SOLANGE ZANOTTI

Open Source Initiative

Tesi di laurea in giurisprudenza
dell'Università di Pavia
discussa nell'anno accademico 2002-2003

INDICE

p.

INTRODUZIONE

Open source: il termine, ancora poco conosciuto in Italia, descrive una realtà ormai affermata e preponderante nel mondo del software; mentre nell'ambito del tradizionale software proprietario, il codice sorgente, vale a dire il contenuto creativo del programma, è segreto e noto solo ai tecnici che lavorano al suo sviluppo, quello open source (letteralmente sorgente aperto) è software il cui codice sorgente è reso pubblico, in modo che chiunque possa aggiungere il proprio contenuto. L'idea rivoluzionaria di condivisione del software insita nel concetto d'open source, non solo ha sconvolto la tradizionale idea di copyright, ma ha introdotto un nuovo modo d'intendere la proprietà intellettuale. Tale nuovo fenomeno si è spinto addirittura oltre: ha dato origine ad un modello di sviluppo del software originale e, cosa più importante, ha creato aree di mercato non ancora sfruttate, affermando al contempo un nuovo modo d'intendere il mondo degli affari. In questo nuovo contesto culturale, economico e giuridico, l'Open Source Iniziative si preoccupa di tracciare le linee guida fondamentali per definire cosa debba essere effettivamente considerato software open source.

CAPITOLO PRIMO
ORIGINI, STRUTTURA E ATTIVITA'

Sommario: 1. Origine dell'idea di condivisione del software. – 2. ARPAnet. – 3. Unix. – 4. L' esperienza di Unix a Berkeley. - 5. La situazione a cavallo degli anni ottanta. – 6. Il progetto GNU. – 7. Il manifesto GNU. – 8. La GPL. – 9. I primi anni novanta. – 10. La nascita di Linux. – 11. Un originale metodo di sviluppo del software. – 12. La commercializzazione di Linux – 13. Rapporto tra free software e open source. – 14. L'Open Source Definition. – 15. Il ruolo dell'Open Source Initiative. – 16. Le licenze open source e il pubblico dominio. – 17. Le licenze open source in generale. – 18. La GPL e la LGPL. – 19. La licenza X, BSD e Apache. – 20. La Licenza Artistica. – 21. La NPL e la MPL. – 22. Come scegliere una licenza. – 23. La disputa KDE-GNOME. - 24. I pericoli da evitare.

1. Il concetto di free software non è nuovo. Agli albori dell'informatica, dalla metà degli anni '40 ai primi anni '70, il software era, il più delle volte, distribuito a codici sorgenti aperti affinché gli altri programmatori potessero conoscerne il funzionamento ed apportare modifiche per migliorarne le prestazioni o personalizzarlo; strettamente collegato alla cultura hacker, cominciò a diffondersi in ambiente accademico, soprattutto all'interno degli istituti d'ingegneria e di fisica delle università, per merito di programmatori entusiasti, il cui rapporto col software era di puro piacere: in un clima del simile, ogni nuova realizzazione o miglioramento era visto come qualcosa da condividere, accrescendo soddisfazione e prestigio personali.

L' ambiente di riferimento era quello dei centri di ricerca dei più famosi atenei statunitensi quali il MIT, Stanford University, Canergie-Mellon University, Berkeley ed altri, nonché le strutture di ricerca di importanti società come i Laboratori di Bell e il Palo Alto Research Center di Xerox Per comprendere la nascita e l' evoluzione del software e della cultura hacker, è necessario prendere come punto di riferimento una delle Università americane più prestigiose, la Cambridge University e, al suo interno, quella che fu la culla dei più grandi hacker statunitensi, il Massachusset Institute of Tecnology (MIT).

Una precisazione preliminare, tuttavia, è subito necessaria: l'espressione hacker nel senso di "pirata" è una confusione di termini in gran parte dovuta ai mezzi d'informazione. Le origini del termine hack devono essere ricercate nel vecchio gergo del MIT: l'espressione era stata a lungo usata per indicare gli scherzi elaborati che gli studenti del MIT s' inventavano regolarmente. In seguito, e in ambito più tecnico, il termine cominciò ad indicare un progetto intrapreso non solo per adempiere uno scopo specifico, ma che portava con sé il piacere della partecipazione. Ovviamente per essere un "vero hack", l'impresa di programmazione doveva mostrare innovazione, stile e virtuosismo tecnico. Fu così che con il termine hacker si cominciò ad indicare una persona che amava programmare, a cui piaceva essere bravo a farlo.

Fu proprio all'interno del MIT che andò, lentamente, prendendo forma un'etica, un modo di intendere il rapporto con le macchine che avrebbe finito per influenzare profondamente la cultura informatica successiva. In particolare, la comunità degli hacker sviluppò la convinzione che l'accesso ai computer dovesse essere assolutamente illimitato e completo. L'affermazione, che può parere oggi scontata, non lo era allora, quando le università erano dotate di una sola macchina per l'intera collettività. L'assenza di barriere tra l'hacker e lo strumento di cui egli si serviva nella sua ricerca di conoscenza, diventò quindi, il metodo più efficace per favorire il libero scambio delle informazioni.

L'avversione per qualsiasi barriera fisica o regola che impedisse l'accesso alle macchine, determinò altresì la tendenza a dubitare di qualsiasi autorità e a promuovere il decentramento. L' etica hacker avversava, in generale, la burocrazia: che fosse industriale, governativa o universitaria, era in ogni caso inconciliabile con lo spirito di ricerca. Nell' ottica hacker, la burocrazia si basava su regole arbitrarie, agli antipodi degli algoritmi logici, con cui operano le macchine, sulle quali si basava per rafforzare il proprio potere. La comunità hacker, al contrario, si caratterizzava per essere un sistema decentrato, all'interno del quale non c' era nessun leader ad impartire ordini: chiunque avrebbe potuto scovare un punto debole nel sistema e imbarcarsi in un ambizioso intervento di modifica e miglioramento del programma. La comunità hacker si distingueva, quindi, per un atteggiamento meritocratico, che dimostrava come l'attenzione fosse posta sul potenziale dell'individuo di far progredire lo stato dell'hackeraggio e sulla capacità di creare programmi innovativi degni d'ammirazione.

All'epoca, gli hacker del MIT non guardavano al software da loro creato come software libero, dal momento che tale concetto non era ancora stato enucleato, ma in realtà di questo si trattava. Era pratica diffusa che programmatori d'altre università o di qualche società convertissero il programma elaborato dagli hacker del MIT per adattarlo al proprio sistema: operazione attuabile poiché a questi era data la possibilità di vedere il codice, leggerlo, modificarlo o prelevarne alcune parti. Qui sta il nucleo dell'intera cultura hacker:

tutta l'informazione era libera, giacché, solo un libero scambio d'idee, avrebbe consentito lo sviluppo di una maggiore creatività complessiva. Tale radicata convinzione derivava dall'osservazione del modo in cui un ottimo computer o un valido programma lavorava: pensando ai bit[1] binari come elementi che si muovono lungo il percorso più logico e diretto a svolgere il loro compito, si arrivava a pensare al computer come ad uno strumento che beneficia di un libero flusso d'informazioni.

La nascita ufficiale della cultura è 1961, anno in cui una società di recente fondazione, la Digital Equipment Corporation (DEC), consegnò al MIT, assolutamente gratis, un nuovo computer: prodotto dell' evoluzione informatica, si trattava del Pdp-1, il primo minicomputer progettato per la ricerca scientifica e l' elaborazione matematica. Occupandosi gli hacker del miglioramento di software e sistemi esistenti, aziende come la DEC cominciarono a seguire la politica di donare macchine e programmi a quelle entità che costituivano il terreno abituale degli hacker: se questi fossero riusciti a migliorarne il software, tali migliorie sarebbero state riutilizzate dalle stesse aziende, che le avrebbero incorporate alle nuove versioni dei programmi destinate al mercato commerciale. Gli hacker rappresentavano, quindi, una sorta di capitale comunitario, un'unità aggiuntiva per la ricerca e lo sviluppo a costi minimi.

E proprio al computer, o più precisamente al hardware[2] bisogna guardare, in quanto sia la nascita del laboratorio che il suo futuro declino furono fortemente influenzati dall'innovazioni nel campo della tecnologia informatica. Le sorti del hacking al MIT s'intrecciarono, infatti, alla serie di minicomputer PDP della DEC. Furono la flessibilità, la potenza e la relativa economicità di questa macchina che portarono la Cambridge University, così come molte altre università, al loro acquisto: il Pdp-6 prima e il PDP-10 poi rimasero le macchine preferite dagli hacker per più di 15 anni. Anche al MIT si utilizzavano questi elaboratori, con una variante; gli hacker rifiutarono il sistema operativo originale del progetto MAC, programma di ricerca sostenuto dal Ministero della Difesa che aveva dato i natali al Compatible Time Sharing System (CTSS). In segno di protesta, gli hacker del MIT scelsero, infatti, di creare un proprio sistema operativo chiamato ITS (Incompatibile Time-sharing System): il nome suonava particolarmente ironico giacché, in fatto di compatibilità con sistemi e programmi diversi, l'ITS era molto più adattabile del CTSS. La decisione di creare l'ITS aveva anche una connotazione politica. Il sistema operativo era, infatti, stato specificamente progettato per il PDP-6. In questo modo, la macchina sarebbe rimasta ad uso esclusivo degli hacker del MIT, gli unici in grado di utilizzarlo.

L' ITS rappresentò l'incarnazione dell'etica hacker: al contrario di quasi tutti gli altri sistemi time-sharing, l'ITS non usava password**,** giacché era stato progettato per permettere agli hacker la massima libertà d'accesso a qualsiasi file[3] dell'utente. La struttura aperta dell'ITS incoraggiava gli utenti a consultare tali file, cercare bug[4] nei programmi e correggerli. Dominava la convinzione che i programmi non appartenessero agli individui, ma al mondo degli utenti. Questa fiducia era ben rappresentata dal modo di gestire il problema dei crash di sistema provocati intenzionalmente. Un rito hacker piuttosto diffuso era stato quello di penetrare nel sistema time-sharing al fine di bloccare la macchina. Ovviamente più il sistema era controllato, maggiore era la sfida e il piacere che si ricavava a mettere in ginocchio il sistema. Onde prevenire questo genere d'inconvenienti, l'ITS era dotato di un comando, la cui specifica funzione era quella di mandare in crash il sistema: era, infatti, sufficiente digitare "Kill system" e la macchina si sarebbe bloccata. L'idea, vincente e creativa, era quella di togliere il divertimento, rendendo banale il compito: l'ITS pareva dimostrare che la migliore sicurezza era, appunto, l'assenza di sicurezza.

Tale sistema operativo diventò, ben presto, la "palestra" degli hacker del MIT: i programmi scritti dagli hacker li aiutarono a programmare più facilmente e ad elaborare programmi che consentivano di sfruttare la potenza derivante dal massimo impiego della macchina. Per gli hacker, il codice del programma possedeva una bellezza propria: all'interno della comunità era emersa, addirittura, un'estetica dello stile di programmazione. A causa della memoria[5] limitata dei primi computer, gli hacker apprezzavano enormemente le tecniche innovative che permettevano ai programmi di fare operazioni complicate con

[1] Abbreviazione di binary digit (cifra binaria). Le uniche cifre binarie possibili sono 0 e 1. le cifre binarie vanno a formare i numeri binari, i quali si adattano perfettamente alle esigenze del computer, dal momento che molti dispositivi elettronici hanno solo due stati: on e off.

[2] Elementi fisici di un sistema d'elaborazione dati, vale a dire qualsiasi parte di un computer, periferica o dispositivo fisico in genere che è possibile toccare materialmente. La traduzione letterale di hardware, ossia "ferraglia", indica che qualsiasi componente di un sistema di elaborazione dati è solo un insieme di parti elettroniche e meccaniche, inutilizzabile se non c' è il software a farlo funzionare.

[3] Blocco di dati memorizzati su disco, nastro o altro supporto. Può contenere un programma, un documento o una raccolta di dati strutturati.

[4] Letteralmente insetto, indica un errore presente nel programma. Il termine si riferisce ad un episodio che risale alla nascita dei primi computer: nel cercare il motivo del malfunzionamento, i tecnici aprirono l'interno del computer, trovandovi un grosso insetto che provocava errori d'esecuzione del programma.

[5] Spazio interno al computer in cui sono memorizzate le informazioni in corso di elaborazione. Nei primi computer la memoria aveva dimensioni limitate poiché molto costosa.

pochissime istruzioni. Migliorare il programma significava, quindi, ottenere lo stesso risultato con un numero minore d'istruzioni. In alcuni casi, ridurre un programma diventava una competizione accanita per dimostrarsi padroni del sistema: la sfida sarebbe stata, allora, quella di eliminare un certo numero d'istruzioni superflue o, meglio ancora, ripensare l'intero problema e approntare un nuovo algoritmo, che facesse risparmiare un intero blocco d'istruzioni. L'ITS può essere considerata, in definitiva, la più forte espressione dell'etica hacker. Erano molti quelli che pensavano che sarebbe dovuto diventare uno standard nazionale per tutti i sistemi time-sharing: non solo aveva eliminato l'odioso concetto di password e spronato la pratica del debugging, ma aveva dimostrato il potere sinergico proveniente dalla condivisione del software, dove i programmi non appartengono all'autore, ma a tutti gli utenti.

Nel 1968, le più importanti istituzioni d'informatica si radunarono all'Università dello Utah con l'obiettivo dichiarato di concordare un sistema operativo time-sharing da usare sull'ultima macchina DEC, il Pdp-10. Uno dei due sistemi presi in considerazione fu proprio l'Incompatibile Time-Sharing System degli hacker del MIT. Con tutta probabilità, la scelta di un sistema basato sull'etica hacker (che tra le altre cose non garantiva alcuna riservatezza) sarebbe, tuttavia, stata un passo troppo drastico da compiere. La bocciatura dell'ITS deve, quindi, essere ricondotta a una scelta politica, più che tecnica. All' epoca, in ogni caso, la diffusione dell'etica hacker al di là dei confini del MIT, non era considerata un obiettivo prioritario.

2. La nascita di una cultura informatica deve, però, essere ricollegata ad un'altra importante innovazione, ARPAnet, che vide la luce nel 1969. Fondata dalla Defence Advanced Research Projects Agency (DARPA), una sezione del Ministero della Difesa statunitense, è stata la prima rete transcontinentale di computer ad alta velocità. Ideata e realizzata come esperimento nel campo delle comunicazioni digitali, diventò un collegamento tra centinaia di università, esponenti della difesa e laboratori di ricerca: permise ai ricercatori, ovunque si trovassero, di scambiarsi informazioni con velocità e flessibilità senza precedenti, dando un forte impulso alla collaborazione, e accelerando enormemente il ritmo del progresso tecnologico.

La prima rete sperimentale entrò in funzione il 1 settembre del 1969, con la costituzione di quattro nodi presso l'Università della California a Los Angeles, lo Stanford Research Institute, l' Università della California di Santa Barbara e l' Università dello Utah. Il sistema fu, in seguito esteso a tutti i maggiori centri universitari che collaboravano col Dipartimento della Difesa, e ad altri centri di ricerca scientifica che non erano in relazione con organismi militari: vennero così a passare in rete sia i flussi di comunicazione per la ricerca scientifica militare, sia quelli per la ricerca civile; in seguito, un altro ente governativo, la National Science Foundation, creò una terza rete destinata alle comunicazioni tra università per le scienze sociali; negli anni ottanta le tre reti furono riunite con la denominazione di ARPAnet. Se l'iniziale progettazione della rete deve, quindi, essere attribuita al complesso scientifico-militare del Dipartimento della Difesa, il suo primo, libero sviluppo è da attribuirsi al mondo delle Università americane e alle varie comunità accademiche di professori e di studenti, che intuirono le possibilità di comunicazione e di collaborazione che si prospettavano attraverso la diffusione del nuovo sistema. Proprio all'interno dell'ambito accademico si formarono le prime comunità virtuali: tramite la rete trovarono, cosi, espressione, oltre alle relazioni scientifiche, anche le tendenze pacifiste, ambientaliste e libertarie degli studenti di quegli anni; in particolare, ARPAnet fece in modo che gli utenti cominciassero a condividere un certo senso d'appartenenza ad una comunità, e sentissero il bisogno di divulgare ciò che scoprivano.

In campo informatico, ARPAnet si rivelò un eccezionale veicolo di scambio d'informazioni tra università e laboratori di ricerca, permettendo la collaborazione tra le migliori menti del pianeta, con un'effettiva spinta verso la crescita della conoscenza tecnologica, soprattutto nel campo del software. Quest'ultimo era trasferito 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. ARPAnet assolse, quindi, ad un compito di fondamentale importanza: mise in contatto gli hacker di tutti gli Stati Uniti, i quali, fino allora isolati in gruppi sparuti, si scoprirono una vera e propria tribù. Non è da sottovalutare, in questo senso il contributo del Dipartimento della Difesa Americano, il quale chiuse deliberatamente un occhio sul proliferare dei mailing list su ARPAnet: sopportare maggiori costi, era lo scotto che poteva essere pagato per attrarre un'intera generazione di giovani brillanti ed appassionati nel campo dei computer e della programmazione.

Nel 1990, dopo oltre vent'anni d'esercizio, l'attività di ARPAnet cessò. Al suo posto subentrò la NSFnet, che operò sino al 1995. Dopodiché, vari soggetti commerciali subentrarono alle reti regionali NSF, realizzando in breve tempo la completa privatizzazione e provocando la nascita di Internet. La crescita della rete determinò la creazione di un sistema planetario, che attualmente collega circa diecimila reti concernenti servizi commerciali, aziende, enti ed istituzioni private, università e centri di ricerca, uffici pubblici e statali, ed altre reti varie d'ambito locale.

Dal punto di viste giuridico, se la rete faceva, inizialmente, parte della struttura militare e la sua funzione era soltanto quella di essere al servizio della difesa degli Stati Uniti, a partire dal 1995, la stessa è stata edificata all'interno di un regime civile di tipo privatistico, che ha permesso una comunicazione libera,

poco costosa e non sottoposta a vincoli, limiti, autorità, regolamentazioni, sanzioni. Ancora oggi, infatti, non esiste alcuna autorità internazionale che detti una disciplina specifica per Internet. Con la rete si è quindi realizzato il paradosso di una tecnologia che, nata per essere al servizio del maggior complesso militare del mondo, è, in una prima fase, diventata luogo virtuale di diffusione della controcultura accademica per poi trasformarsi in zona franca delle illimitate comunicazioni globali. In questo spazio virtuale ha preso corpo una comunità planetaria, all'interno della quale trovano realizzazione nuove forme di collaborazione e di solidarietà.

3. L' anno di nascita di ARPAnet fu anche l' anno in cui un hacker dei Laboratori Bell, di nome Ken Thompson, inventò Unix, nell' intento di realizzare un sistema operativo più semplice da usare e da programmare rispetto ai complessi sistemi esistenti. Mentre Thompson si dedicava alla costruzione di Unix, un altro hacker dei Laboratori Bell, Dennis Ritchie, lavorava alla creazione di un nuovo linguaggio di programmazione chiamato "C" che avrebbe dovuto essere usato con una versione Unix di Thompson ancora allo stato embrionale. Al pari di Unix, anche C fu progettato per essere piacevole, facile da usare, nonché flessibile. Si stava compiendo un passo epocale: i due hacker furono infatti tra i primi a capire che la tecnologia dell' hardware aveva raggiunto un livello di maturità tale da poter scrivere in C un intero sistema operativo: nel 1974 il nuovo ambiente software era regolarmente installato su numerose macchine di diversa tipologia. La possibilità d'installare un sistema operativo completo su macchine diverse rappresentò, a quel tempo, un evento senza precedenti, che portò con se implicazioni rivoluzionarie: se Unix poteva presentare la stessa interfaccia e le stesse funzionalità su macchine di diverso tipo, avrebbe, di conseguenza, potuto fungere da ambiente software comune per tutte; ciò significava che gli utenti non avrebbero più dovuto pagare per nuovi software, appositamente progettati, ogni volta che una macchina diventava obsoleta, e che gli hacker erano in grado di utilizzare gli stessi strumenti software da una macchina all'altra. Tecnicamente, si parla di portabilità, vale a dire d'adattabilità di un programma da un computer ad un altro.

Oltre alla portabilità, Unix e C presentavano altri notevoli punti di forza: entrambi si fondavano sulla filosofia " Keep it simple, stupid!". Unix fu, infatti, strutturato come un pacchetto flessibile di semplici programmi pensati per combinarsi in vari modi secondo le caratteristiche dell'elaboratore. L' intera struttura logica di C, d'altro canto, avrebbe potuto essere memorizzata da un qualsiasi programmatore, senza dover ricorrere continuamente a manuali. Queste caratteristiche fecero sì che Unix fosse presto adottato dalla maggior parte delle università, dei laboratori di ricerca informatica ma soprattutto da una moltitudine di hacker. Unix aveva, infatti, integrato, al suo interno, una specifica task 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 di utenti Unix che diede un grosso impulso al fenomeno della diffusione e condivisione del software; seppur lento, questo metodo di trasmissione consentì a qualsiasi utilizzatore di questo sistema operativo di comunicare senza dover accedere ad ARPAnet, riservato, a quel tempo, a pochi soggetti istituzionali ben determinati. La nascita di una vera e propria comunità di utenti Unix furono anche facilitata da una circostanza particolare. AT&T, il colosso telefonico cui faceva capo i Bell Labs, in cui nacque e fu sviluppato inizialmente Unix, fu accusata nel 1949 dalla Divisione Antitrust del Dipartimento di Giustizia statunitense di aver violato lo Sherman Antitrust Act. Ciò portò nel 1956 ad un consent decree che diffidava AT&T dall'intraprendere qualsiasi attività commerciale diversa dai servizi telefonici e telegrafici. Fu per non violare tale disposizione che AT&T decise che, nel caso del software, la propria politica sarebbe stata di licenziarlo, come consentito dal decreto, ma non di svilupparvi una politica commerciale. In quest'ottica Unix fu quindi fornito - a pagamento - a chi lo richiedesse, ma a scatola chiusa, senza in pratica nessun servizio correlato né d'assistenza né di correzione di bug. Senza alcun supporto, la nascente comunità di utenti Unix fu in un certo senso costretta dalla necessità a sviluppare una cultura di solidarietà e condivisione, mettendo in comune idee, informazioni, programmi, modifiche sia hardware che software. Al contrario di ciò che avevano fatto gli hacker del MIT con il PDP-6 e l'ITS, i programmatori di Unix non costruirono il loro sistema operativo su una macchina specifica. Scelsero, invece, un approccio più generico: concentrandosi sugli standard che tenevano uniti le numerose sotto-componenti del sistema, crearono un sistema operativo che avrebbe potuto essere velocemente modificato e adattato alle diverse macchine su cui veniva installato. Se è vero che Unix mancava dell'eleganza e dell'estetica di programmi come l'ITS, tuttavia poteva vantare una flessibilità ed economicità, che ne garantirono la rapida diffusione.

4. Sviluppato, originariamente, dai giovani ricercatori dei Laboratori Bell, l'adattabilità e l'economicità di Unix sfuggirono alle grande aziende di software, cosicché il sistema operativo si sviluppò nell' ambito dell' informatica accademica. E' stata, in particolare, l'Università della California di Berkeley ad unire indissolubilmente il suo nome a quello di Unix. La prima installazione risale al gennaio del 1974:considerati i problemi iniziali, determinanti si rivelarono la disponibilità e l' aiuto dei ricercatori dei

Laboratori Bell, nel miglioramento del software disponibile a Berkeley. L'anno successivo, due studenti appena laureati, Bill Joy e Chuck Haley, si fecero notare per il lavoro svolto sul nuovo sistema operativo; si dedicarono, in particolar modo, al miglioramento dell'interprete del Pascal, trasformandolo nel sistema di programmazione scelto per gli studenti. La gran richiesta di copie del sistema che ne scaturì spinse Joy a fondare, all'inizio del 1977, la Berkeley Software Distribution (BSD).

La storia del sistema Unix, e del BSD in particolare, ha per prima dimostrato il potere del rendere il sorgente disponibile agli utenti: una volta attribuita loro la facoltà di osservare il funzionamento del programma, questi non si sono quasi mai limitati a un uso passivo del sistema, ma hanno lavorato attivamente al fine di scovarne gli errori, migliorarne le prestazioni e la funzionalità e perfino per aggiungere nuove caratteristiche. Tale metodo di sviluppo non era altro che il risultato di una tradizione particolarmente radicata all'interno della comunità informatica dell'Università di Berkeley: il suo codice di comportamento prevedeva, infatti, che i suggerimenti raccolti ritornassero alla comunità, un aspetto, quest'ultimo, che, si sarebbe rivelato fondamentale per il modello di sviluppo open source. Anziché vagliare le informazioni attraverso cinque o sei strati intermedi prima di arrivare agli sviluppatori, accadeva che lo sviluppatore prendesse contatto direttamente con gli utenti per scoprire le loro necessità. E' proprio focalizzando l'attenzione su quest'aspetto, che diventa palese come il software libero non sia mai stato, quindi, solo una questione di licenza, ma un modello, un processo di sviluppo che ha coinvolto gli utenti sin dall'inizio. Fino alla fine degli anni settanta, i computer furono considerati strumenti di ricerca: il software prodotto per il loro funzionamento e per le varie applicazioni circolava liberamente, come un qualsiasi prodotto di conoscenza scientifica ed i programmatori erano pagati per il fatto di programmare e non per i programmi che producevano. Tutti i progetti di sviluppo cooperativo di software erano intrapresi e procedevano su basi informali. Non era sentito come una priorità, il problema dei diritti proprietari o delle eventuali restrizioni all'uso del software. AT&T stessa non aveva mai mostrato, fino allora, alcun interesse a sfruttare la sua posizione di privilegio. Tuttavia, la grossa diffusione di Unix e il suo continuo miglioramento, grazie ai contributi di università e singoli utilizzatori, furono percepiti dall'azienda come un fenomeno utile ma sostanzialmente incontrollabile, e potenzialmente dannoso a livello economico. Fu così che la licenza che accompagnava, nel 1979, la Settima Versione di Unix limitò lo studio del codice sorgente, bloccando il suo sviluppo in diversi ambienti universitari. Con quest'azione AT&T intraprese, di fatto, la strada che avrebbe portato alla vera e propria commercializzazione di Unix. La Settima Versione fu, infatti, l'ultima sviluppata dai laboratori Bell; dopodiché, tutti i seguenti rilasci di Unix AT&T furono sviluppati e gestiti da un altro gruppo, con espliciti fini commerciali. Con la commercializzazione di Unix, i ricercatori dei Laboratori Bell non furono più in grado di fungere da standard per la continuazione della ricerca; dato, però, che i ricercatori continuavano a modificare il sistema, essi si resero immediatamente conto che avrebbe avuto bisogno di un'organizzazione che potesse produrre versioni di ricerca: a causa del suo originario coinvolgimento con Unix e il proprio passato di distributrice di strumenti basati su tale sistema, Berkeley si ritrovò a ricoprire il ruolo precedentemente svolto dai Laboratori Bell.

Lo sviluppo di Unix, portato avanti dall' università di Berkeley, fu influenzato dai piani della DARPA (Defence Advanced Research Prjecy Agency). Come abbiamo visto' uno dei primi successi della DARPA era stato quello di strutturare una rete di computer a livello nazionale, che aveva permesso di collegare insieme tutti i maggiori centri di ricerca. Si era reso presto evidente, tuttavia, che molti dei computer di questi centri avrebbero dovuti essere rimpiazzati: il costo maggiore di questa sostituzione era costituito dal trasferimento del software di ricerca sulle nuove macchine. Furono analizzate diverse soluzioni: la scelta di un solo rivenditore hardware si rivelò impraticabile a causa dei differenti bisogni dei gruppi di ricerca, ma soprattutto a causa dell'interesse a non dipendere da un solo produttore. I pianificatori di DARPA, alla fine, decisero che la soluzione migliore sarebbe stata quella dell'unificazione a livello di sistema operativo. Fu così che, considerata la sua comprovata portabilità, la versione Unix di Berkeley fu scelta come sistema standard.

L'università di Berkeley concluse, quindi, un contratto con la DARPA, grazie al quale ricevette i fondi necessari a sviluppare una versione potenziata della 3BSD. A tale scopo, fu formato il Computer System Research Group (CSRG), il quale, per quasi un quindicennio avrebbe guidato lo sviluppo del software creato a Berkeley. In occasione dell'uscita della nuova versione, nell'autunno del 1980, anche il sistema di distribuzione delle copie fu potenziato, in modo da poter gestire un maggior numero d'ordini. Da questo momento in poi, le versioni Unix di Berkeley modificarono il loro sistema di numerazione. AT&T aveva, infatti, fatto notare che avrebbero potuto crearsi delle confusioni tra clienti con l'uscita commerciale di Unix System V e un'ulteriore uscita a Berkeley denominata 5BSD. Berkeley si mostrò, quindi, d'accordo a cambiare le serie di nomi per le uscite a venire, lasciando valido 4BSD e incrementando il numero iniziale con un decimale.

Seppur condiviso tra ricercatori e programmatori commerciali tramite un'apposita licenza, il codice della BSD conteneva codice proprietario dell'AT&T: ciò significava che tutti i destinatari della BSD dovevano preventivamente procurarsi una licenza del codice sorgente di rete di AT&T. L'aumento dei costi della licenza originaria AT&T spinse, tuttavia, i programmatori di Berkeley a implementare autonomamente

il protocollo di rete TCP/IP, in modo da non dover più richiedere alcuna autorizzazione a AT&T. il nuovo programma fu reso disponibile nel giugno del 1989 come Networking Release 1, il primo codice ridistribuito gratuitamente da Berkeley. I termini della licenza, in seguito conosciuta come licenza BSD, erano ampi: al licenziatario era permesso rilasciare il codice modificato o intatto, sorgente o in forma binaria, senza diritti per Berkeley. Si richiedeva soltanto che le specifiche ai diritti d'autore nel file sorgente fossero lasciate intatte: si voleva, inoltre, che i prodotti contenenti il codice indicassero nella loro documentazione che il prodotto conteneva codice dell' università di Berkeley; sebbene Berkeley richiedesse un contributo di 1000 dollari per un nastro, chiunque avrebbe potuto ottenerne una copia da chiunque altro che già lo avesse ricevuto. La Networking Release 1 fu ridistribuita gratuitamente sulla rete: la sua popolarità fu tale, da spingere i programmatori a preparare una versione ampliata che includesse più codice BSD. All'uopo, si ricorse alla tecnica di sviluppo di massa via rete: furono sollecitati i collaboratori a riscrivere, da zero, tutti i pacchetti Unix, eliminando il codice AT&T. Per tutti quelli che avessero partecipato all'impresa, l'unica soddisfazione sarebbe stata quella di leggere il proprio nome sulla lista dei collaboratori di Berkeley, accanto al nome dell'utilità che avevano riscritto. Un'attenta analisi mostrò che solo sei file richiedevano un'elaborazione più complessa: si decise, quindi, di rendere disponibile il codice di cui già si disponeva, senza attendere la riscrittura dei sei file. La versione gratuita allargata cominciò ad essere disponibile nel giugno del 1991; i termini di distribuzione e il costo erano gli stessi della prima versione. Ad ogni modo, l'intervallo dal Networking release 2 ad un sistema pienamente funzionante non fu lungo. Entro sei mesi dall'uscita, erano già state riscritte le modifiche ai sei file mancanti: fu allestito un sistema completamente compilato e scaricabile per i PC 386 che prese il nome di 386/BSD. Il suo creatore, Billy Jolitz non era, tuttavia, nella condizione di dedicare al progetto il tempo necessario per mantenere aggiornato il flusso di miglioramenti; alcuni d'utenti interessati formarono, allora, il gruppo NetBSD, con l'obiettivo di coadiuvare il mantenimento e il miglioramento del sistema. Le successive versioni presero, quindi, il nome di NetBSD: fino al 1998 circolarono solo sulla rete. In linea con la politica di distribuzione adottata dal gruppo, nessun altro mezzo di distribuzione fu reso disponibile: la NetBSD si rivolgeva, infatti, ai soli utenti tecnici. Accanto alla NetBSD, comparve e si sviluppò un altro gruppo: il FreeBSD**.** Interessato a supportare l'architettura dei PC, sviluppò un sistema che fu distribuito sul mercato su CD-ROM a basso costo. La FreeBSD si rivolgeva, infatti, ad una fascia d'utenti più ampia rispetto a NetBSD, ma meno avanzata tecnicamente[6].

Accanto a questi gruppi organizzati, che ridistribuivano gratuitamente i sistemi creati attorno al Networking Release 2, fu fondata la società Berkeley Software Design Incorporated (BSDI), con l'obiettivo di creare e distribuire una versione commerciale del codice. Subito dopo l'inizio della campagna vendite, tuttavia, l'Unix System Laboratories (USL), una società associata alla maggioritaria AT&T, sporse querela, sostenendo che il prodotto della BSDI conteneva codice proprietario USL, oltre ai suoi segreti commerciali, e diffidando la BSDI dalla vendita dello stesso prodotto. USL cercò, inoltre, di ottenere un'ingiunzione allo scopo d'interrompere le vendite della BSDI fino a quando la causa non fosse terminata: la società sosteneva che, se la vendita fosse continuata, sarebbe stata oggetto di danni irreparabili, dovuti alla rivelazione dei loro segreti commerciali. A quel punto BSDI si difese affermando che stava semplicemente utilizzando i sorgenti distribuiti gratuitamente dall'Università della California, più i sei file aggiuntivi. Il giudice accolse le ragioni della BSDI. USL, chiese quindi, nuovamente, un'ingiunzione sulla distribuzione sia dei prodotti dell'Università sia di quelli della BSDI. Nel dicembre del 1992, il giudice annullò l'ingiunzione e respinse tutte le richieste, tranne due. Queste ultime concernevano i diritti d'autore più recenti e la possibilità di perdita dei segreti commerciali. Il giudice suggerì, altresì, di sottoporre la questione al vaglio di una corte giurisdizionale statale, prima di essere proposta davanti ad una corte federale. L'università della California raccolse il suggerimento e si affrettò a contattare la corte giurisdizionale californiana, sporgendo una controquerela nei confronti dell'USL. Questa sosteneva che l'USL aveva omesso di citare i copyright richiesti dalla Licenza BSD dopo aver usato codice BSD nella propria distribuzione di Unix denominata System V. L'università chiedeva, quindi, che USL fosse obbligata a ristampare tutta la sua documentazione con i ringraziamenti esatti, notificando tutti i suoi licenziatari della sua svista, oltre a pubblicare una pagina pubblicitaria intera su tutte le pubblicazioni finanziarie più note. Subito dopo la presentazione della querela al tribunale, USL cambiò proprietario, da AT&T a Novell. I vertici societari cambiarono rotta e decisero, allora, che di perorare la loro causa sul mercato, piuttosto che in tribunale. Nell'estate del 1993 cominciò la conciliazione, che proseguì fino al gennaio del 1994. Il risultato fu che 3 file furono rimossi dai 18.000 creati nel Networking Release 2, e un numero di cambiamenti minori fu apportato ad altri file. In aggiunta, l'Università concordò di aggiungere i diritti d'autore dell'USL a circa altri 70 file, sebbene questi ultimi continuarono a essere distribuiti gratuitamente.

[6] Negli anni novanta, la FreeBSD ha, poi, sfruttato l'onda di popolarità di Linux. Aggiungendo una modalità simulata , ha, infatti, permesso ai file binari di Linux di essere eseguiti su una piattaforma FreeBSD. Questa caratteristica ha consentito agli utenti FreeBSD di usare il gruppo d'applicazioni, sempre in crescita, disponibili per Linux, sfruttando, al contempo le prestazioni del sistema FreeBSD.

5. La nascita e il rapido proliferare di Unix fece sì che, dalla seconda metà degli anni 70, il panorama informatico fosse dominato da due diverse culture, che vennero inevitabilmente a confrontarsi: gli hacker del PDP-10 da una parte, i seguaci di Unix dall'altra. Un terzo scenario stava, tuttavia, cominciando a prendere forma: nel 1975, infatti, fu immesso sul mercato il primo personal computer. Il nuovo strumento finì, inevitabilmente, per attrarre un'altra generazione di giovani e brillanti hacker. Ecco, quindi, la situazione alle porte degli anni 80: tre culture, simili ma organizzate intorno a diverse tecnologie, si contendevano il primato. La cultura di ARPAnet e del PDP-10 votata e al sistema operativo ITS, il popolo di Unix e C, coi loro PDP-11 e VAX; infine un'orda anarchica, composta per lo più dai primi appassionati di microcomputer. All'epoca, l'ITS poteva ancora vantare il posto d'onore, ma la situazione stava cambiando. Negli anni successivi alla guerra del Vietnam, infatti, i fondi a disposizione di quello che, storicamente era stato il maggiore finanziatore della ricerca informatica, il Dipartimento della Difesa, andavano rapidamente esaurendosi: laboratori e università cominciarono, allora, a rivolgersi al sistema privato, per la ricerca di fondi. Come prevedibile, il Laboratorio d'Intelligenza Artificiale del MIT non tardò ad ottenere il sostegno d'investitori privati; a partire dal 1980, la maggior parte degli hacker del Laboratorio d'Intelligenza Artificiale fu impegnata su un doppio fronte: da una parte il MIT, dall'altra i vari progetti commerciali. Presto, tuttavia, le aziende cominciarono ad assumere direttamente gli hacker, al fine di sfruttarne pienamente il lavoro: parecchi programmatori abbandonarono, quindi, il Laboratorio per firmare accordi di non-divulgazione con aziende produttrici di software. Simili clausole contrattuali garantivano al programmatore l'accesso al codice sorgente del software, in cambio della promessa di mantenerlo segreto. Oggi clausola standard, nei contratti dell'industria informatica, a quel tempo l'accordo di non divulgazione costituì una novità, un segnale del valore commerciale acquisito, per le aziende produttrici, sia dal software stesso sia dalle informazioni necessarie al relativo funzionamento. Il software era diventato, infatti, un bene troppo prezioso: le aziende non sentirono più la necessità di diffonderne i sorgenti, soprattutto per il fatto che tale divulgazione avrebbe garantito ai potenziali concorrenti la possibilità di duplicare un qualsiasi programma a costi irrisori.

La fine del Laboratorio d'Intelligenza Artificiale del MIT avvenne nel 1983, quando la DEC mise fuori produzione la serie PDP-10, per concentrarsi sulle linee PDP-11 e VAX: i programmi che formavano l'ITS diventarono inutilizzabili sui moderni computer, essendo stati progettati come software specifico per il PDP-10. Non solo: a causa della sua scarsa portabilità, l'idea di trasportare il sistema da un hardware all'altro era impensabile.

Il problema principale era, tuttavia, un altro. I primi tentativi di commercializzazione del software mal si conciliarono, infatti, con il valore che per anni aveva guidato e ispirato l'intera attività di programmazione del MIT: l'idea della condivisione del software. La questione della condivisione si rivelò centrale negli eventi che accaddero nei primi anni ottanta, fatti che, non solo posero fine alla peculiare comunità fiorita intorno al Laboratorio d'Intelligenza Artificiale, ma portarono alla creazione di un clone libero di Unix. Già, dai primi tentativi di commercializzazione della tecnologia, la comunità hacker appariva spaccata su due fronti contrapposti. Da una parte, un'azienda software nota come Symbolics, dall'altra la diretta rivale, la Lisp Machine Inc. (LMI): entrambe perseguivano l'obiettivo di lanciare sul mercato la Lisp Machine, un sistema operativo basato sul linguaggio di programmazione Lisp, messo a punto, nel corso degli anni settanta, dai programmatori del Laboratorio d'Intelligenza Artificiale. Il software per la Lisp Machine era di proprietà del MIT, anche se, nel rispetto della tradizione hacker, il codice era a disposizione di chiunque voleva copiarlo: questa pratica finì, di fatto, per limitare l'attività commerciale delle aziende che intendevano ottenere un programma sotto licenza MIT, per poi lanciarlo sul mercato come prodotto specifico. Fu la Symbolics che adottò la strategia più aggressiva: arruolò la maggior parte degli hacker del laboratorio al fine di sviluppare una propria versione della Lisp Machine; i restanti hacker del MIT si allearono con la Lisp Machine Inc.. Tutti tranne uno: Richard Stallman, il cui ruolo fu quello di aggiornare la Lisp Machine del Laboratorio, sulla base delle modifiche apportate dalle due le contendenti; sebbene, infatti, la Symbolics permettesse a Stallman di visionare i sorgenti del suo prodotto, ma non di copiarlo, un "gentlemen's agreement" tra la società e il laboratorio faceva in modo che Stallman, di fatto, potesse prelevare le parti più interessanti del codice. Nel marzo del 1982, tuttavia, i dirigenti della Symbolics posero fine all'accordo: l'intenzione era, chiaramente, quella d'impedire alla diretta concorrente sul mercato, di utilizzare la copia del sistema in dotazione del laboratorio. Stallman imputò, così, alla Symbolics la fine della cultura hacker nata al Laboratorio d'Intelligenza Artificiale e si schierò apertamente con la LMI, impegnandosi a riscrivere da capo ogni funzione e strumento della Lisp Machine. All'epoca, non tutti gli hacker condividevano il punto di vista radicale di Stallman: molti ritenevano, infatti, che con la diffusione della Lisp Machine sul mercato, la Symbolics aveva fatto in modo che i valori dell'etica hacker sulla progettazione informatica non restassero confinati al laboratorio, ma arrivassero ad un mercato di tipo imprenditoriale. Tuttavia, anche chi considerava Stallman una figura anacronistica, non poteva negare la sua abilità nel tenere testa all'intera equipe di programmatori della Symbolics: il lavoro svolto per la LMI non fece che consolidare la fama di

Stallman all'interno della comunità hacker.

La disputa sulla Lisp Machine, nonostante il clamore sollevato, non rappresentò che un evento collaterale, tra le grandi battaglie in corso nel mercato della tecnologia. La miniaturizzazione del computer procedeva a ritmo accelerato, realizzando microprocessori sempre più potenti che avrebbero ben presto consentito d'incorporare le capacità hardware e software in un'unica macchina. Su tali microprocessori giravano una miriade di programmi commerciali, ciascuno protetto da una pluralità di licenze d'uso e di accordi di non divulgazione, che impediva l'accesso al codice sorgente. Il software, una volta, considerato una sorta di decorazione offerta dai produttori di hardware per rendere più appetibili i loro costosi elaboratori, si stava trasformando nella punta di diamante del mercato informatico. Sempre più alla ricerca di nuovi giochi e funzioni, gli utenti andavano dimenticando la tradizionale richiesta di poter aver accesso al codice. Ignorando la cultura hacker e il suo rifiuto per il software in formato solo binario, la maggior parte degli utenti non vedeva motivo di protestare quando le grandi aziende produttrici di software non allegavano al proprio prodotto i file con il codice sorgente. Diventava evidente che, ben presto, le norme etiche non avrebbero più trovato posto, all'interno di un mercato che premiava i programmatori abbastanza veloci da elaborare programmi sempre nuovi, e abbastanza scaltri da porli sotto la tutela legale del copyright.

Arrivato al punto di non poter più competere con la Symbolics, che nel frattempo aveva creato un nuovo tipo di Lisp Machine, Stallman si trovò di fronte ad una difficile scelta etica: abbandonare tutte le obiezioni morali contro il software "proprietario", oppure dedicarsi alla costruzione di un sistema alternativo, non proprietario. Scegliendo la seconda soluzione, Stallman si trasformò, da hacker solitario, in attivista animatore di una crociata che avrebbe applicato i tradizionali valori della libertà, dell'uguaglianza e della fraternità al mondo dello sviluppo del software**.**

6. Nel gennaio 1984, Stallman diede inizio al suo progetto, denominato GNU (dall'acronimo Gnu's Not Unix). Il suo obiettivo dichiarato era quello di creare software libero: il progetto si fondava, infatti, sulla convinzione che la conoscenza alla base di un programma eseguibile, il c.d. codice sorgente, doveva essere gratuita, poiché solo la possibilità di osservare, studiare e manipolare il codice sorgente avrebbe permesso il progresso della scienza informatica. Una volta definiti i propri obiettivi, Stallman decise di rendere il sistema compatibile con Unix, in modo che fosse portabile e che gli utenti di Unix potessero facilmente passare ad esso. Stallman, tuttavia, non ambiva a creare un software tecnicamente migliore: anche se GNU non avesse avuto alcun vantaggio tecnico su Unix avrebbe avuto un vantaggio sociale, permettendo agli utenti di cooperare, sia un vantaggio etico, rispettando la loro libertà. Un anno dopo l'inizio del suo progetto, nel gennaio del 1985, Stallman rese pubblicamente disponibile GNU Emacs, il quale s'impose immediatamente all'attenzione del pubblico come un prodotto nuovo e assolutamente affascinante. Al di là della funzione di editor di testo, permetteva, infatti, di leggere e rispondere alle mail, leggere e scrivere a gruppi di discussione, aprire una shell, far girare il compilatore e fare il debug[7] dei programmi risultanti. Al di là delle indiscusse qualità tecniche, l'uscita di GNU Emacs rappresentò una tappa fondamentale nella storia del progetto GNU in quanto, per la prima volta, gli utenti furono in grado usare e giudicare l'efficienza del software creato. Quest'approccio, rilasciare software, incoraggiando il feedback e le modifiche degli utenti, sarebbe stato centrale per i rapidi progressi del software libero nei successivi 15 anni: la sua adozione in forma estrema per lo sviluppo di Linux è stata, infatti, uno dei fattori chiave della sua veloce crescita e della sua solidità. La seconda conseguenza di rendere disponibile Emacs fu che Stallman poté venderlo su cassetta, il mezzo standard di distribuzione prima dei dischetti; questa affiancava la distribuzione libera su Internet, giacché, a quel tempo, pochi avevano accesso a questo mezzo.

Con GNU Emacs ormai concluso, Stallman poté rivolgere la sua attenzione ad altri progetti, in particolare al problematico compilatore C[8]. Stallman chiamò il suo compilatore GCC (GNU C Compiler), il quale contribuì a fare parlare del progetto GNU poiché dimostrò di essere un software di prima qualità. Com'era successo con Emacs, l'ottimizzazione fu enormemente aiutata dai commenti che Stallman ricevette dagli utenti dopo il rilascio del programma.

Man mano che gli sviluppatori cominciavano a prendere confidenza con GNU Emacs, non solo arrivarono aiuti in denaro, ma anche maggiori richieste di copie su nastro. Ad occuparsi del lato commerciale del progetto GNU, Stallman chiamò qualche collega, formalizzando così, nell'ottobre del 1985, la nascita della Free Software Foundation. Ancora oggi, la Free Software Foundation è un'organizzazione no-profit per lo sviluppo del software libero, che contribuisce a fornire la necessaria professionalità al progetto GNU.

[7] Procedimento di correzione degli errori del programma

[8] Si tratta di uno degli elementi fondamentali di un sistema di tipo Unix. I programmi scritti in C sono file di testo contenenti le stringhe d'istruzioni che seguono le regole del linguaggio. Prima che il computer possa, tuttavia, obbedire alle istruzioni contenute nel codice sorgente, queste devono essere convertite in codice binario, la sequenza di 0 e 1 che un processore può capire. Questo è il lavoro di un compilatore C, una specie di mediatore tra il codice sorgente e il codice binario

Proprio grazie all' intervento della Fondazione è stato possibile realizzare alcuni importanti progetti, tra cui i più importanti sono stati, senza dubbio, una shell e una library[9] di C. La crescente statura di Stallman come programmatore era, tuttavia, controbilanciata dai problemi in cui si dibatteva come project manager. Alle porte degli anni novanta, infatti, mancava ancora il kernel[10].

7. I prodotti nati dal progetto GNU si rivelarono tutti d'altissima qualità tecnica: non deve quindi sorprendere che potessero essere assolutamente appetibili per il mercato, soprattutto in un periodo in cui l'offerta di prodotti per programmatori era scarsa. Ciò nonostante, prodotti così eccezionali come quelli GNU furono distribuiti sotto forma di software libero. Il rigore morale di Stallman gli aveva impedito di firmare accordi di non condivisione, e lo aveva spinto a creare una struttura software libera e autonoma. Per riuscire nel suo intento, si era, persino, licenziato dal laboratorio d'Intelligenza Artificiale: in questo modo negò al MIT qualsiasi possibilità di vantare qualche diritto sulla distribuzione del suo prodotto. Il progetto di Stallman non prevedeva, tuttavia, solo la creazione e la libera distribuzione di software, ma un piano di vero e proprio indottrinamento degli utilizzatori. Scritto nel 1985, il Manifesto GNU conserva, dopo quasi un ventennio, una grand'attualità. Può, a ragione, essere considerato un documento filosofico; contiene, infatti, non solo un'attenta analisi della natura del software, della programmazione, della grande tradizione accademica, ma lancia un messaggio di vitale importanza: a prescindere dalle conseguenze economiche, esistono imperativi etici e morali che impongono di condividere liberamente le informazioni che ci sono trasmesse.

Grazie a tale documento è possibile fare luce sui motivi che hanno condotto alla creazione del programma GNU. L' idea di base è che, chi scrive un programma, deve condividerlo con altre persone che lo apprezzano: si tratta di una regola di comportamento decisamente stridente con la realtà delle cose: di regola, infatti, le aziende produttrici di software adottano, nei confronti dell' utenza, il metodo divide et impera, in modo che ciascun utilizzatore sia spinto a non condividere nulla con gli altri. Con il suo manifesto, Stallman sostiene con vigore l'ideale della solidarietà tra utenti, facendosi portavoce di quella che, ai tempi, è stata l'etica hacker all'interno della comunità del MIT. Quella che nacque come una cultura underground, emerge ora, per il tramite di un unico individuo, in tutta la sua forza dirompente e la sua innovatività.

Dal punto di vista di Stallman, inoltre, tutti gli utenti possono beneficiare della libera distribuzione del software GNU, giacché chiunque è messo nella condizione di disporre dei sorgenti completi del sistema. Di conseguenza, un utente che intende apportare modifiche al programma, è sempre libero di farlo, o di assumere un programmatore disponibile a realizzarle al suo posto. In questo modo, gli utenti non dipendono più da un unico programmatore o da un'unica compagnia, non essendo più, questi ultimi, gli esclusivi detentori del diritto di eseguire modifiche.

Non bisogna inoltre dimenticare che, nella pratica, copiare una qualsiasi parte di un programma è un atto connaturato all'attività di programmazione: i programmatori non creano codice dal nulla, ma si rifanno, di regola, a qualcosa di già esistente. L'attività di prelevamento e modifica di codice implementatati da altri programmatori sono alla base, quindi, dello sviluppo e del progresso della tecnologia.

Oltre ad essere un brillante programmatore, Stallman è universalmente riconosciuto come il portavoce del movimento per il free software. Come lo stesso si è preoccupato di precisare, l'espressione free software è, tuttavia, male interpretata. Il termine inglese free ha un duplice significato: corrisponde sia alla parola "libero" che alla parola "gratuito"; nella concezione di Stallman, tuttavia, il termine non ha niente a che vedere col prezzo del software, concernendo solo la questione della libertà.

Un programma è, quindi, considerato software libero se:

1) l'utente ha la libertà di eseguire il programma per qualsiasi scopo

2) l'utente ha la possibilità di modificare il programma secondo i propri bisogni; affinché tale libertà sia effettiva è, tuttavia, necessario avere accesso al codice sorgente.

3) l'utente ha la libertà di distribuire copie del programma, gratuitamente o dietro compenso

4) l'utente ha la libertà di distribuire versioni modificate del programma, in modo tale che la comunità possa fruire dei miglioramenti apportati

Già al tempo della creazione del software GNU, il punto cruciale della questione era, però, quello di stabilire quando il software poteva considerarsi effettivamente libero. Nel momento in cui Stallman rilasciò al pubblico le componenti del suo sistema operativo GNU, una delle sue maggiori preoccupazioni fu, infatti,

[9] Parte di codice ausiliario che altri programmi possono usare. Separando in questo modo un'intera serie di funzioni comunemente impiegate, i programmi per gli utenti diventano più snelli

[10] Programma di controllo centrale di tutti i sistemi Unix, ha il compito di determinare quali periferiche e applicazioni devono avere accesso al microprocessore e quando.

quella di assicurarsi che i programmi restassero liberi e aperti, man mano che passavano da utente a utente. Si rese immediatamente conto, infatti, che se un programma poteva essere considerato software libero nel momento in cui usciva dalle mani del suo autore, ciò non significava necessariamente che sarebbe stato software libero per chiunque ne avesse avuto una copia. L' esempio emblematico, in questo caso, è rappresentato dall' X Windows System sviluppato al MIT e pubblicato come software libero con una licenza permissiva, era stato rapidamente adottato da diverse società informatiche, che avevano aggiunto X ai loro sistemi Unix proprietari, coprendolo, poi, con un accordo di non diffusione. La definizione delle modalità attraverso le quali GNU sarebbe stata resa disponibile, si rivelò, quindi, di primaria importanza; si voleva, in pratica, accordare a chiunque la possibilità di modificarlo e ridistribuirlo, senza, tuttavia, attribuire il potere di porre limitazioni alla sua ulteriore distribuzione. In altri termini, non sarebbe stato possibile apportare modifiche proprietarie, poiché si voleva che tutte le versioni di GNU rimanessero libere.

8. Stallman si rese subito conto che se il programma da lui creato fosse stato rilasciato sotto dominio pubblico, avrebbe permesso a molte aziende di inserirne il codice nei loro prodotti, al fine di trarne profitto. La risposta di Stallman a questo rischio è stata la GNU GPL; se l'idea essenziale era, infatti, quella di dare agli utenti la possibilità di operare liberamente con il proprio computer, i termini della distribuzione avrebbero, di conseguenza, dovuto evitare che il software GNU fosse trasformato in un prodotto proprietario: tale nuovo concetto di distribuzione è stato indicato con il termine permesso d'autore o copyleft. In pratica, Stallman ha capovolto le leggi del diritto d'autore, per ottenere lo scopo opposto: invece che un metodo per privatizzare il software, il copyleft diventava lo strumento per mantenerlo libero. Il nucleo centrale del concetto di copyleft deve essere ricercato nella possibilità di dare a chiunque il permesso di eseguire, copiare, modificare il programma e distribuirne versioni modificate, ma senza il permesso di aggiungere restrizioni. Solo in questo modo, le libertà essenziali che definiscono il free software sono garantite a chiunque ne abbia una copia. Tuttavia, affinché un permesso d'autore sia efficace, anche le versioni modificate devono essere libere: ciò fa in modo che, una volta pubblicato, ogni lavoro basato sul software GNU sia reso disponibile per la comunità. Un problema connesso al precedente è quello della combinazione di un programma libero con codice non libero. Dal momento che una tale combinazione finirebbe, dal punto di vista di Stallman, per essere non libera, tutto ciò che è aggiunto o combinato con un programma protetto da permesso d'autore deve essere tale che il programma risultante sia anch' esso libero e protetto da permesso d'autore. La specifica implementazione di permesso d'autore utilizzata per la maggior parte del software GNU è la GNU General Public License (GNU GPL). Tale licenza genera un'ulteriore fondamentale conseguenza: la libertà di modificare un programma implica, infatti, che il codice sorgente del software sotto copyleft sia sempre disponibile; è questa un requisito essenziale, se si vuole mettere altri soggetti nella condizione di apportare migliorie al programma. Tramite la GPL, Stallman non si è, tuttavia, limitato ad impedire la creazione di derivati proprietari, ma è riuscito ad esprimere l'etica hacker in un linguaggio informale, comprensibile tanto per i giuristi che per gli hacker. In definitiva il lavoro di Stallman si è rivelato rilevante, non solo perché ha anticipato molti dei processi che hanno prodotto il successo di quello che poi è diventato il sistema operativo GNU/Linux, ma perché fornisce un fondamento etico allo sviluppo dell'intera storia del software libero e dell'open source.

9. Quando Windows 3.1 fu lanciato nel giugno 1992, consolidò il dominio di Microsoft nell' ambiente desktop. Quando, poi, le aziende passarono da programmi per DOS a quelli funzionanti sotto Windows, si creò, una frattura nel mondo del software. Microsoft riuscì, allora, a sfruttare questo passaggio per conquistarsi la leadership nei settori del word processing e dei fogli elettronici lanciando nel giro di poco tempo programmi come Excel e Word, che tutti conosciamo. Tuttavia, Windows 3.1 non era il solo sistema operativo prossimo a essere completato nel 1991. Microsoft Windows New Technology, meglio conosciuto come Windows NT, era stato avviato già nel 1988 nel tentativo di creare un sistema operativo che sbaragliasse Unix, il quale, nel corso degli anni, si era affermato come il sistema operativo leader a livello aziendale. Nel corso degli anni ottanta erano stati molteplici i tentativi di commercializzare versioni proprietarie di Unix: alcune distribuzioni si basavano sulle versioni AT&T, altre sulle BSD; il problema, tuttavia, era che il prezzo delle licenze, tendeva a rimanere alto anche dopo l'eliminazione della dipendenza dalla licenza originaria AT&T e - cosa peggiore - non erano resi disponibili i codici sorgenti, rendendone quindi impossibile, o quantomeno molto complessa, la modificabilità e la redistribuibilità. Quest'ultimo aspetto derivava da quella che i sostenitori del software libero hanno sempre considerato una pecca della licenza BSD, vale a dire il fatto che rendesse possibile la redistribuibilità del software anche in forma di codice binario, comprensibile solo dalla macchina. A partire dalla fine degli anni 80, Unix cominciò, così, a perdere la sua originaria unità; ogni rivenditore ne offriva, infatti, una versione leggermente differente: questo significava, non solo che il software per le applicazioni doveva essere riscritto più volte, ma, cosa più importante, che gli utenti erano vincolati al software di un unico venditore. Tutto ciò rivela la miopia delle strategie di mercato, messe in atto dai produttori di versioni proprietarie di Unix, i quali hanno,

imprudentemente, trascurato la risorsa rappresentata dall'universo di utilizzatori/sviluppatori, di fatto, in rete già diversi anni prima dell'avvento di Internet. Queste lacune consentirono ad un sistema operativo obiettivamente inferiore a livello tecnologico, come Microsoft Windows, di affermarsi sul mercato. Avvenimenti cruciali si stavano, tuttavia, verificando al di fuori della sfera d'interesse più prossima a Microsoft. Se il periodo pionieristico della rete è individuabile, infatti, negli anni settanta e ottanta, la sua vera e propria nascita può essere fatta risalire al 1990, quando, in Europa, fu compiuto il passo progettuale decisivo che ne permise la diffusione a livello mondiale. Nel 1991 Tim Berners-Lee, un fisico britannico del CERN, (Centre Europèen pour la Recerche Nuclèaire), rilasciò, infatti, per uso pubblico un sistema d'ipertesto che aveva elaborato nei due anni precedenti. Il sistema, che chiamò World Wide Web, era in grado di organizzare il contenuto dei dati di Internet (all' epoca diffusa ancora su scala ridotta e nota soprattutto negli ambienti universitari) per informazione piuttosto che per posizione, fornendo agli utenti un sistema ipertestuale di interconnessione. Fu allora che, grazie al World Wide Web cominciarono ad essere costruiti anche i primi siti. Il programma, che sfruttato commercialmente avrebbe garantito ai suoi autori profitti incalcolabili, fu messo gratuitamente a disposizione degli utenti della rete. Sebbene le minacce che tale progetto rappresentava, fossero all'epoca, trascurabili, è possibile che Microsoft ne seguisse già l'evoluzione nel 1991. Internet era, infatti, abbastanza ben conosciuta e il Web di pubblico dominio. Ma è sicuramente impossibile che, nello stesso anno, i vertici del colosso di Redmond potessero anche solo sospettare che l'elemento chiave di una sfida altrettanto potente stesse prendendo forma nella camera da letto di uno studente del secondo anno d'informatica a Helsinki.

10. Quando Linus Torvalds, studente d'informatica all'Università di Helsinki, incontrò Unix, nel settembre del 1991, il grande progetto GNU di Stallman, iniziato sette anni prima, era quasi completo. Il solo elemento cruciale a non essere ancora disponibile, il kernel, era in via di sviluppo nella forma di GNU Hurd. Oltre a stimolare il suo interesse per Unix, gli studi in informatica fornirono a Linus un modo per entrarne in possesso. Uno dei libri di testo era, infatti, Operating System. Design and Implementation, cui era allegato un software dimostrativo: si chiamava Minix ed era stato scritto da Andrew Tanenbaum, docente di scienze informatiche alla Libera Università di Amsterdam; nelle intenzioni di Tenenbaum, il software aveva lo scopo precipuo di insegnare agli studenti come funzionasse un sistema operativo attraverso lo studio del suo codice sorgente. A ben vedere, Minix racchiudeva in sé tutte le caratteristiche che gli avrebbero garantito un successo immediato: era un sistema Unix con il codice sorgente per PC, richiedeva un hardware minimo e aveva un libro che ne descriveva il funzionamento. Fin dalla sua uscita, Tenenbaum ricevette sul newsgroup centinaia di richieste di migliorie, che, tuttavia, stentava a prendere in considerazione, spaventato dall'idea che il codice binario diventasse troppo impegnativo per l'hardware minimo che intendeva utilizzare. E' proprio questo l'elemento che determinò la stasi di Minix e che, al contrario, sancì il successivo successo di Linux

Il fatto di potersi basare su un predecessore importante come Minix, non fu l'unico fattore che propiziò la nascita e la popolarità di Linux. Al contrario, la grande diffusione di Linux deve essere attribuita alla contemporanea combinazione di una serie di circostanze assolutamente fortunate e fortuite. Di sicuro, l'avvento del processore Intel 386 e il progressivo abbassamento dei prezzi dei PC che lo utilizzavano, giocarono un ruolo fondamentale: parte del successo di Linux può, quindi, essere attribuito all'hardware, abbastanza potente da permettere di adottare caratteristiche integralmente di tipo Unix e, tuttavia, sufficientemente a buon mercato perché in molti potessero permetterselo. Linux era, alla sua nascita, strettamente legato all'architettura Intel, essendosi sviluppato su un tentativo di comprendere un chip di questa famiglia. Tuttavia, gli eventi successivi misero in luce come, Linux non fosse per niente condannato ad essere un sistema solo per Intel. Linux nacque dall'analisi combinata delle principali caratteristiche processore Intel 386 e della struttura di Minix, nel settembre del 1991. L'aspetto principale del programma risiedeva nell'essere divertente, la giustificazione ultima dell'hacker per ogni tipo di progetto. A livello personale, infatti, Linus non era altro che l'ultimo di una lunga fila di giovani interessati semplicemente a smontare e rimontare le cose per puro divertimento. Tuttavia, ciò che aveva attirato i primi hacker che si erano avvicinati a Linux era stata l'idea di poter combinare le potenzialità di un compilatore libero con la qualità di GCC con un sistema operativo libero Anche se animato dall'intenzione di costruire un vero e proprio sistema operativo, Linus era, infatti, al corrente della disponibilità di gran parte degli strumenti operativi necessari, grazie all'opera di GNU, BSD e altri sviluppatori di software libero. Nel gennaio del 1992, al momento della distribuzione della versione 0.12, la prima contenente una versione integrata del GCC, Linus abbandonò la licenza del primo kernel, per adottare la GPL: pur non animato da alcun intento politico, si trattava di un gesto di lealtà verso il movimento del software libero. GCC non è stato l'unico strumento ad essere adattato nel sistema Linux. Nel marzo del 1994, dopo tre anni di sviluppo, il gruppo di lavoro di Linux era pronto a distribuire il primo vero sistema, Linux 1.0, che comprendeva le versioni pienamente modificate di GCC e GDB, oltre ad un'ampia gamma di strumenti BSD. Fu allora che ci si accorse che Linux era, in effetti, in grado di colmare il buco lasciato nel progetto GNU da Hurd, il kernel a

lungo rinviato, realizzando così l'obiettivo finale di Stallman,un clone di Unix completo e libero.

11. La caratteristica più innovativa di Linux non è rappresentata tanto dall'alto contenuto tecnico del programma creato, quanto dall'affermazione di un modello di sviluppo vincente. Molto del successo di Linux può, infatti, essere attribuito direttamente all'atteggiamento che Linus ha adottato per condurre il progetto: il suo incoraggiare gli altri a sottoporgli le patch[11] e la sua prontezza ad accettarle possono essere considerati fattori determinanti della grande popolarità del suo lavoro A differenza di Windows, venduto dalla Microsoft come una "black box" che non può essere aperta, Linux si presenta completo del suo codice sorgente: chiunque è, quindi, in grado di curiosare all'interno del programma e correggere le cause dei problemi che incontra. La possibilità per ciascun utente di apportare delle modifiche e di partecipare, in questo modo, alla costruzione del sistema è poi amplificata, oggi, dalla grande diffusione della rete: grazie ad Internet, gli errori nel codice possono essere segnalati direttamente a Linus, il quale nello stesso modo può rispondere agli utenti. L'apertura verso i "cacciatori di bug" e un atteggiamento non protettivo nei confronti del codice, si sono, quindi, rivelati elementi cruciali nello sviluppo di un modello dimostratosi vincente. Accelerando il ciclo di revisione delle distribuzioni, Linus si è fatto, inoltre, promotore di un nuovo stile per il rilascio degli aggiornamenti che, oltre a contribuire al rapido miglioramento del programma, n'è diventato il marchio di garanzia. Il rilascio di nuove versioni pubbliche a distanza di pochi giorni ha rappresentato qualcosa senza precedenti nell'industria del software: prima dell'avvento di Linux, persino gli altri progetti di software libero, in grado di ignorare costrizioni d'ordine commerciale, erano soliti adottare un ritmo d'aggiornamento più tranquillo. Questa frenesia nella produzione di codice, è alimentata dai molti hacker che gli inviano consigli e codice: oltre a consentire agli sviluppatori di lavorare sul codice più recente, il rilascio di versioni costantemente aggiornate ha anche l'effetto di emancipare gli sviluppatori, che sentono d'essere parte del gruppo di sviluppo. Nessuno di queste distribuzioni a ripetizione sarebbe stato, tuttavia, possibile senza Internet.: la localizzazione di Linux in Finlandia, patria del colosso della telefonia mobile Nokia e pioniere della connettività alla rete, è, quindi, un ingrediente chiave del successo di Linux. Man mano che Linux è cresciuto in età e statura, altrettanto ha fatto il numero di persone che lo hanno provato e hanno trovato degli errori. Tanto migliore è diventato Linux, tante più persone hanno cominciato ad usarlo; e tante più persone hanno partecipato al debug, tanto più velocemente è migliorato: un circolo virtuoso che continua, ancora oggi, a guidare lo sviluppo di Linux ad un ritmo vertiginoso. Linux ha sfruttato quello che si sarebbe potuto chiamare il "metodo Minix", meglio di quanto facesse Minix stesso, perché Tenenbaum permise che il suo sistema operativo crescesse lentamente e solo sotto determinate direzioni. Linus, al contrario si è mostrato da subito disponibile a modificare il suo kernel e a farne la base di modifiche a più ampia portata. Il metodo di lavoro adottato da Linus ha permesso a Linux di rimanere stabile e crescere continuamente: le parti di codice che avrebbero dovuto aggiungere nuove importanti funzioni al sistema, non sono mai state semplicemente integrate al kernel; dimostrando notevole lungimiranza, Linus ha, invece, utilizzato le lezioni apprese dal nuovo codice per estendere il kernel esistente, in vista di possibili sviluppi futuri. Il ricorso a tale metodo ha fornito agli sviluppatori una base più solida per lavorare, e assicurato che la struttura del sistema potesse migliorare ulteriormente con l' aggiunta di nuove funzioni. In particolare, quest'approccio ha dato i suoi frutti al momento del porting di quella che si sarebbe dimostrata essere una delle aggiunte strategicamente più importanti alla gamma dei programmi di Linux: il sistema X Window[12]. Proprio l'arrivo di X può essere considerato una delle tre pietre miliari nella storia di Linux, insieme alla Virtual Memory (VM), aggiunta nel dicembre del 1991, e al networking. Può sembrare strano che un sistema nato e sviluppatosi attraverso Internet, per i primi 18 mesi di vita non sia stato in grado di accedere alla rete. Linux nasceva, infatti, nel momento topico dell'era Internet, quello del passaggio dagli ambienti accademici verso

[11] Parti di codice scritte per correggere gli errori presenti nel programma.

[12] All' inizio, Linux era controllato attraverso una semplice shell come la Bash di GNU. Questa funzionava in modo piuttosto simile a MS-DOS: l'utente era chiamato a digitare i comandi, i quali avrebbero stimolato determinate risposte del kernel. Questo tipo d'interazione macchina-utente era, da qualche tempo, una caratteristica standard per il mondo di Unix. Il mondo di Apple Macintosh o di Microsoft Windows, allora alla versione 3.1 si trovava, invece, in totale contrasto. Qui, il controllo era effettuato direttamente, cliccando su icone grafiche a video. Da molti anni Unix possedeva le fondamenta di un approccio grafico del genere; questo era chiamato X perché era il successore di un sistema a finestre chiamato W ed era un elemento standard sugli Unix commerciali. Diversamente dai sistemi Macintosh o Microsoft Windows, la sua principale funzione non era di rendere il controllo più semplice, ma di consentire la vista e il controllo di più programmi simultaneamente attraverso diverse finestre X aperte nello stesso momento. In un certo senso X era molto più potente di Apple Macintosh e Microsoft Windows: i contenuti di una finestra di X non si limitavano alla visualizzazione dei programmi che giravano sulla macchina stessa; X, infatti, era stato progettato come un sistema in rete di finestre che consentiva la visualizzazione e la gestione dell'output da programmi che giravano su altre macchine collegate tra loro in rete. Non deve quindi stupire che l'elaborazione del porting di X Window fosse stata intrapresa contemporaneamente alla progettazione della gestione della rete all':il sistema X Window e il networking di Linux erano, infatti, profondamente legati: proprio il desiderio di far funzionare X spinse gli sviluppatori a intraprendere il progetto del networking.

il successo di massa. Faceva quindi la sua apparizione giusto in tempo per cavalcare il successo che Internet avrebbe cominciato a riscuotere da lì a poco. Il collegamento alla rete, tuttavia, non ha costituito inizialmente una priorità per gli utenti di Linux: il suo sviluppo era, infatti, affidato a persone che frequentavano le università e che da lì avevano la possibilità di connettersi. Le macchine con Linux, al contrario, erano private e non erano collegate alla rete: non risentivano, pertanto, della mancanza del TCP/IP, una sorta di programma ponte tra il sistema operativo e la rete, le cui funzioni non erano ancora state inserite nel kernel. E' importante rilevare, tra l'altro, che Linux possiede una delle rare implementazioni di TCP/IP non derivate da quella di Berkeley. Nel periodo in cui fu realizzato il networking di Linux, infatti, l'azione legale dell'AT&T contro Berkeley era ancora in corso e la comunità di Linux, al fine evitare inconvenienti simili, decise di scrivere il codice TCP/IP partendo da zero.

Il movimento di Linux aveva e tuttora ha una gerarchia formale in base alla quale operazioni di programmazione importanti possono essere distribuite: si tratta di un'apparente debolezza, che si è rivelata, tuttavia, un punto di forza. Chiunque si dimostri interessato ad un programma particolare, è invitato a scriverlo o a migliorarlo. Il primo volontario che intraprese lo sviluppo del codice del networking del kernel fu Ross Biro; tuttavia, la pressione esercitata all'interno della comunità Linux per ottenere sviluppi nel campo del networking, aggiunte a un notevole carico di lavoro, lo convinsero ben presto ad abbandonare il progetto. Fu Fred Van Kempen che gli successe nell'accollarsi il compito di sviluppare il networking. Van Kempen affinò il codice di Ross, rendendolo abbastanza fruibile; la comunità degli utenti tuttavia reclamava, diventando, via via, sempre più impaziente; com'era già successo per Ross, la pressione si fece sempre più crescente. Fu allora che Linus prese una decisione insolita: approvò lo sviluppo parallelo del codice del networking da parte di un'altra equipe di sviluppatori, guidata da Alan Cox. Un'impostazione di questo tipo poteva apparire molto pericolosa poiché avrebbe potuto facilmente determinare una biforcazione del codice, vale a dire un'evoluzione in due direzioni separate. Tale sdoppiamento non avrebbe fatto altro che confondere gli utenti e spaccare il fronte degli sviluppatori in fazioni contrastanti, con il risultato che lo sviluppo del sistema avrebbe perso il suo slancio. Alla fine il rischio fu più che ricompensato e un networking sicuro e utilizzabile fu inserito com'elemento standard nel kernel; sebbene Linus non potesse impedire che il programma evolvesse in una certa direzione, egli aveva l'autorità, legittimata sia dal conoscere quel programma meglio di chiunque altro, che dalla sua comprovata abilità nella gestione dello sviluppo del kernel, di accettare una derivazione del codice nelle release ufficiali. Linus, in quella situazione, si comportò sostanzialmente da arbitro. Sostenne Cox, inserendo il codice da questi elaborato nella distribuzione standard, perché il suo metodo di lavoro coincideva con la filosofia che Linus aveva sposato si dai primi mesi di vita di Linux: "Prima fallo funzionare, poi perfezionalo". Questa mossa non avrebbe potuto non ripercuotersi sul lavoro di Van Kempen: gli sviluppi del suo team non avrebbero più potuto contare su un'ampia base d'utenti che, utilizzando il codice, lo avrebbero collaudato. La scelta di Linux legittimò Cox quale leader dello sviluppo del networking del kernel; la breve lotta che scaturì da questo sviluppo a due vie produsse, oltre alla definitiva consacrazione di Cox come vice di Linus, un importante quanto necessario perfezionamento del modello di sviluppo di Linux: l'episodio aveva, infatti, sollevato importanti questioni, non solo tecniche; ci si chiedeva, in sostanza, quale sarebbe stato il modo più efficiente di dirigere il team di sviluppo. Alcuni eminenti esponenti, tra cui Tenenbaum, erano dell'idea che una squadra di programmatori doveva essere organizzata come un'equipe di chirurghi, composta di un leader e dei suoi assistenti. Non così per Linus: la sua risposta al problema di come mantenere il controllo sembrava quella di non mantenerlo affatto. Egli conosceva il sistema meglio di qualunque altro: questa era, in definitiva, l'unica forma di controllo che Linus aveva effettivamente esercitato. Il problema del controllo è strettamente collegato al problema del copyright: Linus ha voluto che il proprio sistema fosse rilasciato sotto la GNU GPL, in quanto questa consente a chiunque di accedere al codice sorgente e di modificarlo, a patto che le modifiche siano rese pubbliche e messe a disposizione dell'utenza; questo significa che, se qualcuno decidesse di creare una versione alternativa di Linux, neanche Linus stesso potrebbe opporsi, purché ne venga fornito il codice sorgente: in questo modo, Linus sarebbe in grado, in qualunque momento, di trasferire le modifiche da lui approvate dal Linux alternativo a quello ufficiale.

Un'attenta analisi del fenomeno Linux, sembrerebbe suggerire, quindi, che la caratteristica fondamentale del sistema non sia tanto tecnica, quanto sociologica. Fino all'avvento di Linux, era, infatti, pensiero comune che qualsiasi programma complesso doveva essere sviluppato in modo attentamente coordinato, da un ristretto numero di persone ben collegate fra loro. Questo modo di operare era, ed è tuttora, tipico sia del software commerciale sia dei grandi progetti di software libero portati avanti dalla Free Software Foundation. Linux si è evoluto in modo completamente differente. Fin dalla nascita è stato, casualmente, preda di hacking da parte di un vasto numero di volontari collegati solo tramite Internet: la qualità è stata quindi mantenuta non da rigidi standard, ma dalla semplice strategia di proporre periodicamente delle idee e di ricevere opinioni in merito da centinaia d'utenti ogni giorno, creando una sorta di rapida selezione sulle modifiche introdotte dagli sviluppatori.

La grande crescita di Linux è coincisa con un altro fenomeno: la scoperta di Internet da parte del grande pubblico. I primi anni novanta hanno visto, infatti, l'inizio di una fiorente industria dell'Internet

Provider, che ha fornito connessioni al pubblico per pochi dollari al mese; dopo l'invenzione del World Wide Web, la già rapida crescita di Internet è accelerata ulteriormente, catalizzando l'interesse della comunità degli sviluppatori, la cui attività è stata rivolta, nella seconda metà degli anni novanta allo sviluppo di Linux e alla diffusione di massa di Internet.

12. Verso la fine del 1993, Linux era in grado di competere per stabilità e affidabilità con molti Unix commerciali, ospitando una grande quantità di software. Un effetto indiretto di questo sviluppo, fu quello spazzare via la maggior parte dei piccoli fornitori di Unix commerciali. Linux offriva standard tecnici elevati, tuttavia il modo in cui era distribuito era, nel 1993, ancora piuttosto inadeguato. Già a quel tempo c'era, però, già chi, intuendone le potenzialità, considerava il sistema Linux un metodo di sviluppo applicabile al settore della distribuzione: tale modello prevedeva che un gruppo di persone collegate tra loro tramite Internet, si occupasse, ciascuna di una serie di dettagli che, sommati, avrebbero costituito una versione per la distribuzione. Il primo ad intuire le potenzialità di questo modello fu il programmatore Unix Ian Murdock: estremamente colpito dalla funzionalità di Linux, era rimasto, tuttavia, deluso dal proliferare di distribuzioni malamente realizzate. La decisione che prese fu, allora, quella di raccogliere personalmente i migliori strumenti software in circolazione, al fine di dare vita ad una distribuzione autonoma, che prese il nome di Debian**.**

A tale progetto va, sicuramente, il merito di aver avvicinato Stallman alla comunità Linux: questi, decise, infatti, di finanziare il progetto tramite la Free Software Foundation. Tale sostegno economico trasformò Debian, da piccolo progetto interessante, ad un fenomeno imposto all'attenzione della comunità. Ben presto, tuttavia, Stallman cominciò a pretendere di esercitare il suo potere decisionale, spingendo affinché la distribuzione Debian fosse chiamata GNU/Linux' e rilevando come la comunità di Linux non sarebbe esistita se la Free Software Foundation non avesse distribuito gratuitamente il software GNU. In realtà, il progetto GNU non forniva un input diretto al kernel di Linux, limitandosi a creare gli strumenti di lavoro per i programmatori; inoltre, Debian non utilizzava solo programmi GNU. Se Murdock si dimostrò disponibile verso quelle che potevano essere considerate le richieste "politiche" di Stallman, non fece altrettanto con quelle prettamente tecniche. Ad ogni modo, nel 1996 abbandonò il progetto, passando le redini ad un altro programmatore Unix, Bruce Perens.

Oltre a Debian, diverse distribuzioni di Linux, sia commerciali sia non commerciali, sono state, poi, sviluppate. Una distribuzione Linux consiste, di regola, nel kernel Linux cui è aggiunto il software di supporto, la gran parte del quale è stato sviluppato dal progetto GNU; a differenza dei tradizionali sistemi operativi, con Linux è possibile scegliere tra diverse distribuzioni, ognuna delle quali fornisce, oltre al kernel, una collezione di applicazioni, utilità ed un software di installazione. Tra le tante distribuzioni disponibili, quelle che in qualche modo si distinguono sia per notorietà sia per diffusione, sono le americane Red Hat Linux, Turbo Linux, Slackware e Caldera OpenLinux, la tedesca SuSe Linux, la francese Mandrake Linux e Debian GNU/Linux. Ognuna di queste distribuzioni differisce dalle altre per varie caratteristiche come i metodi d'installazione, gli strumenti per l'amministrazione del sistema e altro. Tra le differenze più indicative ci sono i mercati di riferimento: in pratica alcune distribuzioni sono più esplicitamente orientate verso determinati tipi d'utente; accade così che l'utente meno esperto si indirizzeà verso Red Hat o SuSE, mentre l'utente di una power Workstation sceglierà Turbo Linux Workstation Edition.

13. Nel febbraio del 1996 si tenne, a Boston, la "Conferenza sul software liberamente distribuibile". Sponsorizzato dalla Free Software Foundation, l'evento prometteva d'imporsi come il primo ambito tecnico esclusivamente dedicato al software libero. Tra gli invitati, spiccava il nome di Eric Raymond; pur non essendo responsabile d'alcun progetto o azienda, Raymond si era costruito una solida reputazione, all'interno della comunità hacker, in qualità d'importante collaboratore di GNU Emacs, nonché di curatore del The New Hackers Dictionary, versione cartacea dello Jargon File, il lessico hacker in circolazione da un decennio. Poiché membro del comitato organizzativo della conferenza stessa, Raymond sostenne vivamente la proposta di mettere di fronte, per la prima volta, Richard Stallman, rappresentante del contingente "anziano" degli hacker legati all'ITS e a Unix, e Linus Torvald, portabandiera di una generazione più giovane ed energica di sviluppatori. Fu proprio nel corso di questo faccia a faccia che apparve evidente che la leadership della cultura hacker stava passando da Stallman a Linus e, di conseguenza, che una nuova generazione di programmatori stava nascendo.

La conferenza diede a Raymond lo spunto per scrivere "la Cattedrale e il Bazar": interessante teoria su Linux e sulla sua metodologia di sviluppo, lo scritto mette a confronto lo stile manageriale del progetto GNU con quello di Linux. Raymond è stato, tra l'altro, autore di numerosi articoli di taglio antropologico che illustrano il fenomeno del free software e la cultura che vi è cresciuta intorno: tali scritti che furono i primi del genere, portarono alla ribalta un fenomeno fino allora sconosciuto.

Raymond fu invitato ad esporre i suoi scritti al Linux Kongress, raduno d'utenti Linux svoltosi in

Germania nella primavera del 1997. Tra i presenti in sala, Tim O' Reilly, editore specializzato in libri d'informatica, rimase talmente colpito dall'intervento di Raymond da invitarlo alla prima Perl Conference, organizzata per il mese di agosto a San Jose, in California; anche se incentrato sul linguaggio di programmazione Perl, l'evento avrebbe toccato anche altre tematiche connesse con il software libero. Considerando il crescente interesse commerciale nei confronti di Linux e del server web Apache, la conferenza avrebbe pubblicizzato il ruolo del software libero nella costruzione dell'intera infrastruttura di Internet. Era chiaro che il software libero stesse diventando qualcosa di più che un fenomeno emergente. L'unico elemento ancora assente, in questo scenario, era l'autocoscienza comunitaria. Proprio per favorirne il nascere, O' Reilly guardava all'acuta analisi di Raymond come un ottimo elemento per stimolare tale consapevolezza. Linguaggi di programmazione come Perl, software per Internet come Apache, BIND e Sendmail, erano lì a dimostrare che lo sviluppo del software libero non si limitava al progetto GNU. La conferenza vide, tra i presenti, i dirigenti di una nota azienda produttrice di software, la Netscape. Impressionata dal discorso di Raymond, nel gennaio del 1998, l'azienda annunciò l'imminente pubblicazione del codice sorgente del proprio prodotto di punta, il browser Navigator, nella speranza di motivare gli hacker a collaborare alle future migliorie. Netscape chiese, quindi, a Raymond di collaborare alla stesura di una licenza per il loro free software. In quella circostanza, Raymond suggerì di adattare la nuova licenza alla Guida Debian, poiché solo in quel modo avrebbe potuto essere presa sul serio come free software.

Era stato Bruce Perens, nel luglio del 1997, a proporre il contratto sociale Debian e la Guida Debian del Free software, al fine di rendere evidente la linea di condotta scelta, nella distribuzione di GNU/Linux. A causa della molteplicità di licenze che comportavano la gratuità, e della scarsa chiarezza della politica intrapresa dai produttori con riguardo al free software, era, infatti, diventato un problema definire cosa realmente fosse da considerare software libero. Il contratto sociale era, quindi, finalizzato a documentare l'intenzione di Debian di costituire la propria distribuzione interamente con free software, mentre la Guida rendeva facilmente possibile classificare il software come libero o non libero, tramite il confronto tra la licenza software e la guida stessa.

Quando la dirigenza di Netscape definì il saggio "La cattedrale e il Bazaar" il principale responsabile della decisione di rendere liberi i sorgenti del browser Navigator, il suo autore assurse al livello di celebrità hacker. Raymond era deciso a sfruttare l'attenzione dei media, che l'evento aveva suscitato: il fine era quello di convincere altre aziende a seguire l' esempio di Netscape. Il punto critico della questione era, ormai da qualche tempo, rappresentato dal significato della parola "free". I meccanismi della lingua inglese avevano, infatti, creato un malinteso: il termine free significa gratuito, ma anche libero. Quando Stallman parlava di free software, si riferiva ad un software libero relativamente ai diritti, non al valore economico del prodotto. L'espressione free risultava, quindi, avere, soprattutto in ambito commerciale, una valenza intimidatoria: nonostante i tentativi di chiarire il vero significato del termine, il messaggio non riusciva a passare. La maggior parte dei dirigenti d'azienda, imbattendosi, per la prima volta, nel termine free software lo interpretava come sinonimo di "a costo zero". Raymond fu tra i primi a capire che, fino a quando questo malinteso cognitivo non fosse stato superato, il movimento del software libero avrebbe avuto vita difficile, nonostante l'uscita di Navigator. Raymond temeva, in particolare, che la mentalità conservatrice dell'ambiente degli affari fosse scoraggiata dal grado di libertà sostenuto da Stallman, che era al contrario popolarissimo fra i programmatori di mentalità più liberale. Era impressione di Raymond che ciò stesse impedendo lo sviluppo di Linux e del software open source nel mondo del business, laddove esso fioriva, invece, nell'ambiente della ricerca.

Nel febbraio del 1998, Raymond convocò una riunione presso gli uffici della VA Reseach: l'ordine del giorno era di trovare un termine in grado di sostituire free software. Fu proprio in quest'occasione che, per la prima volta, fu utilizzata l'espressione open source, la quale, a differenza del termine free software, dimostrava un atteggiamento più amichevole rispetto al mondo imprenditoriale. Il termine s' impose, all'attenzione generale, qualche mese più tardi, al Freeware summit; l'obiettivo della conferenza era quello di rendere comprensibile, da parte delle società di software, quella che, fino allora, era stata un'attività marginale e frammentaria. I leader presenti alla manifestazione concordavano sulla necessità di adottare un'impostazione più adatta al contesto commerciale, in modo da consentire una più vasta diffusione del software libero e dei suoi prodotti. Si trattava di una vera e propria operazione di riposizionamento della comunità, che fu resa possibile dall'adozione di quello che può essere considerato un vero e proprio brand, un nome semplice e orecchiabile che avrebbe avvicinato la comunità del software libero alla realtà imprenditoriale. Open source, appunto. Tutto quello che serviva a vendere l'idea di software era un volto simpatico e un messaggio intelligente. Invece di combattere il mercato a viso aperto, come aveva fatto Stallman, Raymonds, Torvalds e gli altri nuovi leader della comunità stavano adottando un atteggiamento più rilassato e conciliatore. Quello che fino a quel momento era stato considerato il leader della comunità del software libero e che era stato estromesso dalla manifestazione, Richard Stallman, non tardò a chiarire la sua posizione: benché riconoscesse l'utilità del nuovo termine nel descrivere i vantaggi tecnici del software libero, riteneva che il movimento open source avrebbe finito per allontanare l'attenzione dall'idea di libertà, concetto, in primo luogo, sociale ed etico. I due termini, sebbene descrivano circa la stessa categoria di

software, dicono, quindi, cose differenti sul software e sui valori. Per questo motivo, il progetto GNU continua a usare il termine free software per esprimere l'idea che la libertà è più importante della tecnologia

Oltre al nome, Raymond era intenzionato a fissare, una volta per tutte, cosa potesse essere considerato software open source; dal punto di vista di Raymond, la guida Debian era senza dubbio, il documento più adatto a definire l' open source: era tuttavia necessaria una denominazione più generale e la rimozione dei riferimenti specifici a Debian. Fu Perens che si occupò delle modifiche al documento: il risultato fu l'Open Source Definition. Lo stesso Perens, il quale aveva formato per Debian un ente chiamato Software in the Public Interest, si offrì di registrare un marchio per Open Source, in modo da poter associare il suo uso alla definizione. Poco dopo la registrazione del marchio, apparve, però, chiaro che Software in the Public Interest avrebbe potuto non essere la dimora migliore per il marchio Open Source e trasferì la proprietà del marchio a Raymond. Da allora Raymond e Perens hanno formato l'Open Source Iniziative, un'organizzazione esclusivamente destinata alla gestione della campagna Open Source e della sua certificazione di marchio. Il compito dell'organizzazione è, a tutt'oggi, quello di monitorare il corretto uso del termine open source e di fornire una definizione adeguata del concetto per tutte quelle aziende interessate a realizzare programmi propri. L'Open Source Initiative ha, quindi, scelto un approccio più pragmatico rispetto a quell'adottato dal movimento per il software libero. Entrambe mirano ad una più ampia diffusione delle licenze open source: a differenza del movimento per il software libero, tuttavia, l'Open Source Initiative guarda all'industria del software come ad un alleato, ai fini della diffusione del software open source. L'Open source Initiative, ha, infatti, intuito, che le licenze open source potrebbero essere concretamente utili all'interno del mondo degli affari: essendo il software open source revisionato e testato da un'intera comunità di sviluppatori, ecco che, facendo leva su un bacino di sviluppatori potenzialmente illimitato, è possibile scrivere e sviluppare programmi per elaboratore più velocemente e a un prezzo più basso.

14. L'Open Source Definition è una carta dei diritti dell'utente di computer. Definisce, in particolare, quali diritti una licenza software deve garantire per essere certificata come Open Source. La fissazione di criteri determinati ha un importante seguito economico: i produttori che non rendono open source i loro programmi trovano difficile competere con chi lo fa, giacché gli utenti imparano ad apprezzare quei diritti che avrebbero dovuto essere loro da sempre.

L'Open Source Definition non è intesa per essere un documento di valore legale. Affronta, tuttavia, una delle questioni più spinose relative alla realtà open source: la licenza. E' proprio questa, infatti, che determina il modo in cui la comunità svilupperà e distribuirà il software, e determinerà l'evoluzione del progetto riguardante il programma.

Il documento Open Source Definition formula i nove criteri che una licenza di distribuzione del software deve seguire, per essere definita open source. I primi tre racchiudono le caratteristiche fondamentali alla base della nuova metodologia di sviluppo del software. Essi, in particolare, riguardano:

1) Il diritto di fare copie del programma e venderle o cederle, senza dover pagare nessuno per questo privilegio.

2) Il diritto d'accesso al codice sorgente, condizione necessaria per apportare correzioni e modifiche: il codice sorgente costituisce, infatti, un preliminare necessario alla riparazione o alla modifica di un programma. Là dove il codice sorgente manchi, l'Open Source Definition prevede specificamente che "siano disponibili mezzi ben pubblicizzati per scaricarlo, senza costi aggiuntivi, da Internet".

3) Diritto di creare e distribuire prodotti derivanti dalle modifiche effettuate. La manutenzione dei programmi è attività connaturata a quella di programmazione: senza migliorie, riparazione degli errori e porting su nuovi sistemi, il software serve a poco; la modifica si rivela quindi indispensabile per la manutenzione.

Deve, inoltre, essere permessa la distribuzione di un'opera modificata sotto gli stessi termini di licenza dell'opera originale. Ciò non significa, tuttavia, che ciascun produttore di un'opera derivata sia costretto ad usare gli stessi termini della licenza, ma solo che possa farlo quando lo voglia. Sul punto, esistono rilevanti differenze fra licenze: mentre la BSD consente di mantenere private le modifiche, la GPL non lo ammette.

Quelli affermati dall'Open Source Definition sono diritti importanti, poiché consentono di garantire uno stesso livello di tutela a tutti gli individui che collaborano alla creazione di software. La ragione del successo di una strategia che può suonare alquanto comunista proprio nel momento in cui il fallimento del comunismo stesso è visibile ovunque, deve essere cercata, soprattutto, nel fatto che l'economia dell'informazione è sostanzialmente diversa da quella degli altri prodotti: i costi della copia di un'informazione come un programma software è, infatti, infinitesimo.

I restanti criteri elencati nell'Open Source Definition riguardano esigenze secondarie e sono finalizzati a chiudere le eventuali scappatoie che potrebbero crearsi. Essi riguardano:

4) L'integrità del codice sorgente dell'autore. Si tratta di una clausola che dà agli autori la possibilità d'imporre una separazione tra le modifiche e l'opera originale, senza naturalmente proibire le prime. L'obiettivo è quello di scongiurare l'eventualità che certe modifiche siano percepite come opera dell' autore originale, gettando discredito su questi.

5) Il divieto di discriminazione contro persone o gruppi.

6) Il divieto di limitare l'uso del programma in uno specifico settore o per uno scopo determinato.

7) La distribuzione della licenza. Tutti coloro ai quali il programma è redistribuito hanno gli stessi diritti, senza necessità d'esecuzione di una licenza aggiuntiva. La licenza, non richiedendo alcuna firma, è automatica.

8) La mancanza di una specifica correlazione tra licenza e prodotto. I diritti concernenti un programma non devono dipendere dall'essere il programma parte di una particolare distribuzione software. Ciò significa che un particolare prodotto, classificato come open, source non può essere considerato gratuito solo se lo si usi con una determinata marca di distribuzione. Il software deve rimanere gratuito anche nel caso in cui sia separato dalla distribuzione software originaria.

9) Il divieto di contaminare altro software per il tramite della licenza. Quest'ultima, infatti, non può imporre restrizioni ad altri prodotti software che siano distribuiti insieme a quello licenziato. Per esempio, la licenza non può pretendere che tutti gli altri programmi distribuiti sullo stesso media siano software open source. Il concetto di derivazione non deve, in questa sede, essere confuso con quello d'aggregazione. Si parla di derivazione quando un programma incorpora parti d'altri programmi: quest'aspetto è trattato alla sezione quattro dell'Open Source Definition. Si parla, invece, d'aggregazione quando più programmi vengono inclusi in uno stesso media, per esempio un CD-ROM: ed è proprio all'aggregazione di prodotti software che si riferisce la sezione nove.

15. L'Open Source Initiative si occupa dell'aggiornamento dell'elenco delle licenze che si conformano all'Open Source Definition, la quale, è bene ricordarlo, definisce quali sono le qualità essenziali del software open source. Il largo uso del termine open source ha creato una certa difficoltà nel tracciare i confini dell'ambito, all'interno del quale un determinato programma può essere definito open source: poiché, tuttavia, la comunità ha bisogno di una via sicura di conoscere se un prodotto software sia effettivamente Open Source, la Open Source Iniziative si è preocccupata di registrare un contrassegno di certificazione, OSI certified, a tale scopo; ha, inoltre, creato un simbolo di certificazione grafico, che può essere usato in alternativa. Il contrassegno OSI certified è, quindi, il mezzo che l'Open Source Initiative utilizza per certificare che la licenza sotto cui il software è distribuito si conformi all'Open Source Definition. Chiunque volesse valersi del contrassegno OSI certified sul proprio software, può farlo, distribuendo il software sotto una delle licenze approvate che si trovano sulla lista, e contrassegnando il software in modo appropriato.

Le licenze classiche come GPL, LGPL, BSD, e MIT sono le più comunemente utilizzate per il software open source, anche se molte altre licenze sono state sottoposte all'approvazione e alla revisione da parte dell'Open Source Initiative. Quest'ultima incoraggia gli sviluppatori ad utilizzare una licenza già approvata per la distribuzione del software, ovviamente se si è sicuri di aver compreso a pieno i termini della licenza. Non fornendo alcuna assistenza legale, l'Open Source Initiative incoraggia, inoltre, a chiedere una consulenza in materia.

16. Le licenze GNU GPL, BSD, X Consortium, MPL e la Licenza Artistica sono tutte da considerarsi conformi all'Open Source Definition. Prima di considerarle singolarmente, è necessaria, tuttavia, una precisazione: le licenze elencate devono essere tenute separate dal concetto di pubblico dominio. Un programma diventa di pubblico dominio nel momento in cui l'autore rinuncia a tutti i sui diritti di copyright. A quel punto, l'ipotetico utilizzatore, potendo considerare il software alla stregua di una proprietà personale, può utilizzarlo come meglio crede: potrebbe, addirittura, scegliere di rilicenziarlo, rimovendo, in tal modo, quella versione dal pubblico dominio, o togliendo il nome del suo autore e trattandolo come opera propria. Per rendere privato un programma di pubblico dominio è sufficiente dichiarare un copyright e applicare la propria licenza, oppure semplicemente dichiarare "Tutti i diritti riservati"; quando, al contrario, non si vuole che terzi operino delle modifiche che possano rimanere riservate, si può ricorrere ad una licenza di software libero, e in particolare alla GPL: in questo modo la versione da cui si è partiti rimarrà di pubblico dominio, ma la propria versione sarà sotto una licenza che dovrà essere osservata da chi la usa o ne derivi altre. Al

contrario di quanto accade per i programmi di dominio pubblico, quindi, i programmi free software sono molto protetti da diritti e sottoposti a licenza: solo si tratta di una licenza che concede all'utente diritti più ampi di quelli solitamente attribuiti attraverso tale istituto giuridico.

17. I programmi per elaboratore sono, solitamente, distribuiti nella forma di codice oggetto. Di conseguenza, nel momento in cui si decide di acquistare una copia di un qualsiasi programma proprietario, si riceve un CD-ROM, sul qual è stata impressa una lunga serie di 0 e 1, che complessivamente, forma il codice oggetto del programma. A questo punto l'utente tipo intende solo installare il software sul proprio computer e fare girare il programma, non avendo alcun interesse ad osservare, ed eventualmente modificare, il codice sorgente. Nel momento in cui distribuiscono il programma sotto forma di codice oggetto, le aziende produttrici di software non possono che condividere gli interessi dell'utente tradizionale: non solo vogliono, infatti, che il programma sia comodo da usare, ma vogliono anche, e soprattutto, proteggere il proprio codice sorgente dalla divulgazione; dal momento che il codice sorgente è comprensibile all'uomo, questo, se rivelato, potrebbe, infatti, essere esaminato da un qualsiasi programmatore, il quale scoprirebbe non solo come funziona il programma, ma anche il modo in cui si sono espresse le capacità, gli sforzi e la creatività del programmatore originale. Nel momento in cui il software è distribuito sotto forma di codice sorgente, qualunque programmatore sufficientemente capace è, quindi, messo nella condizione di prelevare e riutilizzare le parti più innovative di un programma scritto da altri. Dal punto di vista delle aziende produttrici di software si potrebbe sostenere che, in questo modo, si perde il vantaggio competitivo dato dal primo programma, poiché il secondo avrebbe le stesse caratteristiche innovative, ma sarebbe sviluppato ad un costo minore. A differenza delle licenze tradizionali, lo scopo precipuo delle licenze open source non è, tuttavia, quello di ottenere un vantaggio economico dalla concessione del programma in uso, ma quello di consentire una libera distribuzione tra il pubblico del codice sorgente dei programmi per elaboratore; di regola, infatti, un certo numero di sviluppatori lavora sul codice sorgente, creando migliorie e modifiche, che sono restituite alla comunità, per essere liberamente condivise. Mentre nel caso di software proprietario, i miglioramenti e le correzioni degli errori sono il frutto del lavoro dei dipendenti di una singola azienda, nell'ambito del software open source, è un'intera comunità di sviluppatori sparsi in tutto il mondo che apporta migliorie e scova errori; nel caso in cui , poi, il singolo utente non sia soddisfatto del lavoro svolto da altri, è libero di modificare il software a suo piacimento. Lo sviluppatore originale agisce, quindi, in questo contesto, come una sorta di arbitro, controllando come il nuovo codice viene inserito nel programma originale ed assicurandosi che qualsiasi modifica o correzione sia ben scritta, per essere testata all'interno del codice ufficiale.

Le licenze open software sono, inoltre, accomunate dal fatto di declinare qualunque garanzia. Come chiarisce il testo della GPL " L'intero rischio concernente la qualità e le prestazioni del programma è dell'acquirente…il quale si assume il costo di ogni manutenzione, riparazione o correzione necessaria, qualora il programma si riveli difettoso". Inoltre, l'autore del programma e i terzi che possono modificare e ridistribuire il programma nei termini consentiti dalla GPL " 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 derivanti dall'uso o dall'impossibilità di usare il programma: ciò comprende la perdita di dati, la corruzione dei dati, l'inabilità del programma a lavorare insieme ad altri programmi". Lo scopo precipuo di clausole simili è quello di proteggere il proprietario del software da qualunque responsabilità connessa al programma. Dal momento che il programma è ceduto a costo zero, si tratta una richiesta ragionevole: l'autore non riceve del programma una fonte d'entrata sufficiente per sostenere un'assicurazione sulle responsabilità ed eventuali spese legali. Nel caso in cui gli autori di software open source perdessero il diritto di rifiutare tutte le garanzie e si trovassero ad essere citati in tribunale in base alle prestazioni dei programmi che hanno scritto, smetterebbero di fornire software gratuito.

18. La GPL è un manifesto politico tanto quanto è una licenza software: la maggior parte del testo è, infatti, intesa a spiegare la motivazione ideologica dietro la licenza. Essendo, tuttavia, stata stilata con l'assistenza di giuristi, la GPL è assai meglio scritta della maggior parte delle altre licenze open source. Pur non richiamando alcuna delle clausole contenute nella sezione quattro relative alla possibilità, riconosciuta all'autore originale, di imporre una separazione tra le modifiche e l'opera originale, soddisfa l'Open Source Definition; La GPL, inoltre, non solo non permette di mantenere private le modifiche apportate, prevedendo che queste debbano essere distribuite sotto GPL, ma non ammette neanche l' incorporazione di un programma sotto GPL in un programma proprietario, laddove per proprietario s'intende ogni programma con una licenza che non garantisca tanti diritti quanti la GPL.

La LGPL è un derivato della GPL, escogitato per le librerie software. A differenza della GPL, un programma sotto LGPL può essere incorporato entro un programma proprietario.

19. La licenza X, la BSD e l'Apache devono essere tenute ben distinte dalla GPL e dalla LGPL: queste licenze consentono, infatti, di fare quasi tutto ciò che si vuole con il software licenziato sotto di esse; tale fondamentale differenza ha radici storiche: il software coperto dalle licenze X e BSD, era, infatti, originariamente sovvenzionato con sussidi del governo degli Stati Uniti; si riteneva, pertanto, che, siccome i cittadini avevano già pagato il software tramite il sistema fiscale, dovesse essere loro garantito il diritto di fare del software tutto ciò che volessero.

In particolare, a differenza della GPL, tali licenze consentono di mantenere private le modifiche licenziate. In altre parole si può ottenere il codice sorgente di un programma sotto licenza X, BSD o Apache, modificarlo e quindi venderne versioni binarie, senza distribuire il codice sorgente delle modifiche e senza applicarvi le licenze suddette. Il programma risultante rimane comunque Open Source, dal momento che la Open Source Definition non richiede che le modifiche debbano sempre ricadere sotto la licenza originale.

20. Originariamente sviluppata per il linguaggio di programmazione Perl, la Licenza Artistica è stata, successivamente, adoperata per altri prodotti software. Si tratta di una licenza formulata in modo piuttosto impreciso, poiché, pur imponendo determinati requisiti, fornisce poi delle scappatoie che rendono facile aggirarli. Per esempio, la Licenza Artistica richiede che le modifiche apportate al programma siano ridistribuibili gratuitamente, salvo poi fornire una scappatoia che permette di mantenere le stesse private, e perfino di porre sotto dominio pubblico parti del programma. Probabilmente è questa la ragione per cui quasi tutto il software sotto Licenza Artistica ha oggi una seconda licenza, offrendo la scelta fra la Licenza Artistica e la GPL

21. Nel gennaio del 1998, la Netscape Communications Corporation annunciò l'intenzione di rilasciare il codice sorgente del suo browser web Netscape Communicator: si trattò di un annuncio di importanza capitale, in quanto, per la prima volta, una grande azienda informatica adottava il modello di sviluppo, distribuito e globale, tipico del sistema Linux e di altri progetti open source. Una delle questioni più importanti da affrontare fu, come abbiamo visto, quella della licenza. Nel comunicato che annunciava il rilascio del codice sorgente del suo prodotto di maggiore successo, la Netscape si era preoccupata di precisare che sarebbe ricorsa ad una licenza che avrebbe permesso la modifica e la ridistribuzione del codice sorgente e garantito la piena disponibilità delle versioni modificate, sulla base della licenza GNU GPL. In pratica, Netscape si era dichiarata pronta ad un passo epocale senza sapere che forma avrebbe avuto uno degli elementi chiave dell'intero progetto: la licenza sotto la quale sarebbe stato diffuso il codice. I vertici della Netscape avevano ben compreso che uno sviluppo di tipo open source avrebbe funzionato solo se la licenza fosse riuscita ad incoraggiare contributi esterni: con il riferimento alla GPL, la società aveva dimostrato le sue buone intenzioni, tuttavia trovare un compromesso tra Netscape e la comunità hacker rappresentava, in ogni caso, una sfida. La verità era che la GPL non costituiva una scelta praticabile, giacché avrebbe significato che tutto il codice collegato sarebbe dovuto essere distribuito sotto la stessa licenza. Ciò non era, tuttavia, possibile, giacché porzioni del codice di Communicator erano utilizzate per altri programmi di Netscape, e altre parti erano date in licenza a terzi. Alla fine, questo fastidioso problema legale fu risolto attraverso la creazione di due diverse licenze: la Netscape Public License (NPL) e la Mozilla Public License (MPL).

La prima, che non è una licenza open source, è utilizzata per i programmi Netscape. Per molti aspetti funziona come la GPL, tuttavia conferisce alla società alcuni privilegi speciali; in primo luogo, il diritto di rilasciare codice a terzi, senza che si trasmettano automaticamente le sue proprietà GPL: Netscape ha, infatti, la possibilità di mantenere private le modifiche apportate, migliorarle e rifiutarsi di restituire il risultato. Inoltre la NPL consente a Netscape di inserire il codice sorgente originale di Communicator in altri prodotti che, per due anni, non possono essere soggetti alle condizioni previste per l'open source.

La MPL è identica alla NPL, eccetto che nei diritti speciali: si tratta di una licenza maggiormente diretta a promuovere la comunità. Entrambe consentono, in ogni caso, di mantenere private le modifiche apportate.

22. Supponiamo che un individuo, un gruppo di sviluppatori o ancora una società voglia realizzare un progetto open source. Il primo passo è, ovviamente, quello di scrivere il programma che viene, poi, rilasciato sotto una licenza. Proprio la scelta della licenza rappresenta una delle decisioni chiave dell'intero progetto: insieme alla qualità del codice e alla reputazione dello sviluppatore, è proprio questa che determinerà se il software avrà successo all'interno della comunità dei programmatori. Nella scelta della licenza da applicare al proprio prodotto software, è necessario tener conto di una prima regola fondamentale: non è, in linea di massima conveniente formulare una licenza nuova se è possibile fare ricorso a una già esistente; la

proliferazione di licenze diverse e incompatibili opera, infatti, a discapito del software open source, soprattutto quando l'eventuale incompatibilità fra licenze non permette di combinare parti di programmi provenienti da prodotti diversi.

Non c' è dubbio, inoltre, che se la licenza scelta è idonea a soddisfare gli interessi del singolo, ciò non significa che si ponga allo stesso modo nei confronti della comunità; la scelta della licenza interessa, quindi, inevitabilmente, diversi soggetti: innanzitutto la comunità degli sviluppatori, le cui possibilità di mettere mano al codice sono determinate proprio dalla scelta della licenza; anche gli utilizzatori finali, che potrebbero essere interessati a sfruttare il maggior numero di applicazioni possibili, sono influenzati dal tipo di licenza adottata,; la scelta della licenza finisce, inoltre per interessare anche gli altri progetti open source, che vanno a competere o a completare il progetto in esame. In questo senso, un programma rilasciato sotto GPL potrebbe rivelarsi inutile per un altro progetto open source licenziato sotto BSD, a causa dell'incompatibilità fra le due licenze; anche la possibilità di profitto di chi vende programmi o fornisce supporti è influenzata dalla scelta della licenza.

Nella selezione della licenza da utilizzare, lo sviluppatore è, poi, chiamato a valutare la gamma di benefici che il progetto potrebbe portare. Questi ricomprendono:

- La motivazione intrinseca fornita dalla sfida intellettuale
- La gratificazione personale, e gli incentivi concernenti possibili offerte di lavoro
- La necessità di risolvere un problema concreto.
- La possibilità di trarre benefici materiali, edificando, per esempio, un operazione commerciale intorno al progetto.

Lo sviluppatore dovrà, quindi, valutare in che modo questa gamma di motivazioni influenzerà tutta una serie di condizioni generali. Nella scelta della licenza è, infatti, necessario prestare attenzione ad una serie di fattori:

a) Il progetto open source potrebbe non avere successo, se isolatamente considerato: potrebbe, in altre parole, avere bisogno dell'appoggio di prodotti software complementari. Questi ultimi possono essere, a loro volta, open source, ma possono anche non esserlo. La licenza finisce, quindi, per influire sul modo in cui i diversi prodotti software si combinano; per esempio, non ammettendo la combinazione con programmi proprietari, la GPL scoraggia la potenziale utenza commerciale. La scelta della licenza dipende, inoltre, dall'individuazione del settore in cui s' intende essere attivi: nel caso si operi in un ambito che utilizzi licenze restrittive, intendendosi con questo termine le licenze che non consentono modifiche proprietarie al programma, lo sviluppatore sarà spinto a scegliere una licenza dello stesso genere, al fine di garantire la possibilità di combinare i diversi programmi. Un progetto che adotti una licenza restrittiva, potrebbe, invece, non avere successo in un settore dominato da progetti rilasciati sotto BSD, consentendo quest'ultima di rendere proprietarie le modifiche apportate al programma.

b) I programmi rilasciati sotto una licenza non restrittiva potrebbero diventare preda d'aziende commerciali, le quali potrebbero aggiungere parti di codice proprietario e chiudere poi l'intero lavoro. Sebbene il programma risultante possa essere superiore, il soggetto commerciale avrebbe in ogni caso inciso negativamente sulle dinamiche del progetto open source; la chiusura del codice potrebbe non essere socialmente pregiudizievole (costituendo per esempio il suo passo logico successivo o ravvivando interesse in una tecnologia di scarso richiamo), tuttavia l'atto di chiudere il codice priverebbe, in ogni caso, la comunità di tutta una serie di benefici che il progetto avrebbe potuto garantire (per esempio, gli utenti potrebbero dover pagare per il programma finale e non essere in grado di adattarlo alle proprie esigenze). A dire il vero, neanche le licenze restrittive possono considerarsi immuni dal problema appena descritto, giacché, dopotutto, le aziende commerciali potrebbero osservare il codice del prodotto per poi riscriverlo in forma proprietaria; a differenza della protezione garantita dal sistema dei brevetti, infatti, la proprietà intellettuale tutela esclusivamente l'espressione e non le idee ad essa sottostanti.

c) Non bisogna sottovalutare l'impatto dei brevetti sul software: l'instaurazione di cause legali per violazione di brevetto potrebbe ostacolare lo sviluppo del progetto e, in definitiva e scoraggiare sia gli sviluppatori sia i potenziali utenti.

In ultimo luogo, lo sviluppatore dovrà chiarire quali esigenze di tipo tecnico intenda realizzare:

a) Qualora desideri che chi ha apportato delle modifiche al proprio software, ne rimandi il codice sorgente, è chiamato ad applicare una licenza che lo prescriva come la GPL o la LGPL; in caso contrario, gli è possibile fare ricorso alla licenza X o Apache: a chi utilizzerà il programma sarà, in tal modo, consentito mantenere private le eventuali modifiche apportate.

b) Qualora intenda, invece, autorizzare altri soggetti a far confluire il proprio programma nel suo software proprietario, utilizzerà la LGPL che lo permette esplicitamente senza consentire al pubblico di rendere privato il codice; in alternativa, userà la Licenza X o Apache, che permettono che le modifiche siano mantenute private.

c) Se desideri consentire, a chi lo voglia, di comprare versioni non –open source del

proprio programma sotto licenza commerciale, doterà il software di doppia licenza: la GPL potrebbe, in tal caso, essere la licenza open source.

23. E' datata 1997 la disputa che, causata dall'avvento dei due sistemi software GNOME e KDE diretti entrambi a costruire un'interfaccia grafica per il desktop, ha minacciato di dividere la comunità open source, e di Linux, in particolar modo. Verso la fine degli anni novanta, Linux era ormai, per definizione, il sistema degli hacker; la quasi totale assenza di applicazioni rivolte a un'utenza non esperta si stava, tuttavia, rivelando preoccupante, in quanto metteva in dubbio la capacità della metodologia open source nello sviluppare quel tipo di programmi. Per questo motivo, quando Matthias Ettrich, studente d'informatica all'università di Tubinga, in Germania, annunciò l'idea di un desktop grafico completo e gratuito per il sistema Linux, la notizia fu accolta con grande entusiasmo. Il progetto KDE (K Desktop Environment) utilizzava Qt come toolkit. Qt, che sta per Quasar Toolkit, era prodotto dalla Trolltech, una piccola azienda norvegese fondata nel 1994 e con sede ad Oslo. Pur essendo un programma proprietario, Qt era offerto in versione gratuita per chi produceva free software con sistemi operativi che supportassero X Windows, come Linux, appunto; i motivi di tale politica erano sia pragmatici sia ideologici. Da una parte, infatti, vigeva la convinzione che distribuire una versione gratuita, avrebbe permesso agli utenti di cominciare ad utilizzare Qt e conoscerne i vantaggi. Dall'altra, gli stessi fondatori della società si erano mostrati intenzionati a dare il loro contributo alla comunità free software, avendo loro, per primi, apprezzato l'uso del software libero; pur essendo gratuito e disponibile nella forma di codice sorgente per tutti i programmatori free software, Qt non poteva, in ogni caso, essere considerato free software, secondo l'accezione data al termine da Richard Stallman e dalla Free Software Foundation. Il problema chiave era costituito dalla mancanza della licenza GPL; per quanto i problemi della licenza di Qt apparissero evidenti, la prospettiva di un desktop grafico per Linux era, tuttavia, così attraente che molti utenti furono disposti a chiudere un occhio sulla sua natura non-open source. Ciò non impedì che attorno al progetto KDE si concentrasse una delle più intense battaglie scoppiate all'interno della comunità open source: fu proprio in quest'occasione che si palesarono le due correnti di pensiero che fluivano all'interno del movimento. Da una parte si schierarono, infatti, i c.d. pragmatici, come Ettrich, convinti che ci fossero solo due tipi di software: quello buono e quello cattivo. Dal punto di vista di tale fazione, la questione della licenza passava inevitabilmente in secondo piano, rispetto a quella dell'efficienza del prodotto, dal momento che l'obiettivo fondamentale di qualsiasi programmatore èra quello di costruire software ben funzionante.

L'altra fazione, quella dei puristi, era guidata da Richard Stallman. Come i pragmatici, anche i puristi erano convinti che esistessero solo due tipi di software; a contrario dei pragmatici, tuttavia, dividevano il software tra libero e proprietario, ponendo l'accento sull'idea della libertà e relegando le questioni tecniche ad un ruolo secondario. Il fatto che Qt fosse privo dei diritti fondamentali garantiti dalla GPL (la possibilità di modificare il codice sorgente e di ridistribuire le modifiche), era fonte di preoccupazione per i sostenitori della fazione purista, poiché qualunque prodotto creato utilizzando Qt avrebbe avuto fondamenta non libere. La fazione purista cercò, quindi, di convincere Trolltech ad applicare i termini della licenza GPL a tutti quelli che intendevano sviluppare software libero, mantenendo una licenza commerciale per coloro che, al contrario, volevano creare prodotti destinati alla vendita; Trolltech vide, tuttavia, un rischio in quest'operazione: i termini della GPL avrebbero consentito ad un vasto team di sviluppatori di prelevare il codice sorgente di Qt e di svilupparlo più velocemente di Trolltech. Se era vero che Trolltech avrebbe, comunque, mantenuto il diritto d'integrare tali modifiche, il flusso di sviluppo alternativo avrebbe finito per portare a una biforcazione del progetto. Trolltech non adottò, quindi, la GPL; la comunità del free software decise, allora, che l'unica soluzione possibile era, a quel punto, quella di creare un nuovo ambiente desktop basato interamente su software libero. Nell'agosto del 1997 fu annunciato l'avvio del progetto GNOME (GNU Network Object Model Environment), che si propose, quindi, come il concorrente interamente free di KDE. Il gruppo di sviluppatori, guidati da Miguel De Icaza, scelse di utilizzare come toolkit Gtk+ che, sebbene meno stabile e maturo di Qt, era completamente libero.

A quel punto, sebbene non incline a soddisfare le richieste dei sostenitori di GNOME, Trolltech cominciò a adottare un atteggiamento più conciliante. In prima battuta, istituì, nell'aprile del 1998, la KDE Free Qt Foundation, cui affidò il compito di assicurare che una versione libera di Qt sarebbe sempre stata disponibile, anche nel caso di cessione della società. Quest'evoluzione servì a placare i timori di molti, ma ancora non affrontava il problema che maggiormente interessava l'ala purista del movimento, quello della licenza. Alla fine Trolltech decise di formulare, grazie anche alla preziosa consulenza di Eric Raymond e Bruce Perens, una nuova licenza per Qt, la Q Public License (QPL), rilasciata nel novembre di quello stesso anno.Grazie all'avvento dell'Open Source Definition, quindi, Trolltech non fu più costretta a scegliere tra l'adozione della GPL o il mantenimento della proprietà, ma fu in grado di adottare una licenza che soddisfasse i canoni richiesti per qualunque licenza open source, lasciandogli, al contempo, il controllo della tecnologia. Oggi sia KDE sia GNOME stanno portando avanti il loro processo di sviluppo: non esiste alcun'effettiva biforcazione tra i due progetti e nessuna spaccatura all'interno del movimento Linux, ma solo

differenze nell'impostazione e nei risultati.

24. I maggiori pericoli per il movimento open source sembrano provenire dall'interno della comunità stessa: è piuttosto diffusa, infatti, la tendenza a diluire la definizione d'open source, in modo da ricomprendere prodotti solo parzialmente gratuiti. E', in pratica, ciò che è avvenuto con la libreria Qt in KDE prima che TrollTech rilasciasse il proprio prodotto sotto una licenza open source. Bisogna, inoltre, considerare l'ipotesi che le aziende produttrici di software potrebbero decidere di danneggiare il movimento open source, rilasciando software gratuito quel tanto che basta per attrarre utenti, ma senza garantire le piene libertà dell'open source. E' probabile che, al fine di difendere i propri mercati, le grandi software house faranno ricorso a due principali strategie: copyright e brevetti. Faranno, inoltre, ricerca in nuove direzioni dell'informatica e cercheranno di brevettare tutto quanto potrà prima che si possano sfruttare quelle tecniche nel free software; chiuderanno, quindi, fuori i concorrenti open source con le concessioni sui diritti di brevetto.

I pericoli provengono, tuttavia, anche da altre direzioni: si supponga che qualcuno fornisca del software che contenga un espediente diretto a sconfiggere la protezione in un sistema Linux; si supponga, poi, che la medesima persona resti in attesa che il prodotto venga largamente distribuito, per poi pubblicizzarne la vulnerabilità agli attacchi alla sicurezza. Un evento simile potrebbe ridurre la fiducia generale nel software open source. Gli utenti potrebbero, infatti, convincersi che un sistema libero possa essere più vulnerabile agli attacchi alla sicurezza, rispetto ad un sistema chiuso. Si potrebbe obiettare che il software proprietario ha la sua parte d'errori di sicurezza e che il modello a codice sorgente aperto dall'open source rende più facile scoprire questi errori. Qualunque bug che comparisse in Linux sarebbe riparato poco dopo essere stato scoperto, laddove un omologo, in un sistema proprietario, o non sarebbe mai scoperto o dovrebbe aspettare a lungo il rimedio. Tuttavia, il modello di sviluppo open source ha bisogno di rinforzare ancora la propria difesa contro gli attacchi alla sicurezza del sistema. Identificare chi contribuisce alla creazione del software e alle modifiche potrebbe rappresentare la soluzione, poiché permetterebbe di valersi del diritto penale contro attentati alla sicurezza.

CAPITOLO SECONDO
PROBLEMATICHE GIURIDICHE

Sommario: 1. La situazione attuale. – 2. Il software come opera dell'ingegno. – 3. I soggetti del diritto. – 4. L'oggetto del diritto. - 5. L'open source e la tutela apprestata dalla Legge sul diritto d'autore. - 6. Diritti d'utilizzazione economica: a) diritto di riproduzione. - 7. b) Diritto di distribuzione. - 8. c) Diritto di diffusione. - 9. d) Diritto d'elaborazione. - 10. In particolare: la GPL. - 11. La LGPL. - 12. La Licenza artistica. – 13. La BSD e l'Apache. - 14. Le licenze MIT e X Consortium. - 15. La QPL. – 16. Le utilizzazioni libere. - 17 I contratti informatici. -18. La licenza d'uso come contratto informatico. - 19. La licenza d'uso nel contesto open source. - 20. Qualificazione giuridica della licenza d'uso. - 21. La licenza open source come contratto a titolo gratuito. - 22. La mancanza di garanzia e la responsabilità contrattuale. - 23. Il profilo della responsabilità extracontrattuale. - 24. Il programma open source può essere considerato un'opera collettiva? – 25. Il problema della violazione dei termini della licenza. – 26. Considerazioni conclusive sul metodo di sviluppo open source.

1. Il software è un prodotto costituito da un insieme coordinato e strutturato d'istruzioni digitali. Ha delle caratteristiche economiche che lo accomunano alla risorsa informazione: è, quindi, un bene immateriale che per essere utilizzato, prodotto e accumulato deve essere incorporato in un supporto; se è vero che tra supporto e prodotto c'è una stretta integrazione, questi vanno, tuttavia, tenuti logicamente distinti per comprendere i problemi che sorgono per la sua tutela giuridica.

Il software è opera dell'ingegno. E' una questione di politica legislativa stabilire fino a che punto tale opera vada tutelata in quanto tale, cioè se debba essere assicurata, o meno, l' esclusività dell' utilizzazione, se debba essere, o meno, brevettabile, se debba essere inquadrata esclusivamente come opera protetta dal diritto d' autore o altro. Non c' è dubbio che tutti gli ordinamenti sono chiamati a rispondere a tale delicato problema di fondo, destinato ad incidere sulla realtà industriale.La tutela giuridica del software incontra, in ogni caso, due problemi principali: uno legato alla riproducibilità e trasmissibilità a costo prossimo allo zero, in un contesto globalmente interconnesso; l'altro alla disomogeneità delle normative nazionali ed internazionali di tutela. Problema molto sentito è quello dei brevetti sul software e su concetti quali teorie matematiche ed algoritmi, consentiti dal sistema giuridico statunitense. Attualmente in Europa è usata solo la pratica del copyright, ma il dibattito in Commissione Europea è aperto: diventa così difficile, se non impossibile, controllare efficacemente il rispetto dei diritti di proprietà e dei contratti; una simile situazione non può che dare luogo ad una persistente situazione d'incertezza da un punto di vista giuridico. In generale queste difficoltà nascono dal fatto che, negli ultimi anni, le istituzioni economiche si stanno evolvendo ad una velocità decuplicata rispetto alle epoche precedenti, in un contesto globalizzato e intorno alla risorsa informazione, che è un bene immateriale. Le risposte politiche e giuridiche, poste in essere per far fronte al cambiamento, sembrano non comprendere né la natura del cambiamento né la natura della risorsa intorno alla quale quest'ultimo si sta strutturando e i rimedi disposti risultano inefficienti se non peggiorativi: la legge italiana sul copyright n'è, appunto, un esempio.

2. Il software è protetto, all'interno del nostro ordinamento, quale opera dell'ingegno, dal diritto d'autore. Il principio della protezione del software quale opera letteraria è stato sancito dal d.lgs. n. 518 del 29 dicembre 1992, il quale, recependo la direttiva europea 91/250, ha inserito il testo delle norme comunitarie sulla tutela del software all'interno dello schema della l. n. 633 del 22 aprile 1941 (la c.d. Legge sul Diritto d'Autore). Tale integrazione ha fatto sì che al software siano, oggi, applicati tutta una serie principi generali, originariamente previsti per categorie d'opere di natura completamente differente: ciò ha creato problemi interpretativi che si sono spesso riflessi nella redazione di contratti ad oggetto informatico.

3. L' acquisto del diritto d'autore sul programma opera, senza necessità di alcuna formalità preliminare, a favore del creatore effettivo, il quale diventa, in questo modo, soggetto legittimato alla negoziazione di contratti concernenti i diritti d'autore sul software.

Si ritiene, tuttavia, consigliabile, in sede di formazione del contratto, verificare la legittimazione a contrarre di chi assume d'essere cessionario del diritto d'autore, poiché, nella pratica, accade spesso che l'autore conceda in uso il programma e/o ceda uno o più diritti di sfruttamento economico a terzi (nella maggior parte dei casi si tratta di soggetti collettivi). Tanto più che, nel caso in cui il soggetto che concede o

trasferisce i diritti sull'opera non sia l'effettivo creatore del programma, il contratto non produrrà i suoi effetti e l'acquirente in buona fede potrà solo agire nei confronti del cedente per il risarcimento dei danni.

E' necessario, in secondo luogo accertare che non vi siano altri soggetti contitolari. Se il programma è creato con l'apporto inscindibile di più soggetti, ci si trova, infatti, davanti, salvo patto contrario, ad un regime di comunione tra coautori a parità di quote (art. 10 l.a.). Si applicano, in questo caso, le norme che regolano l'istituto della comunione: occorre, di conseguenza, il consenso di tutti i contitolari per il trasferimento di uno o più diritti di sfruttamento economico del programma (art. 1108 c.c.), laddove per la concessione dei diritti d'uso si ritiene debbano applicarsi le maggioranze richieste dalla legge, salvo patto contrario. La difesa del diritto morale d'autore può, al contrario, essere sempre esercitata individualmente da ciascun autore.

Nel caso in cui il contributo creativo di più persone sia scindibile e distinguibile, si ricade nella fattispecie dell'opera collettiva, in virtù della quale è considerato autore del programma, nella sua interezza, chi ha diretto e organizzato la propria creazione, fermo restando in capo a coloro che hanno realizzato le singole parti del programma la titolarità del diritto d'autore sulle stesse e la facoltà di utilizzare separatamente i rispettivi contributi. La questione se il software open source rientri o no nella categoria delle opere collettive sarà trattata in seguito.

Nel caso del software creato dal lavoratore dipendente di una software house, il problema si pone in modo diverso. L'art. 12-bis l.a. stabilisce, con specifico riferimento al software, che nel caso di creazione del programma da parte del lavoratore dipendente nell'esecuzione delle proprie mansioni o su istruzioni impartite dal suo datore di lavoro, sia quest' ultimo il titolare dei diritti esclusivi d'utilizzazione economica dell'opera, laddove il lavoratore dipendente resta, in ogni caso, titolare del diritto morale d'autore. La legge fissa, quindi, una presunzione di cessione automatica dei diritti d'utilizzazione economica, a favore del datore di lavoro, facendo tuttavia salva la possibilità di patto contrario.

4. A norma dell'art. 2 n. 8 l.a., oggetto della tutela apprestata dal diritto d'autore sono "i programmi per elaboratore, in qualunque forma espressi purché originali, quale risultato di creazione intellettuale dell'autore". La protezione accordata dal diritto d'autore è, in particolare, limitata alla sola forma espressiva dell'opera, in quanto compiuta e originale, non estendendosi alla tutela del suo contenuto; invero, nel caso del programma per elaboratore, è spesso difficile distinguere contenuto e forma espressiva: è probabilmente questo il motivo per cui il legislatore del 1992 ha ricompreso nell' ambito di tutela anche "il materiale preparatorio per la progettazione del programma", escludendo, invece, espressamente " le idee e principi che stanno alla base di qualsiasi elemento del programma, compresi quelli alla base delle sue interfacce".

La difficoltà di circoscrivere la tutela del software esclusivamente al carattere creativo della sua forma espressiva, ha generalmente spinto la giurisprudenza[13] a valutare con minore rigore il carattere dell'originalità, necessario ai fini dell'applicabilità del diritto d'autore. Tale orientamento ha finito con l'aumentare la possibilità d'effettiva confondibilità tra programmi e il rischio di contraffazione. Con riguardo alla tutela del software, infatti, il vero problema è stato, ed è tuttora, quello di applicare i principi propri delle opere dell'ingegno in tema di creatività, ai programmi per elaboratore. Il legislatore italiano, infatti, nel ricomprendere il programma per elaboratore tra le opere dell'ingegno, ha previsto che a questi fosse accordata la protezione di cui alla legge n. 633/1941 "purché originali quale risultato di creazione intellettuale dell'autore" (art. 2 n. 8). Con riferimento al requisito dell'originalità, giurisprudenza ritiene che, al fine di accedere alla tutela giuridica, sia sufficiente riscontrare nell'opera un quid novi, unitamente a un apporto personale dell'autore: in particolare si è affermato che è originale tutto ciò che non costituisce pedissequa imitazione dell'originale[14]. Tuttavia, chi sostiene la tesi secondo cui siano sufficienti requisiti minimi, in termini di originalità dell'opera, al fine di accordare protezione giuridica, non si accorge che ciò conduce a conseguenze tutt'altro che desiderabili: infatti, allo stesso modo in cui è tutelato un programma non identico a quelli precedenti, così il diritto su questo non può essere fatto valere se non contro di coloro che questo programma hanno copiato integralmente. Sussiste, di conseguenza, una necessaria specularità tra il rigore con cui è valutata la creatività e l'efficacia dell'esclusiva: più alta la soglia d'accesso alla tutela, più ampio l' ambito della privativa idonea, in tal caso, a coprire qualunque plagio sostanziale del programma. Quando invece si afferma che è originale quel programma che non costituisce pedissequa imitazione di un altro, si deve anche concludere che è lecita qualunque copia non pedissequa.

C' è anche un altro aspetto d'importanza centrale: l'avvento dei sistemi digitali ha condotto alla perdita di materialità dell'opera. Proprio tale fenomeno di smaterializzazione minerebbe, in ultima analisi, alla stessa esistenza del diritto d'autore. La tutela garantita dal diritto d'autore è, infatti, posta a presidio, come già detto, della forma espressiva: l'opera dell'ingegno trova protezione in quanto l'idea si esprima o si estrinsechi. Non bisogna, tuttavia, cadere nell'errore di confondere la forma espressiva dell'opera, intesa

[13] Vedi Cass. 13 dicembre 1999 n. 13937, in *AIDA* 2000, 659
[14] Vedi Pretura di Roma, 30 giugno 1988

come sua percettibilità, con la materialità del supporto in cui la creazione è, eventualmente, incorporata. La perdita di materialità tangibile del supporto, porterebbe a prima vista, a concludere che l'idea difetti d'estrinsecazione, che è poi la condizione che consente l'applicazione del diritto d'autore. In realtà, l'avvento del digitale non va a minare al fondamento del diritto d'autore, giacché "manifestazione" è concetto diverso e distinto da quello di "materializzazione": mentre il primo conduce all'immediata percettibilità dell'idea originale ed è requisito necessario per la tutela apprestata dal diritto d'autore, il secondo si limita a consentire la conoscibilità del fatto della creazione. Non è, quindi, l'evanescenza del supporto, quale conseguenza dell'evoluzione tecnologica, a minare all'esistenza del diritto d'autore. La smaterializzazione del supporto non impedisce, infatti, la nascita del diritto. La creazione viene espressa, in codice binario, attraverso una sequenza di 0 e 1, che non è altro che una delle infinite forme di espressione dell'opera. Non è il diritto, ma l'opera stessa ad essere vulnerabile: in forma digitale,essa è facilmente manipolabile. Il modello open source, invece di prevenire e combattere le inevitabili conseguenze dell'avvento dell'era digitale, ne sfrutta le caratteristiche, consentendo sia il libero e gratuito accesso ad ogni opera che la riproducibilità dello stesso.

5. E' necessario, a questo punto, guardare un po' più specificamente all'oggetto della mia trattazione: l'open source. Tale fenomeno, avente ad oggetto esclusivamente il software quale opera dell'ingegno, rappresenta, una volta calato nell'ordine giuridico, una fonte piuttosto atipica rispetto, sia alle legislazioni nazionali, sia a quelle internazionali, sia, infine, al diritto convenzionale. Rientra, infatti, nel regime dell'autoregolamentazione e comporta soluzioni negoziali non riconducibili alle fattispecie tipicamente regolate dalle legislazioni in materia di diritto d'autore. In tale contesto, in particolare, l'autore rinuncia alla titolarità delle situazioni giuridiche attive di natura patrimoniale.

Dal punto di vista del contenuto, il diritto d'autore raggruppa, tanto i diritti d'utilizzazione economica dell'opera, quanto i diritti sull'opera a difesa della personalità dell'autore. Da una parte, quindi all'autore è riconosciuto il diritto esclusivo di utilizzare economicamente l'opera in ogni forma e modo, laddove per "utilizzazione economica" non deve intendersi la mera utilizzazione a scopo di lucro, ma ogni utilizzazione che possa avere riflessi sulle probabilità di guadagno di cui l'opera è suscettibile, in quanto bene idoneo allo scambio. In questa prospettiva, all'autore sono riconosciuti una serie di diritti esclusivi di carattere patrimoniale (art. 13 e segg.: diritto di riproduzione, di diffusione, di distribuzione, d'elaborazione) che rappresentano una serie di prerogative indipendenti l'una dall'altra, e che hanno per oggetto l'opera nel suo complesso e in ciascuna delle sue parti. Dall'altra, all'autore spetta la titolarità di un diritto morale d'autore che prescinde dalle vicende connesse al suo sfruttamento economico, permanendo comunque in capo all' autore per il fatto stesso della creazione. Tale diritto comprende, oltre al diritto alla paternità dell'opera, il diritto all'integrità dell'opera stessa, il diritto a pubblicare l'opera ovvero a mantenerla inedita (art. 20 e segg. l.a.), e il c.d. diritto di pentimento, che si concreta nell'esercizio del diritto di ritiro dell'opera dal commercio. Quale diritto di carattere personale, è intrasmissibile, imprescrittibile, irrinunciabile, come tale non può formare oggetto di un contratto. E', infatti, assolutamente pacifico, in giurisprudenza[15], il principio secondo cui il diritto morale d'autore è, ai sensi dell'art. 21 l.a., inalienabile e, di conseguenza, insuscettibile di rinunzia, con conseguente radicale nullità di qualsiasi eventuale patto abdicativo Sono invece liberamente acquisibili, alienabili e trasmissibili "in tutti i modi e forme consentiti dalla legge" i diritti di utilizzazione economica, che quindi possono formare oggetto di pattuizioni contrattuali; l'elenco dei diritti esclusivi è, peraltro, meramente esemplificativa, riconoscendo la Legge sul Diritto d'Autore la possibilità per l'autore di sfruttare economicamente l'opera " in ogni forma e modo" (art. 12 l.a.): l'unica condizione richiesta, in proposito, è la prova scritta del trasferimento (art. 110 l.a.). All'interno della realtà open source, quindi, l'assenso dell'autore non può che limitarsi al giuridicamente rinunciabile, vale a dire ai diritti d'utilizzazione economica dell'opera. Come già precisato, il sistema delineato dal codice civile (art. 2577) e dalla legge speciale sul diritto d'autore (art. 12-24) raggruppano sotto il contenuto del diritto d'autore, tanto i diritti d'utilizzazione economica dell'opera, quanto i diritti sull'opera, a difesa della personalità dell'autore. Il diritto morale d'autore comprende, in particolare, il diritto alla paternità dell'opera. Al riguardo, l'art. 2577 c.c. comma 2 stabilisce che, anche dopo la cessione dei diritti d'utilizzazione economica, l'autore può rivendicare la paternità dell'opera e opporsi a qualsiasi deformazione, mutilazione o altra modificazione dell'opera stessa, che possa essere di pregiudizio al suo onore o alla sua reputazione. L'art. 6-bis della Convenzione Internazionale di Berna sembra andare addirittura oltre, riconoscendo all'autore dell'opera il diritto di rivendicarne la paternità e di opporsi ad ogni deformazione, mutilazione od altra modificazione, come anche ad ogni altro atto a danno dell'opera stessa, che rechi pregiudizio al suo onore od alla sua reputazione. E' proprio in questo punto che la licenza open source va a confliggere con il sistema tradizionale tracciato dal codice civile, in unione con la Legge sul Diritto d'Autore. La licenza, non dimentichiamo, è un contratto con cui l'autore del programma per elaboratore, volontariamente, rinuncia

[15] Pretura di Milano, ord. 10 novembre 1992, in *AIDA* 1993. Vedi anche Pretura di Milano, ord. 25 ottobre 1991, in *AIDA* 1992, *67/3*

all'esercizio esclusivo dei diritti d'utilizzazione economica riconosciutigli dalla legge. Ciò significa che, per il tramite della licenza, l'autore concede al licenziatario di copiare, diffondere, distribuire, elaborare e, in particolare, modificare la sua opera. E' sufficiente, tuttavia, guardare al disposto del comma secondo dell'art. 2577 c.c. per comprendere come non ogni modifica sia possibile. Il giuridicamente rinunciabile non può che limitarsi ai diritti d'utilizzazione dell'opera, non potendo estendersi al diritto morale d'autore che è, per definizione, intrasmissibile e irrinunciabile. Sarà, pertanto, necessario accertare il tipo d'adattamento apportato dal licenziatario, giacché saranno lecite solo quelle modifiche che non andranno a ledere all'onore e alla reputazione dell'autore.

6. Nella rinuncia volontaria al diritto d'utilizzazione dell'opera sono implicite diverse rinunce, che riguardano quei diritti che la legge accorda, in via esclusiva all'autore dell'opera: in primo luogo il diritto di riproduzione dell'opera (art. 13 l.a.), diritto che ha per oggetto la moltiplicazione in copie con qualsiasi mezzo (nel caso del software, il supporto comunemente utilizzato è il cd-rom). La stessa Convenzione di Berna prevede all'art. 9 che gli autori delle opere protette hanno il diritto esclusivo di autorizzare la riproduzione delle loro opere in qualsiasi maniera e forma, riservando, tuttavia, alle legislazioni dei Paesi dell'Unione la facoltà di permettere la riproduzione delle predette opere in taluni casi speciali, purché una tale riproduzione non rechi danno allo sfruttamento normale dell'opera e non causi un pregiudizio ingiustificato ai legittimi interessi dell'autore.

Infine, il d. lgs. 518/ 1992, attuativo della direttiva 91/250/CEE relativa alla tutela giuridica del software, enuncia le attività oggetto dell'esclusiva dell'autore del programma per elaboratore: il legislatore comunitario, prima, e quello nazionale, poi, si sono, infatti, trovati costretti a "riscrivere" quelle che sono le prerogative del titolare del diritto d'utilizzazione economica dell'opera, poiché la semplice trasposizione dei tradizionali principi del diritto d'autore non avrebbe consentito una privativa sul software così estesa come quella che emerge dal d.lgs. 518/92, ma avrebbe, al contrario, legittimato diverse attività di terzi. Ciò detto, adottando una licenza open source, l'autore di un programma per elaboratore rinuncia all'esercizio di una serie di diritti di più ampia portata rispetto a quelli previsti,in generale, a tutela dell' opera dell' ingegno, e che gli avrebbero consentito un maggior controllo sulla propria opera.

L'analisi dell'art. 64-bis rivela come le facoltà cui l'autore del programma per elaboratore rinuncia sono ben lontane da quelle che riguardano le altre opere dell'ingegno. Il punto a) dell'art. 64-bis riserva all'autore la riproduzione del programma; se la norma si limitasse a questo, non direbbe niente di diverso da quanto stabilito dall'art 13 l.a.: l'autore del programma si limiterebbe a rinunciare al diritto esclusivo di moltiplicazione in copie dell'opera. In realtà, all'art. 64-bis punto a) per riproduzione del software s' intende molto di più di quanto disciplinato dall'art. 13 l.a.. Innanzitutto la norma fa riferimento alla figura della riproduzione "temporanea" del programma, operazione che si sostanzia nel semplice passaggio del programma nella memoria dell'elaboratore e che consente il funzionamento del programma stesso. La norma insiste, inoltre, su operazioni quali " il caricamento, la visualizzazione, l'esecuzione, la trasmissione o la memorizzazione": si tratta d'attività tipicamente effettuate da un qualsiasi elaboratore, che quindi non hanno necessariamente a che vedere con la produzione di una copia. Una privativa di così ampia portata fa in modo che, nel momento in cui opera la rinuncia, l'autore del programma abdichi a facoltà di più ampia portata rispetto a quelle garantite dalla legge sul diritto d'autore per le altre opere dell' ingegno. Sembrerebbe quasi che l'obiettivo del legislatore sia stato, con l'art. 64-bis, quello di riservare, non solo la riproduzione, ma anche l' uso del programma; non sembra possibile opporre a tale conclusione che l' art. 64-ter disciplina i diritti dell' utilizzatore, dal momento che tale norma si applica solo al "legittimo acquirente", vale a dire a un avente causa del titolare dell' esclusiva. Grazie ad una così ampia nozione di riproduzione, il legislatore non ha sentito il bisogno di riservare al titolare attività quali la diffusione, per la quale dovrà farsi riferimento alla norma generale, sancita per tutte le opere dell'ingegno, dall'art. 16 l.a..[16]

Il diritto di riproduzione costituisce, quindi, una facoltà dell'autore distinta dalle altre, non essendo in esso compreso il concetto di divulgazione e di messa in commercio, ma solo quello di fabbricazione: rinunciando a tale diritto, l'autore abdica, quindi, alla facoltà di esercitarlo attraverso la repressione delle riproduzioni, che, di conseguenza, non possono più essere considerate illecite, prima che queste siano poste in commercio.

7. In secondo luogo, l'autore abdica al diritto esclusivo di distribuzione (art. 17 l.a.), concernente la facoltà di "mettere in commercio, di porre in circolazione o comunque di mettere a disposizione del pubblico, con qualsiasi mezzo e, a qualsiasi titolo, l' opera o gli esemplari di essa". Come ha precisato la Corte di Cassazione[17], il diritto esclusivo di mettere in commercio si concreta nel porre in circolazione, a

[16] Vedi infra par. 2.6.2

[17] Vedi Corte di Cassazione, 18 febbraio 1993, in AIDA 1993

scopo di lucro, esemplari dell'opera; ha così modo di esprimersi il profilo patrimoniale del diritto d'autore, giacché il prezzo di ciascuna copia del corpo materiale, nel quale l'opera intellettuale si concretizza, spetta, in via esclusiva, all' autore. La Corte di Cassazione ha, tuttavia, posto l'accento sul fatto che l'esclusività del diritto riguarda solo il primo esito delle copie dell' opera e non le successive, le quali possono essere liberamente vendute o donate dall'acquirente. Ne consegue che, in relazione alla vendita o donazione delle opere successive, la violazione del diritto esclusivo di distribuzione dell'opera dell'ingegno si realizza solo nel caso in cui l' autore abbia esercitato il diritto di ritiro dell' opera dal commercio di esemplari lecitamente riprodotti (art. 2582 c.c., art, 142 l.a.), mentre, al di fuori di tale ipotesi, il commercio realizza la violazione del diritto di distribuzione solo se collegato ad un'attività di abusiva riproduzione, che, appunto, si esteriorizza e assume rilevanza attraverso il primo atto d'immissione nel circuito distributivo.

Anche a proposito del diritto di distribuzione dell'opera, occorre fare riferimento a quanto stabilito dall'art. 64-bis lett. c) con particolare riguardo al software. A norma dell'art. 17 l.a., infatti, il diritto esclusivo di mettere in commercio ha per oggetto l'attività di porre in circolazione l' opera e gli esemplari di essa, a scopo di lucro: sembrerebbe , al contrario, potersi considerare libera ogni altra forma di circolazione dell' opera. Per il software sembra valere un principio diverso, dal momento che l'art. 64-bis lett. c) non contiene alcun riferimento allo scopo lucrativo. Come per la riproduzione, anche qui, l'ambito della privativa ha un raggio più ampio rispetto a quello delineato dalle norme generali sui diritti d'utilizzazione economica dell'opera dell'ingegno. In particolare, secondo quanto previsto dall'art. 64-bis lett. c), l'autore abdica al diritto esclusivo di effettuare o autorizzare ogni forma di distribuzione al pubblico del programma originale o di copie di esso, inclusa la locazione. L'art. 64 bis lett. c) contiene inoltre un'esplicita normazione dell'"esaurimento del diritto": istituto disciplinato in modo esplicito per le invenzioni industriali, consiste nel limitare la facoltà esclusiva di trarre profitto dall'opera alla prima vendita degli esemplari dell'opera stessa. A questo proposito la norma in questione afferma che " la prima vendita di una copia del programma nella Comunità Economica Europea da parte del titolare ei diritti, o con il suo consenso, esaurisce il diritto di distribuzione": l'acquirente di tali esemplari è, quindi, legittimato a disporre degli stessi; tuttavia, lo stesso art. 64-bis lett.c) restringe l'ambito in cui solitamente opera l'esaurimento: viene, infatti, riconosciuto al titolare un diritto di controllare l'ulteriore locazione del programma o di una copia di esso. Vi è, in sostanza, una parte dell'esclusiva che non si esaurisce con la vendita di una copia. Aderendo ad una licenza di software libero, il titolare dei diritti d'utilizzazione economica del programma abdica, in forza del disposto dell'art. 64-bis lett.c), alla possibilità di esercitare un controllo, non solo sulla vendita della propria opera, ma anche sulla locazione della stessa.

Sempre con riguardo al diritto di distribuzione, il legislatore italiano, con l'introduzione del nuovo testo dell'art. 17 l.a. (apportata tramite d.lgs. 685 del 16 novembre 1994), si è preoccupato di precisare che non costituisce esercizio del diritto di distribuzione "la consegna gratuita, effettuata o consentita dal titolare, di esemplari di opere a fini di promozione ovvero a fini di insegnamento o di ricerca scientifica". Si può, dunque, ritenere che non provochi esaurimento comunitario del diritto sull'opera, la distribuzione del programma come "shareware", vale a dire con rinuncia temporanea dell'autore al diritto di distribuzione del programma originale o di una sua copia, per tutto il periodo di prova concesso all'utente; questi, da parte sua, può utilizzare gratuitamente il programma o un suo esemplare al fine di decidere se acquistarlo. In definitiva, quindi, come nel modello disegnato per il software tradizionale, l'autore non potrà limitare contrattualmente la consegna gratuita di copie del programma ai fini suddetti, così, nel modello open source, l'autore non ha necessità di consentire la stessa in un'apposita clausola contrattuale.

Guardando il problema più da vicino, in materia di distribuzione e di riproduzione, la GPL afferma al paragrafo 1 che "è lecito copiare e distribuire copie letterali del codice sorgente del programma così come viene ricevuto, con qualsiasi mezzo"; la stessa pone, tuttavia, delle condizioni:

a) La chiara riproduzione di un'appropriata nota di copyright e d'assenza di garanzia su ogni copia. In questo modo viene ribadito il mantenimento del diritto morale d' autore in capo al creatore dell' opera originale; ma non solo: il fine della nota è, altresì, quello di rendere immediatamente chiaro all'utilizzatore che, a contrario dei prodotti software tradizionali, quello che ha ricevuto è un prodotto privo di qualsiasi garanzia.

b) che si mantengano intatti tutti i riferimenti alla licenza e all'assenza di ogni garanzia.

c) che si dia ad ogni altro destinatario del programma coperto da licenza, una copia della licenza insieme al programma.

La GPL si cura di precisare, al paragrafo 6, che ogni volta che il programma coperto dalla licenza 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 i termini e le condizioni della GPL, non essendo consentito imporre restrizioni ulteriori all'acquirente, nel suo esercizio dei diritti garantiti dalla licenza stessa.

La GPL consente, inoltre, di copiare e distribuire il programma (o un'opera basata su esso) sotto forma di codice oggetto o eseguibile purché ricorrano alternativamente le seguenti condizioni:

a) Il programma sia corredato del codice sorgente completo, in una forma leggibile dal calcolatore. La GPL si preoccupa di precisare che tale sorgente sia fornito, secondo le regole fissate dalla stessa GPL, 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 tal caso, la GPL prevede che si possa richiedere un compenso non superiore al costo del trasferimento fisico della copia, la quale deve essere fornita, anche in questo caso 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 nel caso di distribuzioni non commerciali e solo se il programma è stato ottenuto sotto forma di codice oggetto o eseguibile (paragrafo 3).

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 diventa, poi, facoltativo per l'acquirente. Tali condizioni si rendono necessarie poiché si ritiene che il codice sorgente sia la forma preferenziale usata per modificare un'opera.

8. Il diritto esclusivo di diffusione dell'opera fa sorgere delicate questioni giuridiche. A norma dell'art. 16 l.a., tale diritto ha per oggetto " l'impiego di uno dei mezzi di diffusione a distanza, quali il telegrafo, il telefono, al radiodiffusione, la televisione ed altri mezzi analoghi". Originariamente, quindi, il diritto esclusivo di diffusione aveva poco a che fare con il software quale opera dell'ingegno, non potendo questa, per la sua particolare natura, essere diffusa attraverso l'uso dei mezzi indicati dall'art. 16 l.a.. L'avvento delle reti telematiche ha, però, radicalmente modificato lo scenario: oggi, non solo il software, ma, più in generale, le opere dell' ingegno possono raggiungere qualsiasi utente collegato alla rete tramite il proprio computer, in ogni angolo del globo. Diventa quindi necessario chiarire se Internet può essere ricompreso tra i mezzi di diffusione a distanza elencati dall'art. 16 l.a. (in particolare tra i c.d. "mezzi analoghi"). Premettendo che non si tratta di un'elencazione tassativa, a mio avviso, si può concludere che le reti telematiche costituiscano, in effetti, un mezzo di comunicazione a distanza, potendosi considerare la capacità di raggiungere utenti collocati in vari punti del globo, l'elemento unificante di tali mezzi. Allo sviluppatore di software spetta, quindi, un diritto esclusivo di diffusione dell'opera: quale titolare di tale situazione giuridica soggettiva, egli ha, di conseguenza, la possibilità di rinunciarvi attraverso l'adesione al modello open source. Diversamente dalla legge sul diritto d'autore italiana, la convenzione di Berna parla, all'art. 11-bis, del diritto esclusivo dell'autore di autorizzare: la radiodiffusione della propria opera o la comunicazione al pubblico di essa mediante qualsiasi altro mezzo atto a diffondere senza filo segni, suoni od immagini; anche in questo caso, a mio avviso, è possibile far rientrare la rete telematica tra i mezzi di diffusione "senza fili". L'opportunità di diffondere il programma attraverso la rete, genera delicati problemi strettamente legati ad una più vasta possibilità d'accesso all'opera e, di conseguenza, ad una ridotta possibilità di controllo dell'autore sulla stessa. Alle origini del diritto d'autore, infatti, le possibilità di fruizione dell'opera da parte del pubblico e, quindi, di rientro economico per l' autore e per chi provvedeva alla diffusione, erano date dalla riproduzione e dalla messa in commercio di un certo numero di esemplari dell'opera su un supporto che variava a seconda della natura dell' opera stessa. Già l'avvento del sistema di radiodiffusione aveva costituito un punto di svolta per la disciplina del diritto d'autore; la possibilità di fruire dell'opera per effetto d'energie e senza un supporto materiale, aveva reso necessario l'intervento del legislatore: era, quindi, stato riconosciuto all'autore il diritto di diffusione a distanza dell'opera, quale diritto distinto e diverso da quello d'esecuzione e di rappresentazione in pubblico. Oggi, grazie all' innovazione tecnologica e all'avvento dei sistemi digitali, le opere dell'ingegno possono essere diffuse attraverso le reti telematiche, evitando i passaggi della catena produzione – distribuzione – vendita o noleggio del supporto che contiene l'opera stessa. L'uso della rete consente, in ultima analisi, a chiunque sia ad essa collegato, non solo di ricevere, in qualunque momento, l'opera richiesta, ma anche di registrarla nella memoria del proprio computer, di riprodurla e di ritrasmetterla ad altri utenti. Mentre per il software tradizionale, si è, quindi, posto il problema della tutela del diritto d'autore, il software open source ha, al contrario, utilizzato le immense potenzialità delle rete per diventare una realtà globale e consolidata.

Tra l'altro, il diritto esclusivo di messa a disposizione del pubblico trova, oggi, un formale riconoscimento nell'art. 3 della Direttiva 2001/29/CE relativa al diritto d'autore nella società dell'informazione. La norma in questione dichiara che gli Stati membri riconoscono agli autori il diritto esclusivo di autorizzare o vietare qualsiasi comunicazione al pubblico " compresa la messa a disposizione del pubblico delle loro opere, in maniera tale che ciascuno possa avervi accesso dal luogo e nel momento scelti individualmente". Il diritto in questione non coincide con il diritto alla diffusione dell'opera sancito dall'art. 16 l.a., ma si rinviene nella definizione del diritto esclusivo di distribuzione dell'opera, il quale ha ad oggetto, tra l'altro, il diritto "di porre in circolazione o comunque a disposizione del pubblico, con qualsiasi

mezzo ed a qualsiasi titolo, l'opera". Il problema che il legislatore italiano dovrà affrontare, in sede d'attuazione della Direttiva 29/2001, sarà, quindi quello di inserire nella legge sul diritto d'autore, quale nuovo diritto, quello di comunicazione al pubblico di cui all'art. 3 della direttiva, modificando, di conseguenza, il testo dell'art. 17 l.a.

Non bisogna, tuttavia, dimenticare che il diritto dell'autore di disciplinare l'accesso alla sua opera, deve essere bilanciato con la libertà dell'utente di accedere ai contenuti di Internet. L' accesso all'opera avviene, infatti, senza l'intermediazione di un supporto (la cui vendita o noleggio avrebbe garantito all'autore un la corresponsione di una somma di denaro), ma attraverso un servizio reso all'utente, che è, quindi, posto nelle condizioni di utilizzare l'opera in un modo che può significativamente incidere sulla vita economica della stessa. D' altra parte però, la possibilità di utilizzare l'opera attraverso la rete telematica, consente diversi tipi di fruizione, alcuni dei quali non rientrano nell'esclusiva del diritto d' autore, quale per esempio la semplice raccolta di dati informativi (vedi anche le libere utilizzazioni). Di conseguenza, solo quando l'accesso all'opera va ad interferire con alcuno degli esclusivi diritti d'utilizzazione economica sulla stessa riconosciuti all'autore, questi è legittimato ad intervenire contro ogni tipo d'elusione, neutralizzazione o aggiramento delle misure tecnologiche adottate per disciplinare l'utilizzo dell'opera. In particolare, la Direttiva CE, all'art. 2, riconosce all'autore il diritto di riproduzione, precisando che esso consiste nel diritto esclusivo di autorizzare o vietare la riproduzione diretta o indiretta, temporanea o permanente, in qualunque modo o forma, in tutto o in parte dell'opera.[18]

9. Infine, l'adesione al modello open source comporta, per l'autore, la rinuncia al diritto esclusivo d'elaborazione, il quale, ex art. 4 l.a., comprende tutte le forme di modificazione, elaborazione e di trasformazione dell'opera che danno luogo alle cosiddette elaborazioni di carattere creativo, quali le traduzioni in altra lingua, le modificazioni che costituiscano un rifacimento sostanziale dell' opera .

Con riferimento alla disciplina dettata specificamente per il software, l'autore di programmi che possono essere classificati come software libero, rinuncia volontariamente all'esclusiva prevista dall'art. 64-bis l.a., il quale sancisce il diritto esclusivo dell'autore ad autorizzare ogni traduzione, trasformazione, adattamento e modificazione del programma, nonché la riproduzione dell' opera che ne consegue. Anche le attività riservate in forza del punto b) dell'art. 64-bis l.a. mostrano, quindi, un discostamento dalla tradizione del diritto d'autore. L'art. 18 l.a. distingue, infatti, tra diritto esclusivo di introdurre modifiche e diritto d'elaborazione. Qualora l'elaborazione mostri carattere creativo, essa origina, come appena detto, una nuova opera che si affianca a quell'originaria. Lo sfruttamento economico dell'opera elaborata è subordinato al consenso del titolare, mentre si ritiene che le medesime attività restino lecite nell'ambito personale dell'utilizzatore. Quest'ultima possibilità sembra da escludersi per il software, posto che l'art. 64-bis lett. b) subordina ogni attività d'adattamento, intervento spesso necessario ai fini dell'utilizzazione del programma, ad autorizzazione. Come ho appena accennato, il riconoscimento di una specifica esclusiva d'elaborazione in capo all'autore (art. 4 l.a.), impedisce, in materia di software tradizionale, di accordare protezione, a qualsiasi elaborazione o trasposizione del programma, anche nel caso sia realizzata con un apprezzabile contributo creativo.

L' elaborazione creativa è oggetto d'autonoma protezione da parte della legge sul diritto d'autore, purché sia soddisfatta un'importante condizione: che la tutela accordata non rechi "pregiudizio ai diritti esistenti sull'opera originaria" (art. 4 l.a.). Ciò significa che l'elaborazione del programma è tutelata se creativa, purché autorizzata dal titolare dei diritti sul programma elaborato. Il diritto d'elaborazione, che va quindi tenuto distinto dal diritto di riproduzione, pone intorno all'opera un'area di tutela che va oltre l'opera stessa nelle sue forme riproduttive, comprendendo la creazione d'opere che possono essere, giuridicamente, attribuite ad altri autori, ma che essendo, in ogni caso, collegate con l'opera originaria, non possono essere divulgate e diffuse senza il permesso dell'autore di quest'ultima.

L'applicazione del principio del consenso dell'autore ha creato problemi d'applicazione della legge sotto vari profili: innanzitutto a causa l'eccessiva durata della protezione accordata al programma per elaboratore, la quale risulta manifestamente sproporzionata rispetto alla durata commerciale dello stesso. Si tratta di un'incongruenza generata dalla miope assimilazione del software alle opere letterarie. In secondo luogo, perché è implicito del progresso informatico il ricorso ad algoritmi e sequenze logiche già utilizzati in

[18] Si tratta di un diritto sottoposto, a norma della stessa direttiva, a una serie di eccezioni. Ex art. 5, infatti, sono esclusi dal diritto di riproduzione, gli atti di riproduzione temporanea "privi di rilievo economico proprio, che sono transitori o accessori e parte integrante o essenziale di un procedimento tecnologico, eseguiti all' unico scopo di consentire a) la trasmissione in rete tra terzi con l'intervento di un intermediario b) un utilizzo legittimo dell' opera. Le eccezioni riguardano, inoltre, particolari categoria di utenti quali le biblioteche pubbliche, istituti di istruzione, musei, archivi che non mirano ad alcun vantaggio economico o commerciale diretto o indiretto. In ultimo luogo, l'art. 3 lascia alla discrezionalità degli stati membri la possibilità di disporre in merito ad una lunga serie d'eccezioni al diritto di riproduzione. Non c' è dubbio che la norma non gioverà all'armonizzazione delle normative degli Stati membri.

altri programmi. Non è raro, infine, il caso d'utenti che provvedono autonomamente alla manutenzione del programma o che si rivolgono a società d'assistenza non autorizzate dal titolare del diritto d'autore sul programma. L'utente rimane, in tale ipotesi, teoricamente soggetto al rischio di azione legale da parte dell' autore del programma, il quale potrebbe addirittura lamentare la lesione del diritto morale d' autore (art. 2577 comma 2).

Non bisogna dimenticare che, abdicando al diritto esclusivo d'elaborazione e, con esso, agli specifici diritti di modifica e traduzione, l' autore si preclude la possibilità di vietare ad altri di apportare all'opera qualsiasi modificazione, anche modesta. Gli interventi creativi, in ogni caso collegati al programma originario, non richiederanno quindi più il previo consenso dell'autore. Il programma per elaboratore nasce aperto alla modifica, al contrario di quanto succede per le opere artistiche o letterarie; è, infatti, dotato di un'intrinseca strumentalità moltiplicativa: non è rara la possibilità che il programma produca strumenti destinati, in un circolo virtuoso, a produrne altri. Il modello open source non fa altro che facilitare lo scambio di tali programmi, promovendo la libera, immediata e gratuita fruizione dell' opera. La facilità dell'accesso ha, poi, condotto ad un notevole aumento della creatività complessiva, facendo sì che autori e utilizzatori si trovassero sovente a posizioni invertite ad ogni scambio, laddove questo segnasse un nuovo balzo inventivo.

Il modello del software libero elimina, in ultima analisi, la figura del monopolista di scambio del prodotto e spinge le restanti due figure della relazione creativa, l'autore e il consumatore a scambiarsi di ruolo. Chi decide di aderire al modello open source consegue gratuitamente la disponibilità del bene. In cambio, ove apporti delle modifiche, s'impegna, tuttavia, a riportare gli estremi identificativi del programma originario sulla nuova opera, rimettendo, a sua volta, gratuitamente questa a disposizione della comunità. Non è un caso, quindi che tale modello, provenga dagli Stati Uniti, il cui ordinamento giuridico non esalta la necessità di riconoscere il principio generale dell'attribuzione della titolarità all'uomo autore. Il diritto d'autore, una volta che l'opera sia pubblicata, consiste in un diritto di sfruttamento di durata determinata (calcolata dalla data di pubblicazione dell'opera) e garantita dalla legge solo se l'autore manifesta la sua volontà di godere di una tale protezione. In un sistema siffatto, l'autore, con la pubblicazione, abbandona parte dei suoi diritti esclusivi e assoluti sull'opera. Inoltre la protezione del diritto morale dell'autore non rientra nella particolare disciplina di tale privativa di sfruttamento: è, invece, ricompresa nell'ambito della tutela generale, di diritto comune, dei diritti della personalità.

10. Le licenze di software libero, generalmente considerate complessi documenti legali, non sono, al momento, ancora passate al vaglio della giurisprudenza (neanche negli Stati Uniti, dove il fenomeno è più radicato). La mancanza di pronunce giurisprudenziali fa sì che permangano significativi dubbi sulla loro interpretazione. Al fine di agevolarne la classificazione, si è soliti distinguere tre classi di licenze: quelle restrittive, come la LGPL, che richiedono che le modifiche del programma siano rese liberamente disponibili, quelle estremamente restrittive, come la LGPL, che implicano che il programma non possa essere compilato con altri programmi proprietari e infine quelle non restrittive, come la BSD, che non impongono le condizioni elencate[19]. Sotto un altro punto di vista, le licenze di software libero si dividono in due grandi categorie: quelle con permesso d'autore e quelle senza. Le prime, tra le quali spicca per importanza e diffusione la GPL, insistono sul fatto che le versioni modificate di ciascun programma protetto dalla licenza debbano essere, a loro volta, software libero[20]. Le licenze senza copyleft, al contrario, ammettono che le modifiche al programma possano essere rese private.

Si potrebbe ritenere che il modo più semplice di rendere un programma libero sia quello di porlo sotto dominio pubblico. Una scelta simile consente la condivisione del programma e dei miglioramenti apportati allo stesso, tuttavia, offre, a chiunque lo desideri, la possibilità di convertire il programma in un prodotto software proprietario. Gli utenti che, a questo punto, ricevono il programma, non possono più godere delle facoltà che l'autore originario aveva loro attribuito. E' proprio qui che intervengono le licenze dotate di copyleft: queste, infatti, garantiscono la possibilità di distribuire il codice sorgente del programma e qualunque modifica ad esso apportata, ma solo sotto i termini della stessa licenza open source utilizzata per quel programma. In questo modo, mentre il programma e le modifiche si trasmettono da soggetto a soggetto, quest'ultime continuano a rimanere open source. Questo ingegnoso stratagemma prende forma nella GPL, la licenza con copyleft per eccellenza. Nel preambolo si legge, infatti, che "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".

Con particolare riguardo al diritto di modifica, la GPL afferma, al paragrafo 2, che "è lecito modificare la propria copia o copie del programma, o parte di esso, creando un'opera basata sul programma, e copiare o distribuire tali modifiche o tale opera" secondo i termini fissati dalla stessa GPL.

[19] Vedi anche Josh Lerner e Jean Tirale, The Scope of Open Source Licensing su http://www.nber.org/papers/w9363.
[20] In particolare, la GPL impone che ciascuna versione modificata ricada sotto GPL.

Anche in questo caso, tuttavia, la licenza si preoccupa di fissare determinate condizione, al cui soddisfacimento è subordinata la facoltà di modifica dell'opera. E', infatti, necessario:

a) Indicare chiaramente nei file che si tratta di copie modificate e la data di ogni modifica. In questo modo, l'opera creata dall'autore originario viene tenuta distinta dalle successive modifiche, senza pregiudizio del diritto morale d'autore.

b) 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. Oltre ad essere l'aspetto più controverso della licenza, è questa la clausola che caratterizza la GPL come una licenza dotata di copyleft.

c) Se, normalmente, il programma modificato legge comandi interattivamente quando viene eseguito, fare in modo che all'inizio dell'esecuzione interattiva usuale, esso stampi un messaggio contenente un'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 la 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.

L' obiettivo della clausola che prevede il copyleft è quello di evitare che il codice del programma sia reso privato. Si prenda ad esempio il sistema operativo Linux: questo rappresenta il risultato del lavoro svolto da centinaia di capaci programmatori nel corso di diversi anni. Si tratta di un prodotto flessibile al punto, da essere utilizzato in un gran numero di ambienti commerciali. Senza la clausola copyleft, chiunque potrebbe ottenere il codice di Linux, apportare talune migliorie, quindi licenziare il sistema operativo modificato sotto un modello proprietario, per esempio vendendo copie del codice oggetto del nuovo sistema operativo, senza rivelare il codice sorgente.

La clausola riguardante il copyleft si applica all'opera modificata nel suo complesso; se, tuttavia, sussistono parti identificabili dell'opera modificata che non siano derivate dal programma e che possono essere ragionevolmente considerate lavori indipendenti, la GPL e i suoi termini non si applicano a queste parti, quando queste vengono distribuite separatamente. E' tuttavia la un'altra locuzione ad essere potenzialmente catastrofica per l'utilizzatore ignaro dei problematici risvolti della licenza GPL. Quest' ultima, infatti, afferma: "Se però queste parti (non derivate dal programma e ragionevolmente identificabili come lavori indipendenti) sono distribuite all'interno di un prodotto che è un'opera basata sul programma, la distribuzione di quest'opera nella sua interezza deve avvenire secondo i 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". Tale conclusione, non bisogna dimenticare, è rafforzata da quanto disposto in precedenza: "È necessario 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". Quindi, se parte di codice protetto dalla GPL viene incorporato in un prodotto software proprietario, si potrebbe sostenere che il prodotto software proprietario, nel suo complesso, diventi open source e debba essere licenziato sotto la GPL. Tale evenienza illustra a dovere l'importanza di comprendere a pieno il significato della licenza open source e i suoi potenziali pericoli. Usare programmi rilasciati sotto la licenza GPL implica, infatti, una pianificazione attenta, al fine di evitare situzioni nelle quali una società potrebbe trovarsi costretta a cedere il proprio lavoro, cosa che potrebbe accadere se parte di un programma licenziato sotto GPL diventasse parte di un altro prodotto. La stessa GPL si preoccupa di precisare che 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 della stessa. In pratica, il ricorso alla GPL assicura all'autore la possibilità di esercitare il diritto di controllare la distribuzione di programmi derivati dal programma o che lo contengano.

11. Una variante della GPL, conosciuta come LGPL, concede una maggiore flessibilità riguardo alla possibilità di combinare insieme codice aperto e codice chiuso: in particolare permette di unire il programma con altri programmi che non siano rilasciati sotto una licenza open source. E' la stessa LGPL che si preoccupa di precisare che quando un programma è unito ad una libreria, la combinazione delle due è legalmente chiamata "combined work", un derivato della libreria originale. La GPL permette tali collegamenti solo se la combinazione complessiva risponde ai criteri di libertà in esso fissati: ciò significa, in buona sostanza, che il programma, complessivamente considerato, deve essere distribuito sotto GPL . La LGPL è, invece, più permissiva in materia di unione fra la libreria e altro codice. In particolare, al paragrafo 5 la LGPL stabilisce che un programma che non contiene derivati della Libreria, ma è disegnato per funzionare con la libreria, essendo ad essa collegato, è chiamato "work that uses the Library", ovvero opera che utilizza la Libreria. Tale opera, se presa separatamente, non può essere considerata un derivato della Libreria: si pone quindi al di fuori dello scopo della licenza.

In ogni caso, prosegue la LGPL, collegare un'opera che utilizzi la Libreria con la Libreria stessa, crea un programma eseguibile che può essere considerato un derivato della Libreria, giacché contiene delle

porzioni di questa, e che, quindi, è coperto dalla licenza. Per quel che riguarda i termini della distribuzione, poi, la sezione 6, stabilisce che, in deroga a quanto sancito dalla sezione 5, è possibile combinare o collegare un'opera che utilizza la Libreria con la Libreria al fine di creare un programma che contenga porzioni della Libreria, e distribuire quel programma sotto i termini di una licenza a scelta.

12. Il programma Perl fu originariamente reso disponibile dal suo creatore, Larry Wall, sotto GPL. Questi, tuttavia, si accorse, ben presto che i termini della licenza risultavano troppo restrittivi: sviluppò, quindi, quella che è comunemente definita la Licenza Artistica. In forza di tale licenza, gli utenti sono liberi di sviluppare prodotti commerciali basati sul codice di Perl. La Licenza Artistica statuisce, infatti, che è possibile distribuire il programma, originale o modificato, insieme con altri programmi, come parte di una distribuzione software più vasta, purché il programma non sia pubblicizzato come proprio prodotto. Del resto, La possibilità, riconosciuta, di combinare insieme codice proprietario e codice open source non incontra limitazioni.

Per quel che riguarda, nello specifico, la possibilità di modificare il programma, la Licenza artistica stabilisce, alla sezione 3 che è possibile modificare la propria copia del programma, purché sia inserita una nota in ogni file modificato che chiarisca quando e come il file ha subito delle modifiche. Si richiede inoltre che sia soddisfatta almeno una delle seguenti condizioni:

a) Rendere le modifiche di pubblico dominio o, altrimenti, liberamente disponibili per esempio inviando le rilasciando le suddette modifiche su Usenet o su un media equivalente, oppure autorizzando il detentore del copyright a includere le modifiche nella versione standard del pacchetto.

b) Usare il programma modificato solo all'interno della propria società o organizzazione

c) Rinominare ogni programma eseguibile che non sia standard in modo tale che i nomi non siano in contrasto con i programmi standard eseguibili, che devono essere forniti.

13. Licenze libere, non restrittive, senza permesso d'autore, come la BSD e la licenza Apache pongono condizioni sicuramente meno stringenti rispetto alla GPL. Entrambe le licenze menzionate, infatti, consentono la redistribuzione e l'uso del codice in forma binaria e in forma sorgente, con o senza modifiche, purché siano rispettate talune semplici condizioni; innanzitutto richiedono che le redistribuzioni del codice contengano la nota di copyright prevista dalla stessa licenza (Copyright (c) 2000 The Apache Software Foundation. All rights reserved), la lista di condizioni e la rinuncia a qualsiasi garanzia, prevista dalla stessa. La documentazione dell'utente finale inclusa nella redistribuzione deve includere il seguente riconoscimento

> "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)."

In alternativa, tale riconoscimento può comparire sullo stesso software.

La licenza utilizzata per la BSD ha creato, fino al 1999, particolari problemi pratici. La causa del problema deve essere ricercata nella clausola pubblicitaria; questa richiedeva che tutto il materiale pubblicitario che aveva fatto menzione di caratteristiche del software doveva contenere un particolare riconoscimento consistente in una semplice proposizione:

> This product includes software developed by the University of California, Berkeley and its contributors.

La clausola pubblicitaria non rendeva il software non libero, tuttavia creava problemi pratici; tale clausola pubblicitaria fu, inizialmente, utilizzata solo dalla BSD: includere tale semplice frase non costituiva, allora, una questione pratica rilevante. Se gli altri sviluppatori avessero usato la clausola pubblicitaria inserita nella BSD, inclusa la parte che si riferisce all'Università della California, non avrebbero reso la questione particolarmente fastidiosa. Tuttavia, come ci si poteva aspettare, gli altri sviluppatori non copiarono la clausola pubblicitaria letteralmente, ma la cambiarono, sostituendo " University of California" con il proprio nome o quello dell'istituzione in cui lavoravano. Quando si cominciò a combinare insieme programmi, coperti da licenze diverse, in un unico sistema operativo, il problema è emerso: la versione 1997 di NetBSD ha richiesto 75 diverse proposizioni, rese necessarie dalle varie licenze che proteggevano le singole componenti del sistema, ciascuna delle quali faceva riferimento ad autori o gruppi di autori differenti. Prima gli sviluppatori della FreeBSD, nel maggio del 1998, poi l'Università della California nel giugno del 1999, hanno rimosso la clausola pubblicitaria dalle loro licenze originali[21]. Oggi esiste una versione rinnovata della BSD (così come della licenza Apache) che non contiene la clausola incriminata. La BSD modificata, così come la licenza Apache versione 1.1 sono licenze per software libero semplici, permissive, senza permesso d'autore, compatibili con la GNU GPL. L' espulsione della clausola pubblicitaria non fa, tuttavia, venir meno

[21] Grazie soprattutto alla mediazione di Richard Stallman

la liceità delle licenze ancora contenenti la clausola pubblicitaria. Ovviamente la modifica apportata alla BSD non ha alcun effetto su altre licenze che imitano lo stile della BSD originaria: tuttavia, l'inversione di rotta potrebbe convincere altri sviluppatori ad eliminare la clausola pubblicitaria dalle loro licenze.

14. Anche semplici licenze senza permesso d'autore come la licenza MIT e la licenza X Consortium (compatibili peraltro con la GPL) permettono a chiunque ottenga una copia del software protetto dalla stessa, di trattare il software senza restrizioni, compresa nessuna limitazione al diritto di usare, copiare, modificare, fondere, pubblicare, distribuire, sublicenziare, e/o vendere copie del software, e di permettere alle persone alle quali il software è fornito di farlo; l'unica condizione richiesta è l'inclusione della licenza e della nota di copyright in ciascuna copia o porzione di software distribuita.

15. Una particolare famiglia di licenze è quella introdotta dalle società commerciali che hanno deciso di rendere pubblico parti del proprio codice proprietario. Tra queste, la QPL (ideata da Trolltech per la propria Libreria Qt) consente di copiare e distribuire il software purché l'intero pacchetto sia distribuito includendo la nota di copyright, e le rinunce, così come rilasciate dallo sviluppatore originale. La QPL crea, tuttavia, rilevanti inconvenienti di tipo pratico, dal momento che i sorgenti modificati possono essere distribuiti solo sotto forma di patch. Si tratta di una licenza per software libero senza permesso d'autore, incompatibile con la GNU GPL: ciò significa che non è possibile non è possibile combinare insieme un programma rilasciato sotto GPL e uno rilasciato sotto licenza QPL; è tuttavia, possibile rilasciare sotto GPL un programma che utilizza una libreria X coperta dalla licenza QPL: se si è detentori del copyright sul programma, è possibile risolvere il conflitto attraverso una nota del tipo

> Come eccezione, avete il permesso di unire questo programma alla libreria X e di distribuire versioni eseguibili se rispettate i termini delle licenza GNU GPL per quanto riguarda tutto il software nell'eseguibile, eccetto X.

16. Dall'area della rinuncia operata dall'autore del programma, attraverso la licenza di software libero, vanno espunte una serie di utilizzazioni, regolate dall'art. 64 ter e quater, che sono già qualificate come libere con riferimento al software tradizionale. Si tratta di una serie di facoltà d'uso che, di regola, non necessitano di preventiva approvazione da parte del titolare del diritto d'autore. In alcuni casi, tuttavia, il legislatore fa salva la possibilità di patto contrario, attribuendo, quindi, al titolare del diritto d'autore la facoltà di limitare e proibire tali libere utilizzazioni attraverso specifiche clausole contrattuali.

Innanzitutto l'art. 64 ter comma 1 l.a., prescindendo dall'autorizzazione dell'autore, consente la riproduzione, la traduzione, l'adattamento, la trasformazione e ogni altra modificazione del programma, qualora tali attività siano "necessarie" all'uso dello stesso e " conformemente alla sua destinazione". Ai fini dell'individuazione della destinazione del programma dovrà farsi riferimento al contenuto degli accordi e delle dichiarazioni negoziali delle parti. La norma sembra, a prima vista, rappresentare un temperamento di quella nozione allargata di "riproduzione" idonea ex art. 64-bis lett. a), ad includere nell'esclusiva anche l'uso del programma. Considerando che l'uso richiede sempre una riproduzione, il suo fine sembrerebbe essere, quindi, quello di permettere al legittimo acquirente, quelle attività indispensabili affinché il programma esplichi la propria utilità. L'intento di temperare l'estensione della privativa sembra ancora più evidente con riferimento al punto b) dell'art. 64-bis, il quale consente gli adattamenti del programma necessari all'uso privato del legittimo acquirente; a ben vedere, tuttavia, tale risultato poteva essere raggiunto indipendentemente dalla previsione dell' art. 64-ter: il titolare, nel disporre del programma o di una copia di esso, può autorizzare l'acquirente a svolgere attività altrimenti riservate. Riproduzione, traduzione e adattamento sono, altresì, consentite, sempre ex art. 64 ter comma 1, in caso di "correzione degli errori". Al fine di poter ricomprendere tale attività nell'ambito delle libere utilizzazioni è, tuttavia, necessario, interpretare il concetto d'errore in senso restrittivo, includendo solo quegli errori che impediscono l'uso del programma conformemente alla sua destinazione. Nel momento in cui la correzione diventa un'attività di miglioramento o d'aggiornamento del programma, si rientra, per quanto riguarda il software tradizionale, nell'area di applicabilità dell' esclusiva di elaborazione dell' autore e, per il software libero, nell' area della rinuncia dell' autore. L' art. 64 l.a. fa salvo, per entrambe le ipotesi, il patto contrario: ciò significa che, con riferimento ad un programma software tradizionale, al titolare del diritto d'autore residuerà la possibilità di limitare l'area delle suddette attività, in sede contrattuale. Clausole simili, al contrario, non potranno trovare spazio, in una licenza di software libero, perché contrastanti con la volontà dell'autore di consentire la modificazione dell'opera originale.

L' art. 64 ter comma 2 attribuisce, poi, all'utilizzatore del programma la facoltà di creare una copia di riserva "qualora tale copia sia necessaria per l'uso": la copia, tuttavia non deve circolare autonomamente rispetto al suo originale. Tale facoltà non può essere proibita attraverso l'inserzione di una specifica clausola contrattuale, che sarebbe, di conseguenza, viziata di nullità.

L' art. 64 ter comma 3 attribuisce, infine, al legittimo utilizzatore la facoltà di osservare, studiare e sperimentare il funzionamento del programma dall'esterno (vale a dire durante le operazioni di caricamento, visualizzazione, esecuzione, trasmissione o memorizzazione) al fine di determinarne idee e principi; anche in questo caso ogni pattuizione contraria è nulla. Esula dall'ambito di operatività dell' art. 64 ter comma 3 la facoltà di studiare il programma dall interno, tramite l' attività di decompilazione e decodificazione del codice oggetto, vale a dire del contenuto creativo del programma espresso in un linguaggio comprensibile solo per la macchina: tali attività si realizzano per il tramite di riproduzioni e traduzioni del programma le quali, non solo risultano estranee alla sua normale utilizzazione, ma sono altresì dirette a decifrare il c.d codice sorgente, vale a dire il contenuto creativo del programma espresso in un linguaggio comprensibile all'uomo. Aderendo al modello open source, l'autore consente all' utente di procedere alla decompilazione e decodificazione del codice oggetto che, permettendo di giungere al codice sorgente, offre all'utilizzatore la possibilità di apportare modifiche e miglioramenti al programma. Tuttavia occorre precisare che, ex art. 64 quater, le attività di decompilazione e decodificazione non esulano, in ogni caso, dalle utilizzazioni libere: tali operazioni sono, infatti, consentite nel momento in cui tali operazioni siano indispensabili per ottenere le informazioni necessarie per consentire l'interoperatibilità con altri programmi, di un programma creato autonomamente. In tal caso l'osservazione e lo studio del programma dall'interno non richiedono una licenza open source per essere lecitamente attuate. Lo stesso art. 64 quater si preoccupa di fissare le condizioni sotto le quali le suddette attività sono consentite. Si richiede, in primo luogo, che siano effettuate dal licenziatario o da altri soggetti che abbiano il diritto di usare una copia del programma oppure, per loro conto, chi è autorizzato a tal fine. In secondo luogo si richiede che le informazioni necessarie ad ottenere l'interoperabilità non sia già state accessibili ai predetti soggetti. Infine, è necessario che le attività delle quali la decompilazione consta siano limitate alle parti del programma originale necessarie per conseguire l'interoperabilità.

17. La vertiginosa espansione della tecnologia informatica, negli ultimi decenni, ha dato vita a nuovi rapporti contrattuali aventi ad oggetto nuovi beni e servizi. Le relative tecniche di contrattazione, che si sono ben presto rivelate in tutta la loro complessità, in quanto a varietà di soggetti e prestazioni coinvolte nella negoziazione, hanno finito per richiamare l' attenzione della dottrina[22], che ha introdotto l' espressione "contratti informatici". Con quest'espressione si fa riferimento ai tutti i contratti riguardanti beni (hardware o software) o servizi (computer services) che rientrano nel comparto dell'informatica. Una volta identificata la categoria dei contratti informatici, seppur per il tramite di un elemento unificante piuttosto labile, è possibile guardare alla classificazione di tale tipologia contrattuale. I tipi negoziali rilevanti nel settore dell'informatica sono, in definitiva, i contratti che riguardano l' hardware e i contratti aventi ad oggetto il software. Tali beni possono essere forniti dalla stessa impresa con un unico contratto, oppure con contratti separati (e collegati), o ancora, possono essere forniti da imprese produttrici distinte. La circolazione dell' hardware e del software, inoltre, ne comporta anche la manutenzione e l' assistenza, che possono, a loro volta, essere oggetto di specifici contratti (c.d. mantainance contrats). Esiste infine la possibilità che l' hardware, il software e i relativi servizi di assistenza e manutenzione vengano forniti all' utente dalla medesima impresa con un unico contratto, attraverso la fornitura di un intero sistema informatico. Se si guarda all'oggetto del contratto, si possono, quindi, distinguere i contratti che hanno ad oggetto l' hardware e quelli che hanno ad oggetto il software. Proprio i contratti di fornitura del software presentano una certa complessità, la quale va ricollegata proprio alla controversa natura giuridica dell'oggetto del contratto. Si discute, infatti, se si tratti di bene materiale o immateriale. Al riguardo, sembra prevalere l'opinione che ritiene che il software abbia natura di bene immateriale[23]. Si è sottolineato, in proposito, come il programma per elaboratore possieda un proprio contenuto d' intellettualità (ossia la forma del programma, la manifestazione espressiva usata per dare soluzione a un problema) che necessita di un supporto materiale per estrinsecarsi. Tale supporto, sia esso un nastro, un disco, o simili, assolve ad una funzione meramente strumentale al godimento del contenuto del programma. I contratti di cessione dl software costituiscono una parte considerevole della contrattualistica del settore. Esaminando tale tipo di contratti, un dato importante deve essere subito evidenziato: i produttori di programmi,al fine di evitare che il programma venga duplicato e messo in commercio o altrimenti ceduto, preferiscono utilizzare forme contrattuali attraverso le quali la proprietà del programma non venga trasferita. Sono quindi soliti concedere in uso il programma dietro il pagamento di un canone forfettario o periodico, imponendo contrattualmente all'utente diverse

[22] DE NOVA, *L'oggetto del contratto d'informatica*: *considerazioni di merito*, in *Dir. Inf.* 1986, p. 803 ss.
[23] Vedi Cass. 19 ottobre 1990, in Dir. Inf. 1991, p. 186 ss.

limitazioni, come non riprodurre il programma in alcun modo, non cederlo a terzi neppure a titolo gratuito, utilizzarlo solo su un determinato elaboratore e via dicendo. Il contratto più diffuso è certamente la licenza d'uso.

18. Normalmente utilizzata per il software commercializzato in serie, la licenza d'uso può essere definita come quel contratto con il quale il creatore/fornitore di software cede ad un altro soggetto, il c.d. licenziatario, di regola dietro corrispettivo, il diritto di utilizzare, in modo non esclusivo e per un determinato periodo di tempo, il programma oggetto di licenza. La licenza è, quindi, istituto giuridico diretto a creare un vincolo obbligatorio tra l'autore dell'opera, che rimane titolare degli autonomi diritti riconosciutigli dalla legge in via esclusiva, e chi gode della concessione secondo determinate modalità di sfruttamento dell' opera. I diritti di proprietà e sfruttamento economico del bene immateriale del software, quale opera tutelata, rimangono in capo al proprietario, che è anche creatore e concedente (art. 1 l.a.); il diritto di sfruttare il bene materiale, rappresentato dal singolo esemplare del software, compete, invece, all'utilizzatore-licenziatario. A costui, altresì, spettano, in via accessoria e strumentale, anche i diritti relativi al supporto fisico nel quale il programma è incorporato. La più attenta dottrina[24] ha, peraltro, osservato come, nella maggior parte dei casi, si parli di "licenza" impropriamente:il contratto di licenza, infatti, si sostanzia nella concessione al licenziatario non solo della facoltà di godere di una certa idea creativa, ma anche di sfruttarla economicamente. Ciò si ricollega alla natura del programma: trattandosi di un bene immateriale protetto dal diritto d'autore, i relativi rapporti negoziali sono rapporti tra l'autore e gli altri soggetti. Tra questi rapporti è possibile ricomprendere contratti con cui l'autore concede ad un altro soggetto di compiere atti che altrimenti sarebbero coperti dal diritto esclusivo d'autore. Solo nel momento in cui si concede la facoltà di sfruttare economicamente il software, il contratto potrà essere qualificato come "licenza". Nella maggior parte dei casi, in ogni modo, tale facoltà non è concessa dal titolare del diritto, preferendosi consentire unicamente l'utilizzazione personale del programma. Nella pratica, quindi, i tipi principali di contratto ad oggetto informatico sono, da una parte il contratto di cessione di diritti sul software, che la dottrina più attenta qualifica come vere e proprie "licenze", e il contratto di licenza d'uso; laddove il primo ha ad oggetto la volontà dell'autore di trasferire uno o più diritti d'utilizzazione economica, il secondo ha ad oggetto la volontà dell'autore di concedere il solo uso temporaneo del programma, senza rinunciare ai suoi diritti di sfruttamento economico dello stesso. Non bisogna, da ultimo, confondere la titolarità dei diritti d'autore sul programma con la titolarità dei diritti (di proprietà, locazione o altra natura) sul supporto meccanico che contiene il programma ovvero una sua copia: come previsto dall'art. 109, infatti, la cessione di uno o più esemplari dell'opera non comporta, salvo patto contrario, la cessione dei diritti d'utilizzazione economica dell'opera stessa.

19. All'interno del contesto sopra appena delineato, il software open source si pone in una posizione molto particolare. La licenza d'uso è, tradizionalmente, un contratto che ha come scopo quello di porre delle condizioni maggiori rispetto a quelle che la legge impone a tutela del possessore del copyright su un particolare tipo di software, a svantaggio di qualunque eventuale utilizzatore che intenda oltrepassare i limiti posti. Nulla si oppone, tuttavia, alla possibilità che la licenza ponga condizioni minori di quelle fissate dalla legge. Quindi, nello stesso modo in cui protegge il diritto del possessore di copyright di stabilire delle condizioni restrittive ai fruitori della propria opera, così la licenza può tutelare il diritto dello stesso soggetto di decidere di rendere l'opera liberamente disponibile per chiunque ne sia interessato. Di regola, poi, il licenziatario utilizza una versione del programma in forma eseguibile dalla macchina (c.d. codice macchina), ma non il codice sorgente, intelleggibile all'uomo. Di conseguenza, non può conoscere il funzionamento intrinseco del software, trovandosi nell'impossibilità tecnica di apportare ad esso qualsivoglia modifica o aggiunta. Onde consentire la modificabilità del programma, la licenza open source rende l'opera disponibile nella sua forma più accessibile, vale a dire sotto forma di codice sorgente.Viene in pratica ribaltata la funzione per la quale l'istituto giuridico della licenza è stato creato: mentre le licenze della maggior parte dei programmi sono, dirette a privare l'utente della possibilità di apportare modifiche al programma e di condividerlo con altri, le licenze open source attribuiscono al concessionario il diritto di copiare, modificare, diffondere e distribuire il programma. Chi utilizzasse software libero od open source in modo diverso dalle condizioni prescritte da una licenza open source, tralasciando di far conservare alla stessa o alle sue modifiche le proprie caratteristiche di libertà, opererebbe un illecito perché contravverrebbe alla volontà dell'autore e possessore del copyright. In tal senso la GPL stabilisce che "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, distribuire il programma non è autorizzato, e farà terminare automaticamente i diritti garantiti da questa licenza". D' altro canto, la stessa si preoccupa di

[24] FRIGNANI, *I contratti di licenza nel diritto italiano*, *in Dir. Autore* 1979, p. 516 ss.

precisare, tuttavia, che ogni acquirente che riceva copie, o diritti, coperti dalla GPL da persone che l' abbiano violata, non vedranno invalidata la loro licenza, purché si comportino conformemente ad essa.

20. Sotto un profilo strettamente giuridico, la questione forse più interessante è quella relativa alla qualificazione giuridica della licenza d'uso e alla sua collocazione sistematica. Nessun dubbio sussiste sul fatto che, ai sensi e per effetto dell'art. 1321, si tratti di un contratto. La questione della qualificazione giuridica affiora, tuttavia, in tutta la sua problematicità, allorquando ci si voglia spingere oltre e definire che tipo di contratto sia la licenza d' uso e quale ne sia la causa.

Sul punto, possono essere individuati in dottrina due diversi orientamenti.

Un primo orientamento, riconducendo la licenza d'uso nell'ambito d'applicazione di figure negoziali tipiche, arriva a negarne la stessa autonomia giuridico-dogmatica. In particolare, secondo tale dottrina, il contratto di licenza d'uso configurerebbe una vera e propria locazione di bene mobile (software) e finirebbe, quindi, per essere interamente disciplinata dalle norme dettate dal codice civile che alla locazione si riferiscono (artt. 1571-1614)[25]. Secondo tale schema, il titolare del bene locato (il software) non trasferisce quel bene al licenziatario, ma s'impegna a far godere a questi le capacità e le potenzialità insite nel bene immateriale software. Ecco quindi che il software non individua una determinata fattispecie contrattuale giuridicamente tipica, ma si limita a configurare una vicenda avente come specifico contenuto l'attribuzione ad un soggetto della facoltà di godimento sul software.

Il secondo orientamento, comunemente ritenuto preferibile, parte da un dato incontrovertibile: l'assenza di una norma giuridica positiva di riferimento; non essendo disciplinata dal nostro ordinamento giuridico positivo, la licenza d'uso è un contratto atipico[26], che viene posto in essere dalle parti in forza del generale principio di autonomia contrattuale di cui all' art. 1322 c.c.. Nella sua struttura di base, può, quindi, essere utilmente assimilata a una locazione di bene mobile, avente ad oggetto il diritto (personale di godimento) d'utilizzazione della singola riproduzione del programma messa a disposizione del licenziatario. A tale qualificazione non osterebbe la natura di bene immateriale del software, sia perché le norme del codice non prevedono alcun'esclusione in questo senso, sia perché requisito fondamentale dell'oggetto della locazione è dato dal fatto che il bene sia in grado di assicurare un godimento continuato, presupposto soddisfatto dal software. Nello schema della locazione, s'inseriscono, tuttavia, elementi atipici che consentono di differenziare la licenza d'uso dalla locazione. L'applicabilità della disciplina della locazione richiederà un attento giudizio di compatibilità che tenga in dovuto conto le peculiarità dell' oggetto contrattuale, ossia il bene immateriale del software. Si configurano come elementi atipici rispetto al contratto di locazione sia le modalità di pagamento del corrispettivo, che nel caso della licenza avviene, normalmente, in un' unica soluzione, sia della durata di regola, a tempo indeterminato (laddove nel contratto di locazione, la durata non può eccedere i trent' anni).

21. Ci si potrebbe chiedere, a questo punto, se la licenza in generale, e quella open source in particolare, possano essere utilmente inquadrate, nel nostro ordinamento, come licenza a titolo gratuito. La qualificazione della licenza d'uso come contratto atipico pone dei problemi in ordine alla possibilità delle parti contraenti di strutturarne la causa secondo il modello gratuito.

In questo quadro generale, s'inserisce la sentenza del Tribunale di Palermo del 29 maggio 1997, la quale ammette, seppur con argomentazioni contraddittorie, la possibilità di stipulare una licenza d'uso a titolo gratuito. Afferma, infatti, il tribunale, che l'atto con cui il creatore dell'opera dell'ingegno (software) cede gratuitamente ad un terzo la facoltà d'uso del bene immateriale, riservandosene la titolarità, è qualificabile come licenza d'uso, anche nel caso in cui sia utilizzata "strumentalmente" per realizzare un intento liberale. Innanzitutto, tale sentenza non arriva ad una chiara qualificazione giuridica della fattispecie contrattuale. Se da un lato, infatti, il giudice qualifica la licenza d'uso come contratto tipico (non fornendo, peraltro, alcun riferimento legislativo), dall'altro ne riconosce la neutralità della causa, facendo propendere per l'atipicità e la sua riconduzione all'interno della sfera d'applicabilità dell'art. 1322 comma 2 c.c.. Il problema è, in generale, quello di accertare se l' onerosità sia tratto essenziale della licenza d' uso e quindi se sia ammissibile, accanto alla tradizionale fattispecie onerosa, la licenza d'uso a titolo gratuito. A tale proposito, il Tribunale di Palermo, nella sentenza citata, afferma che la licenza d'uso di software si configura normalmente quale contratto lucrativo; facendo riferimento alla rinunzia al compenso e collegando tale rinunzia alla licenza d'uso, il giudice finisce, infatti, col riconoscere, implicitamente, la natura onerosa della fattispecie in esame. Su un piano logico, prima che giuridico, infatti, non è possibile rinunciare a

[25] LEONE, *La concessione del software tra licenza e locazione*, in ALPA e ZENO-ZENCOVICH, *I contratti d'informatica. Profili civilistici, tributari e di bilancio*, Milano, 1987, p. 354.

[26] BIN, *L'equilibrio sinallagmatico nei contratti informatici*, in ALPA e ZENO-ZENCOVICH, *I contratti d'informatica*, cit., p.64

qualcosa di cui non si sia già titolari. Il problema dell'ammissibilità della licenza d' uso a titolo gratuito s' inserisce, a questo punto, nell' ambito della più generale questione dell' ammissibilità, nel nostro ordinamento, dei contratti gratuiti atipici. A norma dell'art. 1322 c.c., le parti contraenti non solo sono lasciate libere di determinare il contenuto del contratto, ma possono anche concludere contratti atipici (" che non appartengono ai tipi aventi una disciplina particolare") purché siano diretti a realizzare interessi meritevoli di tutela secondo l' ordinamento giuridico. L' unico limite all'autonomia contrattuale delle parti è, quindi, costituito dalla meritevolezza degli interessi perseguiti. Tale limite si traduce, sostanzialmente, nell'accertamento, svolto dall'autorità giudiziaria, della sussistenza o meno, nei negozi atipici di volta in volta posti in essere dalle parti private, di un interesse meritevole di tutela. In questa prospettiva, la questione dell'ammissibilità dei contratti gratuiti atipici si traduce nella necessità di individuare, per ciascuno di essi, una causa idonea a giustificare giuridicamente l' atto di autonomia privata. L' art. 1322 non distingue, tuttavia, nell'ambito dei contratti atipici, tra quelli a titolo oneroso e quelli a titolo gratuito. La dottrina prevalente[27] è orientata ad ammettere la possibilità di contratti atipici gratuiti, stante la generale formulazione della norma di riferimento, appunto l'art. 1322, applicabile a qualsiasi negozio, a prescindere dalla suo carattere gratuito o oneroso. Una volta ammessa l'astratta configurabilità del contratto gratuito atipico, si pone il problema di individuarne i limiti d'ammissibilità. Al fine di superare il giudizio di meritevolezza di cui all'art. 1322, è necessario che il negozio atipico gratuito posto in essere dalle parti contraenti sia teso a realizzare l interesse patrimoniale[28] di chi sopporta il sacrificio, a condizione che a questo si faccia riferimento nel contesto del negozio[29]. Nel negozio gratuito atipico, quindi, lo spostamento patrimoniale a favore del beneficiario trova la sua giustificazione, sotto il profilo causale, in un vantaggio economicamente valutabile, dell'autore del negozio. E proprio la diversa molla causale è il tratto distintivo tra la figura negoziale considerata e l'istituto della donazione: mentre in tema di donazione, l'arricchimento del destinatario è ammesso dall'ordinamento giuridico sulla base di un interesse di carattere non patrimoniale, nel primo caso lo spostamento presuppone un interesse patrimoniale a capo del disponente. Al fine di porre in essere un negozio gratuito atipico, non sarà, quindi, richiesta alle parti l'adozione dell'atto pubblico (art. 782 c.c.). Questo è necessario, per legge, solo quando lo spostamento patrimoniale è motivato da spirito di liberalità, giacché, in questo caso, sorge la necessità di tutelare la sfera giuridico-patrimoniale del disponente da attribuzioni non sufficientemente ponderate. Lo spostamento patrimoniale risulta, quindi, svincolato da condizionamenti formali quando causalmente giustificato attraverso il collegamento ad un interesse di carattere patrimoniale del disponente. Nei confronti dei contratti atipici gratuiti, la posizione della giurisprudenza è diversa a seconda che si tratti di prestazioni di facere ovvero di dare. Nel primo caso, l'orientamento prevalente è nel senso di ammettere la possibilità per i privati di concludere contratti atipici gratuiti[30]. Al contrario, nel caso di prestazioni di dare atipiche, la giurisprudenza appare decisa a negarne rilevanza giuridica con conseguente comminatoria di nullità per difetto di causa: tale presa di posizione è indice di una generale diffidenza del nostro ordinamento verso le attribuzioni gratuite. Il criterio seguito per negare rilevanza giuridica a tale tipo di negozio è quello di ricondurre qualsivoglia pattuizione gratuita avente ad oggetto un dare, nell'ambito delle convenzioni a prestazioni corrispettive o, in alternativa, nell' ambito delle liberalità donative. Una volta esclusa la qualificazione giuridica del negozio traslativo gratuito come donazione, per difetto di forma e una volta esclusa pure la convenzione, in quanto contratto a prestazioni corrispettive, il negozio gratuito atipico avente ad oggetto una prestazione di dare risulta viziato di nullità per la mancata indicazione della controprestazione[31]. Si può, quindi, concludere che la qualificazione della licenza d'uso come contratto atipico non precluderebbe, quindi, di per sé, la possibilità di ammetterne la configurazione secondo il modello gratuito ove fosse possibile individuare un interesse patrimoniale del disponente.

A tal proposito, tuttavia, il tribunale parla d'intento liberale del cedente, escludendo quindi la presenza di un interesse patrimoniale. Il giudice, inoltre riconosce come "strumentale" l'uso della licenza d'uso, suggerendo, in tal modo, la qualificazione della fattispecie come donazione indiretta. Si dovrebbe, quindi, parlare, nel caso considerato, di una donazione indiretta nella quale il negozio mezzo atipico e gratuito, sarebbe nullo per difetto di causa giacché mancante del fondamentale interesse patrimoniale. La decisione, pur nella sua incoerenza, conferma il dubbio che, di fronte ad una licenza d'uso gratuita, le categorie giuridiche tradizionali rimangano inadeguate. La sua riconduzione agli usuali schemi che regolano la circolazione dei beni in via definitiva (vendita, donazione, ecc.) o temporanea (cessione in godimento) deve confrontarsi, infatti, con la natura del bene che ne forma l'oggetto, il software: suscettibile di un infinito

[27] CARINGELLA, *Alla ricerca della causa nei contratti gratuiti atipici*, in *Foro it.,* 1993, I, 150
[28] App. Roma, 19 gennaio 1967, in *Foro it*., Rep. 1967, voce *Obbligazioni e contratti*, n. 63.
[29] Cass. 21 giugno 1965, n. 1299, in Giur. It., 1967, I, 1, 214.
[30] Vedi Cassazione 22 gennaio 1976 n. 185, in Foro it. 1976, I, 618; Cass. 14 settembre 1976 n. 3150, in Foro it., 1977, I, 1988.
[31] Vedi Cassazione, 20 novembre 1992, in *Foro it*. 1993, I, 1506

godimento senza esaurimento o distruzione dell'idea, si dimostra tuttavia idoneo a presentarsi come bene-valore in sé tutte le volte che ne sia ceduto l' uso.

L' inadeguatezza degli schemi tradizionali può essere, quindi, ricondotta alla peculiare natura del bene "software"; sebbene commerciabile attraverso supporti fisici, questo rimane, infatti un bene immateriale: come tale, le sue potenzialità non possono essere totalmente influenzate dal supporto, la cui incidenza deve rimanere limitata al buon funzionamento del prodotto. Il software conserva, infatti, i caratteri propri dell'opera dell'ingegno: proprio a tali caratteri l'interprete dovrà guardare al fine di individuare gli strumenti idonei alla sua circolazione nel sistema economico.

22. Tratto fondamentale delle licenze di software libero è l'assenza totale di qualunque tipo di garanzia per l'utilizzatore: si tratta di un compromesso ragionevole, dal momento che l'autore non riceve del programma una fonte d'entrata sufficiente per sostenere un'assicurazione sulle responsabilità ed eventuali spese legali. Al riguardo è utile guardare alla formulazione, completa ed esauriente della GPL, la quale, al paragrafo 11, prevede che: " 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".

La questione della responsabilità contrattuale viene in evidenza, all'interno della nostra giurisprudenza[32], con riguardo al contratto di fornitura di sistema informatico: questo è stato definito come un contratto atipico avente ad oggetto la fornitura inscindibile di un software personalizzato e di un hardware compatibile, con la previsione di un'obbligazione di risultato.

In materia, riveste una certa importanza la sentenza del 13 marzo 1993 pronunciata dal tribunale di Torino; con la decisione in esame, il giudice ha risolto il problema della qualificazione del contratto atipico di fornitura di sistema informatico, ricorrendo allo schema dell'unico contratto misto[33], nonostante la possibilità di ricorrere, considerata la pluralità di negozi, allo schema del collegamento tra contratti. Sebbene il regolamento contrattuale sia, solitamente, redatto in un unico testo - anche se nulla vieta di interpretare unitariamente una pluralità di testi[34], come succede con la decisione in esame-, nel contratto misto le prestazioni pattuite sono, di regola, plurime e riconducibili ad una pluralità di schemi contrattuali tipici[35] fusi in un' unica causa per la realizzazione di un interesse unitario sul piano pratico-economico. Il tribunale si è espresso a favore dell'unitarietà della vicenda contrattuale complessiva: la fornitura di un elaboratore elettronico con software personalizzato, a prescindere dalla terminologia utilizzata nei documenti, non consente " la qualificazione atomistica delle singole pattuizioni, ma configura un contratto unitario". La scelta di qualificare il contratto di fornitura di un sistema informatico, indipendentemente dalla pluralità documentale degli atti negoziali, come un unico contratto misto si rivela decisiva, ai fini della tutela dell'utente finale, soprattutto nel caso d'inadempimento del fornitore del sistema. La rilevanza dell'inadempimento, ex art. 1455, dovrà, infatti, essere valutata, in virtù dell'unicità del contratto, in un'ottica unitaria. Con la decisione in esame, il tribunale di Torino si spinge, quindi, ben oltre la qualificazione del contratto di fornitura di sistema informatico, affermando che l'obbligazione scaturente dal contratto di fornitura di un sistema informatico ad hoc deve considerarsi "di risultato": proprio, per così dire, la personalizzazione del sistema implica una specifica garanzia del soddisfacimento delle esigenze personali del cliente. Il configurarsi di un'obbligazione di risultato in capo al fornitore fa sì che il rimedio della risoluzione per inadempimento possa essere invocato dall'utente, anche in caso di mancato raggiungimento del risultato utile promesso. La semplice rispondenza astratta dei beni informatici forniti alle specifiche tecniche rilasciate dal fornitore non sarà più sufficiente ad evitare la risoluzione per inadempimento, istituto che risulta, in ultima analisi, rafforzato dalla decisione in esame. Proprio a causa dell'utilità pattuita, non è possibile inquadrare la vicenda negoziale in una semplice licenza d'uso: tale schema ricorrerebbe se si fosse alla presenza del c.d. software "standard" immesso in serie sul mercato.

Con la sentenza n. 2661 del 22 marzo 1999, anche la Corte di Cassazione si è espressa, per la prima volta, sulla qualificazione giuridica di un contratto di fornitura di un sistema informatico, unitamente ad un contratto d'assistenza tecnica indispensabile per il buon funzionamento dello stesso. Confermando l'orientamento tracciato dal Tribunale di Torino, la Suprema Corte considera le predette figure negoziali alla stregua di un unico contratto misto, caratterizzato dal concorso di due cause distinte, quella di trasferire il

[32] Trib. Milano, 20 ottobre 1988, in *Dir. Inf.* 1989, p.549; trib. Rovereto, 28 febbraio 1985, in *Dir. Inf.* 1986, p. 590.
[33] DE NOVA, *Risoluzione dei contratti di fornitura di sistemi informatici*, in *Contratti*, 4, 1993, 440 ss.
[34] BIN, Op. cit. 71.
[35] Nel caso di contratto complesso, che si pone in un rapporto di genus a species nei confronti del contratto misto, le prestazioni pattuite sono riconducibili a una pluralità di schemi contrattuali atipici fusi in un' unica causa.

bene contro una somma di denaro, tipica della compravendita e quella di compiere e, rispettivamente, ricevere, un'opera, tipica dell'appalto. Secondo la Corte, infatti, due sono le obbligazioni assunte da chi s' impegna a fornire un prodotto software: da una parte, la fornitura di un sistema informatico completo va configurata come un contratto di compravendita; dall'altra, la prestazione, per un certo lasso di tempo, dell'assistenza tecnica necessaria a dare esecuzione alla promessa di garantire il buon funzionamento del sistema, deve essere riportata allo schema del contratto d'appalto. La Corte ritiene prevalente la disciplina del contratto di compravendita, caratterizzato dall'assunzione, da parte del venditore, dell'obbligo di garantire il buon funzionamento del prodotto, quindi di assicurare un determinato risultato[36]. Di conseguenza, chi fornisce un prodotto software, non può esimersi da responsabilità in caso d'inadempimento dell'obbligo di risultato implicito nel contratto di fornitura di in sistema informatico. Una volta verificato il cattivo funzionamento del prodotto,il giudice dovrà stabilire se tale disfunzione è da attribuirsi all'utente o al fornitore: mentre al primo è sufficiente indicare cattivo funzionamento, sarà l'azienda produttrice di software a dover provare che il cattivo funzionamento non è ad essa imputabile. Nel caso in cui il fornitore non riesca a raggiungere tale prova, l'utente può legittimamente richiedere la risoluzione del contratto a norma dell' art. 1453 c.c., non rilevando che il mancato funzionamento dipenda da vizi della cosa venduta o da incapacità del venditore di farla funzionare.

Dal contratto di fornitura di un sistema informatico completo, si differenzia il contratto di sviluppo di software, nel quale il programma applicativo è appositamente preparato, su ordinazione dell'utente ed in conformità delle sue esigenze. La dottrina che si è occupata del problema[37] ha ritenuto che il contratto di sviluppo software sia configurabile come appalto se chi fornisce il servizio è un imprenditore che, ex art. 1655 c.c., si obbliga allo sviluppo del programma dietro un corrispettivo in denaro, organizzando personalmente i mezzi necessari, gestendo tale servizio a proprio rischio e obbligandosi a raggiungere un risultato. Ricorre, invece, lo schema del contratto d'opera intellettuale, stante l'impiego d'intelligenza e preparazione tecnica in misura prevalente rispetto al lavoro manuale, quando l'individuo si obbliga a compiere, contro un corrispettivo in denaro, un'opera o un servizio con lavoro prevalentemente proprio e senza vincoli di subordinazione nei confronti del committente (art. 2222). Solo nel caso in cui si configuri un contratto d'appalto, ricoprendo chi fornisce il software la figura d'imprenditore, sarà configurabile in capo al medesimo soggetto un obbligo di assicurare il buon funzionamento del programma. L'obbligazione nascente dal contratto d'appalto è, infatti, un'obbligazione di risultato. Di conseguenza lo sviluppatore risponderà contrattualmente del cattivo funzionamento del software davanti all'utilizzatore. L'obbligazione nascente da un contratto d'opera intellettuale, invece, è un'obbligazione di mezzi. In questo caso, a norma dell'art. 2236 c.c., i diritti del committente, in caso di cattivo funzionamento del software, si riducono alla tutela concessa nei casi in cui sia stata promessa un'obbligazione di mezzi e non un'obbligazione di risultato. Il prestatore d'opera, in altre parole, non risponde dei danni, se non in caso di dolo o colpa grave.

23. Affermando l'assenza di qualunque garanzia, la GPL, al paragrafo 12, stabilisce che: "Né il detentore del copyright, né le 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". La GPL si preoccupa, quindi, di precisare che nella nozione di danno sono inclusi i danni generici, speciali o incidentali, come pure i danni derivanti dall'uso o dall'impossibilità di usare il programma; ciò comprende, senza tuttavia limitarsi a questo, la perdita e la corruzione di dati, le perdite sostenute dall'acquirente o da terzi e l'incapacità del programma ad interagire con altri programmi, anche se il detentore o altre parti sono state avvisate della possibilità di questi danni. Si deve, quindi, desumere che, nel caso in cui si verificano dei danni derivanti da difetto del programma, non troverà applicazione la specifica disciplina dettata in tema di responsabilità extracontrattuale, connessa al danno derivante dal difetto del prodotto (in questo caso il software).

In materia, la normativa di riferimento è costituita dalla Direttiva della Comunità Economica Europea adottata il 25 luglio 1985 per l'armonizzazione delle legislazioni in materia di responsabilità da prodotto difettoso, recepita in Italia dal d.p.r. 224/1988. L'applicabilità al software dei principi della responsabilità da prodotto, così come definiti dalla Direttiva Comunitaria, applicata in Italia, ma anche, come vedremo, dalla giurisprudenza statunitense, incontra, tuttavia un ostacolo preliminare nella difficoltà di considerare il software alla stregua della nozione di prodotto così come definito nelle fonti predette. Parte della dottrina[38], nel risolvere il problema della natura giuridica del fenomeno, ha finito per classificare il software quale "servizio": ha, di conseguenza, escluso l'applicabilità della Direttiva, sostenendo che, quando si parla di

[36] Anche nel caso in cui la Corte avesse considerato prevalente la disciplina del contratto d'appalto, l'appaltatore sarebbe stato gravato dello stesso obbligo.

[37] MOSCATI, *L'appalto di sistemi*, in ALPA e ZENO-ZENCOVICH, *I contratti d'informatica*, cit. p. 211; BONAZZI e TRIBERTI, Guida *ai contratti d'informatica, Vol. II*, Milano 1985; LANZILLO*, I contratti di fornitura di elaboratori elettronici*, in ALPA e ZENO-ZENCOVICH, *op. cit.,* p. 349.

[38] ALPA, *Computer e responsabilità civile*, Milano 1985, p. 69.

responsabilità da prodotto, si deve fare esclusivo riferimento al difetto di "beni mobili"[39]. Tale limitazione è stata, tuttavia, superata da un'altra parte della dottrina[40]: se la fornitura di programmi per elaboratore può essere considerata alla stregua di un servizio, il software, in quanto tale, può, a ragione, essere considerato alla stregua di un prodotto, in quanto bene che non si esaurisce con la prima utilizzazione, ma che comporta una sua vita autonoma oltre l' uso che ne faccia l' utente.

La questione dell'applicabilità della Direttiva comunitaria non deve, tuttavia, essere sopravvalutata: i principi in materia di responsabilità da prodotto, enunciati in sede comunitaria, non si allontanano, infatti, drasticamente, da principi generali enunciati dal nostro ordinamento interno (art. 2043)[41]. Il d.p.r. 224/1988 ha recepito la direttiva comunitaria in materia di responsabilità da prodotto difettoso, adeguandosi ai principi fondamentali, in essa affermati: questi sono essenzialmente due e riguardano la definizione del difetto e le ipotesi di esonero del produttore. In particolare, l'art. 5 del d.p.r. 224/1988 sancisce il principio generale del difetto di fabbricazione, secondo il quale il prodotto si considera difettoso qualora non offra la sicurezza offerta normalmente dagli altri esemplari della medesima serie (e che quindi ci si potrebbe legittimamente attendere); assume, poi, particolare rilievo l'art.6 il quale, oltre ad escludere la responsabilità nel caso in cui manchi il requisito dell'immissione in commercio, o allorché il difetto non esista al momento in cui il produttore ha messo il prodotto in circolazione, o non ha fabbricato il prodotto per la vendita né lo ha fabbricato o distribuito nell' esercizio di un' attività professionale, stabilisce il principio fondamentale secondo cui è esclusa la responsabilità del fornitore "se lo stato delle conoscenze scientifiche e tecniche, al momento in cui il produttore ha messo in circolazione il prodotto, non permetteva ancora di considerare il prodotto come difettoso". Sono queste le norme cui fare riferimento al fine di stabilire la natura della responsabilità prevista dalla Direttiva Comunitaria e, di conseguenza, dal d.p.r. 224/1988, e per verificare se e in quale misura tale responsabilità si discosti dai principi generali previsti, per l'imposizione dell'obbligazione risarcitoria nell'ipotesi di responsabilità extracontrattuale, dal nostro ordinamento giuridico. In materia di responsabilità dell'imprenditore, il mancato rispetto dello stato dell'arte, quindi l'omessa osservanza degli standard raggiunti dal progresso scientifico e tecnologici, non può essere riferito a un generico concetto di buona fede, ma deve essere identificato nell' omissione di quei requisiti di professionalità relativi all' attività imprenditoriale svolta. Si richiede, quindi, all'imprenditore di conformarsi agli standards raggiunti dal progresso scientifico e tecnologico, poiché, solo in questo caso, egli può affermare di aver adempiuto agli obblighi di diligenza riferiti al suo status: la Direttiva comunitaria, così come il d.p.r. 224 configurano, di conseguenza, una responsabilità per colpa professionale[42]. A ben guardare, il tipo di responsabilità così delineata non si discosta, sostanzialmente, dai principi tradizionali della responsabilità per colpa, di una responsabilità intesa, in altre parole, come sanzione per punire il responsabile che ha omesso di adottare le cautele imposte dallo stato dell'arte, e per prevenire danni dello stesso tipo. Una responsabilità così configurata si allontana, perciò, dall'ipotesi di una responsabilità obiettiva di carattere essenzialmente risarcitorio. Il principio d'imputazione dell'obbligazione risarcitoria fatto proprio dalla Direttiva, finisce, così, per non discostarsi drasticamente dai principi fissati dall'art. 2043 c.c. in materia di fatti illeciti. Presupposto dell'imposizione dell'obbligazione risarcitoria è la presenza dell'elemento del dolo o della colpa in capo a chi commette il fatto illecito: l'art. 2043 c.c. non si limita, cioè, a riferirsi alla mancanza di buona fede, vale a dire alla diligenza media del buon padre di famiglia. A questo punto, è utile precisare come, nel nostro ordinamento, la nozione di colpa, nei fatti illeciti extracontrattuali, è fatta derivare da principi di diritto comune contenuti nel codice penale: in particolare, a norma dell'art. 43 c.p., la colpa consisterebbe in una negligenza, imprudenza, imperizia o inosservanza di specifiche regole di condotta. Ecco quindi che la colpa viene in rilievo, nel caso considerato, come una forma d'imperizia: si traduce, cioè nel mancato rispetto dello standard di professionalità imposto all' imprenditore nell' esercizio dell' attività svolta.

La diligenza di carattere professionale s'identifica, quindi, nel rispetto della prudenza, della perizia e dell'osservanza delle leggi, regolamenti, ordini, discipline la cui violazione è sanzionata dall'art. 43 c.p., attualmente individuato come criterio unitario della colpa derivante da atto illecito in tutte le ipotesi di responsabilità, penale e civile[43]. Tale impostazione trova conferma nell'attenta lettura dell'art. 1176 c.c. in materia d'adempimento delle obbligazioni, ma anche nell'estensione di tali principi, attuata dalla

[39] In particolare, tale orientamento dottrinale, ricomprendendo il software nella categoria dei "servizi", ha escluso l' applicabilità della Direttiva comunitaria, sulla base dell' immediata consumabilità, nonché la natura contrattuale del rapporto fra fornitore e utente.

[40] *TROIANO, La responsabilità per prodotti difettosi,* in Le Nuove Leggi d'Italia Commentate, 1989, p. 510

[41] ARCADU e RINALDI BACCELLI, *Prodotto difettoso e responsabilità per danni derivanti dall'esercizio del software*, in *Rivista del diritto commerciale e del diritto generale delle obbligazioni*, 1992, I, p. 91 ss.

[42] ARCADU e RINALDI BACCELLI, in *op. cit.*, p. 112.

[43] Cass., 29 luglio 1949 n. 2035, in *Rep. Foro it.*, 1949, voce *Responsabilità Civile*, n. 51; Cass., 7 marzo 1952 n. 611, in *Rep Foro it.*, 1952, voce *Infortuni sul lavoro e malattie professionali*, n. 258; Cass., 23 gennaio 1954 n. 161, in *Rep. Foro it.*, 1954 voce *Responsabilità Civile*, c.2227, n. 59.

giurisprudenza, all'ambito della responsabilità extracontrattuale in genere e di responsabilità da prodotto difettoso in particolare. Ai fini dell'adempimento dell'obbligazione l'art. 1176 richiede, infatti, in caso d'attività professionale, l'impiego della diligenza riferita alla particolare attività svolta: il modello di riferimento non è più l'uomo medio[44], ma il professionista medio. Non bisogna, inoltre, dimenticare che la Cassazione, con la sentenza n. 4004 del 21 ottobre 1957[45] ha sancito che al produttore devono essere applicati i principi della responsabilità professionale di cui al comma 2 dell'art. 1176, applicabile sia in materia contrattuale che extracontrattuale: ciò equivale ad affermare che il produttore deve rispettare il progresso scientifico e tecnologico, vale a dire conformarsi allo stato dell'arte.

A questo punto, la distinzione tra responsabilità contrattuale e responsabilità extracontrattuale non può che essere ricercata nella diversa distribuzione dell'onere della prova: mentre nella seconda ipotesi il danneggiato è chiamato a provare la colpa del responsabile, nel primo caso dovrà provare soltanto l'inadempimento, restando a carico del responsabile la prova della non imputabilità della causa. Ribaltando tale impostazione, la Direttiva comunitaria stabilisce, all'art. 8 che il danneggiato deve provare il danno, il difetto e la connessione causale tra difetto e danno (l'elemento oggettivo della responsabilità), laddove sul produttore incombe l'onere di provare i fatti che possono escludere la responsabilità. In particolare, il responsabile dovrà provare d'aver adottato le misure necessarie per adempiere all'obbligazione o evitare il danno, avuto riguardo della situazione concreta nell' ambito della quale il danno è stato prodotto[46]. La diligenza e la cura devono essere verificate con riguardo alla natura dell'attività svolta: sarà quindi rappresentata dalla diligenza professionale del buon produttore secondo lo sviluppo scientifico e tecnologico. Sollevando il danneggiato dall'onere di dimostrare la colpa del produttore, vale a dire l'elemento soggettivo della responsabilità, la Direttiva sembrerebbe prospettare per il responsabile una situazione deteriore rispetto a quella prevista dall'art. 2043: tuttavia, non bisogna dimenticare che, nel 1964, la Cassazione con la sentenza n. 1270 del 25 maggio 1964[47], ha ritenuto applicabile la presunzione di colpa del produttore nell' ipotesi di responsabilità da prodotto.

Alla luce di quanto detto, possiamo affermare che sia la Direttiva comunitaria che, di conseguenza il d.p.r. del 1988, si esimono dall'accogliere il principio della responsabilità obiettiva del fornitore (esclusa l'ipotesi di difetto di fabbricazione), limitandosi ad imporre soltanto l'inversione della prova riguardo all'impiego della cura e della perizia connessa al rispetto della professionalità. Una volta ammesso che il software può essere oggetto della disciplina dettata dalla Direttiva e comunque, una volta appurato che i principi dettati in sede comunitaria non si discostano poi dai principi generali sanciti dal nostro ordinamento in tema di responsabilità extracontrattuale, i problema da risolvere diventano essenzialmente due

- Verificare se e in quale misura la fornitura di software soddisfi quei requisiti di produzione da cui dipende, nell'ambito della responsabilità extracontrattuale, l'applicazione di quell'indirizzo giurisprudenziale che afferma l'inversione dell'onere della prova a carico del responsabile;
- Definire i requisiti di professionalità e perizia che consentano al fornitore di esonerarsi da qualsiasi responsabilità, nonostante il verificarsi del danno.

Nel momento in cui la responsabilità extracontrattuale concerne la produzione di beni destinati al mercato, l'onere della prova a carico del danneggiato riguarda l'elemento oggettivo della responsabilità (esistenza del danno, del difetto, del nesso di causalità), mentre sarà il responsabile a dover dimostrare di aver adottato tutte le misure necessarie per evitare il danno con riferimento all'attività professionale svolta. Tuttavia, in tale contesto, è necessario precisare che per il difetto di fabbricazione, che si verifica, a norma dell' art. 5 d.p.r., nel momento in cui un esemplare non rispetti i requisiti di sicurezza degli esemplari della stessa serie, il difetto risulta in re ipsa e non deve essere dimostrato. E' questa la vera novità della tutela raggiunta dal consumatore nella produzione di massa che impone al produttore la responsabilità per difetto di fabbricazione: è la responsabilità professionale nel rispetto del progresso scientifico e tecnologico che impone di risarcire i danni provocati dal difetto di fabbricazione, giacché la professionalità impone di verificare la corretta esecuzione del processo produttivo. Analizzando la realtà industriale, si può notare come il programma per elaboratore sia, nella maggior parte dei casi, il risultato di un accordo tra imprenditore e utente, secondo le diverse e specifiche esigenze di quest'ultimo; ne consegue che maggiore è la sofisticazione del software, in relazione alle specifiche esigenze del singolo, tanto minore appare la probabilità della produzione in serie, che costituisce la fondamentale premessa per l'applicabilità dei principi relativi al difetto di fabbricazione, cui la Direttiva Comunitaria, come applicata in Italia, riconnettono una vera e propria responsabilità obiettiva. Pur potendosi configurare, in astratto, il requisito dell'immissione in commercio del prodotto difettoso, che consentirebbe l'applicabilità della disciplina generale in materia di danno da prodotto difettoso, ben difficilmente potrebbe configurarsi il difetto di fabbricazione che

[44] Al comma 1, l'art 1176 richiede, ai fini dell'adempimento dell' obbligazione, l' impiego della diligenza del buon padre di famiglia.
[45] Vedi Giur. It., 1958, I, 1, 187
[46] Cass. 9 luglio 1984 n. 4020
[47] Integralmente riportata in ALPA e MESSONE, La responsabilità del produttore, Milano, 1980, p. 26

presuppone il prodotto di serie: tale rilevante ipotesi risulta, quindi, esclusa dall'ambito d'applicabilità della disciplina relativa alla responsabilità per difetto di fabbricazione, configurandosi, al suo posto, un difetto di progettazione o di errata informazione circa l' uso del servizio[48]. La Direttiva comunitaria in tema di responsabilità da prodotto non prevede la distinzione tra dei diversi tipi di difetto del prodotto. E' stata la prassi giurisprudenziale statunitense, seguita poi dagli altri paesi, a distinguere per prima tra:

a) Difetto di fabbricazione, che investe il vizio del singolo esemplare rispetto agli altri della stessa serie.

b) Difetto di progettazione, che investe tutti gli esemplari prodotti.

c) Difetto di informazione che si verifica quando il produttore omette, in tutto o in parte, di fornire all' utente le informazioni necessarie circa l' uso del prodotto.

Al riguardo, bisogna notare come il legislatore italiano abbia adottato una scelta peculiare, introducendo la distinzione del difetto di fabbricazione. Va, in ogni caso, sottolineato che, le riserve circa l'applicabilità della responsabilità da prodotto difettoso, e dei criteri connessi al difetto di fabbricazione, non comportano l'immunità del fornitore di software nei confronti di terzi, ma soltanto l'applicabilità dei criteri generali derivanti dal diritto comune in materia di fatti illeciti (artt. 2043 e ss. c.c.). Escluso, quindi, il ricorrere della responsabilità per difetto di fabbricazione, bisogna chiedersi se tale responsabilità sia configurabile nel caso di difetto d'informazione o di progettazione, e quindi se, in questi casi, trovi applicazione il criterio dell'inversione dell'onere della prova. La risposta sembra essere affermativa, considerata la professionalità dell'attività svolta e la generale diffusione delle tecniche di computerizzazione in tutto il mondo. In definitiva, anche nell'ambito dei programmi per elaboratore si sono andati formando standards di carattere professionale (basti pensare alla loro brevettabilità che si traduce, in definitiva, nell'omologazione di un determinato procedimento produttivo), cui l'imprenditore deve conformarsi nello svolgimento della propria attività. Per quel che riguarda, poi, la definizione dei canoni di professionalità e perizia, che consentono al produttore di esonerarsi da responsabilità, si ritiene[49] che la mancanza di una legislazione che definisca le caratteristiche del software, od ometta di prevedere specifiche abilitazioni professionali per gli addetti all'elaborazione dei programmi, non può pregiudicare la possibilità di configurare il canone di diligenza del fornitore del software con riguardo alla natura dell'attività svolta. In effetti, anche se tale mancanza rende la configurazione del predetto canone molto più complessa, la professionalità dell' imprenditore deve conformarsi, anche in questo caso, allo sviluppo delle conoscenze scientifiche e tecnologiche, vale a dire agli standards industriali.

Passando brevemente ad analizzare la situazione statunitense, in materia di responsabilità da prodotto, la codificazione della responsabilità obiettiva trova riconoscimento nella sezione 402/A del Restatement of the Law of Torts (II) del 1965, la quale stabilisce che chiunque venda un prodotto in condizioni difettose, irragionevolmente pericolose per il consumatore o l'utente è responsabile per il danno fisico che ne derivi all'ultimo consumatore o utente[50]. La giustificazione della responsabilità obiettiva è da, tuttavia, ricercare nella parte (secondo paragrafo) che prevede l'imposizione dell'obbligazione risarcitoria, indipendentemente dalla cura profusa dal produttore nella produzione del prodotto. La giurisprudenza statunitense ha, inoltre, evidenziato come il requisito dell'irragionevolezza del pericolo ingenerato dal prodotto immesso sul mercato finisce per subordinare la nascita dell'obbligazione risarcitoria al mancato rispetto di un parametro che si sostanzia nell'obbligo di conformarsi al progresso scientifico e tecnologico (definito come stato dell'arte), quindi, in definitiva, alla professionalità del costruttore. Proprio il richiamo al parametro della ragionevolezza ripropone il modello tradizionale del rispetto della diligenza, qualificata dall' esercizio dell' attività svolta che comporta l' applicazione dei principi della negligence, cioè della proponibilità, cioè dell' azione proponibile dal terzo danneggiato nella sua persona o nei suoi beni, secondo i principi del common law. Per la giurisprudenza degli Stati Uniti, quindi, nell'ambito dell'action in tort (azione extracontrattuale), i principi della ragionevolezza e della negligence coincidono sia che la domanda venga proposta in base alla responsabilità per colpa, sia che essa sia fondata sulla responsabilità obiettiva. Il canone dell'esercizio della dovuta cura nella progettazione e costruzione del prodotto (rilevante nell'action in negligence) si traduce nella verifica del se egli ha posto in commercio un prodotto irragionevolmente pericoloso (presupposto della strict liability).

Nel corso degli anni settanta si è verificata, tuttavia, nella giurisprudenza statunitense, una significativa inversione di tendenza intesa ad accentuare il carattere obiettivo dell' obbligazione risarcitoria: la giurisprudenza ha, infatti ritenuto non più necessario, ai fini dell' imputazione, il requisito irragionevolezza del pericolo determinato dal prodotto; ha, in questo modo, configurato in capo al produttore una responsabilità che prescindendo dalla ragionevolezza del pericolo stesso, quindi della rispondenza o

[48] ARCADU e RINALDI BACCELLI, *op. cit.*, p. 122.

[49] ARCADU e RINALDI BACCELLI, op. cit.

[50] ai fini della configurabilità della responsabilità in capo al venditore, la norma richiede, altresì, il ricorrere di determinate condizioni: innanzitutto che la vendita del prodotto sia esercitata professionalmente; inoltre che il prodotto raggiunga l' utente o il consumatore privo di modifiche sostanziali rispetto alle condizioni in cui è stato venduto.

meno del prodotto rispetto a determinati standards di professionalità, assumeva inequivoci connotati di carattere obiettivo. Alla tendenza intesa ad inasprire i criteri d'imputazione dell'obbligazione risarcitoria negli USA, nel 1979, ha reagito il potere federale mediante l'emanazione dell'Uniform Product Liability Act che reintroduce, salvo per il difetto di fabbricazione, i principi della responsabilità professionale mediante un'analitica elencazione delle condizioni cui è subordinato il diritto al risarcimento del danno da prodotto difettoso. Reintroducendo il richiamo allo stato dell'arte, l'UPLA ha escluso la responsabilità del produttore in caso d'impossibilità di scoprire il difetto alla luce delle conoscenze scientifiche e tecnologiche.

In conclusione, sia per la legislazione USA che per quell'italiana (che si adegua a quella comunitaria in materia di responsabilità da prodotto) il difetto di fabbricazione costituisce la tipica ipotesi di responsabilità obiettiva. In materia di software, tale ipotesi si verifica tanto più difficilmente quanto più sia complesso il software in relazione alle esigenze del singolo utente. La differenziazione del prodotto non favorisce la realizzazione di prodotti di serie inducendo a configurare ogni software come un prodotto di carattere originale: tale stato di cose, se da un lato non permette la standardizzazione, dall'altro esclude l'automatismo della responsabilità derivante dal difetto di fabbricazione. E' tuttavia configurabile una responsabilità per difetto di progettazione o d'informazione. I criteri d'imputazione della responsabilità, sia per quanto riguarda la determinazione del canone di diligenza e della perizia sul piano sostanziale, sia per quanto attiene alla distribuzione dell'onere della prova non possono prescindere dal rispetto dei criteri della responsabilità professionale, che, per altro verso, consegue all'immissione del bene o del servizio sul mercato, facendo sorgere la necessità di valutare in concreto:

- Il grado di standardizzazione dei sistemi o e dei metodi, anche in relazione alle specifiche richiesta dei singoli utenti (ben potendo un programma prodotto in serie essere adattato secondo le esigenze del singolo individuo).
- La difficoltà e la novità dell'opera, anche con riguardo alle particolari limitazioni di responsabilità previste dall' art. 2236 c.c. in connessione con la prestazione dei professionisti[51].
- La possibilità di adottare criteri di presunzione di colpa, in relazione alle circostanze, così come da tempo stabilito dalla giurisprudenza italiana per il difetto del prodotto in applicazione dell'art. 2043.

24. La possibilità, attribuita all'utilizzatore del programma dalla licenza open source, di copiare, elaborare, modificare e ridistribuire il programma di software libero, consente di guardare all'opera come un insieme di contributi di più soggetti. A questo punto, la domanda è: può un programma software open source essere considerata un'opera collettiva?

La legge italiana nulla dispone in materia di programmi come opere collettive nonostante la Direttiva CEE 91/250, attuata, in Italia, con il d. lgs. 518/1992 (che, ricordiamolo, ha esteso al software la normativa dettata per la protezione delle opere dell'ingegno) consenta agli Stati che prevedono tale istituto – tra cui l' Italia – di disciplinare la materia[52]. Le opere collettive, così come definite dalla legge sul diritto d'autore, costituiscono, infatti, una singolare categoria del sistema italiano, sebbene esse si ricolleghino a quei recueils de different oeuvres, di cui parla la Convenzione Internazionale di Berna all'art. 3 alinea 3.

Per rispondere alla domanda iniziale è, quindi, necessario fare riferimento a quello che la legge sul diritto d'autore stabilisce, in generale, sulle opere collettive. A norma dell'art. 3 l.a., l'opera collettiva, costituita dalla riunione di opere o di parti di opere, è quella che ha carattere di creazione autonoma, come risultato della scelta e del coordinamento di opere varie a un determinato fine letterario, scientifico, didattico, religioso, politico, artistico[53] e ciò indipendentemente dai diritti d' autore sulle opere o parti di opere di cui essa è composta. Si parla, quindi, d'opera collettiva, quando i singoli contributi non si compenetrano reciprocamente, ma restando distinti e autonomi, sono destinati ad una presentazione d'insieme per un fine unitario e organico, che l'opera persegue. Ne deriva che il software open source può essere qualificato come opera collettiva, solo quando sia possibile individuare singolarmente l'apporto di ciascuno. Stando così le cose, l'omissione del nostro legislatore parrebbe, a prima vista, da condividere: sembra, infatti, difficile

[51] Prevede infatti l' art. 2236 c.c. che "se la prestazione implica la soluzione di problemi tecnici di speciale difficoltà, il prestatore d' opera non risponde dei danni, se non in caso di dolo o colpa grave.

[52] Il d.lgs n. 518/1992 non riproduce neppure il par. 2 dell'art. 2 della Direttiva, secondo il quale "allorché un programma per elaboratore è creato congiuntamente da un gruppo di persone fisiche, esse sono congiuntamente titolari dei diritti esclusivi". Il disposto comunitario si può, tuttavia, ritenere rispettato, dal momento che la legge n. 633/1941 già conteneva una norma sulle opere in comunione (art. 10)

[53] A proposito della finalità perseguita, potrebbe a prima vista sembrare che il programma per elaboratore non persegua alcuno degli scopi elencati dall'art. 3. A prescindere dalla tassatività o meno di tale elencazione, non bisogna dimenticare che con la legge 518/1992 ha sancito il principio di protezione del software quale opera letteraria. Lo scopo a cui è destinato può, quindi normativamente, essere fatto rientrare nel fine letterario.

immaginare che un programma per elaboratore, sia esso open source o meno, possa riunire parti di creazioni autonome che non si compenetrino reciprocamente e per le quali può essere sempre distinto l'autore di ognuna di esse. In quest'ottica, perfino il software complesso, per il quale lavorino diversi gruppi di programmatori, sembra destinato ad essere ricomposto in una fase finale e ad essere utilizzato in maniera tale che le diverse parti interagiscano fra loro. A mio parere, la compenetrazione fra programmi per elaboratore non deve essere identificata con l'interazione, essendo questa, in ogni caso, caratteristica costante di un'opera collettiva in cui le varie componenti sono chiamate a lavorare insieme. La compenetrazione non deve neanche essere identificata con l'utilizzo, da parte di un programma, di codice proveniente da altri programmi, giacchè, nella maggior parte dei casi, tale codice viene rivisitato per sfociare, poi, in un'elaborazione creativa. In definitiva, occorre verificare se l'opera, nel suo insieme, possa essere scomposta in tante parti aventi una vita propria e che possano essere attribuiti ad autori distinti. A questo proposito potrebbe essere utile ricordare quanto previsto dalla GPL (sicuramente la licenza legalmente meglio formulata oltre che la più diffusa) in tema di libertà di modifica. Si stabilisce, infatti, in questa sede, che è lecito modificare la propria copia o copie del programma, o parte di esso, creando un'opera basata sul programma, e copiare o distribuire tali modifiche o tale opera purché venga indicato chiaramente nei file che si tratta di copie modificate e la data di ogni modifica (paragrafo 3 lettera a). Tale previsione permette di mantenere ciascuna modifica apportata al programma distinta e scindibile dalle altri parti di cui esso consta, e consente, quindi, di qualificare l' opera risultante come opera collettiva. Il testo della GPL, tuttavia, si spinge oltre: ammette, infatti, che possano sussistere parti identificabili dell'opera modificata che non siano derivate dal programma e che possano essere ragionevolmente considerate lavori indipendenti[54]. Ne consegue che il software open source può essere considerato opera collettiva sia quando consti di parti scindibili che derivino dal programma originale, sia quando le parti da cui è formato non siano derivate dal programma.

L'individuazione degli autonomi e scindibili contributi apportati da ciascun autore, non è però condizione sufficiente per poter parlare di opera collettiva. A norma dell'art. 7 l.a. "è considerato autore dell'opera collettiva nella sua interezza colui che ha diretto e organizzato la sua creazione": sembra quindi che per potersi parlare di opera collettiva, con riguardo al software libero, sia necessaria l'individuazione di una figura, per così dire, leader che coordini il lavoro di una pluralità di contributori. Bisognerebbe chiedersi, allora, cosa succederebbe se tale figura non fosse nitidamente distinguibile: se è vero che i progetti software open source, così come quelli proprietari, sono sviluppati da squadre d'esperti programmatori, organizzati al loro interno secondo gerarchie definite, è anche vero che la possibilità di diffusione e trasmissione dell'opera attraverso la rete, rende difficoltosa l'individuazione del soggetto che dirige.

Una volta individuato, a norma dell'art. 7, colui che organizza e coordina la creazione dell'opera, a tale soggetto spetterà l'esercizio del diritto d'autore. Nel sistema del codice civile (art. 2577) e della legge speciale (art. 12-24) si raggruppano sotto il contenuto di tale diritto, tanto i diritti d'utilizzazione economica dell'opera, quanto i diritti sull'opera, a difesa della personalità dell'autore. In particolare, a norma dell'art. 2577 c.c. all'autore è riconosciuto il diritto esclusivo di pubblicare l'opera e di utilizzarla economicamente in ogni forma e modo nei limiti e per gli effetti fissati dalla legge. In caso di cessione dei diritti d'utilizzazione economica, non perde, in ogni caso, la facoltà rivendicare la paternità dell'opera e di opporsi a qualsiasi deformazione, mutilazione o altra modificazione dell'opera stessa, che possa essere di pregiudizio al suo onore o alla sua reputazione. L' art. 12 l.a., a sua volta, dispone che "l'autore ha il diritto esclusivo di pubblicare l' opera. Ha altresì il diritto esclusivo di utilizzare economicamente l'opera in ogni modo e forma", in particolare, con l'esercizio dei diritti esclusivi (diritto di riproduzione, di diffusione, di distribuzione, d'elaborazione) indicati negli articoli seguenti". All'autore dell'opera collettiva, di conseguenza, la legge speciale riserva il diritto esclusivo di pubblicare l'opera, che sarà quindi precluso agli autori dei singoli contributi.

Il diritto d'inedito, vale a dire la facoltà riconosciuta all'autore di separare l'opera dalla sua sfera intima, se e quando gli sembri opportuno, riveste il carattere di diritto della personalità: non dovrebbe di conseguenza rientrare tra le facoltà d'utilizzazione economica dell'opera. Tuttavia è stato rilevato come, pur rimanendo nella sfera del diritto morale d'autore, il diritto di pubblicazione costituisca il necessario presupposto per l'esercizio delle varie facoltà patrimoniali: è, pertanto, dotato anch'esso di un contenuto patrimoniale, anche se solo in potenza.

Nonostante autore dell'opera collettiva debba considerarsi il soggetto che organizza e dirige la creazione dell'opera, resta ferma, in capo a coloro che hanno realizzato le singole parti del programma, la titolarità del diritto d'autore sulle stesse nonché la facoltà di utilizzare separatamente i rispettivi contributi (art. 38 l.a.).

---

[54] La GPL prevede che se queste parti sono 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 della GPL; la stessa non si applica laddove vengano distribuite separatamente.

Se si esclude il disposto dell'art. 7 comma 2, si può notare che i profili concernenti il diritto morale d'autore non trovano, nella legge speciale, una disciplina esplicita, che deve quindi essere ricostruita in via sistematica. Innanzitutto sembra potersi affermare che, quando un'opera nasca dall'apporto creativo di una pluralità di soggetti, ognuno di questi ha il diritto di essere indicato come autore del proprio contributo. Tale principio non è fissato da una clausola generale, ma si ricava dal collegamento sistematico della disciplina di alcune categorie di opere nate dal contributo di più autori: in particolare, si fa riferimento all'art. 10 l.a., che attribuisce il diritto morale a tutti gli coautori dell'opera in comunione, nonché all'art. 48 l.a., in base al quale i quattro coautori dell'opera cinematografica "hanno diritto che i loro nomi, con l'indicazione della loro qualità professionale e del contributo nell'opera, siano menzionati nella proiezione della pellicola".

L' art. 48 l.a. suggerisce, d'altro canto, un modello suscettibile d'applicazione generale ai fini dell'attribuzione ai vari autori della paternità dei singoli contributi. Questa norma stabilisce, infatti, che ogni coautore sia menzionato mediante l'indicazione del proprio ruolo specifico: in questo modo ognuno dei contributi sarà nominato in base alla propria natura. E' anche vero che ciò avviene anche per l'opera collettiva, giacché l'articolo 7 l.a. attribuisce a chi coordina l'opera collettiva il diritto di esserne qualificato direttore, laddove, al comma 2, attribuisce al collaboratore la paternità del contributo da lui realizzato. Tuttavia, è possibile concludere che, in tutti i casi in cui l'opera nasca dall'apporto creativo di più persone, il sistema della legge d'autore vuole che ognuna di esse sia menzionata, e che tale menzione si accompagni all'indicazione del ruolo specifico avuto nella realizzazione dell'esito complessivo.

Per quel che riguarda le facoltà dei singoli contributori di utilizzare economicamente l'opera, in sede separata è necessario fare subito una considerazione: l'art. 4 l.a. protegge le elaborazioni di carattere creativo dell'opera stessa, quali le modificazioni e le aggiunte che costituiscano un sostanziale rifacimento dell'opera originaria. Ben potrebbe accadere che uno dei molteplici contributi dell'opera collettiva sia costituito dall'elaborazione creativa di un'opera protetta: non è raro, infatti, il caso in cui un programma per elaboratore sia sviluppato sulla base di un software preesistente. Come ho gia avuto modo di precisare[55], l'elaborazione creativa, intendendo con questo termine ogni intervento creativo non limitato all'aggiunta di modiche varianti alla mera riproduzione del programma, diventa, quindi, oggetto d'autonoma protezione da parte della legge sul diritto d'autore, purché sia soddisfatta un'importante condizione: che la tutela accordata non rechi "pregiudizio ai diritti esistenti sull'opera originaria" (art. 4 l.a.). Ciò significa, in ultima analisi, che l'elaborazione del programma è tutelata se creativa, purché autorizzata dal titolare dei diritti sul programma elaborato. Il singolo contributore potrebbe, quindi, avere bisogno del consenso di un terzo per poter sfruttare economicamente la propria opera.

Per quel che riguarda il profilo del diritto morale d'autore, relativamente alle opere derivate, è evidente che l'esito creativo complessivamente considerato non dipende soltanto dall'attività dell'elaboratore, ma è in parte imputabile alla precedente attività creativa dell'autore dell'opera base. Si deve, quindi, concludere che quest'ultimo soggetto ha diritto ad essere menzionato nelle forme d'uso anche sull'opera derivata: in particolare, ha diritto ad essere menzionato come autore dell'opera che ha funto da base per la realizzazione dell'elaborazione creativa.

L'opera collettiva è considerata, dalla legge sul diritto d'autore, nella sua interezza e nelle singole parti. Nell'opera collettiva si hanno, dunque, due distinte attività creatrici: l'una che riguarda i singoli contributi, l'altra che caratterizza l'opera collettiva in quanto tale. La creatività riferita all'organizzazione e alla scelta deve avere gli stessi caratteri richiesti per la singola opera dell'ingegno. La "creatività" richiesta dalla vigente legge per la tutela del diritto d'autore non deve essere identificata con l'originalità in assoluto e neppure con la "novità", concetto proprio dell'invenzione industriale protetta, ma piuttosto con l'individualità della rappresentazione e, da un punto di vista soggettivo, con la "personalità", vale a dire la capacità di sentire e di esprimere in modo personale un'idea. Nel caso in cui all'interno del programma non siano individuabili parti distinte e inscindibili, ma lo stesso sia creato con l'apporto inscindibile di più soggetti, non ci si troverà più davanti a un'opera collettiva, bensì, salvo patto contrario, ad un regime di comunione tra coautori a parità di quote (art. 10 l.a.).

Il programma creato con il contributo inscindibile e indistinguibile di più soggetti è definito, con termine generale, opera in collaborazione in senso proprio: la collaborazione è considerata alla stregua di un "fatto" che si manifesta nella volontaria associazione di più attività creative al fine della creazione, in comune, di un'opera costituente un'unità inscindibile. D'opera in collaborazione si parla, anche, nel caso di contributi distinti e capaci di vita indipendente: si parla in dottrina di "opere composte", indicando con questo termine le opere costituite da contributi creativi, dovuti a diversi autori, che abbiano, in misura più o meno ampia, una loro autonomia e una loro indipendenza artistica, economica e giuridica, ma che, nel momento in cui si presentano combinati insieme, danno luogo ad una nuova opera di carattere unitario risultante dalle varie parti. In questo caso si ammette che la collaborazione possa essere successiva, nel senso che può non essersi manifestata durante il processo creativo dell'opera unitaria. La legge comprende in tale categoria d'opere in collaborazione quelle drammatico-musicali, le composizioni musicali con parole, le

[55] Par. 9

opere coreografiche e pantomimiche (art. 33-37 l.a.), e le opere cinematografiche (art. 44-50), le quali, possono essere ricondotte anche alla categoria delle opere in collaborazione in senso proprio. Escludendo la tassatività dell'elencazione operata dal legislatore speciale, anche i programmi per elaboratore possono essere ricompresi all'interno della categoria delle opere composte.

In ogni caso, l'opera in collaborazione non deve essere confusa con l'opera collettiva poiché, in quest'ultima ipotesi, l'attività creativa, svincolandosi da quella che ha generato i singoli contributi compresi nell'opera collettiva, si concentra sul risultato della scelta e del coordinamento di altre opere; ciò non toglie, naturalmente, che una collaborazione possa manifestarsi anche in tal genere di opere, nel momento in cui, in concreto, la peculiare attività creativa che caratterizza l'opera collettiva promani da più autori.

Come stabilito dall'art. 10 l.a., nel caso di opera creata con il contributo indistinguibile e inscindibile di più soggetti,si applicano le norme che regolano l'istituto della comunione (artt. 1100 e ss. c.c.): si renderà, di conseguenza, necessario il consenso di tutti i contitolari per il trasferimento di uno o più diritti di sfruttamento economico del programma (art. 1108 c.c.), laddove per la concessione dei diritti d'uso si ritiene debbano applicarsi le maggioranze richieste dalla legge, salvo patto contrario. Sempre a norma dell'art. 10 l.a., la difesa del diritto morale d'autore potrà, al contrario, essere sempre esercitata individualmente da ciascun autore.

25. Uno dei problemi potenzialmente più rilevanti in materia di licenze di software libero, è quello relativo alle possibili conseguenze giuridiche derivanti dal mancato rispetto dei termini della licenza da parte del licenziatario. Poniamo l'ipotesi, tra l'altro neanche remota, che un programmatore scarichi da Internet un programma distribuito sotto una licenza open source, apporti una qualche modifica, dopodiché rilasci l'intero programma risultante sotto una licenza non open source. In altre parole, supponiamo che il programmatore renda il programma proprietario: tramite l'utilizzo di una licenza non open source egli potrà, a suo piacimento, limitare la possibilità di riprodurre, modificare e ridistribuire il programma da parte degli utenti successivi. Il problema, a questo punto, è quello di configurare giuridicamente una situazione di questo tipo. In particolare, bisognerà chiedersi se venga in rilievo l'ipotesi di una violazione contrattuale, piuttosto che quella della contraffazione. In entrambi i casi, diventa preliminarmente necessario chiarire a quale tipologia contrattuale la licenza open source debba essere ricondotta. Mi sembra, in questo senso, preferibile la tesi che riconduce la licenza allo schema dei contratti atipici, per il quale il codice non detta, quindi, una disciplina specifica. La disciplina applicabile alla licenza deve quindi essere ricercata, in primo luogo, nel disposto dell'art. 1322 c.c.: in forza di tale norma le parti sono lasciate libere di determinare il contenuto del rapporto giuridico patrimoniale tra loro insorto, entro i limiti imposti dalla legge. Detto ciò, al fine di verificare se vi sia effettivamente stata violazione della volontà delle parti, occorrerà guardare agli accordi intervenuti tra loro. Una distinzione è, a questo punto, necessaria: laddove, infatti, il programma sia stato distribuito sotto i termini della BSD, o di un'altra licenza non restrittiva, non sarà, di regola, configurabile alcuna violazione contrattuale. Questo tipo di licenza consente, infatti, al licenziatario di apportare al programma modifiche proprietarie: ciò significa che il programma, una volta modificato, potrà essere rilasciato sotto una licenza diversa da quella che copriva il programma originale. Nel caso in cui quest' ultimo sia distribuito sotto GPL o altra licenza con copyleft, il discorso si fa differente. Il paragrafo 3 punto b) della GPL ammette, infatti, la possibilità per il licenziatario di apportare delle modifiche al programma; richiede, tuttavia, che ogni opera che sia derivata, in parte o nella sua interezza, dal programma o da parti di esso sia concessa nella sua interezza in licenza gratuita ad ogni terza parte, secondo i termini della stessa GPL. Nel caso in cui, le modifiche apportate dal licenziatario non siano distribuite sotto GPL e, in ogni caso, rese libere, ci si trova, quindi, di fronte ad una violazione degli accordi intervenuti tra le parti. Del resto, la stessa GPL, nella sua accurata formulazione, si preoccupa di precisare che deve essere considerato illecito ogni tentativo di modificare o distribuire il programma in modi diversi da quelli espressamente previsti dalla licenza stessa. Non solo: chiunque ponga in essere condotte non conformi al dettato della licenza, per esempio chiudendo il codice del programma, decade dai diritti previsti dalla licenza stessa. E' chiaro quindi che chi distribuisce il programma modificato sotto una licenza non open source, pone in essere una condotta contrattualmente illecita.

A questo punto, bisogna verificare quali siano i rimedi, posti dall'ordinamento, a tutela dell'autore che vede il suo codice subire delle modifiche proprietarie. Escludendo il ricorso alla domanda di risoluzione per inadempimento, non obbligandosi il licenziatario ad alcuna controprestazione nei confronti dell'autore, l'autore mirerà ad impedire l'uso della licenza. Modificare e distribuire il programma in modo diverso da quelli espressamente previsti dalla licenza è, infatti, comportamento non più coperto da un atto autorizzatorio dell'autore, titolare esclusivo dei diritti d'utilizzazione economica. Mancando un'autorizzazione ad apportare modifiche proprietarie al programma e a distribuire lo stesso sotto una licenza non open source, l'autore sarà, quindi, legittimato ad esperire azione di contraffazione, anche se non sulla base di un diritto proveniente dal diritto d'autore; il meccanismo ostativo trova, infatti, in questo caso, la sua giustificazione nella mancanza del consenso dell'autore in ordine alla chiusura del codice del programma da parte del licenziatario.

Non si potrà, quindi, a mio avviso, parlare di contraffazione diretta: qui non è, infatti, la modifica ad essere, di per sé abusiva, ma l'atto di chiudere il codice. La violazione contrattuale consiste, pertanto, nella chiusura del codice da parte del licenziatario e si concreta nella violazione del contenuto di un diritto esclusivo dell'autore.

I rimedi giuridici a cui l'autore potrà ricorrere saranno quelli apprestati, in sede civilistica, dall'art. 156 l.a.: intendendo impedire la continuazione di una violazione, già avvenuta, un diritto d'utilizzazione economica, l'autore agirà, quindi, in giudizio per ottenere che il suo diritto sia accertato e la violazione interdetta.

Non dimentichiamo che, oltre ad essere titolare di una serie di diritti esclusivi d'utilizzazione economica, l'autore del programma per elaboratore è, altresì, titolare di un diritto morale d'autore sull'opera. Essendo l'opera, sotto questo punto di vista, espressione della personalità dell'autore, si potrebbe pensare al diritto alla paternità come ad un diritto a non vedere snaturata la propria opera. Se la volontà dell'autore, espressa tramite il ricorso ad una licenza open source, è quella di rendere libera il propria programma, la chiusura del codice da parte del licenziatario potrebbe essere interpretata, in ultima analisi, come una violazione del diritto morale dell' autore a non vedere compromesse quelle qualità che rendono l' opera una creazione personale. Tra l'altro, nel caso di successiva commercializzazione dell'opera, la giurisprudenza[56] ammette il ricorrere del periculum in mora necessario per l'adozione di un provvedimento d' urgenza, che, a norma dell'art. 700 c.p.c, inibisca l'ulteriore distribuzione degli esemplari di un' opera che violino il diritto morale d'autore. E', tuttavia necessario, che l'ulteriore distribuzione comporti un aggravamento della situazione lesiva e non sia agevole monetizzare il danno conseguente.

Sempre con riguardo alle misure esperibili dall'autore, e con particolare riguardo al caso in cui il programma possa essere identificato con un' opera collettiva[57], la giurisprudenza ha chiarito che non può essere accolta domanda di sequestro o distruzione degli esemplari in commercio di un' opera quando la stessa non sia sequestrabile, né ex art. 161 comma 2 l.a. trattandosi di opera che risulti dal contributo di più soggetti, né ex art. 700 c.p.c. poiché una simile disposizione urterebbe con il carattere interinale della tutela prevista da questo articolo e con il disposto dell'art. 169 l.a., in tema di residualità di tale provvedimento[58]. L'art. 169 l.a. ammette, infatti, a tutela dei diritti che riguardano la paternità dell'opera, la sanzione della rimozione o della distruzione delle opere solo quando la violazione non possa essere convenientemente riparata mediante l'aggiunta o la soppressione delle indicazioni che si riferiscono alla paternità dell'opera stessa o con altri mezzi di pubblicità.

Quanto alla questione riguardante l'opera collettiva, forma nella quale potrebbe concretizzarsi un programma open source, le ragioni per cui l'art. 161 comma 2 l.a. stabilisce che il sequestro non possa, di regola, venire concesso per le opere che risultano dal contributo di più persone, risiede nella volontà di evitare che un provvedimento così grave possa recare danno a persone estranee alla violazione del diritto d'autore, a causa del quale il sequestro è stato richiesto. Alcuni degli autori dell'opera potrebbero avere, infatti, interesse a vedere la loro opera pubblicata e distribuita. Non a caso, la medesima norma prevede l'inoperatività del divieto quando la violazione del diritto è imputabile a tutti i coautori[59]. Il sequestro potrebbe, quindi, essere concesso nel caso in cui l'opera sia creata, con contributi inscindibili, da più soggetti in comunione[60]. La giurisprudenza ha affermato che l'inapplicabilità del sequestro non preclude, in ogni caso, la possibilità di emanare un provvedimento d'urgenza, ex art. 700c.p.c., di inibitoria di un atto di illecita violazione del diritto d'autore[61]. A ben vedere, tuttavia, l'inibitoria, come il sequestro, appare idonea a ledere gli autori non responsabili della violazione: in questo senso l'inibitoria sembrerebbe risolversi in un espediente teso ad aggirare il divieto di ledere un interesse che il legislatore ha, invece, inteso tutelare.

26. Come ho già avuto modo di rilevare a proposito del fenomeno Linux[62], l'elemento veramente interessante nella comprensione del fenomeno open source, è rappresentato dal tipo di procedimento utilizzato per la realizzazione di programmi liberi. Tale metodologia di sviluppo è attentamente analizzata e commentata nel saggio La Cattedrale e il Bazar di Eric Raymond . Nel suo scritto, Raymond sembrerebbe, a prima vista, sostenere la tesi secondo cui i sistemi operativi e, in generale i programmi di maggior rilievo, debbano essere costruiti come cattedrali: debbano, cioè, essere minuziosamente creati da singoli individui o da ristretti gruppi di programmatori che lavorano autonomamente e senza diffondere versioni beta prima del tempo. La sua indagine empirica, tuttavia, finisce, inevitabilmente, per evidenziare che lo stile di sviluppo c.d. a bazaar, proprio del modello open source, funziona, e funziona in modo eccellente. Tale metodo di

[56] Pret. Milano, ord. 10 novembre 1992, in AIDA 1993,
[57] Par. 24
[58] Pret. Milano, ord. 10 novembre 1992, cit.
[59] Il divieto non opera anche nel caso di particolare gravità
[60] Pret. Roma, ord. 3 luglio 1975, in IDA, 143.
[61] Pret. Catania, 11 febbraio 1991, in AIDA 1992,
[62] Cap. 1, par. 11

sviluppo sfrutta, infatti, gli apporti di un bacino di utenza che, grazie soprattutto all'avvento della rete, si rivela potenzialmente illimitato: incoraggiando gli apporti esterni, si fa, quindi, in modo che ciascun utente interessato sia messo in grado di scovare errori, suggerire soluzioni e dare il proprio contributo per il miglioramento del codice. Non solo: sottoponendo il codice all'attenzione di una pluralità di soggetti, i tempi del debugging[63] si riducono drasticamente. Come si può notare, tale metodologia open source consta di una pluralità d'elementi, destinati a sollevare questioni giuridiche ed economiche delicate che vanno, quindi, attentamente ponderate.

Innanzitutto, se ciascun utente è legittimato a metter mano al codice e ad apportare modifiche e miglioramenti, diventerà di primaria importanza stabilire se tutti quelli che contribuiscono allo sviluppo del software possano essere considerati autori del programma, o se l'autore dovrà essere identificato, unicamente, nel creatore del programma originario. La distinzione si rivela decisiva, giacché, solo nel primo caso, a norma dell'art. 10 l.a., trovano applicazione le norme che regolano l'istituto della comunione (artt. 1100 e ss. c.c.): il consenso di tutti i contitolari diventerà, quindi, necessario, ai fini del trasferimento di uno o più diritti di sfruttamento economico del programma (art. 1108 c.c.). A questo proposito, non si può non rilevare come la quasi totalità dei progetti open source venga gestita da una mantainer, che è solitamente la persona che ha iniziato il progetto: si tratta di una figura di importanza primaria, in quanto dotata del potere di gestire e coordinare il progetto, quindi di accettare o rifiutare le richieste di miglioramenti, di applicare le patch, di controllare le imperfezioni e di interagire con altri progetti, quando necessario. Nel momento in cui tale figura è individuabile, sembrerebbe possibile caratterizzare il software come opera collettiva; a tal fine è, tuttavia necessario il ricorrere di un'ulteriore condizione: i contributi dei vari sviluppatori devono essere tra loro distinti e scindibili. Qualora tali requisiti siano soddisfatti, spetterà al mantainer la paternità dell'opera complessivamente considerata, nonché la titolarità dei diritti d'utilizzazione economica dell'opera stessa. Resta, tuttavia, ferma, in capo a coloro che hanno realizzato le singole parti del programma, la titolarità del diritto d'autore sulle stesse nonché la facoltà di utilizzare separatamente i rispettivi contributi (art. 38 l.a.)

Una volta individuato l'autore dell'opera spetterà a tale soggetto l'esercizio degli esclusivi diritti d'utilizzazione economica attribuitigli dalla legge speciale sul diritto d'autore. Non solo: l'autore sarà anche l'unico soggetto a poter decidere se e in che modo i terzi possono utilizzare la propria opera. In particolare, ricorrendo ad una licenza open source, l'autore del programma sceglierà di rendere liberamente disponibile la sua opera, concedendo al licenziatario il diritto di copiare, modificare, diffondere e distribuire il programma. Quello svolto dalla licenza è, quindi, un ruolo fondamentale: sebbene il procedimento di sviluppo non riguardi di per sé la licenza, è proprio quest' ultima che fornisce le condizioni che fanno funzionare il processo di sviluppo open source. In particolare, la GPL e le altre licenze che comportano i sottoscrittori l'obbligo di restituire alla comunità modifiche e miglioramenti, assicurano uno sviluppo collettivo e comune. Anche licenze come la BSD e l'Apache, rispondendo ai dettami dell'Open Source Definition, rientrano nell'area delle licenze open source: a differenza della GPL, tuttavia, tali licenze consentono di mantenere private le eventuali migliorie apportate: non richiedono, quindi, che le modifiche siano distribuite sotto la medesima licenza open source. A ben vedere, quindi, la comprensione dei termini e delle condizioni della licenza diventa, di conseguenza un passaggio fondamentale nella creazione e nello sviluppo di prodotto software che voglia essere open source.

Un ulteriore tratto caratteristico delle licenze open source è rappresentato dalla mancata previsione di qualunque garanzia a carico dell'autore del programma. Tale mancanza può essere ricondotta all'incontrollabile fruibilità del software open source: se è vero, infatti, che l'autore si avvale del contributo di una pluralità di soggetti, è anche vero che lo stesso non dispone degli strumenti necessari per controllare tutti i soggetti che manipolano e modificano il codice. Attraverso l'esclusione di qualunque garanzia, l'autore si sottrae, quindi, alla responsabilità nascente dall'eventuale cattivo funzionamento del software.

Una delle differenze fondamentali tra lo stile di sviluppo a cattedrale e lo stile di sviluppo a bazaar è la presenza di rilasci frequenti. L'obiettivo primario del primo modello è, infatti, quello di fornire all'utente un prodotto il più perfezionato possibile: le varie release sono, quindi, temporalmente distanziate l'una dall'altra, in modo tale da consentire un debugging completo tra una versione e l'altra. Mentre per lo stile a cattedrale l'individuazione degli errori e, in generale dei problemi riguardanti la programmazione, diventano operazioni complesse che richiedono mesi d'attenta analisi da parte di team di esperti programmatori, nella concezione a bazaar, l'ampia frequenza dei rilasci fa in modo che gli eventuali errori vengano sottoposti all' attenzione di migliaia di volenterosi co-sviluppatori, che scandagliano ogni nuova versione. La rapidità di diffusione, e l'entità dell'apporto dato dall'utenza finisce, quindi, per minimizzare le eventuali disfunzionalità causate dagli errori, che diventano, di conseguenza, fenomeni marginali. Il metodo di sviluppo a bazaar consente, quindi, al gruppo di sviluppatori di riconoscere velocemente le imperfezioni e di indirizzarle tempestivamente nel processo di sviluppo. Solitamente, gli sviluppatori utilizzano le prime versioni di sviluppo nel loro lavoro quotidiano, procedendo, in tal modo, alla loro sperimentazione sul campo. In tale contesto, diventa, tuttavia, problematica, la qualificazione di una figura indispensabile per la

[63] Procedimento di ricerca degli errori presenti nel programma.

metodologia di sviluppo dell'open source: quella del beta-tester, vale a dire del soggetto che, pur non apportando delle modifiche al programma, si occupa della sperimentazione dello stesso e dell'individuazione dei bug; se è vero, infatti, che, non contribuendo alla scrittura del codice, tale soggetto non può considerarsi autore d'alcun contributo originale al programma, è anche vero che senza i beta-tester le modifiche non sarebbero possibili. E', preferibile, a mio parere, la tesi secondo cui la figura del beta-tester, non apportando alcun intervento creativo all'opera, non può essere assimilata a quella dell'autore del programma.

Il metodo di sviluppo open source, così considerato, è stato adattato ad un certo numero di realtà diverse, all'interno delle quali lo stesso assumerà caratteristiche peculiari.In primo luogo, tale metodo è stato inserito nella realtà dell'azienda: in questo caso, è la stessa software house che si preoccupa di controllare lo sviluppo, rivestendo il ruolo di principale mantainer del progetto; nel contesto aziendale, si rivela, quindi, di primaria importanza chiarire chi sia il soggetto legittimato alla negoziazione di contratti concernenti i diritti d'autore sul software. Nel caso del software creato dal lavoratore dipendente nell'esecuzione delle proprie mansioni, o su istruzioni impartite dal suo datore di lavoro, l'art. 12-bis l.a. stabilisce, con specifico riferimento al software, che sia proprio il datore di lavoro il titolare dei diritti esclusivi d'utilizzazione economica dell'opera, pur rimanendo, il lavoratore dipendente, titolare del diritto morale d'autore. Opera, in questo caso, una presunzione di cessione automatica dei diritti d'utilizzazione economica a favore del datore di lavoro, salva, tuttavia, la possibilità di patto contrario. Sebbene le aziende spesso promuovano e sovvenzionino progetti open source e debbano, quindi, essere considerate le uniche responsabili del progetto open source, accade spesso che il singolo individuo sia, effettivamente, il mantainer dell'opera: è anzi probabile che, nel caso in cui cambiasse azienda, quella persona continuerebbe a essere il mantainer ufficiale del codice. Per esempio, si pensa sempre a Linus Torvalds come mantainer di Linux e mai a Transmeta, fino a tempi recenti il suo datore di lavoro, né ad altre società che potrebbero assumerlo in futuro. E', quindi, necessario che l'azienda assuma una posizione ben precisa, chiarendo, contrattualmente, se i dipendenti possano limitare la loro collaborazione unicamente a progetti open source o se ci si aspetti che questi partecipino anche a progetti proprietari. Sarà inoltre chiamata a definire quando e come i propri dipendenti possano partecipare a progetti open source nel loro tempo libero, oltre a stabilire accordi di riservatezza. In alcuni casi sarà possibile delimitare confini precisi tra il lavoro per la compagnia e gli hobby dei suoi dipendenti, in altri invece tale demarcazione non sarà possibile. In altri casi ancora, ciò che per un dipendente è iniziato come un passatempo potrebbe trasformarsi in un progetto importante per l'azienda.

L' azienda, tuttavia, non è l'unica realtà all'interno della quale un progetto open source può essere sviluppato. Spesso, infatti, si ricorre alla creazione di fondazioni, quali mezzi per finanziare progetti di grandi dimensioni: in tale contesto, le aziende e gli individui che hanno interesse a far sì che il progetto venga portato a termine possono donare dei fondi a un' organizzazione no-profit. Non solo la fondazione, ma anche un comitato può essere utilizzato per progetti di grandi dimensioni. In contrasto con l'ipotesi standard in cui il singolo mantainer è responsabile delle scelte che si fanno, in questo caso le statuizioni più importanti vengono, di regola, prese in base a un consenso comune. Svanisce, quindi, la figura del singolo coordinatore dell'opera, escludendo, di conseguenza, la possibilità di guardare al risultato come ad un'opera collettiva. E' questo il caso in cui è possibile guardare all'opera come il risultato di contributi inscindibili di una pluralità di soggetti, cui spetterà, in egual misura, la titolarità del programma risultante.

I progetti di piccole dimensioni vengono, a loro volta, gestiti da singoli individui e ospitati su siti Internet specializzati[64]. Con riguardo a tali progetti, sarà più agevole l'individuazione del mantainer e soprattutto dei singoli contributi che vanno a formare l'opera, nel suo complesso considerata.

[64] il più conosciuto è SourceForge, anche se ultimamamente molti progetti si stanno spostando su Savannah, un archivio per free software ospitato dal progetto GNU

CAPITOLO TERZO
IL PROFILO SOCIOLOGICO

Sommario: 1. La proprietà e l'open source. - 2. Il ruolo della ricompensa. – 3. La reputazione come principale ricompensa. – 4. Gli incentivi alla reputazione. – 5. Il valore dell'umiltà. – 6. I motivi di conflitto e la risoluzione delle controversie. – 7. L'inserimento di sviluppatori open source all'interno di un'impresa commerciale.

1. Una volta analizzate le questioni giuridiche relative alla realtà open source, diventa interessante indagarne alcuni profili più strettamente sociologici: in particolare, ci si può chiedere quali sono le motivazioni che spingono centinaia di programmatori a rinunciare a cospicue retribuzioni, in cambio della possibilità di partecipare a un progetto open source.

A tale scopo, è, in primo luogo, necessario chiarire qual è il rapporto che lega uno sviluppatore ad un determinato progetto. La definizione di tale rapporto sembrerebbe complicata dal fatto che, nell'ambito delle nuove tecnologie digitali, l'istituto giuridico della proprietà perde le proprie tradizionali connotazioni: l'oggetto del diritto, non solo, diventa liberamente accessibile attraverso la rete, ma si trasforma in una realtà malleabile e replicabile all'infinito. Nel contesto open source, tuttavia, il problema dell'individuazione del proprietario del programma e della definizione del contenuto del suo diritto deve essere subito ridimensionato: il detentore del progetto è, infatti, comunemente identificabile nel soggetto che detiene il diritto esclusivo di redistribuire le versioni modificate. Stando così le cose, è possibile fare una prima importante considerazione: se è vero che le licenze open source consentono a chiunque di modificare il programma, è, tuttavia, necessario tenere distinti gli aggiustamenti ufficiali, approvati dalla comunità e incorporati nel software in evoluzione da parte di chi dirige il progetto, da quelli realizzati da soggetti estranei al progetto e non inseriti nella versione ufficiale, che sono, in genere, malvisti all'interno della comunità.

Una volta chiarito il rapporto che lega il programmatore a un determinato progetto, il problema si sposta sul come e sul quando nasce questo progetto. La cultura open source distingue, tradizionalmente, tre modi di acquisire la proprietà di un programma[65]:

1) Avviare la nascita del progetto. In questo senso, deve essere considerato proprietario del progetto, a titolo originario, chi scrive una massa critica di codice da sottoporre alla comunità di sviluppatori o, in ogni caso, chi, sin dall'origine, coordina la stesura del programma.

2) Ottenere la proprietà del progetto dal proprietario precedente. A tale proposito, le usanze della comunità impongono a quest'ultimo, quando non sia più in grado di curare lo sviluppo o il mantenimento del progetto, di eleggere al ruolo di successore, il programmatore ritenuto più competente tra quelli più vicini al progetto.

3) Reclamare un progetto vacante, facendo pubblicamente presente il disinteresse o la scomparsa del proprietario. In questo caso, le consuetudine comunitarie richiedono al potenziale proprietario di fare quanto possibile per rintracciare il precedente detentore, quindi di rendere apertamente note le proprie intenzioni (per esempio all' interno di uno o più newsgroup). In generale, l'acquisizione risulterà tanto più consolidata, quanto maggiori saranno gli sforzi per consentire al proprietario di riprendere il controllo del progetto.

L'evoluzione di tali consuetudini, seguite con notevole coerenza dalla totalità dei membri della comunità, ha finito per ripercuotersi in modo positivo sui comportamenti di tutti i membri della comunità: è riuscita, infatti, a stimolare, non solo una maggiore responsabilità collettiva, ma anche una maggiore attenzione nel preservare i nomi degli sviluppatori che partecipano a ciascun progetto e la cronologia delle modifiche. Sembrerebbe, quindi, che nella realtà open source, la rinuncia ai diritti d'utilizzazione economica dell'opera, sia bilanciata da un pressante interesse a tutelare il diritto alla paternità dell'opera.

2. Se l'atto di delimitare i confini della propria opera, di migliorarla e di difenderne l'atto di proprietà, è di così fondamentale importanza per la comunità degli sviluppatori, lo scopo è quello di ottenere, da tale sforzo, un qualche tipo di provente. A seconda del modo di acquisizione della proprietà, tale ricompensa deve, quantomeno, bilanciare le energie spese per avviare il progetto, i costi sostenuti per mantenere le

[65] Nel suo saggio "Colonizzare la Noosfera", Raymond suggerisce un interessante parallelismo fra le convenzioni degli hacker in tema di proprietà e la teoria della proprietà terriera di Locke. Su www.apogeonline.com/Openpress

cronologie delle documentazioni che provano i passaggi di consegne da un mantainer all' altro, o il tempo speso per i pubblici annunci e per il periodo di attesa prima di potersi impossessare di un progetto vacante. Ci si deve, quindi, chiedere da cos'è costituita tale ricompensa e, di conseguenza, a cosa aspirano i programmatori che partecipano ad un progetto open source. E' da escludere, in primo luogo, che il premio possa essere rappresentato dalla conquista di una posizione di potere all'interno della comunità, non potendo sussistere, all'interno di una rete globale, rapporti di forza. Gli sviluppatori non mirano, inoltre, ad ottenere un compenso materiale, dal momento che il modello di sviluppo non prevede un'equivalente al denaro. A tale proposito, tuttavia, è necessaria una precisazione: esiste, in effetti, una possibilità d'arricchimento economico per chi opera nell'ambito open source; può accadere, infatti, che la reputazione acquisita all'interno della comunità consenta allo sviluppatore di ottenere migliori posti di lavoro o richieste di consulenza, nel campo dell'industria software tradizionale. Sebbene tal effetto collaterale sia raro e marginale, esso pone l'accento su un elemento fondamentale per la comprensione dei motivi che spingono gli sviluppatori a aderire a progetti open source: la reputazione.

3. Per comprendere il ruolo svolto dalla reputazione, all'interno della cultura open source, è necessario caratterizzare quest'ultima come una cultura del dono[66]; in quanto tale, essa si distingue per la relativa abbondanza delle risorse, essendo potenzialmente illimitato sia lo spazio sul disco sia l'ampiezza della banda della rete, sia da ultimo la potenza dei computer. Tale disponibilità di mezzi incide, inevitabilmente, sulle relazioni economiche tra gli individui, rendendo, in primo luogo, difficile l'instaurarsi di rapporti di comando: è, infatti, improbabile che un qualsiasi soggetto possa ergersi ad autorità e decidere il modo in cui distribuire le risorse. L'abbondanza rende, inoltre, inutili le relazioni di scambio, eliminando la necessità d'interagire con altri soggetti al fine di ottenere ciò di cui si ha bisogno. Ne consegue che, in un'economia del dono, lo status sociale dell'individuo è determinato, non dalla possibilità di accedere al potere coercitivo o di controllare le cose, ma da ciò che si regala. Se si parte, poi, dal presupposto che gli esseri umani nascono predisposti a competere fra loro per il miglioramento dello status sociale, si deve concludere che, all'interno di tale cultura, l'unico ambito disponibile per la competizione sarà quello della reputazione fra i consociati.

Sono, chiaramente, diverse le ragioni per cui si tende a conquistare un certo prestigio, all'interno della propria comunità di riferimento: una buona reputazione è, innanzitutto, un ottimo metodo per attirare l'attenzione e la collaborazione degli altri; nel caso in cui l'economia del dono sia, poi, collegata ad altri tipi d'economie, il prestigio potrebbe consentire di ottenere uno status più elevato anche all'interno di tali diversi ambienti.

Nella cultura open source, l'elemento della reputazione trova la sua massima espressione: i programmi che si devolvono alla comunità sono, infatti, prodotti complessi, e costituiscono la prova materiale del tempo e delle energie profuse nel progetto. A differenza di quel che accade per i beni materiali, il loro valore non è, tuttavia, facilmente stimabile: è, pertanto, determinato dal giudizio critico dei soggetti più esperti in materia, vale a dire degli altri sviluppatori. Dal momento che proprio una buona reputazione è l'unico modo per ambire ad un miglioramento dello status sociale, lo sviluppatore è, perciò, spronato a creare prodotti di qualità al fine di ottenere il consenso della comunità.

Senza tralasciare il personale piacere che lo sviluppatore trae dalla programmazione, la ricompensa principale derivante dalla collaborazione in un progetto open source è, quindi, costituita dalla reputazione. Ovviamente, il lancio di un nuovo progetto o la partecipazione a progetti innovativi garantisce al programmatore la possibilità di ottenere una maggior stima all'interno della comunità. E' inoltre più facile avere successo e guadagnare prestigio creando nuovi strumenti, piuttosto che gareggiando con un programma che ha già avuto successo. Esiste insomma una distanza ottimale dai progetti rivali più simili: avvicinandosi troppo, c'è il rischio che il programma offerto alla comunità perda di valore e venga percepito come scadente, con conseguente pregiudizio della reputazione di quanti hanno collaborato al progetto. Allontanandosi troppo, al contrario, si corre il pericolo che il programma risultante sia comprensibile o utile solo al suo autore: anche in questo caso, il dono è percepito come scadente e gli sforzi dell'autore non gli fanno guadagnare in reputazione all'interno della comunità. Tutto ciò rivela che, al fine di seguire un percorso di posizionamento ottimale, il progetto deve andare a riempire i vuoti funzionali prossimi a una linea di frontiera immaginaria, rappresentata da un prodotto innovativo e utile. In tale contesto, un ruolo a parte è rivestito dai c.d. killer di categoria: si tratta di progetti così ben riusciti che porsi in competizione con essi, risulta troppo difficile. L'esempio classico è rappresentato da GNU Emacs: le sue varianti riempiono così perfettamente la nicchia riservata ad un editor di testo, che nessuno ha mai intrapreso un progetto alternativo. La tendenza a creare strumenti sempre più utili e innovativi, insieme alla periodica comparsa di killer di categoria, ha portato all'individuazione di filoni prevedibili nell'avvio dei progetti. Se negli anni settanta la maggior parte del software open source era rappresentata da demo e giochi divertenti, gli anni ottanta hanno visto l'intensificarsi della spinta creativa verso lo sviluppo di strumenti per Internet, spinta che

[66] Vedi ancora "Colonizzare la Noosfera"

si è protratta negli anni novanta insieme a quella relativa ai sistemi operativi. In ognuno di questi casi, esaurite le possibilità dei filoni precedenti, è stato affrontato un nuovo e più complesso livello di problemi. Le tendenze attuali mostrano che l'open source sta ora dirigendo il suo interesse verso programmi per un'utenza non esperta.

4. Se la reputazione rappresenta la massima ricompensa per tutti quelli che si dedicano allo sviluppo di software open source, le consuetudini sviluppate dalla comunità hacker riguardo ai modi d'acquisizione della proprietà[67], costituiscono gli strumenti per massimizzare gli incentivi della reputazione, giacché assicurano che i riconoscimenti vadano a chi siano veramente dovuti e non a qualcun altro. E', infatti, configurabile tutta una serie di comportamenti osteggiati dalla comunità, poiché lesivi della reputazione del programmatore. Esiste, in primo luogo, una forte pressione sociale contro i progetti divaricanti: oltre a dare luogo a quella che gli sviluppatori considerano un'inutile duplicazione del lavoro, la biforcazione[68] espone, infatti, i collaboratori precedenti al rischio di perdita della reputazione[69]. In contraddizione con la convinzione condivisa secondo cui chiunque può modificare il programma, la cultura open source ha, infatti, sviluppato una serie di consuetudini secondo cui spetta al proprietario il diritto esclusivo di approvare le modifiche d'apportare al programma e includerle nella versione ufficiale. In tal modo si evita che la sua reputazione sia ingiustamente danneggiata qualora altri soggetti si approprino indebitamente del suo lavoro o apportino modifiche non approvate. E' inoltre avversata la pratica di distribuire patch, non approvate dal titolare del progetto: ancora una volta, lo sviluppatore può andare incontro ad un'ingiusta perdita di reputazione; anche qualora il codice ufficiale fosse perfetto, al proprietario del progetto potrebbero, infatti, essere rimproverati gli errori presenti nelle patch. In terzo luogo, la rimozione del nome di uno sviluppatore dalla cronologia di un progetto o dall'elenco di quanti hanno collaborato, non avviene mai senza l'esplicito consenso dell'interessato; se ciò accadesse, sarebbe commesso, nel contesto culturale, un crimine gravissimo: non solo lo sviluppatore sarebbe ingiustamente privato della reputazione derivatagli dall'aver partecipato a un progetto, ma il responsabile della cancellatura finirebbe per apparire indebitamente come colui che dona alla comunità il proprio lavoro. A ben vedere, tutti i comportamenti sopra descritti hanno conseguenze che vanno oltre la sfera individuale del singolo sviluppatore: arrecano pregiudizio all'intera comunità, poiché affievoliscono nei consociati la convinzione che regalando il proprio lavoro si otterrà una ricompensa.

5. Una volta verificato che il prestigio è l'elemento centrale nei meccanismi di ricompensa della cultura open source, è interessante chiarire le ragioni per cui tale prestigio rimane qualcosa di non dichiarato. La comunità open source è fortemente meritocratica: non potendo esistere rapporti di forza all' interno della rete, non è dotata di una gerarchia formale sulla base della quale distribuire le varie operazioni di programmazione; da ciò consegue che chiunque sia interessato allo sviluppo di un particolare progetto, è invitato a scriverlo o a migliorarlo; quello che viene prodotto rappresenta, poi, all'interno della comunità, il biglietto da visita di ciascuno: è forte, infatti, la convinzione che la qualità del prodotto dovrebbe parlare di per sé, con la conseguenza che il programma migliore è quello che funziona in modo semplice e viene riconosciuto dagli altri sviluppatori come un lavoro ben fatto; ecco come, in un sistema comunitario basato sulla valutazione dei pari grado, l'umiltà assurge al ruolo di valore primario: vantarsi del proprio lavoro diventa, in tale contesto, un atteggiamento soppresso, in quanto impedisce di concentrarsi su ciò che veramente conta, vale a dire la qualità del lavoro svolto. A ben vedere, il valore dell'umiltà risulta, altresì, funzionale al mantenimento di un progetto di successo: i potenziali collaboratori sono, infatti, propensi ad affidare il loro lavoro a mantainer dotati della sufficiente umiltà di riconoscere il codice ben scritto e di allegarlo alla versione ufficiale, dando così i riconoscimenti dovuti. Per avere successo, il titolare del progetto deve, quindi, dimostrare alla comunità di possedere capacità di giudizio nel valutare il codice proposto da altri.

6. L'affermazione della proprietà è, anche nel contesto open source, un modo per delimitare i confini del proprio territorio e di dichiarare che tali confini saranno difesi. Come nella società civile, la funzione della proprietà è, quindi, quella di prevenire eventuali conflitti, tracciando quel limite che separa i comportamenti pacifici dalle aggressioni; nello specifico della realtà open source, poi, il sostegno della comunità nell'affermazione della proprietà costituisce un modo ancora più efficace di minimizzare i conflitti

[67] Vedi par. 2.
[68] Scissione del progetto in uno o più sottoprogetti.
[69] Rischio che potrebbe essere evitato solo nel caso in cui gli sviluppatori collaborino ad entrambi i progetti derivanti dalla biforcazione.

e massimizzare i comportamenti cooperativi.

Ci si potrebbe, quindi, chiedere quali tipi di controversie, l'affermazione della proprietà è solita prevenire. Con riguardo ai progetti open source le più comuni cause di conflitto riguardano:

- L'individuazione del soggetto chiamato a prendere le decisioni principali.
- I criteri d'attribuzione di meriti e demeriti.
- Le precauzioni da adottare al fine di evitare la biforcazione del progetto.

Nel primo caso, le consuetudini riguardanti la proprietà indicano il proprietario del progetto quale soggetto cui spetta prendere le decisioni vincolanti; la precisa individuazione di un mantainer riduce, inoltre, le possibilità di scissione di progetti. In questi casi, la presenza di un leader carismatico diventa addirittura fondamentale, poiché, non solo assicura la compattezza del gruppo di sviluppo e la fiducia dei programmatori, ma rende meno probabile della fazione dissenziente possa trovare un mantainer altrettanto carismatico Per quel che riguarda il problema della ripartizione dei meriti, il discorso si fa un po' più complicato; è necessario, infatti, distinguere alcuni casi. L'ipotesi più semplice è quella che vede un unico soggetto nel ruolo di proprietario o mantainer del progetto: spetta, in questo caso, a tale individuo il compito di prendere le decisioni e di attribuire meriti e riconoscimenti. Gli unici conflitti possibili potrebbero nascere dalla questione della successione, nel caso in cui il vecchio proprietario perda interesse nel progetto: al fine di evitare biforcazioni, il mantainer è, qui, chiamato a scegliere, tra le persone più vicine al progetto, quella più adatta a sostituirlo.

Un secondo caso occorre quando sono presenti diverse persone che collaborano al mantenimento, sottostando alle direttive di un c.d. dittatore amichevole, che è proprietario del progetto; s'inserisce, in tale contesto, un nuovo livello di possibili dispute relativamente a chi debba andare il merito e per quale parte del progetto. In situazioni simili, le usanze impongono al proprietario di assegnare correttamente i meriti, tramite appropriate menzioni nei file contenenti la cronologia delle versioni. Si potrebbe, quindi, affermare che il contributo al progetto legittima l'attribuzione, ai collaboratori, di parte della reputazione complessiva; sotto un altro punto di vista, si potrebbe dire che il proprietario commercia varie parti della reputazione complessiva con il contributo altrui.

Si può pensare di complicare ulteriormente tale situazione, ipotizzando che nuovi partecipanti vadano ad aggiungersi ai progetti gestiti da dittatori amichevoli. Si afferma, in quest'ipotesi, la tendenza a dividere i collaboratori in due gruppi: gli ordinari da una parte e i co-sviluppatori dall'altra. La differenza tra le due categorie deve essere ricercata nel fatto che, per divenire co-sviluppatore è necessario assumersi la responsabilità di un sottosistema importante del progetto: il soggetto che accetta tale incarico gestionale arriva a controllare l' implementazione del sottosistema, rimanendo soggetto soltanto alle correzioni del mantainer. E' il caso di notare come questa regola, in pratica, produca una serie di proprietà chiuse all'interno del progetto, svolgendo, altresì, la funzione di prevenzione dei conflitti. Considerata, quindi, l'importanza rivestita da tali soggetti nell'ambito del progetto, il dittatore amichevole è chiamato a consultarli in occasione di decisioni importanti: ciò soprattutto quando la decisione riguarda un sottosistema affidato alla sua responsabilità. Linux è il tipico esempio di dittatura amichevole: grazie, soprattutto, alla modularità del suo kernel, è dotato di un'organizzazione patchwork che consente la gestione individuale d'ogni settore da parte dei suoi programmatori più esperti, il cui lavoro consiste nel filtrare e controllare le patch proposte da altri sviluppatori, per poi sottoporle a Linus. Alcuni progetti molto ampi non consentono una piena aderenza al modello del dittatore amichevole: in alcuni casi tale inconveniente è stato superato facendo dei co-sviluppatori un comitato votante, in altri, adottando una rotazione nella funzione di leader. La comunità non guarda, tuttavia, con favore a queste possibilità alternative, poiché, nel momento in cui le organizzazioni si fanno complesse, diventa, problematico assicurare il rispetto della proprietà e il ritorno del riconoscimento.

In generale, la crescente complessità dell'organizzazione si esprime, all'interno del progetto, tramite la distribuzione, tra i vari co-sviluppatori, dell'autorità su determinati sotto-sistemi e di parziali diritti di proprietà: tali articolazioni possono risultare efficaci nella circolazione degli incentivi, giacché distribuiscono le responsabilità tra un numero maggiore di soggetti; al contempo, tuttavia, provocano una diluizione dell'autorità del leader: una minore capacità di controllo da parte del mantainer genera, di conseguenza, un aumento delle possibilità di conflitto. Diventa, quindi, più probabile che sorgano questioni tecniche relative al design del progetto: si tratta, in ogni caso, di controversie facilmente risolvibili attraverso, il ricorso alla regola territoriale secondo la quale l'autorità fa seguito alla responsabilità; spetta quindi al responsabile di ciascun sotto-sistema la risoluzione dei conflitti nascenti all'interno della sua proprietà. Tale criterio non trova, tuttavia, applicazione nel caso in cui nessuno dei collaboratori contendenti sia proprietario del territorio in cui avviene la disputa: il conflitto è, qui, risolto dal criterio alternativo dell'anzianità: ha, quindi, la meglio chi tra i litiganti ha dedicato maggior tempo e lavoro al progetto. Le regole dell'autorità e quella dell'anzianità sono, di regola, sufficienti a risolvere i conflitti relativi a un progetto: quando ciò non sia possibile, tocca all'autorità leader il compito di risolvere la questione.

In generale si può notare che, nel contesto open source, tutti i meccanismi per la risoluzione dei conflitti poggiano quindi sulla volontà di applicarli da parte della comunità degli sviluppatori

complessivamente considerata. Gli unici strumenti per imporla sono rappresentati dal flame e dall'allontanamento, vale a dire dalla condanna pubblica di quanti hanno infranto le convenzioni e dal successivo rifiuto a collaborare ulteriormente con tali soggetti.

7. Abbiamo visto che, in alcuni casi, la reputazione acquisita dallo sviluppatore supera i confini della comunità open source e gli consente di sfruttare richieste di lavoro o di consulenze da parte delle aziende produttrici di software. Attirate da un metodo di sviluppo distribuito, queste hanno, infatti, iniziato a reclutare i migliori talenti disponibili sul mercato, al fine di avviare progetti open source. L' incontro tra la cultura open source e quella del software proprietario tradizionale ha, in ogni caso, creato questioni d'importanza non marginale: l'open source va, infatti, ad incidere sui metodi tradizionalmente adottati dalle aziende in merito alle risorse umane e allo sviluppo organizzativo; con particolare riguardo al reclutamento del personale, le aziende produttrici di software sono quindi, chiamate ad utilizzare una cura particolare.

Un primo problema è quello concernente i contratti di lavoro. Molte società ricorrono, infatti, nei loro contratti d'assunzione, a clausole negoziali in base alle quali tutto ciò che è sviluppato da un dipendente, diventa automaticamente di proprietà dell'azienda. Nel momento in cui, tuttavia, un'azienda produttrice di software assume un programmatore che ha lavorato con la comunità e che avrà, di conseguenza, profonde convinzioni in merito all'open source, con ogni probabilità si troverà ad affrontare una serie di richieste da parte del dipendente. L'azienda è, pertanto, chiamata a tracciare una politica ben precisa, diretta a governare il personale: oltre a chiarire se i dipendenti possano limitare la loro collaborazione unicamente a progetti open source, deve definire quando e come questi possano partecipare a progetti open source nel loro tempo libero. In alcuni casi sarà possibile delimitare confini precisi tra il lavoro per la compagnia e gli hobby, in altri invece tale demarcazione non sarà possibile. In altri casi ancora, ciò che per un dipendente è iniziato come un passatempo potrebbe trasformarsi in un progetto importante per l'azienda.

Un'altra questione particolarmente spinosa riguarda la proprietà intellettuale dei contributi open source. L'azienda potrebbe voler mantenere la proprietà del copyright, al fine di riservarsi la possibilità d'implementare una licenza doppia, e di distribuire commercialmente il prodotto; in ogni caso, il contratto di lavoro deve chiarire a chi è attribuita la proprietà del copyright che copre i contributi offerti dai propri programmatori.

Più in generale, una chiara comunicazione in merito al modello commerciale adottato dalla compagnia rappresenta una guida ideale per i collaboratori: l'azienda deve pertanto illustrare i suoi obiettivi, ponendo l'accento, se del caso, sulle fonti da cui intende trarre il suo guadagno (ad esempio con il supporto, con i servizi, con l'hardware), anche nell'ipotesi in cui si stia lavorando su progetti che non creino direttamente delle entrate. E', inoltre, chiamata a chiarire quale atteggiamento intende tenere nei confronti della comunità, precisando quali sono le attese dell'azienda riguardo all'etica professionale dei propri dipendenti e come l'azienda stessa intenda lavorare con la comunità.

Analizzando più da vicino la questione riguardante l'arruolamento di sviluppatori provenienti dall'open source, si possono individuare alcuni criteri cui appellarsi per verificare l'idoneità dell'individuo a fare parte del progetto. In primo luogo, non bisogna dimenticare che, di regola, attorno a ciascun settore tecnologico (il kernel, i file system, i networking stack, i driver di periferica) si forma una sotto-comunità. Il criterio di valutazione dell'idoneità del candidato, deve, quindi, essere rappresentato dalla notorietà raggiunta da quest'ultimo, all'interno della sotto-comunità di riferimento, e non della comunità complessivamente considerata. Si potrebbe quindi pensare che l'azienda interessata all'assunzione di programmatori open source, contatti ciascuna sotto-comunità al fine di assumerne i personaggi più noti. In secondo luogo, è necessario tener presente che la maggior parte dei progetti open source è condotta da un mantainer o da una comunità di contributori. Diventa, quindi, di primaria importanza stabilire quali di questi ruoli l'azienda ha bisogno di coprire: disporre di un nutrito gruppo di programmatori è di fondamentale importanza per la buona riuscita del progetto, disporre d'individui di provata competenza, nell'offrire contributi in diversi settori tecnologici, è ancora meglio. Nel caso in cui, l'azienda intenda arruolare un professionista che ricopra un ruolo di responsabilità, è chiamata a cercare tra coloro che hanno già una certa esperienza nel mantenimento di progetti open source. I mantainer costituiscono, infatti, una categoria a sé, dal momento che si richiede loro di combinare la competenza tecnologica con la capacità di dirigere gli altri sviluppatori, di risolvere i conflitti e, quindi, di portare a buon fine lo sviluppo di un progetto. Al fine d'individuare un buon mantainer, è indispensabile verificare che il candidato abbia il giusto grado di visibilità e rispetto all'interno della comunità. A tale scopo, è possibile fare riferimento, per esempio, alle interazioni on-line che avvengono tra il potenziale mantainer e la comunità: esaminando il numero di persone che cerca il parere del candidato è possibile determinare il rispetto di cui gode all'interno della comunità. Può, inoltre, rivelarsi utile analizzare con quanta rapidità altri mantainer accettano i suoi contributi: chi gode di una certa stima vede, di regola, i sui contributi accettati senza alcuna discussione in merito E', inoltre, sempre possibile richiedere un elenco dettagliato dei contributi offerti dal potenziale mantainer ad altri progetti open source. In questa prospettiva, uno dei grandi vantaggi dell'open source risiede, proprio, nel fatto che i progetti sono pubblici,

una condizione che permette di valutare la quantità, la qualità del lavoro svolto e le capacità tecniche del candidato, ancora prima di formalizzare l'assunzione. Una volta individuato il potenziale mantainer, diventa di fondamentale importanza la ricerca di un equilibrio tra le responsabilità di quest'ultimo nei confronti della comunità, e le necessità dell'azienda. Prima di fare una scelta è, quindi, necessario chiedersi come potrebbe essere gestita una situazione in cui il mantainer/dipendente prenda una decisione valida per la comunità open source, ma svantaggiosa per l'azienda. Non bisogna, infatti, mai dare per scontato che assumendo un mantainer, l'azienda acquisti, automaticamente, il controllo di un progetto. Basti ricordare, a tale proposito, che l'unico mantainer di Linux è Linus Torvalds, e non il suo datore di lavoro.

E' possibile, a questo punto, concludere che, grazie al metodo di sviluppo open source e alle consuetudini proprie della comunità che vi ruota intorno, sta prendendo forma un nuovo modello economico, che va oltre le nozioni convenzionali di lavoro e compenso. Per sfruttare a pieno le enormi potenzialità di sviluppo offerte da tale modello, l'industria tradizionale è chiamata a adattare i suoi metodi alle caratteristiche proprie del movimento open source, giacché non solo l'attività di programmazione andrà perdendo la sua connotazione d'occupazione ma il valore del lavoro sarà dato esclusivamente dalla condivisione con gli altri.

CAPITOLO QUARTO
PROFILI ECONOMICI

Sommario: 1.Il ruolo della ricompensa. – 2. La reazione delle aziende produttrici di software. – 3. La fornitura d'assistenza tecnica come soluzione alternativa.

1. Nel capitolo precedente ho analizzato, da un punto di vista sociologico, le motivazioni che spingono un programmatore a partecipare ad un progetto open source. Tali motivazioni possono, ora, essere utilmente esaminate sotto un punto di vista più prettamente economico. L'adesione ad un progetto software, sia esso commerciale od open source, è determinata dalla possibilità di trarne un beneficio: in particolare, tale utilità comprende una ricompensa immediata e una differita. Alla prima categoria è, senza dubbio, riconducibile il compenso in denaro ricevuto per l'attività di programmazione svolta: si tratta dell'ipotesi ricorrente nel caso in cui il programmatore sia impiegato in un'azienda commerciale. Esistono altri tipi di ricompensa immediata che, anche alla luce di quanto detto nel capitolo dedicato ai profili sociologici, sono più vicini ai progetti open source. In quest'ottica, grazie alla libera fruibilità del codice sorgente del programma, uno dei vantaggi derivanti dall'attività di programmazione è individuabile nella possibilità di correggere gli errori del sistema o di adattare il programma alle proprie esigenze personali; in generale, anche il software proprietario può, di per sé, essere adattato secondo le esigenze del singolo utente: si tratta, tuttavia, di un servizio che l'utilizzatore non può portare a termine liberamente e che le aziende produttrici o i fornitori di software offrono in cambio di una somma di denaro. Non si può, infine, trascurare che, nel momento in cui lavora ad un progetto, il programmatore trova più difficile dedicarsi ad attività di programmazione alternative: ciò significa, in altre parole, che su ogni programmatore grava un costo in termini di tempo, la cui quantificazione dipende da quanto piacevole e gratificante sia tale attività. Fin dalle sue origini, la cultura hacker ha insistito sull'importanza di guardare alla programmazione, come ad un'attività di puro piacere e divertimento: in quest'ottica, la possibilità di ottenere un ritorno economico dall'attività del programmare passa in secondo piano.

La seconda categoria di benefici, quella delle ricompense differite, comprende due distinti tipi d'incentivi: la possibilità di ottenere un avanzamento in carriera, da una parte, e la gratificazione personale, dall'altra. Il programmatore che si fa notare attraverso la propria attività di programmazione, può, infatti, ambire a ricevere offerte di lavoro da parte d'aziende commerciali o, in alternativa, ad accedere al mercato dei capitali di rischio, nel caso in cui sia a capo di un'impresa. Per quanto riguarda la gratificazione personale, questa va a scontrarsi, nel contesto open source, con il desiderio di ricevere un riconoscimento da parte degli altri membri della comunità; va da sé, che i programmatori preoccupati di ottenere una buona reputazione fra i propri pari, saranno, generalmente, meno preoccupati di ottenere un ritorno economico dal proprio lavoro e aspireranno a segnalare il proprio talento agli altri sviluppatori, vale a dire a un pubblico differente rispetto a quello cui mireranno coloro che intendono sfruttare il proprio operato per guadagnarsi un'opportunità di fare carriera. In ogni caso, entrambe le categorie d'incentivi ora descritte sono più forti quanto più visibile risulta il progetto: i programmatori preferiranno, quindi, lavorare su programmi che attraggano un gran numero di sviluppatori, onde assicurarsi la possibilità di sottoporre il proprio operato all'attenzione degli altri membri della comunità, del mercato del lavoro o di quello dei capitali di rischio.

A questo punto è possibile verificare in che modo tale gamma d'incentivi influisce sull'attività del programmatore in ambiente open source e, parallelamente, in ambiente proprietario. I progetti commerciali hanno, senza dubbio, un vantaggio nell'ambito delle ricompense immediate: dal momento che la natura proprietaria del codice genera un'entrata monetaria[70], l'azienda trae vantaggio dall'offrire un salario ai propri dipendenti, al fine di poter sfruttare il risultato della loro attività. Dal canto suo, pur non prevedendo la corresponsione di un salario per l'attività di programmazione prestata, il modello di sviluppo open source va, in ogni caso, incontro ad una riduzione dei costi per i programmatori. Ciò avviene in due modi:

- Grazie alla sua libera disponibilità, il codice open source può utilizzato nelle scuole e nelle università a fini didattici: ciò aumenta la possibilità che, nel momento in cui si avvicina ad un progetto open source, il programmatore abbia già una certa familiarità con la materia. Per le sue intrinseche caratteristiche, il modello di sviluppo open source implica, quindi, una riduzione dei costi sostenuti per l'addestramento dei programmatori.
- Il costo derivante dal portare avanti un progetto open source è minore nel momento in cui quest'attività garantisce un vantaggio personale al programmatore stesso o all'azienda, che riesce, per esempio, a adattare un programma alle proprie esigenze personali od organizzative; anche in

[70] Data dalla concessione a terzi di una licenza d'uso sul prodotto.

questo caso, l'abbattimento dei costi è strettamente legato all'apertura del codice sorgente del programma.

Nell'ambito delle ricompense posticipate, il metodo di sviluppo open source gode di un certo vantaggio rispetto all'approccio proprietario, essenzialmente per tre ragioni:

1. Gli osservatori esterni possono valutare, solo in maniera inesatta, la qualità e le funzionalità di un programma sviluppato in un ambito commerciale, giacché non è loro attribuita la possibilità di conoscerne il codice sorgente. In un progetto open source, al contrario, l'osservatore esterno, non solo è messo in grado di poter riconoscere il contributo di ciascun programmatore, ma anche di verificare se il lavoro svolto ha richiesto uno sforzo notevole, se è stato condotto in maniera logica e se il codice possa essere utile per qualche futura attività di programmazione.
2. Avendo la piena responsabilità del proprio sotto-progetto, al programmatore è lasciata piena iniziativa, avendo questi la responsabilità del successo del proprio sotto-progetto.
1. Nel conteso open source, il mercato del lavoro è caratterizzato da una maggiore fluidità: non essendo legati ad un'azienda specifica, i programmatori sono, infatti, liberi di dirigere i propri sforzi verso progetti eterogenei, garantendosi, in tal modo, maggiori occasioni di visibilità.

In questo contesto, bisogna, altresì, sottolineare l'importanza di altri tipi d'incentivi individuali: non solo i migliori progetti open source nascono, infatti, da quello che Eric Raymond[71] definisce "un prurito personale", vale a dire la necessità di dare un soluzione a problemi quotidiani, ma è chiaro che, soprattutto in ambito open source, abbia un certo valore il credito dato agli autori del programma: questo genere di riconoscimento, che trova spazio nei file contenenti la cronologia delle modifiche apportate, permette di ottenere una certa reputazione all' interno della comunità e di ambire, quindi, ai progetti migliori.

2. Una volta verificata l'efficienza del modello di sviluppo proposto dal movimento open source, le grandi aziende software potrebbero volerne sfruttare i requisiti, adottando, alternativamente, una delle seguenti strategie.

- Riproducendo la struttura degli incentivi propri del processo open source, all'interno del contesto proprietario
- Combinando il processo open source al modello proprietario, per ottenere il meglio da entrambi.

E' improbabile che le grandi società software riescano a duplicare la struttura degli incentivi, propria del modello open source; le aziende proprietarie non sono, in primo luogo, in grado di sfruttare i benefici derivanti dalla possibilità di impiegare programmatori già esperti: non essendo liberamente disponibile, il codice dei loro programmi non è in alcun modo utile a scopi didattici. In secondo luogo, non sono nella posizione di poter permettere ai propri utenti di modificare il codice senza mettere a repentaglio i diritti di proprietà intellettuale e, di conseguenza la loro maggiore fonte di profitto. Da ultimo, le grandi compagnie non sono in grado di replicare la visibilità che, grazie all'apporto di Internet, un certo programma può raggiungere in ambito open source.

Le aziende potrebbero, tuttavia, cercare di trasportare la struttura delle c.d. ricompense differite all'interno dell'ambiente proprietario; se è vero che la maggior parte delle compagnie fa in modo che i propri dipendenti non diventino eccessivamente visibili, onde evitare che altre compagnie possano assumerli, è anche vero che garantire loro una certa visibilità, finirebbe per attrarre altri talenti e per incentivare gli altri dipendenti. Un altro terreno in cui le aziende software possono cercare di emulare il metodo di sviluppo open source, è quello della diffusione del codice, quantomeno all'interno della compagnia[72]. Ciò ridurrebbe la possibilità che l'attività di scrittura del codice sia inutilmente duplicata e permetterebbe al programmatore di sottoporre il proprio lavoro a un pubblico più ampio.

Una seconda strategia, che può essere adottata dalle aziende produttrici di software, consiste, come ho accennato, nell'inserire il processo di sviluppo open source in ambiente proprietario. Per esempio, nel gennaio 1999, Hewlett Packard (HP), una delle principali aziende fornitrici di hardware, annunciò, per prima, l'intenzione di fornire ai suoi clienti soluzioni e servizi Internet basati su Linux. Sviluppò e rilascio, quindi, una serie di programmi open source, così da consentire alla comunità di Linux di adattare Linux all'architettura di HP.

Vari sforzi sono stati, inoltre, intrapresi dalle aziende produttrici di software per sviluppare prodotti addizionali attraverso un approccio tipicamente open source: uno degli sforzi più visibili, in questo senso, è stata la decisione di Netscape di rendere il suo browser liberamente disponibile. Anche la creazione di un'organizzazione come Collab.Net può essere interpretata come un tentativo d'inserire il modello di sviluppo open source all'interno delle aziende commerciali. Fondata dal gruppo Benchmark Partners,

[71] Vedi La Cattedrale e il Bazaar su www. Apogeonline.com/Openpress

[72] Non è raro il caso in cui regole aziendali interne vietino ai gruppi di programmatori di trasmettere il proprio lavoro ad altre squadre di programmatori.

Collab.Net è un'impresa che si propone di organizzare progetti open source per tutte quelle aziende che si prefiggono di sviluppare parte del software da loro prodotto secondo il modello di sviluppo open source. In particolare, oltre a gestire uno speciale mercato on-line, l'organizzazione si occupa di redigere i contratti per le aziende, di offrire aiuto per la risoluzione delle controversie. In definitiva, il progetto può essere letto come uno sforzo di certificare i programmi di sviluppo open source interni alle aziende

3. Molte aziende commerciali hanno elaborato strategie per capitalizzare sul movimento open source. In poche parole, si aspettano di ottenere un profitto in quei settori di mercato, in cui la cui domanda di prodotti software è alimentata dal successo di programmi open source complementari; non essendo possibile appropriarsi dei miglioramenti apportati ai prodotti open source, le compagnie commerciali cercano di beneficiarne indirettamente in un segmento di mercato complementare e proprietario.

Un'altra strategia si è dimostrata, tuttavia, più semplice e proficua: consiste nel fornire servizi complementari e prodotti che non sono offerti, in modo soddisfacente, dalla comunità open source; società come Red Hat e Va Linux incarnano perfettamente questa strategia.

Dal punto di vista delle aziende produttrici di software, una volta creato un prodotto che l'utenza vuole acquistare, produrre copie e distribuirle diventa un'attività dal costo marginale risibile. L'industria del software tradizionale si è, così, tradizionalmente concentrata sul perfezionamento del modello astratto di produzione, non curandosi di fornire sostegno tecnico agli utenti, che utilizzavano, in concreto, il prodotto.

Fino a tempi recenti, il concetto d'assistenza software è stato, pertanto, molto sottovalutato e considerato come un sottoprodotto, generato da errori commessi nel processo di produzione. In altre parole, la strategia adottata dalle grandi aziende produttrici di software è stata quella di massimizzare i profitti, riducendo al minimo gli investimenti sull'assistenza all'utenza. Questo stato di cose ha avuto ripercussioni negative, non solo sugli utenti, ma anche sulla qualità del software prodotto: caratteristiche semplici da implementare sono state spesso tralasciate perché giudicate non strategiche. In definitiva, le grandi aziende hanno sfruttato la proprietà intellettuale sui loro prodotti per imporre a milioni d'utenti le loro scelte.

Osservato dal punto di vista del modello open source, il concetto d'assistenza deve essere, tuttavia, rivalutato. Il software open source è, per definizione, gratuito, in quanto liberamente scaricabile da siti internet specializzati; non essendo, quindi, possibile ottenere un profitto attraverso concessione di licenze d'uso sul programma, l'unico modo per cercare di guadagnare è rappresentato dall'offerta di assistenza all'utenza. L'open source unisce gli sforzi dei programmatori di tutto il mondo: le aziende fornitrici di servizi commerciali (personalizzazioni, correzione di bug, assistenza), basate su tale software, hanno la possibilità di capitalizzare sull'economia di scala, sfruttando il largo richiamo di questo nuovo tipo di software e l'ampiezza del suo bacino d'utenza

Di fatto, con il software libero, il guadagno si genera esattamente nello stesso modo che con quello proprietario[73]. Diventa, quindi, di primaria importanza creare un prodotto di valore e commercializzarlo con accortezza e fantasia, prendendosi cura dei clienti; la costruzione di un marchio, che sia sinonimo di qualità e di servizio alla clientela, si rivela, pertanto, la strategia migliore. In particolare, un marketing oculato e ingegnoso suggerisce di offrire alla clientela soluzioni che i concorrenti non possono o non sono in grado di offrire; e proprio sotto questo punto di vista, il modello di sviluppo open source offre un considerevole vantaggio competitivo, producendo software stabile, flessibile e altamente personalizzabile, vale a dire un prodotto di qualità.

La risorsa di maggior valore per le società di software proprietario è la proprietà intellettuale, rappresentata dal codice sorgente del software che detengono. Non è questo il modello economico condiviso dalle nuove aziende specializzate nell'assistenza, dal momento che la loro attività non consiste nel distribuire licenze sulla proprietà intellettuale di cui è titolare. Si può, quindi, concludere che tali aziende non operano nel business del software nel senso tradizionale del termine. Il software open source è liberamente scaricabile dalla rete: le imprese che vogliono approfittare dell'ampio richiamo di questo tipo di programma non possono, quindi, vendere software, vale a dire concedere delle licenze d'uso sul programma. Ciò che vendono è, quindi, supporto: una valida offerta di sostegno tecnico genera un rapporto di fiducia tra venditore e cliente, e fa in modo che l'utente preferirà acquistare una versione ufficiale, fornita da rivenditori specializzati, piuttosto che scaricarla gratuitamente da Internet. Ci si potrebbe chiedere a questo punto, in che settore operano le aziende che forniscono supporto questo tipo di assistenza: la realtà mostra che il software

[73] La realtà dimostra che la maggior parte delle imprese software fallisce, a prescindere dal fatto che sia basata su software libero o proprietario. Dal momento che, fino a tempi recentissimi, le imprese software sono state solo di tipo proprietario, è possibile affermare che il modello IP (Intellectual Property) di sviluppo e marketing del software non garantisce il successo economico. Questo accade perché, se è vero che intraprendere un business nel mondo del software può generare un considerevole profitto, è anche vero che molti rischiano questa strada, solo per avere un'opportunità di successo.

appartiene, oggi, alla categoria dei prodotti di largo consumo; le aziende che intendano commerciare tale tipo di prodotto, potranno, quindi, fare ricorso alle tradizionali regole di politica del marchio associate ai beni di consumo di massa. La sfida che aziende come Red Hat o SuSe sono chiamate ad affrontare è, pertanto, quella di definire, nella mente dei consumatori, cosa sia il software open source: l'obiettivo è quello di creare un marchio che i clienti sceglieranno, fornendo e supportando un prodotto di qualità.

Un'oculata gestione del marchio impone, a questo punto, di posizionare strategicamente il prodotto sul mercato; prendiamo come esempio il mercato dei sistemi operativi: è decisamente affollato e dominato dalla Microsoft. Ardue sono, quindi, le sfide che un nuovo sistema operativo deve affrontare per conquistare una fetta di potenziali acquirenti, quantomeno significativa. Il vero vantaggio competitivo di cui i produttori di sistemi operativi proprietari godono, risiede nella possibilità di controllare il codice sorgente su cui, sia le loro applicazioni, sia quelle dei produttori indipendenti di software, devono essere eseguite. Il rimprovero più comune che l'utenza sembra muovere al del leader del mercato, concerne la sua posizione dominante e, di conseguenza, l'impossibilità di effettuare una libera scelta. Per evadere da questo schema, i produttori indipendenti di software devono ricorrere ad un modello in cui il produttore del sistema operativo non controlli quel sistema operativo. In questo campo, la vera sfida è, oggi, rappresentata da Linux, il quale, tuttavia, non è un vero e proprio sistema operativo, ma una raccolta di componenti open source. Una società come Red Hat prende, quindi, quelle che considera il meglio delle componenti open source possibili per costruire un sistema operativo efficiente. E così acquista compilatori da Cygnus, server Web da Apache, un X Windows System dall' X Consortium e li assembla in un sistema operativo Red Hat Linux certificato e garantito. Il compito di Red Hat è, in definitiva, quello di selezionare il software open source disponibile per farne un prodotto da immettere sul mercato; il controllo sul programma non è, in ogni caso, mantenuto da Red Hat, né da nessun altra azienda che opera in tale ambito. Se un cliente non apprezza la scelta operata da una qualsiasi delle aziende distributrici di software, ha in ogni momento la possibilità di operare una scelta diversa. Questo può succedere perché è proprio l'utente ad avere il controllo del sistema operativo che usa: un simile potere gli deriva dall'avere la licenza di poter aprire e modificare il codice sorgente. Il vantaggio offerto dal software open source risiede, quindi, nella possibilità, data all'utente, di controllare e gestire la tecnologia che utilizzano. Tale facoltà è resa possibile dal fatto che il software open source arriva completo del codice sorgente, può essere utilizzato per qualunque scopo, senza chiedere permessi. Il codice open source è, quindi, solo una caratteristica; il controllo è il vantaggio. Il software open source consente all'utente di scegliere la tecnologia che arriva dentro il computer, richiedendo, pertanto, un livello del tutto nuovo di responsabilità e perizia da parte dell'acquirente.

L'ampia fruibilità del codice dl programma produce un altro considerevole vantaggio: se un fornitore presenta un'innovazione che diventa popolare sul mercato, gli altri produttori la adotteranno immediatamente, e questo perché hanno accesso al codice sorgente di quell'innovazione. La vera potenza del movimento open source sta, quindi, nell'aver creato una pressione unificante verso uno standard unico, e aver rimosso le barriere della proprietà intellettuale, che avrebbero ostacolato questa convergenza.

In definitiva, il tratto fondamentale dell'open source, vale a dire la possibilità di modificare liberamente il software, non solo permette agli utenti di compiere scelte libere e consapevoli, ma abbatte le barriere all'ingresso del mercato, ponendo tutti i concorrenti sullo stesso livello: ne traggono vantaggio, non solo l'avanzamento della tecnologia, ma anche il mercato in generale, come conseguenza di un abbassamento dei prezzi.

Conclusioni

Il software open source e le società commerciali ad esso collegate sono l'investimento del futuro: rappresentano, infatti, la tecnologia su cui, più probabilmente, si concentreranno i capitali di rischio. L'open source non rappresenta, tuttavia, una novità: come abbiamo visto, il movimento per il software libero era già attivo nella seconda metà degli anni ottanta. Le grandi aziende hanno, tuttavia, cominciato ad intuire le grandi potenzialità di questo nuovo modello di sviluppo solo recentemente. Ciò è accaduto essenzialmente per due motivi. In primo luogo perché, storicamente, la parte più visibile e organizzata della cultura hacker, identificabile in Richard Stallman e nella Free Software Foundation, si è caratterizzata sia per uno spiccato fanatismo, sia per un atteggiamento ostile nei confronti del software commerciale; se è vero che la Free Software Foundation è stata per anni il fulcro del movimento per il software libero, producendo un gran numero di validi strumenti software e sponsorizzandone lo sviluppo, è anche vero che ha fatto sì che la cultura hacker fosse percepita dall'esterno come un movimento estremista e anticommerciale.

A tale approccio si contrappone, oggi, una fazione piu' pragmatica, che si schiera, solo moderatamente, contro le grandi aziende produttrici di software: ciò che viene a queste rimproverato non è tanto la chiusura del codice di per sé, quanto il rifiuto ad incorporare nei propri prodotti standard aperti e software open source[74]. Tale orientamento pragmatico si è identificato, nel corso degli anni ottanta e dei primi anni novanta, nei sostenitori delle versioni Unix di Berkeley e gli utenti della licenza BSB: tale fazione non ha, tuttavia, avuto peso e visibilità esterna fino all'esplosione di Linux nel 1993-1994. Linus si è, infatti, imposto all'attenzione globale come un esempio, appoggiando il crescente utilizzo di software open source in ambito industriale e il ricorso a software commerciale d'alta qualità per compiti specifici. Il cambio di rotta verso un atteggiamento più conciliante nei confronti del software commerciale si è, poi, palesato quando Netscape ha annunciato la propria volontà di distribuire pubblicamente i sorgenti del proprio prodotto di punta, il browser Communicator, risvegliando l'interesse del mondo commerciale nei confronti del free software; la cultura hacker, come abbiamo visto[75], si è dimostrata, in quel frangente, decisa a sfruttare l' occasione, ridefinendosi da free software a open source e adottando un'impostazione più adatta al contesto commerciale, in modo da consentire una più vasta diffusione del software libero e dei suoi prodotti.

La seconda ragione del tardivo interesse dell'industria verso l'open source deve essere ricercata nel modello di sviluppo adottato, per anni, dalle grandi aziende produttrici di software; queste non solo hanno posto i loro prodotti sotto copyright, distribuendo, poi, licenze sulla proprietà intellettuale di cui erano, e sono, titolari, ma hanno anche introdotto, nei contratti di lavoro, clausole negoziali in base alle quali tutto ciò che fosse stato sviluppato da un dipendente, sarebbe diventato automaticamente di proprietà dell'azienda. Con riguardo al rapporto tra dipendente e azienda, l'art. 12-bis l.a. prevede, tra l'altro, una presunzione di cessione automatica dei diritti d'utilizzazione economica sul programma creato dal lavoratore dipendente, nell'esecuzione delle proprie mansioni o su istruzioni impartite dal datore di lavoro, a favore di quest'ultimo. I produttori di programmi hanno, quindi, preferito utilizzare forme contrattuali attraverso le quali la proprietà del programma non sarebbe stata trasferita, concedendo in uso il programma dietro il pagamento di un canone forfetario o periodico, e imponendo contrattualmente all'utente diverse limitazioni, quali, per esempio, il divieto di riproduzione, modifica, cessione del programma a terzi. La licenza d'uso, che ha ad oggetto la volontà dell'autore di concedere il solo uso temporaneo del programma, ma che non implica una rinuncia dei diritti di sfruttamento economico dell'opera da parte dello stesso, è stata quindi, tradizionalmente, configurata come un contratto diretto a porre delle condizioni maggiori rispetto a quelle imposte dalla legge a tutela del possessore del copyright, a svantaggio di qualunque eventuale utilizzatore che intendesse oltrepassare i limiti posti. La risorsa di maggior valore per le società di software proprietario è, pertanto, ancora oggi, costituita dalla proprietà intellettuale, rappresentata dal codice sorgente del software che detengono. Questo non viene reso noto dalle aziende produttrici di software, che rilasciano i loro programmi esclusivamente in forma di codice binario, comprensibile solo alla macchina. Non potendo osservare il codice sorgente, l'utente è impossibilitato a comprendere il funzionamento del programma: di conseguenza non può neppure copiarlo o modificarlo; si tratta di facoltà riservate, dalla legge sul diritto d'autore (artt. 12-24), in esclusiva, al titolare del programma Il mercato del software è stato, quindi, finora un mercato monopolistico, con poche grandi aziende a fare da padrone: queste hanno, in pratica, sfruttato la proprietà intellettuale sui loro prodotti per imporre le loro scelte a milioni d'utenti. Il modello open source ha introdotto, a tale riguardo, una consistente novità: mentre le licenze della maggior parte dei programmi sono dirette a privare l'utente della possibilità di apportare modifiche al programma e di condividerlo con altri, le licenze open source attribuiscono al concessionario il diritto di copiarlo, modificarlo, diffonderlo e distribuirlo. In particolare, al fine di consentirne la modificabilità, la licenza open source rende l'opera

[74] La disputa KDE-GNOME rappresenta, per esempio, un interessante terreno di battaglia sul quale le due fazioni sono venute a scontrarsi. Vedi cap. 1, par. 23.
[75] Vedi cap. 1, par. 13

disponibile nella sua forma più accessibile, vale a dire sotto forma di codice sorgente. Nel contesto open source, quindi, lo strumento giuridico della licenza è utilizzato per tutelare il diritto dell'autore di rendere l'opera liberamente disponibile e modificabile per chiunque . Tutto ciò solleva importanti interrogativi giuridici, in primo luogo, riguardanti la titolarità dell'opera, che viene liberamente condivisa, attraverso Internet, tra un numero di utenti potenzialmente illimitato; la diffusa modificabilità del programma impone di chiedersi se l'autore debba essere considerato il singolo creatore dell'opera originale, oppure tutti coloro che collaborano al suo perfezionamento. Ammettere la titolarità del diritto d'autore in capo a tutti i soggetti che collaborano alla creazione dell'opera implica, infatti, la necessità di ricercare il consenso di tutti, ai fini dell'eventuale trasmissione di uno o più diritti d'utilizzazione dell'opera, in adesione alla disciplina dettata in tema di comunione (art. 1108 c.c.). Ciò accade quando i singoli contributi sono a tal punto compenetrati l'uno con l'altro, da essere inscindibili. Nel caso, in cui, al contrario, i contributi dei diversi programmatori siano tra loro scindibili e sia individuabile un soggetto cui attribuire il coordinamento e la direzione dell'opera complessivamente considerata, sembra possibile configurare il programma risultante come opera collettiva. In tal caso, la legge attribuisce a tale soggetto il diritto di utilizzare economicamente l'opera e, in particolare, il diritto di prima pubblicazione. Il diritto d'inedito, vale a dire la facoltà riconosciuta all'autore di separare l'opera dalla sua sfera intima, se e quando gli sembri opportuno, è diritto della personalità: non dovrebbe di conseguenza rientrare tra le facoltà d'utilizzazione economica dell'opera. Pur rimanendo nella sfera del diritto morale d'autore, il diritto di pubblicazione costituisce, tuttavia, il necessario presupposto per l'esercizio delle varie facoltà patrimoniali: è, pertanto, dotato anch'esso di un contenuto patrimoniale, anche se solo in potenza. Ciò significa che, se la licenza open source attribuisce, di regola, a tutti gli utilizzatori, il diritto di diffondere l'opera, tale facoltà è, nella pratica, limitata quando il contributo del singolo s'inserisca, sotto particolari condizioni, in un progetto più complesso.

Il fiorire del movimento open source ha messo in luce come, in realtà, qualcosa stia cambiando nel mondo del software: la più grande arena che le grandi aziende produttrici di software devono ancora conquistare, Internet, non presenta le restrizioni proprie del mercato del software tradizionale; costruita su una potente collezione di standard aperti, la rete è aggiornata grazie alla volontà di singoli individui, non ai capitali delle grandi aziende. Internet, infatti, può essere vista, non solo come il più potente mezzo di diffusione dell'informazione, ma anche come vera e propria impresa open source. Come strumento di diffusione, è riuscita a mettere in comunicazione i programmatori di tutto il mondo, favorendo la nascita di una comunità globale, all'interno della quale il sapere non può che essere condiviso. In campo più strettamente informatico, Internet ha facilitato la libera trasmissibilità dei programmi, inferendo un duro colpo alla tutela del diritto d'autore. Grazie all' innovazione tecnologica e all'avvento dei sistemi digitali, il software e, in generale, tutte le opere dell'ingegno, possono essere diffuse attraverso le reti telematiche, evitando i passaggi della catena produzione – distribuzione – vendita o noleggio del supporto che contiene l'opera stessa. Il software, quale bene immateriale, deve essere, infatti, incorporato in un supporto per essere economicamente sfruttato; la diffusione delle reti telematiche ha, quindi, determinato l'eliminazione dello strumento che, fino ad oggi, ha consentito agli autori di ottenere un ritorno economico dall'esercizio dei diritti d'utilizzazione dell'opera, a loro riservati in via esclusiva dalla legge. L'uso della rete consente, in ultima analisi, a chiunque sia ad essa collegata, non solo di ricevere, in qualunque momento, l'opera richiesta, ma anche di registrarla nella memoria del proprio computer, di riprodurla e di ritrasmetterla ad altri utenti. Tutti i diritti d'utilizzazione economica dell'opera dell'ingegno sono, in pratica, stravolti dall'introduzione del nuovo modello, fondato non più sul concetto d'esclusività della proprietà individuale, ma sulla condivisione del sapere. Mentre per il software tradizionale, si è posto il problema della tutela del diritto d'autore, il software open source, implicando la rinuncia all'esercizio dei diritti d'utilizzazione economica, ha, al contrario, utilizzato le immense potenzialità delle rete per diventare una realtà globale e consolidata. Tra l'altro, la legge sul diritto d'autore detta, con specifico riguardo al software, una disciplina particolare: l'ambito della privativa imposta all'utilizzatore è, infatti, più ampio rispetto a quelle prevista, in generale, per le opere dell'ingegno[76]. Ciò significa, di conseguenza, che ricorrendo ad una licenza open source, l'autore del programma rinuncia ad esercitare una serie di diritti di portata più ampia rispetto a quelli attribuiti al creatore di un'opera dell'ingegno, dalla legge sul diritto d'autore.

In ogni caso, la diffusione delle tecnologie digitali, ha aperto questioni giuridiche di non poco conto. La smaterializzazione del supporto ha, infatti, reso l'opera maggiormente soggetta a modificazioni, quindi più vulnerabile. La possibilità di apportare delle modifiche, attribuita all'utilizzatore del programma attraverso la licenza, incide, come abbiamo visto, sui diritti d'utilizzazione economica dell'opera. Ciò non è tutto: il diritto di modifica finisce, anche, per influire sul diritto morale d'autore, sebbene la licenza open source implichi soltanto la rinuncia ai diritti d'utilizzazione economica dell'opera, ma non al diritto morale d'autore, che rimane inalienabile e intrasmissibile. Tale diritto comprende, in particolare, la pretesa dell'autore a non vedere la sua opera snaturata, giacché espressione della propria personalità; anche la facoltà di modificare l'opera, attribuita in potenza da qualsiasi licenza open source, deve essere, quindi, rivalutata

[76] Vedi cap. 2 par. 6,7,9.

alla luce dei diritti morali d'autore, di cui è titolare il creatore dell'opera originale. In particolare, a norma dell'art. 2577 comma 2 cc., l'autore può, in qualsiasi momento, opporsi ad eventuali deformazioni, mutilazioni o altre modificazioni dell'opera, che possano essere di pregiudizio al suo onore o alla sua reputazione.

Internet, come ho detto, non è solo uno strumento di diffusione e condivisione della conoscenza, ma può essere considerata la vera impresa open source; è, inoltre, la testimonianza del potere del modello degli standard aperti: negli ultimi anni, infatti, i computer hanno compiuto un passo decisivo nell'avvicinarsi al modo in cui le persone comunicano tra loro. Il World Wide Web, ha fornito agli utenti un sistema ipertestuale d'interconnessione, attraverso il quale è possibile costruire siti web. Oggi, non solo i programmatori, ma anche i semplici utenti, costruiscono le interfacce delle loro applicazioni con le parole e le immagini, e non più attraverso controlli specializzati, comprensibili solo per chi ha studiato il software. Un sito web può, quindi, essere allestito da chiunque, giacché il software necessario è gratuito: le specifiche per la creazione di documenti e contenuti dinamici sono, infatti, semplici, aperte e complete di chiara documentazione. Cosa più importante, tanto la tecnologia che l'etica Internet rendono legittimo il copiare dai siti altrui. Una killer application, vale a dire un prodotto che apre porzioni di mercato inesplorate, oggi, non è più un programma di produttività individuale o un software aziendale, ma un singolo sito web. Se cominciamo a pensare ai siti web come applicazioni, si può concludere che essi costituiscano una specie del tutto nuova di software, diretta a computerizzare compiti ingestibili dal vecchio modello informatico.

Quello del software è, oggi, un segmento di mercato maturo, che offre poco spazio all'ulteriore innovazione: il fatto che gli attori in gioco, vale a dire le grandi aziende, abbiano un interesse enorme a mantenere lo status quo, rende loro difficile abbracciare alcunché di veramente nuovo. Il prevalere di Microsoft su IBM, nel dettar legge nell'industria informatica è un esempio classico di come ciò si sia svolto nel passato recente. Alla fine degli anni settanta, IBM ha perso la sua posizione dominante perché non ha colto il trapasso di potere dall'hardware al software. Allo stesso modo, le grandi aziende produttrici di software non si accorgono, al momento, che il software tradizionale non è più l'impulso centrale alla creazione di valore nell'industria informatica. Ai tempi del dominio IBM, l'hardware rappresentava il prodotto di punta del mercato informatico: quasi tutto il software era opera dei costruttori di hardware, o di produttori di software loro satelliti. La situazione è cambiata quando il pc è diventato disponibile come piattaforma e non più solo come marchio di fabbrica: tal evento ha abbassato le barriere d'entrata, permettendo ad una nuova ondata d'imprenditori di accedere al mercato del software. Guardando ai primordi del Web, si può distinguere un'impronta simile. Il monopolio delle grandi aziende software ha reso insostenibilmente alte le barriere d'entrata, escludendo, di fatto, la possibilità che il singolo programmatore produca, da solo, un qualche impatto sul mercato. E' questa, forse, l'osservazione più significativa a proposito del software open source: ha abbassato le barriere per entrare nel mercato del software. Un prodotto nuovo non solo può essere provato gratuitamente, ma può essere personalizzato, essendo il codice disponibile all'ispezione totale degli utenti. Il paradigma di sviluppo open source è una maniera incredibilmente efficiente di mettere gli sviluppatori al lavoro sulle funzioni che contano davvero. Il nuovo software è sviluppato in stretto feedback circolare con gli utenti e le loro necessità, evitando le distorsioni indotte dalle forze di marketing o da decisioni d'acquisto provenienti dall'alto. L'incredibile successo che l'open source sta riscuotendo oggi, rispetto al passato, è dovuto alla rapida diffusione delle informazioni resa possibile da Internet, la quale ha abbassato le barriere all'entrata dell'infrastruttura informatica: oggi il codice sorgente, che rappresenta il contenuto creativo del programma espresso in linguaggio comprensibile all'uomo, non è più distribuito su schede perforate, su floppy disk o su CD-ROM. La vendita di tali supporti avrebbe permesso all'autore di sfruttare economicamente la propria opera. Oggi qualsiasi server Web è una fonte di distribuzione economica praticamente istantanea.

La verità è che l'industria ha bisogno dell'open source, perché ha bisogno dell'innovazione che tale metodo di sviluppo può garantire. L'open source ha attirato l'attenzione delle più grandi società informatiche (IBM, Oracle), poiché è in grado di testare nuovo software con la velocità e la creatività della scienza; anche in campo informatico, infatti, l'innovazione avviene attraverso il metodo scientifico, il quale si fonda, essenzialmente, su un processo di scoperta e di dimostrazione. I risultati scientifici, per essere dimostrati, devono essere replicati: la riproduzione del risultato pone, come imprescindibile presupposto, la condivisione della conoscenza. In campo informatico, l'unico strumento che consente, a persone d'uguale competenza, di replicare i risultati e verificare la validità del programma, è rappresentato dalla distribuzione del codice sorgente. Nel modello di sviluppo open source, la libera distribuzione del codice sorgente del programma conduce, inoltre, a tutta una serie di vantaggi: innanzitutto alla creazione di codici più solidi; la libera disponibilità del programma, fa in modo, infatti, che questo sia sottoposto al controllo di un bacino d'utenza potenzialmente illimitato; ovviamente, maggiore è il numero di persone che si occupano del debugging, più efficiente risulterà il programma. La libera distribuzione del codice sorgente aumenta, inoltre, la creatività collettiva: la possibilità di osservare il funzionamento del programma può, infatti, fornire l'ispirazione per un nuovo progetto che non sarebbe stato concepito senza il primo. Se le grandi aziende guardano, quindi, al movimento open source, come alla fucina delle idee del domani, è tuttavia necessario garantire che i loro

prodotti siano effettivamente open source. E', quindi necessario evitare che queste rilascino software gratuito quel tanto che basta per attrarre utenti, ma senza garantire le piene libertà dell'open source. Qui emerge l'importanza d'istituzioni come Open Source Iniziative e Collab.Net. La prima, nel monitorare il corretto uso del termine open source e nel fornire una definizione adeguata del concetto per tutte quelle aziende interessate a realizzare programmi propri, garantisce che le licenze sotto le quali i programmi sono rilasciati, siano effettivamente open source. La seconda, proponendosi di organizzare progetti per le aziende, verifica che i programmi rispondano, effettivamente, ai canoni open source.

In questo contesto, il compito del giurista è quello d'inserire lo strumento principale attraverso il quale si esplica il fenomeno open source, la licenza, in precisi schemi giuridici. Sotto questo punto di vista, abbiamo visto che la recente giurisprudenza ha ammesso, anche se in termini confusi, la configurabilità della licenza come contratto atipico a titolo gratuito. Ciò sembra, tuttavia, possibile ad una condizione: che sia individuabile un interesse patrimoniale del disponente. Questa soluzione diventa, tuttavia, problematica sotto un duplice punto di vista; innanzitutto, l'individuazione di un interesse patrimoniale implica l'incorporazione del software in un supporto, che, come abbiamo visto, è il mezzo attraverso il quale l'autore è in grado di ottenere un ritorno economico dallo sfruttamento dell'opera dell'ingegno. Lo sviluppo delle tecnologie digitali ha, al contrario, determinato l'inutilità di tale supporto, consentendo la libera trasmissibilità dell'opera attraverso Internet. Diventa, quindi, di primaria importanza sganciare il concetto di gratuità da quello di soddisfacimento d'un interesse patrimoniale del disponente. Non bisogna, inoltre, dimenticare che nel modello di sviluppo open source, l'eventuale soddisfacimento di un interesse patrimoniale rappresenta un'esigenza secondaria rispetto alla ricerca del consenso e della stima degli altri individui appartenenti alla comunità. In questo senso, potrebbe essere utile la messa a punto di codici scritti che traducano quelle che, fino ad oggi, sono state le consuetudini proprie della comunità open source. In tal modo potrebbero essere individuate le pratiche cui ricorrere per la risoluzione dei vari conflitti che possono sorgere in connessione con i progetti open source.

L'esperienza degli ultimi anni ha dimostrato che il metodo di sviluppo open source rappresenta un modello vincente: come tale, è destinato ad essere sempre più impiegato dall'industria del software, che ha bisogno delle idee innovative che tale modello può portare. A tal fine, tuttavia, le grandi aziende produttrici di software sono chiamate ad adeguare le proprie metodologie di sviluppo alle caratteristiche del movimento open source; la creazione di nuovo software è destinata, infatti, a perdere la connotazione di occupazione, dal momento che il compenso monetario non identifica più l' incentivo principale. Il valore del lavoro sarà, quindi, dato dalla condivisione con gli altri. Proprio il valore della condivisione costringerà il legislatore a rivedere la disciplina relativa al software, nel momento in cui sarà chiamato a regolare il fenomeno open source. Fino ad oggi, l'inclusione del software nella categoria delle opere dell'ingegno, ha consentito all'autore del programma di godere di una serie di diritti attribuitigli in esclusiva dalla legge, limitando, in questo modo, le attività di riproduzione, modifica e distribuzione del programma da parte di terzi. In ultima analisi, il modello open source, basato sulla libera condivisione dei programmi attraverso Internet, richiederà l'eliminazione delle prerogative riconosciute all'autore, vanificando, quindi, il ruolo finora svolto dal diritto d'autore, o meglio dai diritti d'utilizzazione economica dell'opera.

BIBLIOGRAFIA

Libri

- AAVV, Open Source. Voci dalla Rivoluzione Open Source a cura di Sam Di Bona, Sam Ockman, Mark Stone, Milano, Apogeo, 1999.
- AAVV, Dizionario di Internet e computer, Milano, Il Sole 24 Ore, 2001.
- AAVV, No Copyright. Nuovi Diritti nel 2000, a cura di Raf Valvola Scelsi, Milano, Shake Edizioni Underground 1994.
- Fink Martin, Modelli di Business per Linux e Open Source, Milano, Paerson Education, 2003.
- Levy Steven, Hackers. Gli Eroi della Rivoluzione Informatica, Milano, Shake Edizioni Underground, 1996.
- Moody Glyn, Codice Ribelle. La Vera Storia di Linux e della Rivoluzione Open Source, Milano, Hops Libri, 2002.
- Williams Sam, Codice Libero. Free as in Freedom, Milano, Apogeo, 2003.

Articoli

- Alpa Guido, "Responsabilità extracontrattuale ed elaboratore elettronico", in *Diritto dell'Informatica e dell'Informazione*, 1986, p. 387.
- Auteri Paolo, "Internet e il contenuto del diritto d'autore", in *AIDA* 1996, p. 83
- Arcadu Giuseppe e Rinaldi Baccelli Guido, "Prodotto difettoso e responsabilità per danni derivanti dall' esercizio del software", da *Rivista del Diritto Commerciale e del Diritto Generale delle Obbligazioni*, 1992, p. 91.
- Catarinella Pierfrancesco, "Appunti comparatistici sul diritto d'autore in Internet" in *Diritto d'Autore*, 2003, p. 343.
- Cerina Paolo, "Contratti internazionali d'informatica e legge applicabile: prime considerazioni", in *Diritto dell'Informatica e dell'Informazione*, 1994, p. 405.
- De Nova Giorgio, "L' oggetto del "contratto d' informatica": considerazioni di metodo", in *Diritto dell'Informatica e dell'Informazione*, 1986, p. 803.
- De Vivo Maria Concetta, "L'informazione in rete: con che diritto?", in *Informatica e Diritto*, 1999, p. 137.
- Fabiani Mario, "Diritto d'autore e accesso a Internet", in *Diritto d'Autore*, 2001, p. 267.
- Frosini Vittorio, "Riflessioni sui contratti d'informatica", in *Informatica e Diritto*, 1996, p.167.
- Lerner Josh e Tirole Jean, "The scope of open source licensing", 2002, in http://www.nber.org/papers/w7600.
- Mirabelli Giuseppe, "I contratti d'informatica. Modelli, tipicità, collegamento", *Diritto dell'Informatica e dell'Informazione*, 1986, p. 791.
- Nadan Christian, "Open source licensing: virus or virtue?", 2002, in www.lexis.com
- Rinaldi Raffaele, "La tutela del software nel D.Lgs. 518/1992, in *Diritto dell'Informatica e dell'Informazione*, 1994, p. 259.
- Ristuccia Renzo e Zeno-Zencovich Vincenzo, "Prime notazioni sulla legge a protezione dl software", in *Diritto dell'Informatica e dell'Informazione*, 1994, p. 233.

- Viterbo Alfredo, "La rete: tecnologia di libertà?", in *Diritto dell'Informatica e dell'Informazione*, 2003, p. 219.

- Zeno-Zencovich Vincenzo "Informatica ed evoluzione del diritto", in *Diritto dell'Informatica e dell'Informazione*, 2003, p. 89.

- Zincone Andrea, "Come prevenire le problematiche inerenti all'utilizzazione del software", in *Diritto d'Autore*, 1993, p. 380.

Saggi:

- Raymond Eric, "Colonizzare la noosfera", 1998, traduzione italiana a cura di Bernardo Parrella, giugno 1999, su www.apogeonline.com/Openpress.

- Raymond Eric, "La cattedrale e il bazaar", 1998, traduzione italiana a cura di Bernardo Parrella, giugno 1999, su www.apogeonline.com/Openpress.

Giurisprudenza

a) Corte di Cassazione

- Corte di Cassazione, 13 dicembre 1999 n. 13937, in *AIDA* 2000, 659.
- Corte di Cassazione 22 marzo 1999 n. 2661, in *Contratti* 1999, p.992.
- Corte di Cassazione 18 febbraio 1993, in *AIDA* 1993, 133
- Corte di Cassazione, 20 novembre 1992, in *Foro italiano* 1993, I, 1506.
- Corte di Cassazione, 19 novembre 1990, in *Diritto dell'Informatica e dell'Informazione* 1991, p. 186.
- Corte di Cassazione, 21 giugno 1965 n. 1299, in *Giurisprudenza Italiana*, 1967, I, 1, 214.
- Corte di Cassazione, 21 ottobre 1957 n. 4004, in *Giurisprudenza Italiana,* I, 1, 187.

b) Tribunali minori

- Tribunale di Palermo, 29 maggio 1997, in *Diritto dell'Informatica e dell'Informazione,* 1998, p. 965, con commento di Marco Montalbano.
- Tribunale di Bari, 4 giugno 1994, in *Diritto dell'Informatica e dell'Informazione* 1995, p. 927, con commento di Emilio Tosi.
- Tribunale di Torino, 13 marzo 1993, in *Diritto dell'Informatica e dell'Informazione,* 1995, p. 382, con commento di Emilio Tosi.
- Tribunale di Milano, 20 ottobre 1988, in *Diritto dell'Informatica e dell'Informazione*, 1989, p. 549
- Tribunale di Rovereto, 28 febbraio 1985, in Diritto dell'Informatica e dell'Informazione, 1986, p. 590.
- Pretura di Roma, ordinanza 31 marzo 1992, in *AIDA* 1992, 99
- Pretura di Catania, ordinanza 11 febbraio 1992, in *AIDA* 1992, 89
- Pretura di Milano, ordinanza 10 novembre 1992, in *AIDA* 1993, 164
- Pretura di Milano, ordinanza del 25 ottobre 1991, in *AIDA* 1992, 67/3
- Pretura di Roma, ordinanza 3 luglio 1975, in IDA 1976, 143