gestione e il salvataggio dei dati (che innalzano i costi di transizione e impediscono l'ingresso di potenziali concorrenti), non hanno alcun incentivo a creare un prodotto perfetto o semplicemente buono, basta infatti che esso funzioni quel tanto che serve, non solo per quanto riguarda alcune caratteristiche generiche come la velocità e l'affidabilità ma anche in relazione alle funzionalità offerte.

Il software Open Source cambia radicalmente questa situazione e apporta tutta una serie di benefici, sia all'industria che agli utenti. Innanzitutto, la disponibilità del codice sorgente e l'adozione di interfacce aperte consente di abbattere fortemente i costi di transizione e quindi di limitare il fenomeno del lock-in. Per capire come ciò avvenga, occorre spiegare brevemente il ruolo e l'importanza delle interfacce nei sistemi informatici. Nessun componente, sia hardware che software, di un prodotto è in grado di operare autonomamente ma necessita di uno strumento per poter interagire con tutti gli altri componenti e con l'utente; questo strumento è appunto l'interfaccia. In questo contesto se ne distinguono tre diversi tipi[139] tuttavia quello più noto è il formato dei documenti che specifica ai programmi come interpretare i dati salvati nel documento stesso.

Se un'interfaccia è aperta significa che le sue specifiche tecniche sono state divulgate pubblicamente e ciò permette di inserirle liberamente all'interno di un qualsiasi codice sorgente; il risultato non è solo la piena compatibilità di qualsiasi software con quella interfaccia ma in genere anche la sua trasformazione in uno standard[140]

---

[138] Shapiro et al. (1999)

[139] Le interfacce informatiche si suddividono in APIs (Application Programmer Interfaces), in interfacce utente (gli strumenti grafici usati per gestire il computer) e nei formati di salvataggio dei documenti.

[140] Un ottimo esempio è fornito dal formato XML utilizzato da OpenOffice che grazie al fatto di essere "aperto" garantisce una piena interoperabilità e sta diventando sempre più uno standard di riferimento. Recentemente anche il programma IDA (Interchange of Data Between Administrations) dell'UE ne ha consigliato l'adozione. http://europa.eu.int/ida/en/document/2604

di riferimento.
Un aspetto molto positivo del software Open Source e dei suoi formati aperti è quindi quello di provocare il passaggio da una competizione *per* lo standard, com'è stato finora, ad una competizione *all'interno* di uno standard[141]. L'apertura del codice sorgente e delle interfacce determina infatti una forte riduzione dei costi di produzione (che nel modello proprietario sono assai elevati) grazie allo sviluppo congiunto con la comunità inoltre permette a potenziali nuovi fornitori di accedere più facilmente al mercato, il tutto a vantaggio della qualità (si concentrano le risorse a favore delle innovazioni e del miglioramento tecnico anziché disperderle in progetti opposti), delle funzionalità e perché no anche del prezzo. La forte dipendenza degli utenti, garantita finora ai produttori dal modello proprietario, viene meno dato che la competizione non avviene più tanto al livello dello sviluppo e della vendita di software ma piuttosto a quello superiore, dei servizi e delle attività complementari ad esso. Il predominio nel mercato non è più assicurato da posizioni dominanti o limitazioni tecniche ma da una migliore qualità dei servizi offerti.

4.5.2 La qualità del software

Uno dei maggiori benefici che si ottiene dall'apertura del codice sorgente e delle interfacce riguarda la flessibilità cioè la possibilità di personalizzare e adattare il software alle proprie esigenze. Gli utilizzatori di software si trovano spesso a dover fronteggiare l'inadeguatezza dei programmi confezionati, disponibili a "basso" prezzo ma assolutamente inadatti a soddisfare le loro specifiche necessità. Essendo molto minore che in passato, l'offerta di know-how e soluzioni personalizzate non è sufficiente a coprire questa inadeguatezza e soprattutto le imprese possono vedersi costrette a riportare all'interno alcuni dei processi precedentemente esternalizzati. Il caso classico è quello dell'azienda che abbia bisogno di un software con certe funzionalità ma che questo diventi improvvisamente indisponibile a causa dell'assenza di potenziali fornitori o dell'obsolescenza delle versioni preesistenti. La scelta, in questi casi, è fra lo sviluppo del software necessario o la chiusura dell'azienda di conseguenza ecco che la stessa potenzia il proprio CED[142] e lo dedica prevalentemente alla produzione di quella soluzione che è indispensabile alla propria sopravvivenza.

Si tratta di una attività non strettamente legata al core business (dato che l'adeguamento delle risorse informatiche va vista come strumentale allo stesso) ma che l'inadeguatezza costringe a svolgere internamente. L'azienda poi non ha interesse a vendere sul mercato queste soluzioni, per quanto sofisticate e potenzialmente cariche di un valore economico esse siano. Farlo richiederebbe consulti legali, una gamma di investimenti e di competenze che sono estranei all'attività per non parlare poi della necessità di coprire i costi che questa scelta implica. E' in situazioni di questo genere che il modello Open Source dimostra la sua validità, da un lato infatti esso offre la flessibilità tanto desiderata, dall'altro permette di limitare gli eventuali costi di sviluppo: stabilito che sarebbe pressoché impossibile vendere il software prodotto, risulta più conveniente regalarne il codice sorgente.

Attraverso Internet è possibile cercare altre imprese, o alle volte singoli privati, che siano interessati a produrre lo stesso software o software con caratteristiche simili. Si creano così delle "joint venture" informali che raccolgono anche gruppi di imprese alla ricerca di una soluzione comune; se poi si decide di pubblicare il progetto e di renderlo Open Source, ecco che una vasta platea di utenti/sviluppatori può contribuire, come si è già visto, al suo sviluppo. Quanto costerebbe avere un gruppo di test e sviluppo così vasto? Per una impresa il cui core business non è nemmeno la produzione del software sarebbe un costo insostenibile. Un caso particolarmente adatto a esemplificare quanto detto finora sulla frequente carenza di flessibilità è quello di XHarbour[143]. Con l'avvento dei Personal Computer nacquero strumenti di gestione simili agli odierni pacchetti di produttività aziendale come ad esempio i fogli di calcolo e i database. Uno dei più diffusi fra questi fu il sistema DBase III che disponeva di un semplice linguaggio di programmazione molto adeguato a scrivere proprio quei programmi di gestione e di un compilatore che ne sfruttava appieno le potenzialità, il Clipper.

L'abbinamento dei due rese il linguaggio una delle piattaforme di sviluppo preferite dalle piccole software house che lavoravano su commessa: il Clipper era infatti estremamente adeguato a produrre software orientato alla

[141] Shapiro et. al. (2003)
[142] Centro Elaborazione Dati.
[143] http://www.xharbour.org/

gestione aziendale per il sistema operativo DOS che era l'architettura dominante all'epoca. L'avvento del sistema operativo Microsoft Windows 95, incompatibile con il Clipper, colse impreparati i suoi sviluppatori di conseguenza tutte le software house che lo avevano adottato come sistema di sviluppo si trovarono con competenze e conoscenze acquisite improvvisamente obsolete. Alcuni produttori cercarono di migliorare quegli aspetti del Clipper che non erano adeguati ai nuovi sistemi operativi generando una serie di "dialetti" che vanno oggi sotto il nome di XBase.

Le imprese che avevano maturato la loro esperienza attraverso il linguaggio Clipper e che avevano un patrimonio di codice, si trovavano nella condizione di vedere da una parte il loro patrimonio depauperarsi a causa della diminuita compatibilità con i programmi meno recenti soffrendo contemporaneamente una crescente obsolescenza delle proprie conoscenze e, dall'altra, una diminuita capacità di fare fronte alle esigenze della propria clientela. A questo punto, l'unica possibilità di sopravvivere era quella di realizzare un dialetto XBase completamente compatibile con il Clipper ma allo stesso tempo capace di incorporare tutte le funzionalità disponibili nelle moderne architetture software e di poter sfruttare quelle che sarebbero venute in futuro. Nessuno degli sviluppatori era in grado di realizzare un simile sistema da solo ma una comunità di piccole imprese e operatori autonomi che erano in contatto, per condividere le loro conoscenze sul Clipper e su Xbase, decise di tentare l'impresa sviluppando un prodotto Open Source: nacque così il progetto Harbour che a distanza di dieci anni continua ad esistere. Sebbene finora si sia ragionato in termini di aziende, in realtà il problema della flessibilità interessa da vicino anche gli utenti privati, basti pensare al caso Fetchmail visto nel terzo capitolo[144].

Un altro aspetto da considerare è l'autonomia che l'Open Source permette di avere dai fornitori sia in termini di assistenza che di aggiornamento del software. La disponibilità del codice sorgente consente innanzitutto di rivolgersi a soggetti via via diversi per l'assistenza e le altre soluzioni informatiche. Se si adottasse come sistema operativo Windows di Microsoft, l'unico soggetto a cui ci si potrebbe rivolgere per l'assistenza sarebbe Microsoft, al contrario un software come GNU/Linux mette l'utente nella posizione di poter scegliere, una volta acquisito il sistema operativo, fra i diversi fornitori di servizi informatici quello che offre il migliore servizio e il migliore prezzo. Inoltre egli non è costantemente costretto ad acquistare gli upgrade (aggiornamenti) messi sul mercato dalla casa produttrice ma anzi può fare affidamento, in caso di necessità, nella comunità pubblica di lavoro che è in grado di affiancare l'impresa nel processo di correzione degli errori e di ampliamento del codice.

4.5.3 Modelli di business

Finora l'industria del software commerciale si è sempre distinta per l'adozione di un modello nel quale i proventi sono direttamente correlati ad un certo prezzo “per copia” e la vendita non ha per oggetto la proprietà del software bensì il diritto al suo utilizzo, secondo i termini della relativa licenza d'uso. I motivi per cui le software house lo hanno scelto sono due: il primo è che esso riserva al fornitore un ampio controllo sulle modalità di utilizzo del software consentendo quindi di offrire fasce di prezzo diverse in base ai diversi tipi di utenza; il secondo è che mantiene il prezzo della licenza indipendente da quello degli eventuali servizi accessori ad essa legati, come il supporto tecnico e la consulenza, per cui le aziende riescono a distinguere le entrate derivanti dalle semplici licenze che si ottengono ad ogni cessione del software da quelle inerenti ai servizi accessori che invece si conseguono nel tempo. Inoltre, sempre per il fatto di non essere legato ai servizi erogati, il prezzo delle licenze può essere negoziato con i clienti in base al valore del software da essi percepito che potrebbe dipendere ad esempio dai soldi che lo stesso può far risparmiare a loro. Tanto più questo risparmio è consistente tanto maggiore sarà il valore percepito.

Dato che il costo marginale di riproduzione di un software è relativamente basso, ipotizzando che il cliente e il mercato nel suo complesso attribuiscano un valore piuttosto alto ad un prodotto software, a parità di altre condizioni, il margine lordo sulle licenze risulta di solito più alto del margine lordo sui servizi, determinando così maggiori profitti per l’azienda. Occorre sottolineare però che l'eccessivo affidamento sul valore di vendita come fonte di reddito può portare a dei “dislivelli” delle entrate nel tempo (specialmente quando il numero dei clienti è relativamente basso e l’entrata media attribuibile a ciascuno di essi è alta), oltre ad esporre l'azienda ai pericoli di

[144] I programmi per la gestione delle posta elettronica non consentivano di inoltrare agevolmente le e-mail al computer di casa così Raymond ne modificò uno esistente, introducendo questa specifica funzione e creando attorno al progetto Fechmail una comunità di sviluppo.

una concorrenza, in termini di prezzi inferiori, da parte di aziende rivali: da queste brevi considerazioni sono sorte molte critiche a questo modello. Secondo molti infatti la ragione del successo e della crescita dell'Open Source non è solo dovuta alla maggiore efficienza del processo di sviluppo ma anche, e soprattutto, dall'aver capito che il software è essenzialmente un servizio piuttosto che un bene in senso stretto. Fra gli oppositori spicca Eric Raymond.

Egli parla a questo proposito di "*illusione manifatturiera*"[145] riferendosi con questo termine alla convinzione, diffusa quanto errata, che il software condivida le caratteristiche di valore di un tipico bene manifatturiero. Egli sostiene che i programmi informatici, come tutte le altre classi di strumenti o beni capitali, abbiano due tipi distinti di valore economico, un *valore d'uso* e un *valore di vendita*; il valore d'uso di un programma è il suo valore economico in quanto strumento mentre quello di vendita è il suo valore in quanto articolo commerciabile. L'illusione manifatturiera si baserebbe su due presupposizioni, condivise sia dagli utenti che dai produttori, secondo le quali il lavoro degli sviluppatori viene retribuito in base al valore di vendita del software e tale valore è proporzionale, ai costi di sviluppo sostenuti per la produzione e al suo valore d'uso. Per smentire queste due ipotesi Raymond fa notare innanzitutto che il codice scritto per la vendita non è che "*la punta dell'iceberg*" rappresentato dalla programmazione. La maggior parte del software diffuso in tutto il mondo è infatti scritto all'interno delle aziende, per un uso interno o in quanto componente di un altro prodotto (il cosiddetto software *embedded*[146]). Inoltre sottolinea come:

"*Gran parte di questo codice, prodotto a livello aziendale, è integrato nel suo ambiente attraverso modalità che rendono molto difficili il riutilizzo e la copia (ciò è vero sia che per "ambiente" si intenda una serie di procedure interne agli uffici di un'impresa, sia che si intenda il dispositivo di iniezione del carburante in una mietitrebbia). Perciò, con il variare dell'ambiente, occorre sempre molto lavoro per l'adattamento del software. Si tratta della "manutenzione" che, come sa qualsiasi ingegnere o perito informatico, rappresenta la stragrande maggioranza (più del 75%) del lavoro per cui sono pagati i programmatori. Di conseguenza, il programmatore passa gran parte del suo tempo (e guadagna gran parte del suo salario) scrivendo e aggiornando codice aziendale che non ha alcun valore di vendita* (Raymond, 1999b)".

Fase dello sviluppo	***Quota dei costi (%)***
Requisiti	3
Progetto	8
Programmazione	7
Testing	15
Manutenzione	67

Tabella 4.7 Costi di sviluppo suddivisi per fase

Fonte: Bloor Research

Per quanto la quota del 75% possa sembrare eccessiva anche altri dati, oltre alla tabella 4.7, sembrano confermare tali affermazioni: in una relazione preparata da una commissione extra-parlamentare per conto del Ministero dell'Innovazione[147] sono riportati alcuni risultati piuttosto significativi. Nel 2001 la Pubblica Amministrazione Centrale[148] ha speso 405 milioni di euro, ossia il 21 % del totale delle spese in IT, in acquisti, manutenzione, leasing e sviluppo di software. Di tale importo la suddivisione degli impieghi vede una netta rilevanza percentuale dello sviluppo del software ad hoc e della relativa manutenzione, pari al 61% del totale. Inoltre in tale relazione è riportato

[145] Raymond (1999b).

[146] Il termine embedded (dall'inglese *incastrato*), sta ad indicare il software destinato a funzionare come componente all'interno di altri prodotti, in particolare dei dispositivi portatili come telefoni cellulari o telecomandi, (Linux è molto utilizzato nei cellulari e nei palmari) ma anche negli elettrodomestici e nelle automobili. Pur rappresentando il volto meno noto dell'industria del software ne costituisce in realtà il segmento più grande e importante. Raymond afferma che lo sia, circa il 95% del software creato, secondo altri invece quest'ultimo raggiunge una quota di poco superiore al 50%.

[147] Ministero per l'Innovazione e la Tecnologia (2003).

[148] Si distingua la Pubblica Amministrazione Centrale (PAC) dalla Pubblica Amministrazione Locale.

un progetto di spesa per un progetto di gestione documentale, protocolli informatici e workflow[149] il cui importo era stimato in 80 milioni di euro su cui le personalizzazioni avrebbero pesato in una percentuale variabile tra il 35 % e il 45 %. Anche l'altra ipotesi secondo cui il valore del software dipende dai costi di sviluppo e dal suo valore d'uso si dimostra falsa:

"*In secondo luogo, la teoria per cui il valore di vendita di un software è collegato ai suoi costi di sviluppo o sostituzione si sfata ancora più facilmente, esaminando il comportamento stesso dei consumatori. Sono molti i beni per cui una proporzione di questo genere regge (prima della svalutazione): cibo, automobili, strumenti meccanici. Ci sono anche molti beni non tangibili il cui valore di vendita è strettamente collegato ai costi di gestione e sostituzione: i diritti di riproduzione di musica, cartine o banche dati, per esempio. Tali beni possono mantenere o addirittura aumentare il proprio valore di vendita, una volta uscito di scena il produttore originale. Al contrario, quando un produttore software cessa l'attività (o se il prodotto sta andando fuori produzione) il prezzo massimo pagato dal consumatore andrà rapidamente verso lo zero, a prescindere dal valore d'uso teorico o dai costi di sviluppo di un equivalente funzionale (per verificare questa affermazione, basta osservare gli scatoloni delle rimanenze al negozio di software più vicino a casa vostra).*

Il comportamento dei rivenditori, quando un produttore chiude, è assai rivelatore. Ci fa capire che i rivenditori sanno qualcosa che i produttori non sanno. Ciò che sanno è che il prezzo pagato dal consumatore è determinato, in realtà, dalle previsioni sul valore futuro dei servizi del produttore (dove "servizi" comprende, in questo caso, personalizzazioni, aggiornamenti e progetti in progressione). In altre parole, il software è in gran parte un'industria di servizi che opera sotto la persistente, ma infondata illusione che si tratti di un'industria manifatturiera (Raymond, 1999b)".

Il motivo per cui, normalmente, si tende a credere il contrario è che questa piccola porzione dell'industria del software, che produce per la vendita, è anche la sola a pubblicizzare i suoi prodotti (i quali si caratterizzano oltre che per una più ampia visibilità, soprattutto per l'assenza o la scarsità di personalizzazione e servizi di assistenza). Tra le principali conseguenze negative dell'illusione manifatturiera vi è il suo sostegno ad un assetto dei prezzi "patologicamente scollato" rispetto alla reale scomposizione dei costi di sviluppo[150]. Se infatti quest'ultimi sono in realtà in gran parte legati alla manutenzione, la comune politica dei prezzi, basata sull'imporre elevati prezzi fissi di acquisto e contributi per l'assistenza decisamente bassi, è destinata a produrre risultati deludenti sia per i produttori che per i consumatori.

La concezione del modello tradizionale infatti, adeguandosi a quello della fabbrica anziché a quello dei servizi, disincentiva il produttore dall'offrire un servizio di alto livello e competente, privilegiando la produzione e la vendita del software. In questo modo l'ufficio assistenza, non essendo interpretato come fonte di reddito, tenderà a trasformarsi in "*una discarica per i meno efficienti e avrà soltanto le risorse strettamente necessarie a non alienarsi un numero eccessivo di clienti*[151]". Per i produttori invece la quasi totale dipendenza dalle entrate dell'attività di vendita può portare, qualora il mercato si trovasse in una fase di recessione, a consistenti tagli delle spese di assistenza. Ciò provocherebbe la fuga dei consumatori verso la concorrenza dal momento che verrebbe a mancare quella parte fondamentale del valore del prodotto costituito dal valore futuro dei servizi forniti dal produttore.

Se nel breve periodo l'azienda può ovviare a questa condizione rilasciando frequentemente degli aggiornamenti al proprio software, che correggono gli errori di programmazione (i cosiddetti *bug-fix*), e facendoli passare per nuovi prodotti, con un prezzo nuovo[152], nel lungo periodo l'unica ancora di salvezza è costituita dall'assenza completa di concorrenza e dalla presenza di un mercato monopolizzato. A tal proposito Raymond fa giustamente notare come queste condizioni si siano verificate spesso nella storia dei sistemi operativi proprietari per il personal computer, dei word processor, dei programmi di contabilità e del software per l'impresa in generale, determinando una dinamica di mercato in cui "il vincitore piglia tutto" e non necessariamente si impongono i prodotti o gli standard qualitativamente migliori.

---

[149] Il workflow è definibile come l'automazione di una parte o dell'intero processo aziendale dove documenti, informazioni e compiti vengono passati da un partecipante ad un altro per ricevere qualche tipo di azione, seguendo un determinato insieme di regole.

[150] Raymond (1999b).

[151] Raymond (1999b).

[152] Raymond (1999b).

Constatata l'inadeguatezza del modello di business finora usato, diventa necessario considerare il software secondo uno schema che ponga le sue basi sul valore d'uso e sulla dicotomia tra servizio e costo, bisogna cioè concepire il software come un servizio e trattarlo come tale. Se l'approccio dei profitti derivanti sul valore di vendita è fortemente basato sugli obblighi e sulle restrizioni (d'uso, di ridistribuzione e di modifica) prescritte dalle licenze d'uso tradizionali, occorre passare ad un modello come quello Open Source dove esse siano impedite o quanto meno limitate. Partendo da questa considerazione è possibile analizzare seppur brevemente alcuni di questi nuovi modelli.

4.5.3.1 Branding and distribution

Uno dei modelli più adottati è il "*branding and distribution*", più noto con il semplice termine di *distribuzione.* Esso consiste nel creare ed offrire ai potenziali clienti una raccolta di vari prodotti Open Source, sotto un marchio noto ed affidabile, attraverso canali di distribuzione diversi. Alla fornitura della distribuzione si accompagna poi l'offerta di un supporto post-vendita, di un'assistenza per l'installazione del prodotto e della relativa manualistica. L'esempio più comune è quello delle distribuzioni basate sul kernel Linux, viste nel paragrafo 1.6. che offrono una vasta scelta di programmi Open Source, in gran parte appartenenti al progetto GNU e quindi sottoposti a licenza GPL. L'azienda che più di tutte le altre si è distinta per l'adozione e la diffusione di questo modello è la Red Hat Software Inc., che è riuscita non solo a conquistare una grossa fetta, sia del segmento server che di quello desktop, ma anche a quotarsi in borsa.

I fattori chiave delle distribuzioni sono innanzitutto la costituzione di un marchio forte al quale i clienti associno un'idea di coerenza, qualità e affidabilità nonché l'offerta di un'ampia serie di servizi di supporto e assistenza dall'alto valore aggiunto, aventi un'importanza ben maggiore della semplice vendita del prodotto dato che quest'ultimo può anche essere reperito gratuitamente o ad un costo esiguo, secondo i dettami della GPL. Uno dei fondatori della società, Robert Young, spiega il modello paragonando il software Open Source alle materie prime:

"*Red Hat non vende licenze sulla proprietà intellettuale di cui non è titolare. Non è questo il modello economico che supporterà i nostri clienti, i nostri dipendenti e i nostri azionisti. La domanda allora è: in quale business ci collochiamo? Per rispondere, ci siamo guardati intorno per trovare, fra le altre industrie, quella a noi più affine. Cercavamo un'industria in cui le materie prime fossero gratis, o almeno liberamente disponibili. Abbiamo considerato l'industria delle materie prime e cominciato a riconoscere alcune idee.* (..) *Tutte le maggiori società che vendono prodotti primari inclusa l'acqua in bottiglia (Perrier o Evian), il sapone (Tide) o la pasta di pomodoro (Heinz), basano le loro strategie di marketing sulla costruzione di marchi forti.*

Questi marchi devono essere sinonimo di qualità, coerenza e affidabilità (..) *Ecco l'opportunità di Red-Hat: offrire convenienza, qualità e soprattutto contribuire a definire, nella mente dei clienti, che cosa può essere un sistema operativo.* (..) *Per bere acqua, nella maggior parte dei paesi industrializzati, basta solo aprire il primo rubinetto che capita: e allora perché Evian vende acqua di rubinetto francese per milioni di dollari su quei mercati? Sotto sotto, sfrutta la paura, largamente irrazionale, che dell'acqua di rubinetto non ci si possa fidare* (..) *Gioca a vantaggio di Evian il fatto che l'umanità quasi per intero beva acqua: noi dobbiamo ancora creare una quantità di consumatori Linux prima di avere un mercato in cui vendere il nostro marchio* (Young, 1999)."

La gratuità stessa del prodotto non è destinata a ritorcersi contro questo modello di business bensì a rivelarsi un indubbio fattore di penetrazione del mercato; le distribuzioni sono prima di tutto degli enti commerciali il cui scopo è quindi il profitto. Il fatto che traggano profitto anche da un qualcosa che non è frutto della loro attività ma del contributo dell'intera comunità che ruota intorno al software libero, ha fatto spesso storcere il naso ai suoi rappresentanti più intransigenti; è infatti piuttosto facile essere tacciati di "arricchirsi con il lavoro degli altri". Diventa quindi fondamentale, per le società di distribuzione, fornire il proprio contributo allo sviluppo del software libero attraverso sovvenzioni alle comunità e agli enti o l'impiego di uomini di garantita professionalità nel settore (remunerati dalle società stesse). Oltre a salvaguardare l'immagine dell'azienda, ne vengono in tal modo garantiti gli sviluppi futuri in quanto il lavoro degli sviluppatori contribuisce al miglioramento del prodotto. Inoltre, così facendo gli eventuali acquirenti della distribuzione avranno la certezza di contribuire attivamente al bene di tutta la comunità,

cosa che non avrebbero eventualmente potuto fare perché non essi stessi dei programmatori.

Infine, occorre precisare che lo stretto rapporto di collaborazione con la comunità non contribuisce solo allo sviluppo di quel progetto specifico (nel caso di Red Hat, il sistema operativo Linux) ma potenzialmente anche ai progetti degli sviluppatori indipendenti, aventi ad oggetto software complementari come ad esempio quelli destinati alla uso in Rete (browser o client di posta), i programmi di grafica e quelli multimediali (giusto per indicare i più noti). La possibilità di garantire l'accesso e il controllo agli strumenti software anche a tali sviluppatori non fa altro che accrescere il valore del software distribuito e il valore percepito dai suoi utenti: il valore della rete costituita dalla base d'utenza cresce data la varietà di software disponibili.

4.5.3.2 Best knowledge here

Questo modello è basato esclusivamente su quella che è risultata essere la componente più importante dei nuovi modelli di business introdotti dal software libero: la fornitura di servizi che possono assumere diverse forme, dalla consulenza alla formazione, fino al supporto tecnico e alla certificazione dei prodotti Open Source. La particolarità di questo modello è che tramite l'utilizzo del software libero vi è la possibilità di competere nel mercato senza dover sopportare alti costi di entrata e di conseguire un'elevata specializzazione. In sostanza è una nicchia in cui possono entrare tutti, ognuno con il suo piccolo "segmento" di consulenza. Oltretutto, l'utilizzo di strumenti liberi permette alle aziende operanti in questo settore di essere sempre aggiornate e pronte alle nuove richieste del mercato.

La condizione su cui ruota l'intero modello è la competenza dei propri dipendenti, organizzata in modo tale da assicurare un servizio affidabile e di eccellenza; essa è di tipo orizzontale cioè non focalizzata su uno o pochi prodotti in particolare ma su tutta una gamma di strumenti informatici a codice sorgente aperto. Questo favorisce spesso la nascita di rapporti di outsourcing (collaborazioni esterne) con le molte aziende che temono, coi loro singoli mezzi, di non riuscire a "tenere il passo" con una tecnologia informatica sempre più complessa ed eterogenea, sia per quanto riguarda il software che l'hardware. Nei fattori chiave di questa attività le competenze stesse occupano il primo posto: le aziende sono altamente specializzate e seguono costantemente l'evoluzione della tecnologia circostante su cui poi si sviluppa la fornitura dei servizi. Le imprese tradizionali che decidono di affidarsi ad esse si levano di dosso il problema dell'aggiornamento delle tecnologie ed esternalizzano la gestione del sistema informativo. Le competenze costituiscono inoltre la discriminante tramite la quale una azienda può prevalere sulle altre: solo attraverso la padronanza di *skills* superiori si può riuscire a prevalere in un mercato dove le fonti di apprendimento non sono nascoste a nessuno.

Occorre poi prendere in considerazione l'aspetto dei costi, anch'esso fondamentale. L'adozione di software Open Source conviene ad entrambe le parti dato che permette di evitare il pagamento di royalties o licenze e di offrire al cliente l'indipendenza dai fornitori. Se egli acquistasse dei servizi accessori relativi a software proprietario non intratterrebbe dei rapporti con il produttore ma al massimo con un concessionario o un esperto certificato, in questo caso invece egli ha la possibilità di colloquiare direttamente con il fornitore del servizio. In questo tipo di consulenza quindi, non essendo la produzione impostata o veicolata dall'alto ma condivisa e gestita in maniera orizzontale secondo un continuo processo di feedback, il rapporto con il cliente è molto più stretto. Analogamente al modello precedente anche qui assume una rilevante importanza l'investimento nella comunità di sviluppo in quanto vitale per la sopravvivenza dell'azienda. Due sono i vantaggi che l'azienda consegue: la possibilità di sfruttare i progetti specifici nei quali si è investito e di accrescere la visibilità dell'azienda stessa, la quale, partecipando ai progetti indipendenti di sviluppo, si crea un nome e una fama.

4.5.4 I costi

Uno degli effetti più rilevanti (e probabilmente il più discusso) del software Open Source riguarda i costi delle soluzioni informatiche ed esiste quindi un forte interesse a capire di quale entità siano questi effetti. Prima di procedere con la relativa analisi occorre sottolineare come questi non siano costituiti solo dal costo di acquisizione dei singoli componenti, che rappresentano una piccola porzione della spesa totale, ma anche da tutta una serie di costi indiretti che vanno valutati in relazione all'intero arco vitale della soluzione informatica. A supporto di questa analisi esistono vari modelli di rilevazione dei costi complessivi di un sistema informatico e tali modelli possono

differire per l'ambito o la modalità di applicazione. Il più utilizzato è indubbiamente il modello del *costo totale di possesso* (Total Cost of Ownership) meglio noto come T.C.O., nel quale le principali voci di spesa vengono correttamente suddivise in costi diretti e indiretti su un periodo di riferimento che varia in genere dai tre ai cinque anni. Rientrano fra i costi diretti quelli imputabili all'acquisto di hardware, software (sistema operativo e licenze d'uso) e infrastrutture di rete nonché quelli sostenuti per la gestione, l'amministrazione e la formazione; i costi indiretti sono invece imputabili alle attività improduttive degli utenti finali e ai tempi di fermo dei sistemi.

Recentemente, sono stati pubblicati diversi studi[153] che mettono a confronto i costi di soluzioni proprietarie con soluzioni Open Source tuttavia i valori esposti sono risultati fortemente dipendenti sia da fattori interni al contesto analizzato (nel caso delle aziende, la presenza di competenze interne, il ricorso all'outsourcing per alcune attività, etc.) che da fattori esterni correlati al particolare mercato nazionale (i costi dei servizi professionali di consulenza). Pertanto, gli studi offrono solo un punto di partenza per la comprensione dei reali costi ICT[154]. Ogni soggetto, sia esso un'azienda o un ente della Pubblica Amministrazione, dovrebbe creare un proprio modello dei costi che sia rappresentativo dello specifico ambiente operativo interno e dei vincoli esterni indotti dal mercato. Per praticità, dato che la maggioranza di questi studi ha per oggetto il confronto fra il sistema GNU/Linux e il prodotto offerto da Microsoft, si ricorrerà prevalentemente ad essi per le prossime osservazioni, passando in rassegna le principali voci di costo.

Innanzitutto Linux, e il software Open Source più in generale, si distingue nettamente nei costi di acquisizione del software. I sistemi operativi e i software proprietari installati su un buon numero di computer possono arrivare a costare anche centinaia di migliaia di euro dato che per ogni macchina occorre sottoscrivere una licenza d'uso. Linux può essere acquisito gratuitamente[155], replicato e installato su un numero qualsiasi di computer senza alcun costo di licenza. Anche se si decidesse di acquistare la licenza di una distribuzione Linux curata da qualche società (ad esempio quella fornita da Red Hat), in modo da avere una maggiore garanzia e un supporto tecnico non limitato alla sola comunità, l'unica licenza registrata potrebbe comunque essere utilizzata su tutti i computer dell'intera rete aziendale. Gli stessi costi che molte società sostengono solo per la gestione e la contrattazione delle varie licenze software vengono di conseguenza annullati. Non si sono invece rilevate quasi mai particolari differenze fra le due soluzioni per quanto riguarda i costi della loro applicazione e del loro utilizzo.

In secondo luogo Linux riduce considerevolmente i costi imputabili all'hardware, in due modi diversi. Come si è già visto nel terzo capitolo i software Open Source vengono di solito sviluppati su piattaforme hardware diverse (il concetto di portabilità) e presentano una struttura modulare che li rende più snelli ed efficienti rispetto ai software proprietari, di conseguenza essi, a parità di operazioni compiute, sono più adattabili e necessitano di inferiori risorse tecniche. Diventa così possibile e conveniente riutilizzare l'hardware che normalmente verrebbe considerato obsoleto, con un ulteriore risparmio sui costi. Occorre considerare poi l'efficienza tecnica che Linux offre in termini di sicurezza, dove presenta una bassa vulnerabilità a virus e ad attacchi informatici[156], nonché in termini di affidabilità

---

153 Fra le società più attive nella realizzazione di questi studi vi sono IBM http://www-1.ibm.com/linux/RFG-LinuxTCO-vFINAL-Jul2002.pdf , Cybersource http://www.cyber.com.au/cyber/about/linux_vs_windows_tco_comparison.pdf e http://www.cyber.com.au/cyber/about/linux_vs_windows_pricing_comparison.pdf nonché Microsoft che ha commissionato diversi studi a società di consulenza come IDC, Forrester e Metagroup sul costo totale di possesso (TCO) di Windows e Linux http://www.microsoft.com/italy/mscorp/facts/default.mspx. Non mancano poi gli studi condotti da enti pubblici come l'Unione Europea http://europa.eu.int/ida/en/document/2623#study. Per un'ampia analisi sul "vero costo di Linux e Windows", si rimanda all'articolo di Salvatore Deodato, disponibile all'indirizzo http://www.mytech.it/pmi/articolo/idA028001043972_2.art .

154 Le difficoltà insite nel calcolo del TCO di un software, dovute alle troppe variabili e alle assunzioni arbitrarie, hanno convinto diversi responsabili IT a rinunciare ad esso come modello di valutazione concentrandosi di più su l'indice ROI (Return On Investments) che esprime la redditività di un investimento. Nel software esso viene usato per esprimere l'efficacia di una certa soluzione in relazione ad un certo lasso di tempo: ad esempio quanto più essa determina dei risparmi futuri tanto migliore è la redditività di quell'investimento. Secondo Chad Robinson, analista di Robert Frances Group, il TCO non prende in debita considerazione i risparmi ottenuti, la flessibilità offerta e il guadagno raggiunto, che sono i motivi per cui si sceglie di fare una migrazione. Inoltre, sebbene l'adozione di Linux possa implicare costi iniziali, per l'addestramento e l'integrazione delle applicazioni (che rientrano nel TCO) essi possono essere valutati nella giusta prospettiva solo guardando al ROI.

155 E' possibile scaricare gratuitamente da Internet le versioni supportate dalla comunità come nel caso di OpenOffice.

156 Nel 2001, secondo un'indagine del sito www.wildlist.org, i virus che colpivano Windows erano circa seicento mentre per

in relazione alla stabilità del sistema. Purtroppo, spesso nelle decisioni d'acquisito di una soluzione informatica non viene attribuito il giusto valore economico all'importanza di questi due aspetti, con il risultato che nel lungo periodo possono sorgere nuovi costi.

Infine, il software proprietario evolve secondo logiche di mercato per cui le nuove versioni dei propri prodotti, che poi nel breve/medio termine risultano essere le sole garantite dal supporto del fornitore, vengono molto spesso rilasciate con certificazioni di compatibilità che impongono all'utente l'aggiornamento di altri componenti software e in alcuni casi il rinnovo delle piattaforme hardware per l'aumento di risorse tecniche richiesto dai nuovi prodotti. Questa completa autonomia permette di salvaguardare gli investimenti fatti in termini di hardware e software senza essere esposti alle scelte strategiche del fornitore. Gli aggiornamenti alla propria distribuzione non possono essere in alcun modo imposti ma vengono sempre eseguiti nei termini e nei tempi stabiliti dall'utente per cui se il servizio di assistenza e supporto ottenuto non è soddisfacente è possibile passare ad altri fornitori senza per questo dover sostenere costi di transizione particolarmente significativi.

A questo punto però occorre evidenziare anche alcuni dei limiti del software Open Source che vengono riconosciuti da un po' tutti gli esperi del settore e che probabilmente sono quelli che finora hanno impedito proprio a Linux di porsi come una valida alternativa ai prodotti proprietari nel segmento desktop. Il primo di questi limiti è l'assenza per Linux di alcuni software specifici, come quelli di progettazione grafica (Autocad), che vengono sviluppati solo per le piattaforme commerciali; anche quando esistono software Open Source simili e che svolgono la stessa funzione spesso rimane il problema di garantire la compatibilità dei vecchi dati.

Il secondo limite riguarda la facilità d'uso concessa dalle interfacce grafiche che alcune società commerciali, come MacIntosh e Microsoft, hanno introdotto con i propri prodotti. Nel mondo Open Source vengono privilegiate la qualità e l'efficienza tecnica a scapito però di questa semplicità d'uso[157]: la conseguenza più diretta è che per il momento occorrono ancora delle competenze tecniche superiori a quelle dell'utente comune e quindi soprattutto nel caso di una migrazione da una soluzione proprietaria (ad esempio da Windows) è molto probabile che si debbano sostenere degli ulteriori costi di transizione, fra l'altro difficilmente valutabili, dovuti all'addestramento del personale e alla consulenza tecnica. Non mancano comunque esempi di piccole e medie aziende, organizzazioni o enti pubblici che avendo delle esigenze limitate al solo sistema operativo e/o a poche applicazioni (per le attività d'ufficio o per la comunicazione in Rete) sono riuscite a compiere il passaggio da soluzioni proprietarie a soluzioni Open Source.

Fra i più significativi vi è sicuramente l'attuale migrazione, seppur parziale[158], del comune di Monaco di Baviera[159] che prevede l'adozione di Linux e OpenOffice al posto di Windows e MS Office su 14.000 computer (Progetto LiMux). Si ricorda poi il caso della Lapam-Federimpresa[160], l'associazione di categoria di esercenti, artigiani e piccole imprese, di Modena e Reggio Emilia. A partire dal 1996, il fornitore dei servizi informativi ha iniziato, visti i consistenti benefici tecnici ed economici, ad adottare Linux sui loro server riscontrando la completa soddisfazione dell'associazione. Il passo successivo è stata la proposta, nel 1999, di riconvertire gli oltre 400 computer usati dalla stessa, con software Open Source. Come nel caso tedesco anche qui la scelta è ricaduta oltre che su Linux su OpenOffice. L'approvazione del progetto e la sua attuazione hanno permesso di risparmiare il costo di tutte le licenze senza per questo determinare costi di transizione[161] rilevanti; anche in questo contesto infatti, le esigenze del cliente erano limitate a strumenti informatici piuttosto comuni di cui l'Open Source può offrire valide e convenienti alternative.

4.6 L'Open Source nella Pubblica Amministrazione

Linux ne esistevano cinque. Ovviamente bisogna considerare la minore diffusione di quest'ultimo tuttavia credo che tali dati siano comunque abbastanza significativi. Risultati simili sono riscontrabili anche nel numero di attacchi informatici a danno dei web server. Si veda http://www.securitystats.com/ .

157 In realtà esistono delle interfacce grafiche anche per Linux che lo rendono simili ai concorrenti Windows (progetti KDE e Gnome). Tuttavia, mentre per l'utente Linux l'interazione grafica è utile ma non indispensabile, un accessorio, ciò è impensabile per gli utenti di Windows da anni abituati ad usare soprattutto il mouse e le icone.

158 Avendo utilizzato per molti anni dei formati proprietari per il salvataggio non è possibile garantire la completa compatibilità con le nuove applicazioni di conseguenza conviene mantenere un numero limitato dei software vecchi. Nel progetto LiMux di Monaco si è previsto, per questo motivo, di mantenere sul 20% dei computer Microsoft Office.

159 Progetto LiMux. http://www.muenchen.de/linux e http://www.muenchen.de/vip8/prod2/mde/_de/rubriken/Rathaus/40_dir/limux/publikationen/clientstudie_kurz.pdf

160 http://www.lapam.mo.it/

161 http://www.yacme.com/casi/metodo.pdf

Negli ultimi anni le potenzialità del modello Open Source hanno favorito la sua diffusione dalla semplice comunità di sviluppo a contesti sempre più vasti, di conseguenza si è avuto anche un allargamento delle tipologie di soggetti coinvolti. L'entusiasmo con cui alcune imprese hanno aderito o quanto meno contribuito ad esso ha infatti spinto anche molti governi e pubbliche amministrazioni ad avvicinarsi e a valutare gli eventuali benefici di una sua adozione. Se in una fase iniziale esse hanno dimostrato un cauto interesse, recentemente invece si sta assistendo a livello internazionale ad un vero e proprio boom dell'Open Source in questo campo. Per comprendere l'importanza del binomio Open Source-Pubblica Amministrazione occorre innanzitutto sottolineare la notevole rilevanza che il soggetto pubblico detiene nel mercato del software, dovuta ovviamente all'ammontare della sua spesa per le soluzioni informatiche.

Lo scorso anno il Ministero per l'Innovazione e la Tecnologia ha diffuso i risultati di un'indagine conoscitiva sull'adozione del software a sorgente aperto[162] e in essa sono stati riportati anche i dati, relativi al biennio 2000/2001, della spesa italiana per l'ICT nei diversi settori economici. La quota complessiva della spesa pubblica, in entrambi gli anni, è stata di circa il 10% del mercato totale (pari a 3031 milioni di euro). Si può quindi affermare che essa rappresenta uno degli attori principali di questo mercato. Per comprendere poi come quali singole voci la compongono conviene approfondire l'analisi. Nel solo 2001 sono stati spesi per il software 675 milioni di euro. Di questi, il 61% è imputabile allo sviluppo, alla manutenzione[163] e alla gestione di programmi custom, sviluppati cioè su commessa per una specifica Amministrazione (o per più Amministrazioni); alcuni esempi di tali prodotti sono i sistemi software sviluppati per l'anagrafe tributaria, il sistema sanitario nazionale, la motorizzazione civile. Il rimanente 39% è invece stato utilizzato per la sola acquisizione delle licenze d'uso di sistemi operativi, sistemi per la gestione di basi di dati (DBMS) e infine di applicativi per ufficio. Per dare ai dati appena citati una maggiore visibilità essi sono stati riportati nella seguente tabella.

VOCI DI SPESA (2001)	IMPORTI (migliaia di euro)	%
Sviluppo e manutenzione di software custom	411750	61
Licenze d'uso	263250	39
Totale	675000	

Tabella 4.8 Spesa per il software nella Pubblica Amministrazione
Fonte: Ministero per l'Innovazione e la Tecnologia (2003).

Ricordando sia la forte esigenza di soluzioni personalizzate che l'ammontare della spesa per le sole licenze d'uso, la Pubblica Amministrazione ha iniziato a guardarsi attorno e valutare la possibilità di adottare soluzioni Open Source. Si può infatti affermare che il primo motivo per cui si è sviluppato questo interesse è stato soprattutto quello economico: l'utilizzo dell'Open Source consentirebbe di ottimizzare la spesa pubblica limitando, come si è già visto in un paragrafo precedente, i costi di acquisto del software (sia per quanto riguarda le soluzioni custom che le licenze). Ovviamente anche in questo contesto l'analisi costi/benefici non può limitarsi ai vantaggi più diretti ma deve appurare se anche nel lungo periodo l'eventuale adozione sia effettivamente proficua[164]. In considerazione però del ruolo che la Pubblica Amministrazione riveste, occorre sottolineare come l'aspetto economico non debba essere l'unico criterio di valutazione per l'adozione di questa soluzione ma si debbano prendere in considerazione anche quelli sociali e politici.

Uno dei compiti fondamentali della Pubblica Amministrazione è ad esempio quello di archiviare, elaborare e trasmettere informazioni riguardanti i cittadini. Il cittadino ha il diritto di accedere facilmente ai propri dati e gli enti pubblici hanno il dovere di garantire l'accessibilità, l'integrità e la riservatezza di questi dati. Deve quindi essere concesso ad ogni cittadino l'accesso alle informazione che lo riguardano, senza per questo costringerlo ad utilizzare uno specifico prodotto software, nemmeno se si trattasse di quello più diffuso. L'unico modo per garantire ciò è quello di archiviare i dati attraverso piattaforme software e interfacce che costituiscano degli standard aperti e riconosciuti (come avviene nell'Open Source). Il ricorso a queste soluzioni non costringerebbe i cittadini ad utilizzare particolari software ma

[162] Ministero per l'Innovazione e la Tecnologia (2003).
[163] Con il termine "manutenzione" si intende l'insieme delle attività volte a modificare il codice sorgente di un programma al fine di mantenerlo allineato rispetto ai requisiti ed all'eventuale evoluzione normativa.
[164] E' il discorso fatto precedentemente per il modello TCO.

permetterebbe di accedere ai dati con applicazioni diverse. Un ulteriore beneficio portato da questo tipo di soluzioni è la garanzia di una più semplice continuità di gestione e di mantenimento delle informazioni archiviate, indipendentemente dalle soluzioni software e dai fornitori scelti.

La forte esigenza di soluzioni personalizzate, create su misura alle attività pubbliche, e di una continua manutenzione delle stesse, richiede l'uso di strumenti altamente flessibili. Questo tipo di necessità mal si concilia con i software proprietari dove solo il fornitore ha accesso al codice sorgente mentre trova una valida soluzione nei software Open Source dove la loro disponibilità permette di compiere la manutenzione e le modifiche, sia internamente, sia utilizzando fornitori di servizi diversi dal fornitore del software (a vantaggio poi della concorrenza e quindi dei prezzi).
L'importanza di poter ispezionare i codici sorgente è dovuta anche alla necessità di controllare che il software risponda a requisiti di sicurezza e non contenga al suo interno funzioni indesiderate, potenzialmente dannose e/o illecite.

Nell'indagine conoscitiva sull'Open Source viene infine citato un ulteriore incentivo, di natura politica, alla sua adozione e diffusione rappresentato dalla possibilità di produrre degli effetti positivi sul mercato dell'ICT in Italia e in Europa. E' facile constatare infatti come l'industria informatica italiana ed europea stia attraversando una profonda crisi: negli ultimi decenni sono andati progressivamente scomparendo (o hanno abbandonato il mercato dei prodotti IT in senso stretto) attori europei globali quali Olivetti, ICL, Siemens/Nixdorf e Bull. Il deficit della bilancia commerciale italiana ed europea nel settore IT[165] segna un preoccupante passivo che ci rende sostanzialmente dipendenti da tecnologie e prodotti sviluppati negli Stati Uniti e in Asia.

L'Open Source potrebbe essere, se adeguatamente accompagnato da investimenti massicci e continui, uno dei possibili strumenti per recuperare il terreno perduto e sostenere la rinascita dell'industria informatica nazionale (ed europea). Ad esempio potrebbe essere il meccanismo per creare piattaforme software condivise in settori industriali importanti come la telefonia mobile o l'industria degli elettrodomestici di nuova generazione. Nel contempo la possibilità di operare su software e formati aperti aprirebbe il mercato a imprese locali permettendo di mantenere in Italia quegli investimenti in soluzioni ICT che normalmente finiscono per finanziare le grandi società d'Oltreoceano e quindi indirettamente l'industria extraeuropea. In Italia comunque, dopo l'indagine conoscitiva qualcosa ha iniziato a muoversi tant'è che si stanno eseguendo già delle sperimentazioni circa l'adozione dell'Open Source nel settore pubblico, in particolare si segnala il progetto europeo COSPA[166] a cui aderiscono la Provincia di Pisa e quella di Genova (proprio nel capoluogo ligure è in atto la migrazione da Office a OpenOffice).

Per quanto riguarda invece la situazione internazionale è riscontrabile un crescente interesse verso questo modello da parte di molti paesi europei, soprattutto Francia e Germania[167] che hanno già adottato software Open Source in molte delle proprie strutture e finanziano addirittura alcuni progetti. In Francia[168] tutte le amministrazioni locali stanno convergendo verso il software Open Source, lo stesso ministero delle finanze sostiene che almeno il 25% dei server è costituito da prodotti Open Source. L'iniziativa OpenCIGREF, riguardante l'adozione di software liberi su sistemi desktop, ha permesso di risparmiare sui costi di gestione e soprattutto sui costi di licenza (risparmi stimati in circa 3 milioni di euro).

In Spagna[169], l'intera regione dell'Extremadura ha completato un progetto di conversione completa all'Open Source software sia nell'apparato amministrativo che nelle istituzioni scolastiche. Molti altri paesi invece per il momento ne promuovono solamente l'utilizzo al fine di favorire la diffusione di formati aperti e di alternative alle attuali offerte proprietarie. Merita infine di essere menzionata la forte valenza che esso può avere per i paesi in via di sviluppo (è il caso soprattutto dei paesi sudamericani e asiatici) caratterizzati da un forte gap tecnologico, il cosiddetto *digital divide*. L'Open Source può rappresentare un ottimo mezzo per limitare tale divario e favorire lo sviluppo economico, con un impatto molto basso sui costi[170].

CONCLUSIONI

[165] Ministero per l'Innovazione e la Tecnologia (2003).
[166] http://www.cospa-project.org/
[167] Ministero per l'Innovazione e la Tecnologia (2003).
[168] Avecedo (2003).
[169] Azais et al. (2001).
[170] Si segnala in merito l'articolo "L'Open Source per lo sviluppo economico" di Guido Sintoni, disponibile all'indirizzo http://www.mytech.it/computer/articolo/idA028001050869_2.art.

Il software libero, evolutosi poi nell'Open Source, ha stravolto il modo di "fare software". Quella che era una consuetudine o un hobby di pochi hacker (la condivisione del codice) si è trasformata, nel giro di quindici anni, in un movimento internazionale e in un modello di produzione del software credibile, quanto meno dal punto di vista tecnico. Nella prima parte dell'elaborato sono state illustrate le vicende storiche che hanno determinato la nascita di un fenomeno solo in apparenza nuovo: si è visto infatti che il movimento era attivo già nella seconda metà degli anni Ottanta come reazione alla crescente pratica (dell'industria informatica) di creare software chiuso.

Ciò nonostante, le aziende, e più in generale il mercato, hanno iniziato ad intuire le potenzialità di questo modello solo negli ultimi anni. Il motivo di questo disinteresse (e del conseguente isolamento della comunità) va ricercato innanzitutto nell'atteggiamento che ha sempre contraddistinto la parte più visibile e organizzata della cultura hacker, identificabile in Stallman e nella Free Software Foundation. L'eccessivo fanatismo e l'ostilità da loro dimostrata nei confronti del software commerciale non hanno fatto altro che relegare la comunità ad un ruolo di secondo piano e ne hanno minato la credibilità. Senza dubbio, alla Free Software Foundation va riconosciuto il merito di essere stata per anni il fulcro del movimento, producendo un gran numero di validi strumenti software e sponsorizzandone lo sviluppo, tuttavia essa ha anche fatto sì che la cultura hacker fosse percepita dall'esterno come un movimento estremista e anti commerciale.

La consapevolezza che queste posizioni finivano in realtà più per danneggiare la causa del software libero che aiutarla, ha spinto l'ala più moderata del movimento a distinguersi e ad indirizzare una critica più costruttiva contro le grandi software house: ciò che ad esse viene rimproverato non è tanto la loro volontà di tutelare il software quanto il loro rifiuto di aprire il codice e incorporare nei propri prodotti standard aperti e software Open Source. A tal proposito un aspetto che va sottolineato è che il software Open Source non debba essere considerato di pubblico dominio: chi lo produce, pur battendosi per la causa della non brevettabilità e della libertà del software in generale, non si oppone alla tutela dello stesso, anzi. Gli ideologi del Free Software utilizzano comunque lo strumento del diritto d'autore trasformandolo però nell'inedito concetto di copyleft con l'obiettivo di restituire agli utenti quelle libertà che nel corso degli anni hanno perso.

L'esistenza di svariate licenze libere, prima fra tutte la GPL, che permettono a chiunque di contribuire senza per questo compromettere la libertà del codice prodotto, conferma quest'idea. La posizione più conciliante verso il mondo commerciale sancita dal passaggio alla nuova denominazione "Open Source", il boom di Linux e la piega presa dalla vicenda Netscape, con l'intenzione dell'omonima società di rilasciare pubblicamente i sorgenti del proprio prodotto di punta, hanno permesso alla fine degli anni Novanta l'inizio di un timido avvicinamento verso l'industria e il conseguimento da parte del movimento di una crescente popolarità, non solo fra gli operatori del settore. L'interesse dei mass media ha permesso di far scoprire e ha portato alla ribalta un nuovo modo di produrre il software, basato sull'esistenza di una comunità operante in Internet e costituita da tutti coloro che contribuiscono più o meno direttamente allo sviluppo.

Il desiderio di contrapporsi al software chiuso ha infatti determinato non solo la creazione di programmi alternativi che rispettassero il concetto di libertà inteso dalla FSF[171] ma indirettamente anche la nascita di una comunità "produttiva" estremamente variegata e fondata sulla cultura hacker, a sua volta portatrice di valori quali l'aiuto reciproco e la solidarietà nonché caratterizzata da regole, comportamenti e sistemi di incentivi impliciti. Una sorta di "micro società" nella quale, avendo a che fare non con il problema della scarsità ma con l'esagerata abbondanza di risorse, prevalgono soprattutto gli incentivi tecnici e sociali più che quelli economici: l'importanza della reputazione all'interno di questo ambiente diventa uno degli aspetti più rilevanti su cui si basa la cosiddetta cultura del dono.

La comunità ha dato vita ad un nuovo modello produttivo nel quale il codice sorgente viene considerato un patrimonio di conoscenze da condividere liberamente, al cui sviluppo tutti possono partecipare e da cui tutti possono prendere, una cosa impensabile per le software house. La produzione di tale conoscenza diventa l'essenza di questo modello in cui la netta distinzione fra produzione e consumo, tipica del modello proprietario, non è più così evidente e dove si accentua l'attenzione alla qualità tecnica del software più che alla riscossione di eventuali diritti. A questo proposito è importante sottolineare il fatto che esso, pur non garantendo la proprietà del codice né quella dell'eventuale investimento sulla comunità di sviluppo, compensa tale (eventuale) perdita limitando i costi di sviluppo e il rischio, che di fatto ricadono anche sulla comunità stessa.

[171] Free Software Foundation.

Proprio la possibilità di ripartire i costi di sviluppo e i rischi è il motivo per cui negli ultimi anni l'industria si è interessata all'Open Source e ha iniziato a considerare sempre più plausibile l'idea di uno sviluppo congiunto con la stessa; questo avvicinamento rappresenta una svolta nel modo di intendere il software e il suo ruolo nel settore. A tal proposito è interessante rilevare come la disponibilità del codice sorgente abbia favorito la nascita di nuovi modelli commerciali, ad esempio il "branding and distribution" adottato da Red Hat, nei quali i concetti espressi finora vengono messi in pratica: il software libero viene visto non come un bene da vendere e sulla cui produzione concentrare gli sforzi ma come la base su cui svolgere la fornitura di servizi e competere liberamente. Proprio la capacità di competere in un mercato aperto sembra essere l'ultima sfida lanciata dall'Open Source all'industria.

Nell'ultima parte dell'elaborato si è visto infatti come il mercato del software commerciale tenda ad assumere un'elevata concentrazione con una concorrenza minima o del tutto assente. Le cause di ciò vanno cercate innanzitutto nella natura del software che determina elevati costi fissi di produzione ed esigui costi di riproduzione nonché in tutta una serie di fenomeni economici che caratterizzano le economie di rete: le esternalità, il feedback positivo e il problema del lock-in. Essi influiscono fortemente sulle scelte dei consumatori e finiscono per favorire la formazione di monopoli o oligopoli, con ovvie inefficienze. Nel caso del software proprietario gli effetti di questi fenomeni vengono poi accentuati proprio dalla segretezza del codice e dall'uso di formati chiusi.

Il fatto di non dover fare i conti con altri concorrenti ha finora permesso alle software house, specialmente a quelle di grandi dimensioni, di basare la loro attività interamente sulla riscossione delle royalties a discapito della qualità dei prodotti e dei servizi offerti. Con l'ingresso sul mercato dell'Open Source anche questa situazione sta cambiando poiché ora sono presenti nuovi fornitori che basano la loro attività su soluzioni aperte di qualità e a prezzi inferiori. Ciò costringe sempre più spesso le "vecchie" società a ripensare le proprie strategie e i propri modelli o quanto meno a migliorare la loro offerta sia in termini di qualità che di prezzo. L'aspetto evidente è che si sta verificando una timida apertura del mercato a nuovi attori e a nuove soluzioni. Questa situazione è riscontrabile soprattutto nel segmento server del mercato dove le soluzioni Open Source hanno incontrato già da tempo il favore delle aziende e più recentemente quello di vari enti pubblici e organizzazioni un po' in tutto il mondo. Proprio la Pubblica Amministrazione si sta rivelando uno dei maggiori sostenitori e fruitori di software a sorgente aperto. Il motivo principale va cercato nella particolare esigenza di soluzioni informatiche specifiche (custom) che offrano quindi flessibilità e una certa autonomia dai fornitori sia in termini di assistenza che di manutenzione; inoltre, occorre ricordare l'attenzione che la stessa riserva alla problematica dei costi sul quale l'Open Source sembra offrire ulteriori vantaggi rispetto alle soluzioni proprietarie. In secondo luogo sempre la Pubblica Amministrazione vede nell'Open Source il mezzo con cui rilanciare l'industria informatica nazionale ed europea da tempo in crisi, limitando così la dipendenza verso l'industria straniera, in particolare quella statunitense.

Dall'analisi compiuta si evince quindi che esso presenta diverse potenzialità e non è azzardato ipotizzare che nei prossimi anni l'industria del software sarà sempre più legata a questo modello. Detto questo, occorre però ammettere anche alcuni dei suoi attuali limiti e sottolineare alcuni dei dubbi che tuttora permangono. Infatti, se i dati disponibili dimostrano che esso può essere fondamentalmente pari al software proprietario in termini tecnici e di pura convenienza economica, è altrettanto vero che esistono ancora poche conferme e molti dubbi, peraltro legittimi, sulla sostenibilità nel tempo del relativo modello economico. Ciò è vero soprattutto in relazione alle società che offrono soluzioni aperte per il segmento desktop del mercato dove, a differenza di quanto accade nel segmento server, l'Open Source, anche con i grandi progetti come Linux e OpenOffice, deve ancora riuscire ad ottenere la piena fiducia di aziende e pubbliche amministrazioni. Questa è probabilmente la vera sfida che, nei prossimi due[172] anni, il movimento dovrà affrontare e cercare di vincere per poter fare quel salto di qualità che gli permetterebbe di uscire dalla nicchia in cui finora è rimasto e di combattere ad armi pari i prodotti proprietari sull'intero mercato.

[172] Nel segmento desktop esiste la dominanza assoluta di Microsoft con il suo sistema operativo Windows tuttavia l'attuale versione (Xp) è sul mercato già da qualche anno e la prossima dovrebbe uscire solo intorno al 2006. Molti pensano che questo ulteriore lasso di tempo dovrebbe essere sfruttato dalla comunità a proprio vantaggio al fine di realizzare una distribuzione Linux specifica per questo segmento e quindi potenzialmente in grado di erodere quote di mercato al gigante di Redmond.

APPENDICE
Testo della licenza General Public License (GNU GPL)[173]

Questa è una traduzione italiana non ufficiale della Licenza Pubblica Generica GNU. Non è pubblicata dalla Free Software Foundation e non ha valore legale nell'esprimere i termini di distribuzione del software che usa la licenza GPL. Solo la versione originale in inglese della licenza ha valore legale. Ad ogni modo, speriamo che questa traduzione aiuti le persone di lingua italiana a capire meglio il significato della licenza GPL.

> This is an unofficial translation of the GNU General Public License into Italian. It was not published by the Free Software Foundation, and does not legally state the distribution terms for software that uses the GNU GPL--only the original English text of the GNU GPL does that. However, we hope that this translation will help Italian speakers understand the GNU GPL better.

LICENZA PUBBLICA GENERICA (GPL) DEL PROGETTO GNU
Versione 2, Giugno 1991

Traduzione curata da gruppo Pluto, da ILS e dal gruppo italiano di traduzione GNU. Ultimo aggiornamento 19 aprile 2000.

Preambolo

Le licenze della maggior parte dei programmi hanno lo scopo di togliere all'utente la libertà di condividere e modificare il programma stesso. Viceversa, la Licenza Pubblica Generica GNU è intesa a garantire la libertà di condividere e modificare il software libero, al fine di assicurare che i programmi siano liberi per tutti i loro utenti. Questa Licenza si applica alla maggioranza dei programmi della Free Software Foundation e ad ogni altro programma i cui autori hanno deciso di usare questa Licenza. Alcuni altri programmi della Free Software Foundation sono invece coperti dalla Licenza Pubblica Generica Minore. Chiunque può usare questa Licenza per i propri programmi.
Quando si parla di software libero (free software), ci si riferisce alla libertà, non al prezzo. Le nostre Licenze (la GPL e la LGPL) sono progettate per assicurarsi che ciascuno abbia la libertà di distribuire copie del software libero (e farsi pagare per questo, se vuole), che ciascuno riceva il codice sorgente o che lo possa ottenere se lo desidera, che ciascuno possa modificare il programma o usarne delle parti in nuovi programmi liberi e che ciascuno sappia di potere fare queste cose.
Per proteggere i diritti dell'utente, abbiamo bisogno di creare delle restrizioni che vietino a chiunque di negare questi diritti o di chiedere di rinunciarvi. Queste restrizioni si traducono in certe responsabilità per chi distribuisce copie del software e per chi lo modifica.
Per esempio, chi distribuisce copie di un programma coperto da GPL, sia gratis sia in cambio di un compenso, deve concedere ai destinatari tutti i diritti che ha ricevuto. Deve anche assicurarsi che i destinatari ricevano o possano ottenere il codice sorgente. E deve

[173] http://www.softwarelibero.it/gnudoc/gpl.it.txt

mostrar loro queste condizioni di licenza, in modo che essi conoscano i propri diritti.
Proteggiamo i diritti dell'utente in due modi: (1) proteggendo il software con un copyright, e (2) offrendo una licenza che dia il permesso legale di copiare, distribuire e modificare il Programma.
Inoltre, per proteggere ogni autore e noi stessi, vogliamo assicurarci che ognuno capisca che non ci sono garanzie per i programmi coperti da GPL. Se il programma viene modificato da qualcun altro e ridistribuito, vogliamo che gli acquirenti sappiano che ciò che hanno non è l'originale, in modo che ogni problema introdotto da altri non si rifletta sulla reputazione degli autori originari.
Infine, ogni programma libero è costantemente minacciato dai brevetti sui programmi. Vogliamo evitare il pericolo che chi ridistribuisce un programma libero ottenga la proprietà di brevetti, rendendo in pratica il programma cosa di sua proprietà. Per prevenire questa evenienza, abbiamo chiarito che ogni brevetto debba essere concesso in licenza d'uso a chiunque, o non avere alcuna restrizione di licenza d'uso.
Seguono i termini e le condizioni precisi per la copia, la distribuzione e la modifica.

LICENZA PUBBLICA GENERICA GNU
TERMINI E CONDIZIONI PER LA COPIA, LA DISTRIBUZIONE E LA MODIFICA

0. Questa Licenza si applica a ogni programma o altra opera che contenga una nota da parte del detentore del copyright che dica che tale opera può essere distribuita sotto i termini di questa Licenza Pubblica Generica. Il termine "Programma" nel seguito si riferisce ad ogni programma o opera così definita, e l'espressione "opera basata sul Programma" indica sia il Programma sia ogni opera considerata "derivata" in base alla legge sul copyright; in altre parole, un'opera contenente il Programma o una porzione di esso, sia letteralmente sia modificato o tradotto in un'altra lingua. Da qui in avanti, la traduzione è in ogni caso considerata una "modifica". Vengono ora elencati i diritti dei beneficiari della licenza.
Attività diverse dalla copiatura, distribuzione e modifica non sono coperte da questa Licenza e sono al di fuori della sua influenza. L'atto di eseguire il Programma non viene limitato, e l'output del programma è coperto da questa Licenza solo se il suo contenuto costituisce un'opera basata sul Programma (indipendentemente dal fatto che sia stato creato eseguendo il Programma). In base alla natura del Programma il suo output può essere o meno coperto da questa Licenza.

1. È lecito copiare e distribuire copie letterali del codice sorgente del Programma così come viene ricevuto, con qualsiasi mezzo, a condizione che venga riprodotta chiaramente su ogni copia una appropriata nota di copyright e di assenza di garanzia; che si mantengano intatti tutti i riferimenti a questa Licenza e all'assenza di ogni garanzia; che si dia a ogni altro destinatario del Programma una copia di questa Licenza insieme al Programma.
È possibile richiedere un pagamento per il trasferimento fisico di una copia del Programma, è anche possibile a propria discrezione richiedere un pagamento in cambio di una copertura assicurativa.

2. È lecito modificare la propria copia o copie del Programma, o parte di esso, creando perciò un'opera basata sul Programma, e copiare o distribuire tali modifiche o tale opera secondo i termini del precedente comma 1, a patto che siano soddisfatte tutte le condizioni che seguono:

a) Bisogna indicare chiaramente nei file che si tratta di copie modificate e la data di ogni modifica.
b) Bisogna fare in modo che ogni opera distribuita o pubblicata, che in parte o nella sua totalità derivi dal Programma o da parti di esso, sia concessa nella sua interezza in licenza gratuita ad ogni terza parte, secondo i termini di questa Licenza.
c) Se normalmente il programma modificato legge comandi

interattivamente quando viene eseguito, bisogna fare in modo che all'inizio dell'esecuzione interattiva usuale, esso stampi un messaggio contenente una appropriata nota di copyright e di assenza di garanzia (oppure che specifichi il tipo di garanzia che si offre). Il messaggio deve inoltre specificare che chiunque può ridistribuire il programma alle condizioni qui descritte e deve indicare come reperire questa Licenza. Se però il programma di partenza è interattivo ma normalmente non stampa tale messaggio, non occorre che un'opera basata sul Programma lo stampi.

Questi requisiti si applicano all'opera modificata nel suo complesso. Se sussistono parti identificabili dell'opera modificata che non siano derivate dal Programma e che possono essere ragionevolmente considerate lavori indipendenti, allora questa Licenza e i suoi termini non si applicano a queste parti quando queste vengono distribuite separatamente. Se però queste parti vengono distribuite all'interno di un prodotto che è un'opera basata sul Programma, la distribuzione di quest'opera nella sua interezza deve avvenire nei termini di questa Licenza, le cui norme nei confronti di altri utenti si estendono all'opera nella sua interezza, e quindi ad ogni sua parte, chiunque ne sia l'autore.
Quindi, non è nelle intenzioni di questa sezione accampare diritti, né contestare diritti su opere scritte interamente da altri; l'intento è piuttosto quello di esercitare il diritto di controllare la distribuzione di opere derivati dal Programma o che lo contengano.
Inoltre, la semplice aggregazione di un'opera non derivata dal Programma col Programma o con un'opera da esso derivata su di un mezzo di memorizzazione o di distribuzione, non è sufficiente a includere l'opera non derivata nell'ambito di questa Licenza.

3. È lecito copiare e distribuire il Programma (o un'opera basata su di esso, come espresso al comma 2) sotto forma di codice oggetto o eseguibile secondo i termini dei precedenti commi 1 e 2, a patto che si applichi una delle seguenti condizioni:

a) Il Programma sia corredato del codice sorgente completo, in una forma leggibile da calcolatore, e tale sorgente sia fornito secondo le regole dei precedenti commi 1 e 2 su di un mezzo comunemente usato per lo scambio di programmi.
b) Il Programma sia accompagnato da un'offerta scritta, valida per almeno tre anni, di fornire a chiunque ne faccia richiesta una copia completa del codice sorgente, in una forma leggibile da calcolatore, in cambio di un compenso non superiore al costo del trasferimento fisico di tale copia, che deve essere fornita secondo le regole dei precedenti commi 1 e 2 su di un mezzo comunemente usato per lo scambio di programmi.
c) Il Programma sia accompagnato dalle informazioni che sono state ricevute riguardo alla possibilità di ottenere il codice sorgente. Questa alternativa è permessa solo in caso di distribuzioni non commerciali e solo se il programma è stato ottenuto sotto forma di codice oggetto o eseguibile in accordo al precedente comma B.

Per "codice sorgente completo" di un'opera si intende la forma preferenziale usata per modificare un'opera. Per un programma eseguibile, "codice sorgente completo" significa tutto il codice sorgente di tutti i moduli in esso contenuti, più ogni file associato che definisca le interfacce esterne del programma, più gli script usati per controllare la compilazione e l'installazione dell'eseguibile. In ogni caso non è necessario che il codice sorgente fornito includa nulla che sia normalmente distribuito (in forma sorgente o in formato binario) con i principali componenti del sistema operativo sotto cui viene eseguito il Programma (compilatore, kernel, e così via), a meno che tali componenti accompagnino l'eseguibile.
Se la distribuzione dell'eseguibile o del codice oggetto è effettuata indicando un luogo dal quale sia possibile copiarlo, permettere la copia del codice sorgente dallo stesso luogo è considerata una valida

forma di distribuzione del codice sorgente, anche se copiare il sorgente è facoltativo per l'acquirente.

4. Non è lecito copiare, modificare, sublicenziare, o distribuire il Programma in modi diversi da quelli espressamente previsti da questa Licenza. Ogni tentativo di copiare, modificare, sublicenziare o distribuire il Programma non è autorizzato, e farà terminare automaticamente i diritti garantiti da questa Licenza. D'altra parte ogni acquirente che abbia ricevuto copie, o diritti, coperti da questa Licenza da parte di persone che violano la Licenza come qui indicato non vedranno invalidata la loro Licenza, purché si comportino conformemente ad essa.

5. L'acquirente non è tenuto ad accettare questa Licenza, poiché non l'ha firmata. D'altra parte nessun altro documento garantisce il permesso di modificare o distribuire il Programma o i lavori derivati da esso. Queste azioni sono proibite dalla legge per chi non accetta questa Licenza; perciò, modificando o distribuendo il Programma o un'opera basata sul programma, si indica nel fare ciò l'accettazione di questa Licenza e quindi di tutti i suoi termini e le condizioni poste sulla copia, la distribuzione e la modifica del Programma o di lavori basati su di esso.

6. Ogni volta che il Programma o un'opera basata su di esso vengono distribuiti, l'acquirente riceve automaticamente una licenza d'uso da parte del licenziatario originale. Tale licenza regola la copia, la distribuzione e la modifica del Programma secondo questi termini e queste condizioni. Non è lecito imporre restrizioni ulteriori all'acquirente nel suo esercizio dei diritti qui garantiti. Chi distribuisce programmi coperti da questa Licenza non e' comunque tenuto a imporre il rispetto di questa Licenza a terzi.

7. Se, come conseguenza del giudizio di un tribunale, o di una imputazione per la violazione di un brevetto o per ogni altra ragione (non limitatamente a questioni di brevetti), vengono imposte condizioni che contraddicono le condizioni di questa licenza, che queste condizioni siano dettate dalla corte, da accordi tra le parti o altro, queste condizioni non esimono nessuno dall'osservazione di questa Licenza. Se non è possibile distribuire un prodotto in un modo che soddisfi simultaneamente gli obblighi dettati da questa Licenza e altri obblighi pertinenti, il prodotto non può essere affatto distribuito. Per esempio, se un brevetto non permettesse a tutti quelli che lo ricevono di ridistribuire il Programma senza obbligare al pagamento di diritti, allora l'unico modo per soddisfare contemporaneamente il brevetto e questa Licenza e' di non distribuire affatto il Programma.
Se una qualunque parte di questo comma è ritenuta non valida o non applicabile in una qualunque circostanza, deve comunque essere applicata l'idea espressa da questo comma; in ogni altra circostanza invece deve essere applicato questo comma nel suo complesso.
Non è nelle finalità di questo comma indurre gli utenti ad infrangere alcun brevetto né ogni altra rivendicazione di diritti di proprietà, né di contestare la validità di alcuna di queste rivendicazioni; lo scopo di questo comma è unicamente quello di proteggere l'integrità del sistema di distribuzione dei programmi liberi, che viene realizzato tramite l'uso di licenze pubbliche. Molte persone hanno contribuito generosamente alla vasta gamma di programmi distribuiti attraverso questo sistema, basandosi sull'applicazione fedele di tale sistema. L'autore/donatore può decidere di sua volontà se preferisce distribuire il software avvalendosi di altri sistemi, e l'acquirente non può imporre la scelta del sistema di distribuzione.
Questo comma serve a rendere il più chiaro possibile ciò che crediamo sia una conseguenza del resto di questa Licenza.

8. Se in alcuni paesi la distribuzione o l'uso del Programma sono limitati da brevetto o dall'uso di interfacce coperte da copyright, il

detentore del copyright originale che pone il Programma sotto questa Licenza può aggiungere limiti geografici espliciti alla distribuzione, per escludere questi paesi dalla distribuzione stessa, in modo che il programma possa essere distribuito solo nei paesi non esclusi da questa regola. In questo caso i limiti geografici sono inclusi in questa Licenza e ne fanno parte a tutti gli effetti.

9. All'occorrenza la Free Software Foundation può pubblicare revisioni o nuove versioni di questa Licenza Pubblica Generica. Tali nuove versioni saranno simili a questa nello spirito, ma potranno differire nei dettagli al fine di coprire nuovi problemi e nuove situazioni.
Ad ogni versione viene dato un numero identificativo. Se il Programma asserisce di essere coperto da una particolare versione di questa Licenza e "da ogni versione successiva", l'acquirente può scegliere se seguire le condizioni della versione specificata o di una successiva. Se il Programma non specifica quale versione di questa Licenza deve applicarsi, l'acquirente può scegliere una qualsiasi versione tra quelle pubblicate dalla Free Software Foundation.

10. Se si desidera incorporare parti del Programma in altri programmi liberi le cui condizioni di distribuzione differiscano da queste, è possibile scrivere all'autore del Programma per chiederne l'autorizzazione. Per il software il cui copyright è detenuto dalla Free Software Foundation, si scriva alla Free Software Foundation; talvolta facciamo eccezioni alle regole di questa Licenza. La nostra decisione sarà guidata da due finalità: preservare la libertà di tutti i prodotti derivati dal nostro software libero e promuovere la condivisione e il riutilizzo del software in generale.

NON C'È GARANZIA

11. POICHÉ IL PROGRAMMA È CONCESSO IN USO GRATUITAMENTE, NON C'È GARANZIA PER IL PROGRAMMA, NEI LIMITI PERMESSI DALLE VIGENTI LEGGI. SE NON INDICATO DIVERSAMENTE PER ISCRITTO, IL DETENTORE DEL COPYRIGHT E LE ALTRE PARTI FORNISCONO IL PROGRAMMA "COSÌ COM'È", SENZA ALCUN TIPO DI GARANZIA, NÉ ESPLICITA NÉ IMPLICITA; CIÒ COMPRENDE, SENZA LIMITARSI A QUESTO, LA GARANZIA IMPLICITA DI COMMERCIABILITÀ E UTILIZZABILITÀ PER UN PARTICOLARE SCOPO. L'INTERO RISCHIO CONCERNENTE LA QUALITÀ E LE PRESTAZIONI DEL PROGRAMMA È DELL'ACQUIRENTE. SE IL PROGRAMMA DOVESSE RIVELARSI DIFETTOSO, L'ACQUIRENTE SI ASSUME IL COSTO DI OGNI MANUTENZIONE, RIPARAZIONE O CORREZIONE NECESSARIA.

12. NÉ IL DETENTORE DEL COPYRIGHT NÉ ALTRE PARTI CHE POSSONO MODIFICARE O RIDISTRIBUIRE IL PROGRAMMA COME PERMESSO IN QUESTA LICENZA SONO RESPONSABILI PER DANNI NEI CONFRONTI DELL'ACQUIRENTE, A MENO CHE QUESTO NON SIA RICHIESTO DALLE LEGGI VIGENTI O APPAIA IN UN ACCORDO SCRITTO. SONO INCLUSI DANNI GENERICI, SPECIALI O INCIDENTALI, COME PURE I DANNI CHE CONSEGUONO DALL'USO O DALL'IMPOSSIBILITÀ DI USARE IL PROGRAMMA; CIÒ COMPRENDE, SENZA LIMITARSI A QUESTO, LA PERDITA DI DATI, LA CORRUZIONE DEI DATI, LE PERDITE SOSTENUTE DALL'ACQUIRENTE O DA TERZI E L'INCAPACITÀ DEL PROGRAMMA A INTERAGIRE CON ALTRI PROGRAMMI, ANCHE SE IL DETENTORE O ALTRE PARTI SONO STATE AVVISATE DELLA POSSIBILITÀ DI QUESTI DANNI.

FINE DEI TERMINI E DELLE CONDIZIONI

Appendice: come applicare questi termini a nuovi programmi

Se si sviluppa un nuovo programma e lo si vuole rendere della maggiore utilità possibile per il pubblico, la cosa migliore da fare è rendere tale programma libero, cosicché ciascuno possa ridistribuirlo e modificarlo sotto questi termini.

Per fare questo, si inserisca nel programma la seguente nota. La cosa

migliore da fare è mettere la nota all`inizio di ogni file sorgente, per chiarire nel modo più efficiente possibile l'assenza di garanzia; ogni file dovrebbe contenere almeno la nota di copyright e l'indicazione di dove trovare l'intera nota.

> <una riga per dire in breve il nome del programma e cosa fa>
> Copyright (C) <anno> <nome dell'autore>
>
> Questo programma è software libero; è lecito redistribuirlo o modificarlo secondo i termini della Licenza Pubblica Generica GNU come è pubblicata dalla Free Software Foundation; o la versione 2 della licenza o (a propria scelta) una versione successiva.
>
> Questo programma è distribuito nella speranza che sia utile, ma SENZA ALCUNA GARANZIA; senza neppure la garanzia implicita di NEGOZIABILITÀ o di APPLICABILITÀ PER UN PARTICOLARE SCOPO. Si veda la Licenza Pubblica Generica GNU per avere maggiori dettagli.
>
> Questo programma deve essere distribuito assieme ad una copia della Licenza Pubblica Generica GNU; in caso contrario, se ne può ottenere una scrivendo alla Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Si aggiungano anche informazioni su come si può essere contattati tramite posta elettronica e cartacea.

Se il programma è interattivo, si faccia in modo che stampi una breve nota simile a questa quando viene usato interattivamente:

> Orcaloca versione 69, Copyright (C) anno nome dell'autore
> Orcaloca non ha ALCUNA GARANZIA; per dettagli usare il comando `show g'.
> Questo è software libero, e ognuno è libero di ridistribuirlo secondo
> certe condizioni; usare il comando `show c' per i dettagli.

Gli ipotetici comandi "show g" e "show c" mostreranno le parti appropriate della Licenza Pubblica Generica. Chiaramente, i comandi usati possono essere chiamati diversamente da "show g" e "show c" e possono anche essere selezionati con il mouse o attraverso un menù, o comunque sia pertinente al programma.

Se necessario, si deve anche far firmare al proprio datore di lavoro (per chi lavora come programmatore) o alla propria scuola, per chi è studente, una "rinuncia al copyright" per il programma. Ecco un esempio con nomi fittizi:

> Yoyodinamica SPA rinuncia con questo documento ad ogni diritto sul copyright del programma `Orcaloca' (che svolge dei passi di compilazione) scritto da Giovanni Smanettone.
>
> <firma di Primo Tizio>, 1 April 3000
> Primo Tizio, Presidente

I programmi coperti da questa Licenza Pubblica Generica non possono essere incorporati all'interno di programmi proprietari. Se il proprio programma è una libreria di funzioni, può essere più utile permettere di collegare applicazioni proprietarie alla libreria. Se si ha questa intenzione consigliamo di usare la Licenza Pubblica Generica Minore GNU (LGPL) invece di questa Licenza.

GLOSSARIO

Hacker. Un hacker è una persona che si impegna nell'affrontare sfide intellettuali per aggirare o superare creativamente le limitazioni che le vengono imposte, in primo luogo nei suoi ambienti di interesse, che solitamente comprendono l'informatica o l'ingegneria elettronica. Esiste peraltro un luogo comune, utilizzato soprattutto dai media, per cui il termine hacker viene associato ai criminali informatici, la cui corretta definizione è cracker; i mezzi usati sono in genere gli stessi ma gli scopi sono opposti: da un lato, gli hacker intendono scoprire per comprendere e imparare, dall'altro, i cracker, intendono scoprire per distruggere.

Il termine originale si è sviluppato al MIT molto prima che i computer divenissero strumenti di uso comune; "hack" indicava una semplice (ma spesso elegante) soluzione. In seguito il termine hack cominciò ad essere usato per indicare ogni intervento o "scherzo" astuto perpetrato dagli studenti del MIT; il responsabile dell'intervento era detto hacker. Al MIT questi termini vengono usati allo stesso modo ancora oggi, senza riferirsi necessariamente ai computer. La cultura informatica si è sviluppata al MIT quando i membri del "Tech Model Railroad Club" iniziarono a lavorare con un elaboratore digitale PDP-1 e applicarono il loro *gergo* ai computer. Attualmente, nella cultura informatica, essere indicati come "hacker" è un complimento, che indica un programmatore abile e preparato mentre nel gergo mediatico è diventato sinonimo di criminale informatico, o cracker. Il termine hacker viene usato, in linea di massima, con i seguenti significati:

● Qualcuno che conosce un modello di interfaccia di programmazione abbastanza bene da essere in grado di scrivere un software nuovo e utile senza troppa fatica, in una giornata o comunque rapidamente.

● Qualcuno che (di solito illegalmente) riesce a oltrepassare o sovvertire la sicurezza di un sistema, di un programma o di una rete, spesso con intenti dannosi. Anche detti "black hat hacker", o "cracker"

● Qualcuno che riesce ad inserirsi in un sistema o in una rete per aiutare i proprietari a prendere coscienza di un problema di sicurezza, anche detti "white hat hacker" o "sneacker". Molte di queste persone sono impiegate in aziende di sicurezza informatica e lavorano nella completa legalità. Gli altri ricadono nella definizione precedente.

● Qualcuno che, attraverso l'esperienza o per tentativi successivi, modifica un software esistente in modo tale da rendere disponibile una nuova funzione. Più che una competizione, lo scambio tra diversi programmatori di modifiche sui relativi software è visto come un'occasione di collaborazione.

Rete (Network). Il concetto di rete è molto vasto e può riguardare contesti molto diversi, si possono infatti distinguere reti di trasporti, reti sociali e di business, reti di telecomunicazione. In relazione al tema trattato in questo elaborato il termine rete è stato usato per indicarne l'uso in ambito informatico, cioè il trasferimento e la condivisione di informazioni e risorse fra più computer; da questo punto di vista si possono distinguere le *reti locali* o LAN (Local Area Network) che si estendono all'interno di un edificio o di un comprensorio, con una estensione entro i 10 km e le *reti geografiche* o WAN (Wide Area Network) che si estende oltre i limiti indicati finora (tipicamente è una rete che copre una nazione, o un continente, se non addirittura il pianeta) come Internet. Nel quarto capitolo lo si è invece utilizzato in senso socio-economico dove la rete sta ad indicare l'insieme di relazioni che si si instaurano fra soggetti, individuali o collettivi, e che ne condizionano il comportamento e le scelte, da qui il concetto di esternalità di rete, per cui il valore della rete e l'importanza dell'appartenenza crescono in modo esponenziale al crescere del numero di soggetti che vi fanno parte.

BIBLIOGRAFIA

ACEVEDO A. (2003), *GNU/LinEx: l'Information Society e il Free Software in Extremadura*, Materiale del convegno "La conoscenza come bene pubblico comune: software, dati, saperi" http://www.csi.it/opensource_ita

AZAIS C., CORSANI A., DIEUAIDE P. (2001), *Vers un capitalisme cognitif*, Parigi, l'Harmanattan.

BENSEN S.M., FARREL J. (1994), Choosing how to compete: strategies and tactics in the standardization, in, *Journal of Economic Perspectives*, 8(2).

BONACCORSI A. ROSSI C. (2001), L'Economia degli Standard e la Diffusione delle Tecnologie. L'Open Source non è un Assurdo Economico, in, *Economia e politica industriale*, 29, pp. 19 – 41 http://www.istitutocolli.org/prosos/materiali/BonaccorsiRossi2001.pdf

DIDONE' L. (2001), *Modelli di Business per il Software Libero*, Trento, Università degli Studi, Facoltà di Economia. http://www.superdido.com/luca/tesi/

EVERS S. (2000), *An Introduction To Open Source Software Development*, Technische Universität Berlin, Fachbereich Informatik, Fachgebiet Formale Modelle, Logik und Programmierung (FLP)

FINK M. (2003), *Modelli di Business per Linux e Open Source*, Milano, Pearson Education Italia Srl

FREE SOFTWARE FOUNDATION (FSF) (2002), *Cos'è il Software Libero?* http://www.gnu.org/philosophy/free-sw.it.html

FREE SOFTWARE FOUNDATION (FSF) (2002), *Perché "Software Libero" è meglio di "Open Source"*. http://www.gnu.org/philosophy/free-software-for-freedom.it.html

FREE SOFTWARE FOUNDATION (FSF) (2004), *Filosofia del Progetto GNU*. http://www.gnu.org/philosophy/philosophy.it.html

GARFINKEL S., STALLMAN R. e KAPOR M. (1991), *Perché i brevetti sono dannosi per il software*, Issues in Science and Technology. Traduzione in lingua italiana http://www.pluto.it/meeting/meeting1999/atti/

GARVIN D.A. (1984), *What does "Product Quality" really mean?*, in "Sloan Management Review", 26, 1, pp.25-44

INGUAGGIATO D. (1999), *L'Open Source Software nella Internet Economy*, Torino, Università degli Studi, Scuola di Amministrazione Aziendale. http://www.liberliber.it/biblioteca/tesi/economia/

LEVY S. (1984), *Hackers, gli eroi della rivoluzione informatica*, Edizioni Shake (1996)

MACHLUP F. (1958), *The Economic Foundations of Patent Law* http://www.ipmall.fplc.edu/hosted_resources/jepson/unit1/aneconom.htm

MINISTERO PER L'INNOVAZIONE E LA TECNOLOGIA (2003), *Indagine conoscitiva sul software a codice sorgente aperto nella Pubblica Amministrazione* http://www.innovazione.gov.it/ita/intervento/normativa/indagine_opensource.shtml

MUFFATTO M., FALDANI M. (2003), Open Source, Padova, Edizioni il Mulino

OLSON M. (1965), *The Logic of Collective Action*, Cambridge University Press, Cambridge Massachusetts.

PERENS B. (1999), The Open Source Definition, in, *Open Sources - Voci dalla rivoluzione Open Source,* Apogeo.

RAYMOND E. (1998a), The Cathedral and the Bazaar, Apogeonline.
http://www.apogeonline.com/openpress/doc/cathedral.html

RAYMOND E. (1998b), Colonizzare *la Noosfera*, Apogeonline.
http://www.apogeonline.com/openpress/doc/homesteading.html

RAYMOND E. (1999a), La Vendetta degli Hacker, in, *Open Sources - Voci dalla rivoluzione Open Source,* Apogeo.
http://www.apogeonline.com/openpress/libri/545/raymondb.html

RAYMOND E. (1999b), *Il Calderone Magico*, Apogeonline
http://www.apogeonline.com/openpress/doc/calderone.html

SHAPIRO C., VARIAN H. (1999), *Information Rules: Le Regole dell'Economia dell'Informazione*, Cambridge, Etas

SHAPIRO C., VARIAN H. (2003), *Linux Adoption in the Public Sector: An Economic Analysis*, Berkeley.
http://www.sims.berkeley.edu/~hal/Papers/OpenSource/

STALLMAN R. (1985), *Il manifesto* GNU, Apogeonline.
http://www.gnu.org/gnu/manifesto.it.html

STALLMAN R. (1996), *Cos'è il Copyleft?*
Traduzione non ufficiale in lingua italiana
http://internet.cybermesa.com/~berny/cosacopyleft.html
Versione ufficiale in lingua inglese
http://www.gnu.org/copyleft/copyleft.html#TOCWhatIsCopyleft

STALLMAN R. (1999), Il progetto GNU in *Open Sources – Voci dalla rivoluzione Open Source,* Apogeonline.
http://www.apogeonline.com/openpress/libri/545/index.html

STALLMAN R. (2000) , *Vendere Software Libero*, Free Software Foundation.
http://www.gnu.org/philosophy/selling.it.html

TORVALDS L. (1999), *Linux History*, Linux International.
http://www.li.org/linuxhistory.php

YOUNG R. (1999), *Regalato! Come Red Hat Software si trovò fra le mani un nuovo modello economico e contribuì a migliorare un'industria*, Apogeonline. http://www.apogeonline.com/openpress/libri/545/young.html

VARIAN H. (1996), *Differential Pricing and Efficiency*, FirstMonday
http://www.firstmonday.org/issues/issue2/different/index.html

WEBER S. (2000), *The Political Economy of Open Source Software*, Berkeley, BRIE Working Paper 140 http://e-conomy.berkeley.edu/publications/wp/wp140.pdf

WIKIPEDIA (2004) http://it.wikipedia.org/wiki/Pubblico_dominio

WIKIPEDIA (2004) http://it.wikipedia.org/wiki/Licenza_Opensource

LICENZA PER DOCUMENTAZIONE LIBERA GNU [174]

GNU Free Documentation License
Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.
This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.
We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.
A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.
A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters)and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.
The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.
The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.
A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available

[174] Versione originale in lingua inglese http://www.gnu.org/licenses/fdl.html#SEC4
Traduzione non originale della versione 1.1 http://www.softwarelibero.it/gnudoc/fdl.it.html

drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".
Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.
The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.
A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.
The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may acceptcompensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.
You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.
If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.
If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque

copy (directly or through your agents or retailers) of that edition to the public.
It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.
You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.
The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.
The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.
In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections
Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.
You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.
If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may

include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.
If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.
See http://www.gnu.org/copyleft/.
Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME.
> Permission is granted to copy, distribute and/or modify this document
> under the terms of the GNU Free Documentation License, Version 1.2
> or any later version published by the Free Software Foundation;
> with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
> A copy of the license is included in the section entitled "GNU
> Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the
> Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.