Indice

0.1 Introduzione

Il fine che questo lavoro si propone è l'analisi, attraverso lo studio di alcuni concreti casi aziendali, dei vari aspetti che caratterizzano il modello economico su cui si basa la produzione di *Free Software* (software libero) e di *Software Open Source*[1] (software con codici sorgenti[2] aperti, liberamente disponibili); nonché dei possibili modelli di business applicabili.

Il software Free ed Open Source ha la particolarità di essere creato da una comunità di sviluppatori/utenti - detti *hacker*[3] - con un sistema di tipo aperto; è reso disponibile a chiunque, completo dei codici sorgenti e protetto da una tipologia di licenze che ne consentono il libero uso, la copia e la redistribuzione in una forma anche modificata, gratis o a pagamento, che sovverte i divieti dei copyright tradizionali. È soggetto ad una evoluzione molto rapida in diretta risposta alle esigenze dei suoi utenti.

La ricca tradizione culturale dei primi sviluppatori/hacker della comunità *Unix*, nata nei laboratori di ricerca delle più famose università americane, comprendeva abilità tecniche altamente specializzate, stabili reti di scambio delle informazioni, norme, gerarchie di status, linguaggi e significati simbolici condivisi. I vincoli informali che ne derivavano, ereditati dagli odierni programmatori delle diverse comunità open source, li incentivavano a partecipare ai progetti di sviluppo che avvenivano in rete attraverso ARPAnet - il primo network militare progenitore di Internet - a considerare i propri programmi come bene pubblico ed a rilasciarli a disposizione di chiunque. Incentivo fondamentale allo scambio di questi doni è il gioco della reputazione che dipende dal giudizio critico dei colleghi.

Le spinte commerciali subite dai primi Unix portarono ben presto alla famosa sfida tra il colosso AT&T/Bell Labs e l'ateneo di Berkeley con la sua Berkeley Software Distribution (BSD) con la vittoria di quest'ultima che

[1] Vedi Glossario
[2] Vedi Glossario
[3] Vedi Glossario

avrebbe portato alla nascita delle prime vere distribuzioni Unix sotto licenza libera: 386/BSD, NetBSD, OpenBSD e FreeBSD.

Accanto ai primi tentativi di commercializzazione di Unix proprietari, gli anni Ottanta videro anche la nascita del concetto di Software Libero - Free Software - e del movimento ad esso collegato capitanato da Richard M. Stallman, fondatore del *Progetto GNU* che prevedeva la costruzione di un intero sistema operativo composto da software libero che fosse compatibile con Unix in modo che potesse essere facilmente portabile - adattabile a diverse architetture hardware - e che gli utilizzatori di Unix non trovassero difficoltà nell'adottarlo.

Per evitare abusi e per poter conseguire il suo scopo di effettiva libertà Stallman, con l'aiuto di esperti legali, mise a punto la *GNU General Public License (GPL)*, una licenza che garantisse a chiunque avesse del software licenziato con essa il diritto e la libertà di usarlo, copiarlo, modificarlo e ridistribuirne le versioni modificate, ma a patto di acconsentire a renderne disponibili i codici sorgenti cioè la forma leggibile dall'uomo, e di non aggiungere restrizioni di qualsiasi tipo a tali versioni.

Negli stessi periodi venne anche creata la *Free Software Foundation (FSF)*, una fondazione con lo scopo di raccogliere fondi per il Progetto GNU attraverso sia donazioni che la vendita di pacchetti di software libero (sia GNU che non GNU) e la relativa manualistica.

Nel 1991 accadde ciò che viene considerato il punto di svolta per il software libero: uno studente all'Università di Helsinki, Linus Torvalds, iniziò a sviluppare sulla base del materiale della Free Software Foundation un kernel[4] libero compatibile Unix per processori 386 che chiamò *Linux*. Verso il 1992, lo combinò con il sistema GNU che mancava ancora di un kernel e mise il risultato in rete secondo le regole del software libero e di quello che nel corso di questo lavoro sarà chiamato Progetto Open Source [5]. Diede così il via ad un processo che attirò l'attenzione di molti hacker in Internet che cominciarono

[4]Vedi Glossario

[5]Vedi capitolo 3, pg. 57

a collaborare con lui, aiutandolo a sviluppare un sistema operativo completo, *GNU/Linux* (od anche solo Linux) con codici sorgenti completamente liberi e redistribuibili.

L'innovazione più importante portata da Linux è stata sicuramente il modo in cui si è sviluppato. In genere ogni sistema operativo o programma di una certa dimensione creato fino ad allora era stato concepito e sviluppato da singoli o gruppi relativamente piccoli di persone coordinati tra loro che lo rifinivano e mettevano a punto anche per lunghi periodi prima di rilasciarne una versione sufficientemente stabile. Linux invece ha sempre potuto contare sull'apporto di un gran numero di volontari coordinati attraverso Internet e sul continuo rilascio diffuso, talvolta anche giornaliero, di versioni aggiornate, comprese le iniziali ed instabili versioni *beta*[6] di solito destinate a pochi eletti. Tali versioni erano - e sono - poi sottoposte al lavoro incessante di verifica e correzione da parte della comunità di sviluppatori e contributori. Ciò ha garantito, secondo molti e a dispetto delle previsioni più diffuse, una qualità impensabile per chiunque fosse abituato a lavorare in qualsiasi altra maniera.

Linux, come altri software liberi od open source, è l'espressione di quella che si può definire *Cultura Hacker*, una (contro)cultura ereditata dalle prime comunità Unix, distribuita nella Rete e molto ricca e diversa, caratterizzata da abilità specializzate, reti di scambio di informazioni, norme, gerarchie di status, linguaggi e significati simbolici condivisi.

Chi si riconosce in questo tipo di cultura attinge quindi ad un background più che trentennale di conoscenze ed insegnamenti ormai dati per acquisiti. Una serie di regole di comportamento e di principï non scritti che l'individuo apprende mediante un processo sia di imitazione che di cosciente documentazione.

In questo tipo di cultura vi è un forte senso di appartenenza ad una comunità. Comunità che dal punto di vista sociologico ed antropologico sembra essere governata dalla *Cultura del dono* [Raymond], la cui caratteristica principale è la sostanziale assenza di problemi connessi alla scarsità di risorse dove

[6] Vedi Glossario

lo status sociale è dato non dal controllo, ma da quanto e cosa si pone in condivisione. In questo caso la risorsa è il software, la cui abbondanza è data dal fatto che è condiviso liberamente secondo gli usi radicati e i principï codificati della comunità. Lo status quindi sarà determinato dal personale contributo dell'individuo al fine che si propone la comunità.

Un importante metro che misura lo status e il successo competitivo in questa realtà, soprattutto nella sua dimensione più recente, è la reputazione. Si riscontra infatti nella comunità un sistema di leadership e regole cooperative implicite che hanno attratto l'interesse di sempre più sviluppatori grazie anche al fenomeno Linux ed alla crescita di Internet. Questo sistema e queste regole, nate abbastanza spontaneamente come conseguenza dell'aggregazione attorno a personaggi o gruppi di riferimento, rispondono al naturale bisogno dell'individuo di vedere in qualche modo ricompensato non solo il proprio lavoro ma anche il proprio atto di altruismo.

Altra componente del modello sociale della comunità direttamente collegata alla reputazione è la motivazione. È questo l'aspetto che aiuta a spiegare perché il software sviluppato in questo contesto e con queste metodologie non risponde ad alcune tra le più comuni caratteristiche del software sviluppato in modo tradizionale. Gli sviluppatori di software libero ed open source, infatti, non sono dipendenti di una società di produzione di software che trovano un incentivo ad occuparsi d'altro con l'aumentare del numero di colleghi o che possono soffrire a causa di imposizioni non condivise provenienti dall'alto.

Nel caso del software libero gli sviluppatori sono a loro volta gli utenti finali del software che, visto che gliene viene data l'opportunità, partecipano attivamente al confezionamento di un prodotto che li soddisfi maggiormente. Ovviamente, oltre all'interesse effettivo, un'altra discriminante è la concreta capacità tecnica che porta ad una sostanziale autoselezione degli sviluppatori.

La nascita del concetto di *Open Source* è un fenomeno abbastanza recente. Già da tempo alcune parti della comunità del software libero ritenevano che il mondo più tradizionalmente commerciale fosse tenuto lontano dalle istanze più radicali di Stallman e della sua Free Software Foundation. Si

perdeva, secondo loro, una grossa occasione non favorendo in qualche modo la diffusione degli ideali "liberi" nel mondo del business.

In questo contesto una *costola* della comunità si staccò dando vita a quella che oggi è conosciuta come *Open Source Initiative (OSI)*. Per cominciare il gruppo si trovò sostanzialmente d'accordo su un punto: ciò che era mancato al movimento del software libero fino ad allora era una concreta campagna di sensibilizzazione sulle proprie istanze. E ad aggravare il tutto, secondo loro, c'era anche la grossa confusione sul significato del termine *free* in free software, che molti interpretavano nel suo significato più immediato ad una prima lettura: gratis. Ecco percè fu coniato il nuovo termine *open source*. Per fornire il movimento di un *manifesto* fu stilata la *Open Source Definition*, una serie di principï e linee guida che una licenza software avrebbe dovuto seguire per essere definita open source.

La Free Software Foundation, con il suo Progetto GNU, e la Open Source Initiative, fanno capo dunque a due diversi movimenti. Sebbene il software libero e l'open source condividano alcune regole e concetti di base, i loro obbiettivi sono abbastanza differenti, pur convivendo nella comunità in una pacifica dialettica.

L'idea fondamentale alla base dell'open source è che *se* si può, tramite Internet o altri mezzi, leggere, distribuire e modificare liberamente del software, grazie alla libera disponibilità dei codici sorgenti, questo migliora e si evolve; viene trasformato e depurato dai cosiddetti bug, gli errori, grazie all'apporto di decine, centinaia e a volte migliaia di sviluppatori o appassionati e ciò avviene ad una velocità impensabile rispetto ai modi tradizionali di produrre software, producendone di migliore.

Il concetto di software libero pone invece l'accento sulla libertà, in particolare sulle quattro libertà fondamentali formalizzate nella Licenza GPL che vedremo nel secondo Capitolo.

In sostanza si tratta di due visioni con priorità diverse: l'una - open source - più pragmatica e possibilista che ha come scopo principale quello di produrre software di maggior qualità; l'altra - software libero - più radicale

e intransigente che ha come imperativo assoluto la produzione di software libero al 100%, senza compromessi.

Linux è oggi utilizzato come sistema operativo, secondo recenti stime [Schenk], da circa 20 milioni di persone e gode del più alto tasso di crescita tra i sistemi operativi. È inoltre il sistema operativo piu usato nel Web (adottato dal 31% dei server contro il 24% di Windows, secondo). Tra i singoli programmi quello che ha avuto più successo è invece Apache, di fatto il più usato tra tutti i programmi web server col 61% sul totale[7]. L'infrastruttura di Internet quindi si affida prevalentemente a software open source e si stima che ogni messaggio di posta elettronica spedito in rete passi attraverso Sendmail[8].

Parte del merito, oltre alla facile reperibilità del codice, si deve al metodo di sviluppo di questo tipo di software che permette a contributori sparsi per il globo di lavorare al codice, aggiungendovi caratteristiche e correggendone le mancanze.

Il software libero ed open source è sviluppato secondo le modalità di quello che viene in genere definito Progetto Open Source[9] che si può definire come *"[un] gruppo di persone che sviluppa software e rende disponibili i propri risultati al pubblico sotto una licenza open source..."* [Evers].

Attualmente sono diverse le categorie di soggetti che sviluppano software secondo queste modalità, tra le principali si possono annoverare: istituzioni scolastiche, istituzioni di ricerca, distributori di software, imprese commerciali, utenti organizzati, utenti privati, governi e pubbliche amministrazioni.

Non ci sono delle regole che stabiliscano i modi di svolgimento tipici di un Progetto Open Source, sono comunque osservabili dei punti in comune, dei paralleli nel modo in cui in genere inizia e si sviluppa un tipico Progetto Open Source. La prima fase è di solito caratterizzata da un iniziale lavoro di

[7] `http://www.netcraft.com/survey`

[8] Server per il trasporto della posta elettronica, sviluppato in origine alla University of California di Berkeley e mantenuto poi da una comunità di sviluppatori free, usato dal 75% circa tra tutti i siti Internet - ogni messaggio attraversa almeno due server

[9] Dove il termine open source identifica non il movimento OSI in particolare, ma la caratteristica fondamentale ed irrinunciabile della disponibilità dei codici sorgenti

sviluppo svolto in prevalenza da singoli utenti o comunque da piccoli gruppi organizzati di utenti privati mossi dalla presenza di un problema da risolvere o di una necessità da soddisfare.

La diffusione dei primi risultati dello sviluppo iniziale tramite i tipici canali in Internet può dare il via ad un fenomeno di interessamento e collaborazione da parte di terzi che vanno ad infoltire il gruppo originario. I continui rilasci dei risultati successivi instaura un circolo virtuoso che rappresenta il Progetto Open Source.

Tra i possibili vantaggi derivanti da questo metodo di sviluppo il più grosso ed inequivocabile sembra essere proprio la disponibilità dei codici sorgenti. Gli effetti immediati di questa fondamentale caratteristica sono, tra gli altri: una continua revisione del codice fatta da molte teste diverse, la possibilità per l'utente non solo di segnalare errori ma anche di fornire modifiche per correggerli, un reale sviluppo distribuito.

Dal punto di vista organizzativo un Progetto Open Source per molti versi si discosta dal modo in cui è organizzato un normale progetto di software proprietario. Quest'ultimo in genere è promosso e sviluppato dal personale di una azienda o da suoi collaboratori sfruttando una serie di risorse e strutture messe a disposizione dall'organizzazione, requisiti questi che assumono un aspetto radicalmente diverso nell'ambito di un Progetto Open Source.

Un Progetto Open Source attinge ad una serie di risorse che potremmo definire atipiche e che per questo sono inizialmente scarsamente individuabili.

L'informazione, in generale, è una delle maggiori risorse per lo sviluppo del software libero, infatti il software non è altro che un particolare tipo di informazione. L'uso dell'informazione è regolato dalle leggi sulla proprietà intellettuale che proteggono l'autore di un'opera di ingegno attribuendogli una serie di diritti su di esso. Per quanto riguarda le opere protette da licenze libere od open source, l'autore, decidendo volontariamente di rimuovere i vincoli alla loro trasmissione, le rende di fatto *legalmente disponibili*. Ed il fatto che non sia materiale deteriorabile, rende la risorsa informazione non vincolata una risorsa inesauribile. Il software è però usato in modo diverso

da ogni altro tipo di informazione e per questo le leggi che lo regolano sono differenti. Tuttavia quando si parla di software libero ed open source, come per le informazioni disponibili senza vincoli, il problema non si pone. Si può porre invece parlando di informazione e software sottoposti a vincoli.

In genere la maggior parte dei Progetti Open Source non dispone di proprie risorse da distribuire e condividere, a parte il risultato del proprio lavoro, il proprio sito e una o più mailing list, il tutto spesso ospitato e/o sponsorizzato da istituzioni educative come le università o da altri tipi di organizzazione interessate al progetto. Ecco quindi che un Progetto Open Source in genere si basa in buona parte sulle risorse personali dei partecipanti.

Esiste anche la possibilità di fondare persone legali come organizzazioni non-profit o associazioni di vario tipo a cui intestare la titolarità di eventuali risorse condivise. Tuttavia ciò comporterebbe un lavoro *amministrativo* poco gradito, ecco perché non sono molti gli esempi di questo tipo.

L'hardware è una risorsa fondamentale addirittura per l'esistenza di un Progetto Open Source e la maggior parte di essa coincide con la categoria delle risorse personali. È possibile distinguere, come vedremo, tre gruppi di hardware in base ad aspetti come proprietà, ubicazione, accesso fisico, accesso remoto ed amministratore: l'hardware personale del singolo sviluppatore, l'hardware generalmente disponibile a tutti i partecipanti al Progetto e l'hardware parzialmente disponibile.

L'aspetto delle risorse umane è fondamentale. La maggior parte dei partecipanti ad un Progetto Open Source in genere vi prende parte in modo volontario e senza essere pagato. Chi sviluppa software per un'azienda, fornisce solo la sua professionalità. L'azienda fornisce l'attrezzatura, l'ambiente, le infrastrutture e tutto ciò che serve, compreso l'aggiornamento e benefici come l'assicurazione od altro. I partecipanti ad un Progetto Open Source in genere utilizzano, come detto, le proprie risorse personali.

Le risorse finanziarie non sono considerate fondamentali in questo contesto. In genere i tipici Progetti Open Source, organizzati su basi volontarie, non hanno entrate o spese. Anche quando entrano in campo imprese interes-

sate ad un Progetto, esse normalmente pagano degli sviluppatori per lavorarci - non il Progetto direttamente - o sponsorizzano delle attività connesse - come meeting, pubblicazioni, ecc - o forniscono infrastrutture web.

Come ogni sistema complesso, anche per il Progetto Open Source si pongono dei problemi di coordinazione per far sì che i vari elementi che lo conpongono funzionino in modo efficiente. La domanda è se possono essere identificati dei processi coordinativi in un contesto di questo genere e quanto essi sono consapevolmente attuati.

Nel terzo capitolo saranno analizzati i processi coordinativi del tipico Progetto Open Source alla luce delle osservazioni fatte da Thomas K. Malone e Kevin Crowston nel loro *The Interdisciplinary Study of Coordination* [Malone]. In particolare sarà dato risalto all'osservazione di aspetti quali: la gestione delle risorse condivise, la gestione delle relazioni produttore/consumatore, la gestione dei vincoli di simultaneità e la gestione delle dipendenze tra compiti e sottocompiti.

Per ciò che riguarda la struttura gerarchica, i Progetti Open Source non sono in genere caratterizzati da un'autorità centrale che ha poteri organizzativi e dispositivi, tuttavia non sono sistemi anarchici. Ci sono dei ruoli ormai riconosciuti ed altri desumibili dal contesto, che permettono un certo grado di organizzazione: il Project Manager del progetto, il mantainer di una specifica componente del progetto, l'amministratore delle risorse - come il sistema di supporto, il software di supporto e i servizi a disposizione del Progetto -, lo sviluppatore del codice e l'utente attivo che non dispone delle capacità dello sviluppatore.

È importante notare che spesso più d'uno tra i ruoli appena elencati possono ritrovarsi nella stessa persona come, al contrario, più persone possono rivestire lo stesso ruolo.

Un Progetto Open Source, come qualsiasi forma di collaborazione organizzata, è caratterizzato inoltre da alcuni strumenti e da alcuni mezzi di comunicazione che permettono lo svolgimento e la coordinazione delle attività che in genere si svolgono *in remoto*. Il mezzo principale di comunicazione

quindi è Internet, al quale sono collegati altri mezzi quali la *posta elettronica*, le *mailing list*, *usenet* ed *IRC*[10].

Tra gli strumenti principali troviamo il *Concurrent Versions System (CVS)* ed il *Bug*[11] *Tracking System*, il primo e un sistema di controllo delle versioni e permette di organizzare e registrare la storia delle modifiche ad un codice sorgente, il secondo è un sistema attraverso il quale è possibile tenere traccia e gestire tutte le segnalazioni sui difetti di un software.

Tuttavia un problema fondamentale nel cercare di capire, stimare e trarre delle conclusioni sul fenomeno dello sviluppo open source è la scarsezza di informazioni su chi sono, ma soprattutto quanti sono esattamente coloro che partecipano a questo sviluppo ed in che modo vi partecipano. Questa scarsezza di informazioni - paradossale se confrontata con la mole di righe di codice e di documentazione prodotti - genera una cronica sottostima delle reali dimensioni di un processo collaborativo che si presenta organico e distribuito.

I dati più recenti a riguardo provengono da uno studio [UNC] del 1999 compiuto dall'Open Source Research Team presso la University of North Carolina (UNC) sulla comunità di utenti/sviluppatori di Linux. Linux infatti è forse l'esempio più significativo di sviluppo di un progetto open source su larga scala. La sua comunità è molto attiva, geograficamente distribuita ed impegnata nella diffusione del software e della documentazione correlata.

Un elemento di enorme utilità per la raccolta e consultazione di questi dati è dato dall'esistenza del Linux Repository, un sito ftp[12] ed uno http[13] ospitati presso il MetaLab di UNC, che raccoglie praticamente tutto il materiale riguardante Linux disponibile in Internet, comprese le più inportanti distribuzioni del codice del kernel, il Linux Documentation Project sulla documentazione, tutto il software collegato e il materiale di supporto.

Lo studio, di cui saranno illustrati i risultati alla fine del terzo Capitolo di

[10] Vedi Glossario
[11] Vedi Glossario
[12] `ftp://metalab.unc.edu`
[13] `http://metalab.unc.edu/pub/Linux`

questo lavoro, conferma una larga partecipazione allo sviluppo delle applicazioni e delle utility - meno complicate rispetto a strutture come il kernel - con un'importante componente europea - riflettendo forse le origini geografiche e culturali di Linux - ed un contributo dalle istituzioni educative che rimane massiccio.

Negli ultimi due anni si è sviluppato un crescente interesse nei confronti di questo tipo di software che ha visto instaurarsi un processo che coinvolge una moltitudine di sviluppatori in tutto il mondo (spesso appartenenti a diverse organizzazioni) nello scambiarsi e condividere i codici sorgenti per sviluppare e migliorare programmi.

Dei vari fattori che hanno favorito l'affermarsi di questo fenomeno, tre in particolare sono quelli considerati da molti i più significativi [Lerner]:

- *La rapida diffusione del software libero ed open source.* Si stima che solo GNU/Linux abbia avuto dal 1998 ad oggi una crescita annua nel numero di utilizzatori del 200% su scala mondiale[14]. Diversi sono inoltre i prodotti open source che dominano la propria categoria, si pensi ad esempio appunto al programma Apache tra i web server in Internet

- *Significativi investimenti di capitale in Progetti Open Source.* Negli ultimi due anni molte tra le maggiori compagnie, tra cui Hewlett Packard, Sun Microsystems, Compaq e IBM hanno investito molto in progetti di sviluppo ed utilizzo di software open source; addirittura dati recenti - Gennaio 2001[15]- parlano per quest'ultima di un terzo degli investimenti per il 2001 destinati allo sviluppo di piattaforme basate su GNU/Linux, mentre Hewlett Packard ha deciso di avvalersi dell'interfaccia grafica Ximian Gnome sviluppata da Ximian per alcuni suoi prodotti[16]. Diverse compagnie specializzate nella commercializzazione di GNU/Linux, come Red Hat e VA Linux hanno raggiunto il traguardo della quotazione al Nasdaq americano, mentre altre compagnie open

[14] `http://www.netcraft.com/survey`
[15] `http://www.linuxdevices.com`
[16] `http://www.ximian.com`

source più orientate ai servizi, come Cobalt Networks, CollabNet, Sendmail e Linuxcare hanno ricevuto cospicui finanziamenti da società di venture capital.

- *Un nuovo modello organizzativo.* La natura collaborativa propria dello sviluppo del software open source rappresenta un fenomeno al quale molti, a cominciare dai media specializzati, guardano come ad una significativa innovazione nell'ambito delle organizzazioni. In molti casi si sta imponendo come valida alternativa al software proprietario sia come prodotto in sé sia come metodo di sviluppo e filosofia di affari; il suo modello collaborativo e di condivisione favorisce la partnership aziendale. Le fonti di guadagno si spostano dalla vendita del prodotto ai servizi ed al supporto tecnico per l'utenza.

Lo sviluppo e l'uso di software libero ed open source trasforma il confine tra impresa ed utente finale che diviene sfumato e costituisce un enorme passaggio di potere dalle imprese ai consumatori.

Il nuovo modello di business legato a questo fenomeno è quello dell'Impresa Open Source che si basa tra l'altro su un patto etico - imposto, più che voluto, dal funzionamento delle licenze libere ed open source - che tutte le parti devono rispettare considerando che la *materia prima* è donata all'azienda dalla comunità e che la logica del meccanismo del dono prevede che questo venga ricambiato.

La struttura dei costi e dei ricavi delle aziende fondate sul software libero o comunque open source è completamente diversa rispetto a quelle tradizionali: la soglia di ingresso è estremamente bassa e basso il costo di produzione ma il bene offerto può essere reperito facilmente e a bassissimi costi o quasi dai possibili clienti. Dal lato del cliente, valutazioni sul Total Cost of Ownership (TCO) [TCO] fanno pendere l'ago della bilancia verso l'acquisto di software libero e open source, innanzitutto per la possibilità di escludere il costo per copia del prodotto, in secondo luogo per la minore componente di costi di sviluppo incorporata. Tutto ciò senza considerare le valutazioni sul-

la maggiore qualità media di questo tipo di software in termini di stabilità, efficienza e scalabilità.

In realtà ciò che viene offerto sul mercato non sono beni ma servizi, il valore di vendita è indiretto. Ad esempio le distribuzioni Linux commerciali come SuSE, Red Hat, Debian, Caldera, ecc. non vendono i singoli bit del sistema operativo ma il valore aggiunto nell'assemblaggio dei diversi software che lo compongono e le garanzie di funzionamento e compatibilità con altri sistemi della stessa marca oltre all'assistenza gratuita per un certo periodo di tempo ed opzioni per l'estensione dell'assistenza stessa. I distributori Linux, inoltre, sono costretti a competere con modalità che apportano benefici ai consumatori e al mercato in generale: infatti la licenza GPL li obbliga a condividere le migliorie e le nuove caratteristiche via via sviluppate che prontamente vengono inserite nei propri pacchetti dalle altre distribuzioni ed evita le frammentazioni.

Alcune aziende fondate sull'Open Source si concentrano prevalentemente sui servizi (supporto, ricerca, sviluppo e formazione) come Linuxcare oppure sulla vendita di documentazione, come ad esempio i manuali sul software open source della casa editrice americana O'Reilly & Associates o della giovane ed italianissima Hops Libri.

Un'altra caratteristica a vantaggio degli utenti ma non immediatamente evidente dei software open source è che questi potranno essere sempre supportati anche se l'azienda che li ha distribuiti è scomparsa o se i programmatori che li hanno sviluppati all'interno di un'azienda non fanno più parte del suo organico: senza la disponibilità dei sorgenti ed il permesso di modifica non sarebbe possibile. Anche la ripartizione del rischio, quindi, può far optare le aziende e gli utenti per questo tipo di software, non solo la ripartizione dei costi di sviluppo.

I vari modelli esistenti di business legati al mondo del software libero ed open source corrispondono in sostanza alle varie tipologie di guadagno legate allo sfruttamento di questo prodotto che vedremo più in dettaglio nel quarto Capitolo e riprenderemo nelle Conclusioni:

- *Distribuzioni Software e supporto.* I distributori vendono copie di software Open Source costruite apposta per essere accessibili anche ai meno esperti o per particolari utilizzi. Accanto alle distribuzioni inoltre sono forniti a pagamento soluzioni di supporto post vendita e una serie di servizi del tipo che vedremo nella seguente categoria.

- *Servizi.* Alcune aziende si specializzano sui servizi. I servizi più diffusi tra quelli prestati sono supporto, customizzazione di software, training e bug-fixing (correzione di bug) a pagamento.

- *Prodotti civetta.* La libera distribuzione del codice sorgente di un prodotto che si posiziona in una fetta di mercato porta l'attenzione verso altri prodotti a pagamento, liberi o proprietari, che costituiscono la vera fonte di guadagno dell'azienda.

- *Software per il funzionamento di hardware.* Alcuni produttori e distributori di hardware, anziché investire somme considerevoli per sviluppare ed aggiornare il software correlato (driver compilatori e linker, applicazioni o interi sistemi operativi), tendono a rilasciarlo come software open source per stimolare la ricerca di maggior compatibilità e supporto ai lori prodotti, aumentandone in questo modo le vendite (come un prodotto civetta).

- *Documentazione.* Manuali, libri, riviste e servizi di news, tutto materiale che diffonde informazioni e quant'altro sul software libero ed open source ai normali prezzi per questi tipi di supporti. La caratteristica di questa documentazione è che spesso è resa anche disponibile liberamente in rete.

In questo lavoro saranno inoltre analizzati a grandi linee alcuni casi di giovani imprese che hanno fatto dell'idea del software libero la ragione della propria avventura imprenditoriale, tutte appartenenti alla categoria delle imprese fornitrici di servizi e supporto tecnico:

- *Prosa s.r.l* (ora Ascensit s.r.l.). Nata dall'idea di un gruppo di amici, che con poche risorse e molta buona volontà è riuscita a farsi conoscere a livello internazionale con *ETLinux*, un software sviluppato - tuttora - secondo le regole dell'open source. La particolarità di Prosa sta nel fatto di essere obbligata dal proprio statuto a sviluppare ed a trattare esclusivamente software libero.

- *Mixad Live s.r.l.*. Un'azienda torinese nata come *spin off* di Mixad (agenzia pubblicitaria) che offre consulenze e lavori di analisi e programmazione di alto livello basandosi anch'essa sul software libero. Tra i suoi lavori più significativi la realizzazione del portale `Punto.it` e `tradinglab.com` di Unicredito per il trading on line.

- *Ximian inc.*, ex HelixCode inc. Una company di Cambridge, Massachusets, basata su venture capital ed ancora nella fase di finanziamento che fonda la propria attività sullo sviluppo di una propria suite di software libero *Ximian Gnome* basata su *Gnome*, un ambiente applicativo libero per sistemi operativi Unix. Il business che intende sviluppare nel prossimo futuro è quello dei servizi collegati al software attualmente prodotto e liberamente distribuito.

Al termine di questo lavoro saranno proposte alcune riflessioni sia personali, sia sulla scorta di quanto la recente letteratura ha rilevato riguardo il fenomeno del software libero ed open source.

Nonostante personalmente mi senta vicino alle istanze ed alla filosofia che sono alla base di questo movimento, ho cercato di proporre i fatti e le argomentazioni nel modo più obbiettivo possibile, non mancando di rilevare le contraddizioni, le fratture ed i punti deboli di una realtà che, per quanto giovane, è troppo spesso rappresentata con toni eccessivamente positivi e trionfalistici soprattutto dagli entusiasti dell'ultim'ora.

La mia intenzione quindi, con questo lavoro, è di rendere un quadro il più fedele possibile di un fenomeno sia culturale che economico che si sta affermando negli ultimi anni, sottolineandone ove possibile le implicazioni

organizzative ed aziendali alla luce del suo recente affacciarsi nel mondo delle imprese.

Capitolo 1

La storia dell'Open Source

1.1 Dagli albori a Unix

Contrariamente a quanto molti oggi potrebbero pensare, il concetto di software proprietario è abbastanza recente, ma soprattutto era quasi sconosciuto ai primi programmatori, coloro che dall'*ENIAC* - primo computer della storia - in poi hanno fatto la storia dei primi pionieristici esperimenti di programmazione.

Agli albori dell'informatica infatti, da 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; nasceva soprattutto all'interno degli istituti di ingegneria e di fisica delle università per merito di programmatori entusiasti, precursori degli hacker, il cui rapporto col software era di puro piacere ed ogni nuova realizzazione o miglioramento era qualcosa da condividere accrescendo soddisfazione e prestigio personali.

1.1.1 Nascita della Cultura Hacker

È datata 1961 quella che per molti ha rappresentato la svolta epocale nel mondo del software: la nascita della *Cultura Hacker*. In quell'anno infatti il Massachussets Institute of Technology (MIT) ha acquistato la sua prima

macchina, il PDP-1 della Digital Equipment Corporation (DEC), divenuta ben presto il passatempo preferito di un nucleo di studiosi che costituivano l'embrione di quello che sarebbe ben presto diventato l'Artificial Intelligence Laboratory (AI Lab), forse uno dei più influenti e produttivi centri di ricerca in questo campo. Il termine stesso di hacker, inteso nella sua accezione originaria positiva di *"qualcuno a cui piace programmare e a cui piace essere chiaro a riguardo"* [Richard M. Stallman], nasce in questo contesto.

L'ambiente di riferimento era quello dei centri di ricerca dei più famosi atenei statunitensi quali appunto il MIT, Stanford University, Canergie-Mellon University, Berkeley ed altri nonchè le strutture di ricerca di importanti società come i Laboratori di Bell e il Palo Alto Research Center di Xerox. Ben presto, dal 1969, tutti questi centri di ricerca poterono usufruire della progenitrice di Internet: *ARPAnet*, evoluzione del progetto di network transcontinentale voluto dal Dipartimento della Difesa degli Stati Uniti - nato come DARPA (Defence Advanced Reseasch Project Agency) - che si rivelò un eccezionale veicolo di scambio di informazioni tra università e laboratori di ricerca, permettendo la collaborazione tra le migliori menti del pianeta, con una vera spinta per la crescita della conoscenza tecnologica, soprattutto nel campo del software.

Ben presto i ricercatori cominciarono a condividere un certo senso di appartenenza ad una comunità e ad una cultura comune, sentendo il bisogno di divulgare ciò che scoprivano e avendone in cambio altre informazioni importanti per il proprio lavoro. Il software prodotto era scambiato liberamente o in cambio di somme irrisorie a titolo di contributo per le spese e spesso chi lo riceveva operava innovazioni e cambiamenti che poi rendeva disponibili a sua volta.

Soprattutto quando l'evoluzione tecnologica nel campo dell'hardware portò alla progettazione e produzione di macchine sempre più potenti, molti di questi gruppi di ricerca decisero di sostituire i sistemi operativi in dotazione, lenti, vetusti e poco affidabili, per svilupparne di originali, più consoni alle esigenze contingenti e più soddisfacenti, anche a livello narcisistico. Anche

quest'ultimo aspetto, oltre all'amore per la ricerca scientifica, va considerato per capire le motivazioni che portarono all'esigenza di divulgazione che caratterizzò quel periodo.

Non è da sottovalutare l'enorme contributo, pur per ovvi motivi interessato, venuto dal Dipartimento della Difesa Americano, che chiuse deliberatamente un occhio sul proliferare delle mailing list su ARPAnet. Queste erano e sono tuttora lo strumento principe di cooperazione e scambio tra gli sviluppatori od anche i semplici amanti di questa cultura. DARPA capì che valeva la pena sopportare dei maggiori costi generali pur di poter attrarre un'intera generazione di giovani brillanti ed appassionati nel campo dei computer e della programmazione... e ovviamente i suoi scopi non erano certo ispirati ad un mero spirito di mecenatismo.

1.1.2 Unix e il linguaggio C

Il 1969 fu anche l'anno in cui Ken Thompson dei Bell Labs di AT&T cominciò a sviluppare Unix nell'intento di realizzare un sistema operativo più semplice da usare, da programmare e quindi più efficace e produttivo rispetto ai complessi sistemi operativi esistenti. Il fatto che nel frattempo un altro *hacker* collega di Thompson alla Dell, Dennis Ritchie, concepisse un nuovo linguaggio di programmazione chiamato *C* da usare sotto Unix che avesse le stesse caratteristiche di flessibilità e semplicità, fece sì che si potesse realizzare per la prima volta il concetto di *portabilità*[1] su cui si concentravano molti degli sforzi comuni di sviluppo dell'epoca. Cioè Unix poteva essere modificato in modo da poter funzionare con le stesse caratteristiche e capacità su macchine diverse. Non era più necessario quindi ridisegnare un intero pacchetto di software ogni volta che una macchina diventava obsoleta e si doveva cambiare con un'altra più moderna.

Sia Unix che C avevano delle caratteristiche rivoluzionarie per l'epoca. Si basavano su un concetto di semplicità estraneo al modo di programmare un po' barocco ed involuto comune a quel tempo. L'intera struttura logica

[1] Dal termine inglese *portability*

di C poteva essere, più o meno facilmente, memorizzata dal programmatore - al contrario della maggior parte degli altri linguaggi - consentendo un ridotto ricorso ai manuali. Mentre Unix era a sua volta strutturato, come un pacchetto flessibile di semplici programmi/moduli[2] pensati per combinarsi in vari modi a seconda delle esigenze, superando il concetto imperante di un unico blocco disegnato specificatamente e su misura per una determinata macchina.

Queste caratteristiche fecero si che Unix fosse presto adottato dalla maggior parte delle università, dei laboratori di ricerca informatica ma soprattutto da una moltitudine di hacker. Unix infatti aveva integrata 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 stimolo al fenomeno della diffusione e condivisione del software. Anche se questo metodo di trasmissione risultava lento, consentiva comunque a qualsiasi utilizzatore di questo sistema operativo di comunicare senza per forza dover accedere ad ARPAnet, appannaggio esclusivo al tempo, come già detto, di pochi soggetti istituzionali ben determinati.

La nascita di una vera e propria comunità di utenti Unix fu anche facilitata da una circostanza particolare. AT&T, il colosso telefonico a cui facevano 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

[2]Vedi Glossario

chiusa, senza cioè nessun servizio correlato né di 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, e ciò fu facilitato dagli strumenti di comunicazione cui ho appena accennato.

È da annotare a margine che in questi anni, più precisamente nel 1975, fu commercializzato il primo *Personal Computer (PC)* - Apple fu fondata appena due anni dopo, nel 1977 - dando vita ad un'altra generazione parallela di hacker il cui linguaggio era il *BASIC*... ma questa è un'altra storia.

1.1.3 Prime spinte commerciali

Ben presto il consistente movimento che si era formato attorno a Unix con la creazione e gli indubbi progressi di alcune sue distribuzioni - in particolare la famosa *Berkeley Software Distribution (BSD)*, nata presso l'ateneo di Berkeley - fecero sorgere un certo interesse per le sue potenzialità commerciali.

Fino a questo momento i computer erano sempre stati 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 od al riuso del software. AT&T stessa, per i motivi sopra spiegati, non aveva mai mostrato fino ad 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, vennero percepiti da AT&T come un fenomeno utile ma sostanzialmente incontrollabile e poten-

zialmente dannoso a livello economico. Fu così che la licenza che accompagnava nel 1979 la *Settima Versione* di Unix limitava lo studio del codice sorgente, bloccando il suo sviluppo in diversi ambienti universitari.

Con questa azione AT&T aveva intrapreso 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; tutti i seguenti rilasci di Unix AT&T furono sviluppati e gestiti da un altro gruppo con espliciti fini commerciali.

1.1.4 La Berkeley Software Distribution (BSD)

La *Settima Versione* del 1979 dei laboratori Bell segnò in qualche modo anche la storia delle distribuzioni sviluppate presso l'Università di Berkeley.

Fino ad allora l'ateneo californiano aveva sviluppato, a partire dal materiale AT&T/Bell delle proprie distribuzioni Unix: la Berkeley Software Distribution (BSD) nel 1977, e la Second Berkeley Software Distribution (2BSD) nel 1978. Nel 1979 venne portato a termine con successo il tentativo di porting della 2BSD sulla nuova macchina della DEC, il VAX 32bit, dando origine a quella che è conosciuta come la 3BSD che segnò il definitivo distacco delle successive Berkeley Distributions dal percorso seguito da AT&T.

Le versioni successive alla 4BSD furono caratterizzate dall'aggiunta del decimale dopo il 4 per evitare possibili confusioni con le versioni AT&T. Un problema con cui Berkeley dovette scontrarsi fino alla versione 4.3BSD fu che necessitava comunque della costosa licenza AT&T, su cui si basava in origine.

1.1.5 Nascita degli Unix liberi

Nel frattempo Berkeley aveva implementato autonomamente il protocollo di rete TCP/IP[3] ed altri strumenti di connessione integrandoli nella propria distribuzione. Tuttavia l'aumento dei costi della licenza originaria AT&T

[3]Vedi Glossario

portarono ad una crescente richiesta da parte di vari soggetti commerciali di avere la disponibilità dei soli protocolli di rete per poter sviluppare dei prodotti specifici da vendere separatamente dal sistema operativo.

Fu così che nel giugno del 1989 Berkeley rilasciò il suo codice di connessione e le utility di supporto sotto il nome di *Networking Release 1*. Si trattava del primo codice liberamente redistribuibile rilasciato da Berkeley; la licenza, in seguito conosciuta come *BSD License*[4], lasciava la libertà di distribuire il codice modificato o meno, in forma sorgente o binaria, senza obbligare alcuno al pagamento di successive royalties a Berkeley. L'unico obbligo era quello di riportare ogni volta nel codice sorgente le note inerenti al copyright originario e i contributori. Berkeley vendeva le copie di questo codice a 1000 dollari l'una, ma chiunque poteva liberamente riceverne una copia da chi già ne possedeva una, favorendone così la diffusione.

Il successo della *Networking Release 1* spinse i suoi sviluppatori a gettarsi con entusiasmo nell'ambizioso progetto di riscrivere da zero tutti i pacchetti Unix per liberarsi definitivamente dei legacci imposti dalla licenza originaria AT&T. Ciò richiese parecchi mesi ed il coinvolgimento di molti appassionati ricompensati solamente dalla soddisfazione di poter vedere il proprio nome con quello degli sviluppatori di Berkeley accanto al nome dell'applicazione riscritta. Il risultato fu la *Networking Release 2*, questa volta un intero sistema operativo funzionante distribuito da Berkeley ancora per 1000 dollari a copia sotto la medesima licenza della precedente versione. Non tardò ad uscire anche una distribuzione dedicata ai più recenti PC basati su architettura 386: la *386/BSD*.

Il successo della distribuzione di Berkeley e la sua libertà finalmente raggiunta, portò alla nascita di parecchi gruppi di utilizzatori che unirono le loro forze con il fine, inizialmente, di mantenere la distribuzione, e poi di svilupparla. Tra i gruppi che ebbero più successo e che legarono il loro nome a delle particolari distribuzioni il primo fu il *NetBSD* che partì dalla 386/BSD per sviluppare un sistema, il NetBSD appunto, che supportasse il maggior

[4]Vedi sezione 2.4.1, pg. 50

numero di piattaforme possibili. Poco dopo fu la volta del *FreeBSD* che si concentrò soprattutto sull'architettura PC. Verso la metà degli anni '90 una costola di NetBSD, *OpenBSD*, si concentrò sul problema della sicurezza dei sistemi.

Accanto a questi gruppi di aggregazione spontanea, nello stesso periodo nacque una società, la *Berkeley Software Design Incorporated (BSDI)*, che intendeva sviluppare e distribuire una versione commerciale supportata del sistema operativo.

Poco dopo l'inizio della sua attività, BSDI fu citata in giudizio da *Unix System Laboratories (USL)*, una controllata di AT&T dedicata alla vendita e allo sviluppo di Unix. USL riteneva che BSDI utilizzasse il suo codice proprietario nonchè segreti industriali. Visto che ad una verifica il giudice constatò che BSDI utilizzava codice liberamente distribuito dall'Università della California più sei files originari che sarebbero stati presto modificati, le richieste di USL furono rigettate e l'Università stessa colse l'occasione di ricorrere in giudizio contro USL per l'aver omesso di citare i copiright richiesti dalla Licenza BSD dopo aver usato codice BSD nella propria distrubuzione di Unix denominata *System V*. Nel frattempo USL fu acquisita da *Novell* che preferì raggiungere un accordo.

1.1.6 Mancato successo degli Unix proprietari

A parte la realtà variegata e articolata delle comunità di utilizzatori, il successo commerciale di Unix nelle sue varie versioni non fu mai esplosivo ed anzi, negli ultimi anni vede un lento ma costante declino.

Gli anni ottanta furono caratterizzati dal tentativo da parte di diversi soggetti di commercializzare versioni proprietarie di Unix, in particolare verso la fine del decennio e gli inizi del successivo tali sforzi furono rivolti al suo porting verso i PC. Alcune distribuzioni si basavano sulle versioni AT&T, altre sulle BSD, ma il problema, a parte un iniziale successo, fu che il prezzo delle licenze tendeva a rimanere alto anche dopo l'eliminazione della dipendenza dalla licenza originaria AT&T e - la cosa peggiore - non venivano resi

disponibili i codici sorgenti, rendendone quindi impossibile o quantomeno molto complessa la modificabilità e la redistribuibilità. Quest'ultimo aspetto deriva da quella che i propugnatori del software libero hanno sempre considerato una pecca della Licenza BSD, e cioè il fatto che rendesse possibile la redistribuibilità del software anche in forma di codice binario[5] e quindi non leggibile.

Tutto ciò dimostrò che i produttori di versioni proprietarie di Unix avevano messo in atto strategie di mercato ottuse, dispersive e poco costruttive, ignorando la risorsa rappresentata dall'universo di utilizzatori/sviluppatori di fatto *in rete* già diversi anni prima dell'avvento di Internet. Queste lacune portarono all'affermarsi sul mercato di un sistema operativo obiettivamente inferiore a livello tecnologico come Microsoft Windows.

1.2 Il movimento free software

Accanto ai primi tentativi di commercializzazione di Unix proprietari, gli anni Ottanta videro anche la nascita del concetto di Software Libero - *Free Software* - e del movimento ad esso collegato capitanato da Richard M. Stallman, fondatore della *Free Software Foundation (FSF)* e del *Progetto GNU*.

1.2.1 Il Progetto GNU

Per contrastare l'insinuarsi di spinte proprietarie nel mondo fino ad allora libero di Unix, Stallman si propose di produrre del software, in particolare un intero sistema operativo su cui poter fare girare tale software, su basi libere. Questo per consentire la rinascita e la sopravvivenza di quella comunità di condivisione nata spontaneamente ai tempi di ARPAnet che purtroppo era stata quasi soffocata dalle spinte proprietarie e dal dilagare di vincoli e licenze che aveva invaso il mercato del software e, di riflesso, anche gli ambienti accademici e di ricerca.

[5] Vedi Glossario

Nel 1984, dopo aver lasciato l'AI Lab del MIT per non sottostare agli inevitabili vincoli che sarebbero stati posti al suo lavoro ivi prodotto, Stallman diede il via a ciò che chiamò *GNU Project* dall'acronimo ricorsivo *GNU's Not Unix*[6]. Intendeva infatti sviluppare un sistema che fosse compatibile con Unix in modo che potesse essere facilmente portabile e che gli utilizzatori di Unix non trovassero difficoltà nell'adottarlo.

Questo sistema GNU non doveva necessariamente essere composto da software originale GNU - sarebbe stata un'enorme perdita di tempo lavorare in quell'ottica - ma avrebbe incluso all'occorrenza anche programmi già esistenti, a patto che fossero software libero e completo dei codici sorgenti[7]. Era questo infatti il comune denominatore dichiarato dell'intero progetto GNU. Ma un software libero e di pubblico dominio avrebbe rischiato di essere utilizzato da qualcuno che lo avrebbe potuto inserire in un programma proprietario, da qui la necessità di proteggerlo con licenze e copyright appositi che ne conservassero le caratteristiche di libertà.

1.2.2 La GNU General Public License (GPL)

Per poter conseguire il suo scopo di effettiva libertà Stallman, con l'aiuto di esperti legali, mise a punto la *GNU General Public License (GPL)*[8], una licenza che, come si vedrà in dettaglio più avanti, mirava e mira tuttora a garantire a chiunque possieda del software sotto di essa il diritto e la libertà di usarlo, copiarlo, modificarlo e ridistribuirne le versioni modificate, a patto di acconsentire a renderne disponibili i codici sorgenti e di non aggiungere restrizioni di qualsiasi tipo a tali versioni.

I termini contrattuali di questa licenza distinguono il software libero da quello conosciuto come shareware - del quale sono resi disponibili solo i codici binari e non i sorgenti, in genere per un periodo limitato di prova -, e dal software di pubblico dominio - sul quale non sono poste restrizioni di alcuna

[6]Trad. it.: GNU non è Unix

[7]Vedi Glossario

[8]Vedi sezione 2.3.2, pg. 46

natura per i successivi utilizzatori del codice sorgente, neppure per evitare abusi.

1.2.3 La Free Software Foundation

Nel 1985 venne anche creata la *Free Software Foundation (FSF)*, una fondazione che ha lo scopo di raccogliere fondi attraverso sia donazioni che la vendita di pacchetti di software libero (sia GNU che non GNU) e la relativa manualistica.

Dei documenti relativi sia alla Free Software Foundation che alla GNU General Public Licence parlerò più diffusamente in seguito[9].

1.3 La nascita di Linux

1.3.1 Il problema del kernel: GNU/Hurd

Per tornare al tema di questo breve excursus storico, durante il 1990 il sistema GNU era quasi completo, l'unico componente fondamentale mancante era il kernel, cioè il cuore del sistema operativo. Gli sviluppatori che negli anni avevano affiancato Stallman avevano deciso di utilizzare come base per quello che sarebbe stato chiamato *GNU/Hurd* (il kernel GNU, appunto), il Mach, un microkernel sviluppato presso la Canergie-Mellon University e la University of Utah. Lo sviluppo di HURD subì numerosi ritardi per l'attesa del rilascio di Mach come software libero, come promesso.

1.3.2 La svolta: Lunus Torvalds

Fortunatamente ben presto avvenne ciò che viene generalmente considerato il punto di svolta per l'intero movimento del software libero. Nel 1991, infatti, uno studente all'Università di Helsinki, Linus Torvalds, iniziò a sviluppare sulla base del materiale della Free Software Foundation un kernel

[9]Vedi capitolo 2, pg. 39

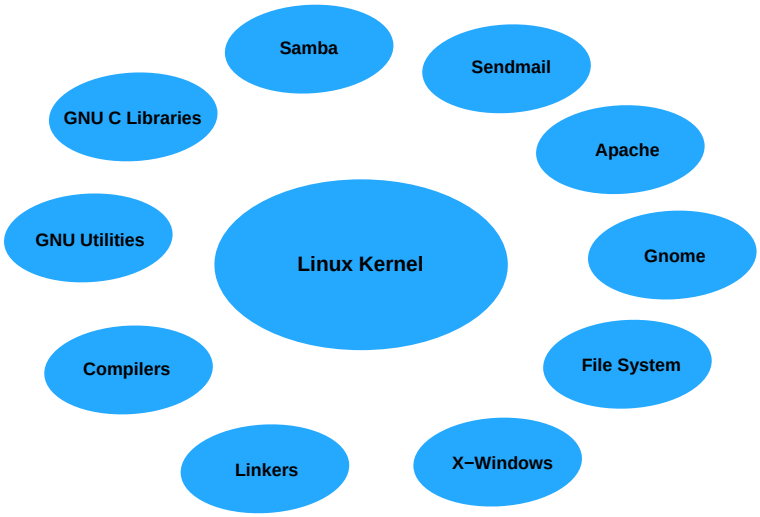

Figura 1.1: Architettura tipica di un sistema GNU/Linux, non molto diverso dagli altri Unix

libero compatibile Unix per processori 386 e lo chiamò Linux. Verso il 1992, lo combinò con il sistema GNU che mancava ancora, appunto, di un kernel e mise il risultato in rete secondo le regole del software libero e di quello che più avanti chiameremo *Progetto Open Source*[10]. Diede così il via ad un processo che attirò l'attenzione di molti hacker in Internet che cominciarono a collaborare con Torvalds, aiutandolo a sviluppare un sistema operativo completo con codici sorgenti completamente liberi e redistribuibili. Il risultato è quello che oggi è universalmente conosciuto come GNU/Linux, vedi Figura 1.1 (od anche, oramai, solamente Linux) su cui si basano sostanzialmente tutte le distribuzioni linux attuali.

1.3.3 Un nuovo metodo di sviluppo

La caratteristica più originale di Linux non fu tanto la creazione da parte di Torvalds del kernel in sé, pur meritoria di plauso, quanto il modo in cui

[10]Vedi capitolo 3, pg. 57

nacque, si sviluppò e si sviluppa tuttora; fu certamente questo modello di sviluppo l'invenzione più importante.

In genere ogni sistema operativo o programma di una certa dimensione creato fino ad allora era stato concepito e sviluppato da singoli o gruppi relativamente piccoli di persone coordinati tra loro che lo rifinivano e mettevano a punto anche per lunghi periodi prima di rilasciarne una versione sufficientemente stabile. Linux invece ha sempre potuto contare sull'apporto di un gran numero di volontari coordinati attraverso Internet e sul continuo rilascio diffuso, talvolta anche giornaliero, di versioni aggiornate, comprese le iniziali ed instabili versioni *beta*, di solito destinate a pochi eletti. Tali versioni erano - e sono - poi sottoposte al lavoro incessante di verifica e correzione da parte della comunità di sviluppatori e contributori. Ciò ha garantito, secondo molti e a dispetto delle previsioni più diffuse, una qualità impensabile per chiunque fosse abituato a lavorare in qualsiasi altra maniera.

Questi due approcci appena descritti sono quelli che Eric S. Raymond definisce con una azzeccata metafora rispettivamente "*a cattedrale*" ed "*a bazaar*"[Raymond1]. Raymond è considerato da molti uno dei massimi teorici del fenomeno del software libero - o meglio della sua successiva evoluzione che lui ha chiamato Open Source Software -, autore di numerosi lavori sulla storia, gli usi e i costumi della comunità hacker - come *Jargon File* [Jargon], *The New Hacker's Dictionary* [Raymond2], *The Cathedral and the Bazaar* [Raymond1] - nonchè egli stesso hacker della prima ora. In un capitolo del suo lavoro più conosciuto, The Cathedral and the Bazaar, egli analizza il successo di un Progetto Open Source, Fetchmail[11] - un programma che preleva la posta elettronica da un server remoto mandandola al proprio client di distribuzione - sviluppato intenzionalmente secondo le modalità sperimentate da Lunus Torvalds con Linux. In questo modo Raymond individua due teorie corrispondenti ad altrettanti fondamentali stili di sviluppo di software: quello *a Cattedrale*, proprio della maggior parte del mondo commerciale e caratterizzato, come detto, dal lavoro di un team relativamente ristretto e

[11] Vedi paragrafo 3.1, pg. 57

chiuso con poche occasioni di interazione con l'utenza finale se non dopo il momento del primo rilascio definitivo; e quello *a Bazaar*, consacrato da Linux e tipico del mondo open source, dove ad un team abbastanza aperto di sviluppatori si affianca il contributo a volte di una moltitudine di utilizzatori che contribuiscono in modo critico alla crescita del programma, grazie alla sua precoce disponibilità e fruibilità sin dalle versioni più sperimentali.

Il fatto che nel giro di un paio di anni Linux potesse competere in stabilità ed affidabilità con i più importanti Unix commerciali, contribuì all'uscita di scena dei distributori più piccoli e deboli di quei sistemi proprietari.

1.3.4 Linux oggi

Linux è oggi utilizzato come sistema operativo, secondo recenti stime [Schenk], da circa 20 milioni li persone e gode del più alto tasso di crescita tra i sistemi operativi. Parte del merito si deve alla facile reperibilità del codice ed al metodo di sviluppo open source, che permette a contributori sparsi per il globo di lavorare al kernel, aggiungendovi caratteristiche e correggendone le mancanze.

Basta semplicemente dare un'occhata al file *Mantainers* allegato al codice sorgente del solo kernel per rendersi conto di quanto è crasciuto il progetto. Quello che è iniziato dal progetto di un singolo sviluppatore si è ingrandito fino a contare più di 100 sviluppatori attivi su scala mondiale, senza contare le migliaia di simpatizzanti che in qualche modo collaborano.

Inoltre si contano una lunga serie di progetti paralleli non inerenti il kernel, ognuno dei quali sviluppato da diversi sviluppatori, che lavorano sui software e le applicazioni che corredano il sistema operativo.

1.3.5 Le distribuzioni

Negli ultimi anni sono state sviluppate diverse distribuzioni di Linux, sia commerciali che non commerciali. Una distribuzione Linux consiste nel kernel Linux più il software di supporto, la gran parte del quale è stato svi-

luppato dal Progetto GNU - infatti quasi tutte le distribuzioni si basano su GNU/Linux. Il software di supporto include compilatori[12], shell[13] e tutta una serie di altre utility e applicazioni che fanno uso del kernel.

A differenza dei tradizionali sistemi operativi, con linux è possibile scegliere tra diverse distribuzioni, ognuna delle quali fornisce, oltre al kernel, come detto, una collezione di applicazioni, utility ed un software di installazione. Tali distribuzioni sono, appunto, distribuite da soggetti che possono essere un'entità commerciale - come per esempio Red Hat -, o gruppi di volontari - come il gruppo Debian.

Tra le tante distribuzioni disponibili, quelle che in qualche modo si distinguono sia per notorietà che 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 di installazione, gli strumenti per l'amministrazione del sistema e altro. Tra le differenze più significative ci sono i mercati di riferimento, cioè alcune distribuzioni sono più esplicitamente orientate verso determinati tipi di utente, così l'utente meno esperto si indirizzeà verso Red Hat o SuSE, mentre l'utente di una power Workstation sceglierà Turbo Linux Workstation Edition.

Debian merita una citazione a parte. Se le altre distribuzioni sono prodotte da aziende che le commercializzano attraverso i normali canali di vendita, Debian si basa prevalentemente sul lavoro volontario di comunità di appassionati che mettono a punto, aggiornano ed implementano una propria distribuzione libera: Debian GNU/Linux. Debian è un progetto iniziato nell'Agosto del 1993 da Ian Murdock per creare una nuova distribuzione *aperta*, non commerciale e conforme allo spirito di Linux e del Progetto GNU. L'iniziativa fu finanziata dalla Free Software Foundation e dal Progetto GNU per un anno. Il piccolo gruppo iniziale di hacker si è negli anni accresciuto fino a diventare una comunità organizzata di utenti e di sviluppatori.

[12]Vedi Glossario
[13]Vedi Glossario

1.4 Un modello sociale

La distribuzione libera di Linux, come di altri software di questo tipo, ha sempre generato nell'osservatore esterno una serie di dubbi e quesiti, a partire dall'equivoco storico sulla sua presunta gratuità.

Già dagli esordi, come accadde per Unix ed altri software nati in ambienti di condivisione, chi organizzava un pacchetto di software per spedirlo a chiunque ne faceva richiesta, poteva facilmente pretendere una somma per copia che andava dal semplice rimborso spese al vero e proprio prezzo fissato con intenti lucrativi. Stallman stesso ad esempio, per autofinanziarsi durante i primi tempi del Progetto GNU chiedeva 150 dollari per ogni copia richiesta via posta ordinaria del suo famoso editor *GNU Emacs*, pur lasciandolo in libera distribuzione in un server ftp[14] per chi lo avesse voluto scaricare dalla rete.

Tuttavia ciò che ad una lettura superficiale stride in un sistema di sviluppo di software del genere è l'apparente difficoltà nel riuscire a trovare una giustificazione anche solo economica a tutto ciò.

Dato per acquisito il risultato qualitativamente superiore di questo sistema se ben applicato - esistono infatti dubbi sulla sua validità in generale[15] - e sorvolando sulle dispute riguardo la *profittabilità* derivante dalla pratica della condivisione e libera distribuzione del software e dal rifiuto del concetto di licenza chiusa, sulle quali torneremo più avanti, è necessario considerare il contesto *sociale* in cui nasce questo fenomeno e i meccanismi che lo governano.

1.4.1 La cultura hacker

> *"I vincoli informali contano. Per dare migliori risposte si deve conoscere molto di più sulle norme di comportamento derivanti*

[14] Vedi Glossario

[15] Vedi paragrafo 5.1, pg. 133

> *dalla tradizione culturale e su come interagiscono con le regole formali."* [North]

Il punto di partenza è la cosiddetta *cultura hacker* e le sue tradizioni, cui abbiamo già accennato, la cui analisi permette di comprendere le dinamiche del fenomeno che andiamo ad osservare e le trasformazioni che ha provocato nelle regole del mercato del software.

Come già detto in questo lavoro il termine hacker è inteso nel suo significato positivo di entusiasta appassionato di software, non accettando il significato negativo di sabotatore che l'uso della lingua gli ha oramai attribuito e per il quale esiste da sempre un termine specifico: *Cracker*.

L'identità dell'hacker e il forte senso di appartenenza che scaturisce da essa, crea una (contro)cultura molto ricca e diversa, comprendente abilità altamente specializzate, reti organizzate di scambio di informazioni, norme, gerarchie di status, linguaggi e significati simbolici condivisi. Gli elementi di tale identità contribuiscono a determinare le caratteristiche principali del comportamento che, se scisso da tali caratteristiche, risulta facilmente incomprensibile a chi ne è estraneo.

A parte la propensione personale a sentirsi - e definirsi - o meno un hacker, chi si riconosce in questo tipo di cultura attinge ad un background più che trentennale di conoscenze ed insegnamenti ormai dati per acquisiti. Una serie di regole di comportamento e di principï non scritti che l'individuo apprende mediante un processo sia di imitazione che di coscente documentazione.

Agli albori, per i primi sparuti gruppi di ricercatori/programmatori, la condivisione delle conoscenze era ovvia ed implicita, uno strumento indispensabile per il progresso scientifico. L'avvento delle licenze e delle varie limitazioni sul software, crearono in quegli stessi ambienti la nascita della consapevolezza dell'esistenza di un problema connesso a tutto ciò, il che fece crescere in molti la consapevolezza di appartenere ad un gruppo, ad una comunità caratterizzata da un comune sentire. ARPAnet funzionò da cassa di risonanza e Unix contribuì a veicolare un certo modo di agire e pensare anche al di fuori di ambienti accademici e di ricerca. La recente esplosione di

Internet non ha fatto che rendere esponenziale la crescita di questa comunità, testimoniata dalla forte espansione di Linux.

1.4.2 La comunità e gli incentivi impliciti

Perché è proprio di comunità che si deve parlare riferendosi agli esponenti di questa cultura. Ovviamente il fatto che gli appartenenti a questa comunità provengano per lo più dal mondo dell'informatica la rende particolare e singolare.

Sociologi e antropologi probabilmente definirebbero questa variegata comunità come un esempio di applicazione della *Cultura del dono*, contrapposta alle altre due principali: Gerarchia di comando e Economia di scambio [Raymond1]. Caratteristica principale della cultura del dono è la sostanziale assenza di problemi connessi alla scarsità di risorse dove lo status sociale è dato non dal controllo, ma da quanto e cosa si pone in condivisione.

In questo caso la risorsa è il software, la cui abbondanza è data dal fatto che è condiviso liberamente secondo gli usi radicati e i principï codificati della comunità. Lo status quindi sarà determinato dal personale contributo dell'individuo al fine che si propone la comunità.

Ci si chiede a questo punto quale sia l'elemento che spinge qualcuno a comportarsi in questo modo e a contribuire alla continua crescita di questo modello sociale, pur convinto delle istanze alla base di tutto ciò e spinto dalla passione per programmare. Qual è il sistema sociale di incentivi impliciti proprio di questa particolare comunità?

Un importante metro che misura lo status e il successo competitivo in questa realtà, soprattutto nella sua dimensione più recente, è la *reputazione*. Anche se non è certo sufficiente per spiegare tutte le caratteristiche della cultura hacker.

Se infatti, almeno nella fase embrionale di questa cultura, era ancora prevalente in alcuni contesti limitati (Bell Labs, MIT AI Lab, Berkeley, ecc) ciò che Weimberg definisce *programmare senza ego*[16] [Weimberg], cioè per il

[16]Dall'inglese *egoless programming*

solo piacere personale derivante dall'atto di programmare, già dalla nascita del progetto GNU ma soprattutto con l'avvento di Linux, si è imposto nella comunità un sistema di leadership e regole cooperative implicite che hanno attratto l'interesse di sempre più sviluppatori grazie anche alla crescita di Internet. Questo sistema e queste regole, nate abbastanza spontaneamente come conseguenza dell'aggregazione attorno a personaggi o gruppi di riferimento, rispondono al naturale bisogno dell'individuo di vedere in qualche modo ricompensato non solo il proprio lavoro ma anche il proprio atto di altruismo.

Un esempio chiarificatore, riportato anche da Raymond in *The Cathedral and The Bazaar* [Raymond1], è ancora una volta il modello di sviluppo di Linux. Linus Torvalds, in qualità di coordinatore del progetto, dimostrò di aver saputo trarre il massimo risultato con uno sforzo ridotto, motivando il lavoro della schiera di utilizzatori/sviluppatori dando loro l'opportunità di soddisfare il loro ego collaborando, anche poco, al progetto e ricompensandoli con la citazione dei contributi effettivi e con un costante rilascio di versioni migliorate grazie al loro lavoro.

Altra componente direttamente collegata alla reputazione è la *motivazione*. È questo l'aspetto che aiuta a spiegare perché il software sviluppato in questo contesto e con queste metodologie non risponde ad alcune tra le più comuni caratteristiche del software sviluppato in modo tradizionale. Prima tra tutte quella che è normalmente conosciuta come la *Legge di Brooks* dall'osservazione di Frederick P. Brooks nel suo *The Mythical Man-Month* [Brooks] secondo cui la complessità e i costi di comunicazione di un progetto aumentano con il quadrato del numero degli sviluppatori che ci lavorano. Molti estensori del metodo di sviluppo open source, primo tra tutti Raymond, hanno banalizzato l'assunto di Brooks riducendolo all'affermazione che aggiungendo manodopera ad un progetto software avanzato si ritarda la sua conclusione. Se ciò fosse vero, essi dicono, come si è sempre dimostrato nei progetti tradizionali, Linux ed altri progetti sviluppati con le stesse modalità non sarebbero stati possibili (o forse Linus Torvalds non conosceva la legge

di Brooks...).

Un modo in cui si spiega un fenomeno del genere è la forte motivazione di chi prende parte ad un Progetto Open Source, che porta ad una sostanziale autoselezione degli sviluppatori. Essi infatti non sono dipendenti di una società di sviluppo software che trovano un incentivo ad occuparsi d'altro con l'aumentare del numero di colleghi. Nel caso del software libero gli sviluppatori sono a loro volta gli utenti finali del software che, visto che gliene viene data l'opportunità, partecipano attivamente al confezionamento di un prodotto che li soddisfi maggiormente. Ovviamente, oltre all'interesse effettivo un'altra discriminante è la concreta capacità tecnica, ecco il significato dell'autoselezione sopracitata.

1.5 Il movimento Open Source

Per spiegare la recente nascita del concetto di *Open Source* e della corrispondente corrente di pensiero nell'ambito del movimento del software libero, bisogna partire da Debian[17]

1.5.1 Debian Social Contract e Debian Guidelines

Debian era (ed è) appunto una distribuzione costituita interamente da software libero e sviluppata su base volontaria. Tuttavia ad un certo punto iniziò ad incontrare delle difficoltà nel definire chiaramente cosa si poteva ritenere libero e cosa no, dal momento che iniziavano a proliferare altre licenze che pretendevano di definire il concetto di libertà. D'altra parte Debian non aveva mai preso in considerazione il fatto di creare una propria politica di software libero dal momento che sin dalla nascita si era appoggiata ai principï della Free Software Foundation.

Bruce Perens, in quanto leader del progetto Debian, nel 1997 propose le bozze di due documenti: il *Debian Social Contract* e le *Debian Free Software*

[17]Vedi sezione 1.3.5, pg. 30

Guidelines. La successiva opera di limatura, correzione e aggiunta che si svolse nel classico *stile hacker* con il contributo della comunità degli sviluppatori Debian, portò alle versioni definitive dei documenti.

Il Debian Social Contract documentava chiaramente l'intento di comporre un sistema interamente a partire da software libero, le Free Software Guidelines rendevano possibile una classificazione del software in libero e non libero mediante una chiara comparazione delle licenze esistenti.

Quando *Netscape* decise di rendere software libero *Communicator*, il proprio famoso programma di navigazione su Internet, contattò Eric Raymond, il quale ritenne che per essere considerato software libero Netscape avrebbe dovuto rispettare i principï sanciti dalle Debian Guidelines.

Questo non fu altro che il punto di partenza di un processo che avrebbe portato alla nascita del concetto di *Open Source*. Raymond infatti già da tempo riteneva che il mondo più tradizionalmente commerciale fosse tenuto lontano dalle istanze di Stallman e della sua Free Software Foundation, con cui egli stesso aveva collaborato. Egli sentiva che si stava perdendo una grossa occasione non favorendo in qualche modo la diffusione degli ideali *liberi* nel mondo del business.

Stallman aveva iniziato il suo Progetto GNU perché credeva - e crede - che la conoscenza che sta alla base di un programma funzionante dovrebbe essere libera, altrimenti il rischio sarebbe quello di una elite dominante nel mondo del software. La conoscenza scientifica, nella sua visione, dovrebbe essere di pubblico dominio, condivisa e distribuita - come avviene con le pubblicazioni scientifiche - ecco la necessità di mantenere liberi e disponibili i codici sorgenti dei programmi. La licenza GPL vuole salvaguardare questo tipo di conoscenza da indebite appropriazioni per fini commerciali estranei al principio del software libero.

Ovviamente molte imprese produttrici di software hanno interpretato le istanze del software libero come integraliste, rifiutando la visione di un prodotto completamente non proprietario che escludesse del tutto la voce di guadagno legata alla vendita di licenze o comunque del diritto d'autore.

1.5.2 Open Source Initiative

In questo contesto una "costola" della comunità free software capitanata da Eric Raymond, Tim O'Reilly e Larry Augustin si riunì in California per cercare una soluzione al problema del messaggio anti-business che traspariva dalle azioni della FSF, impedendo ai più di apprezzare la carica di innovazione e l'occasione insita nel concetto del software libero.

Per cominciare il gruppo si trovò sostanzialmente d'accordo su un punto: ciò che era mancato al movimento del software libero fino ad allora era una concreta campagna di sensibilizzazione sulle proprie istanze. E ad aggravare il tutto, secondo loro, c'era anche la grossa confusione sul significato del termine *free* in free software, che molti interpretavano nel suo significato più immediato ad una prima lettura: gratis. Essi trovarono un nuovo termine che descrivesse il software che volevano promuovere: Open Source, descritto nella Open Source Definition[18], discendente diretta appunto del Debian Social Contract. Risulta quindi evidente lo stretto legame di questo nuovo movimento con lo spirito del progetto GNU, tuttavia differenze sostanziali separano le due concezioni.

Fu così che nacque, sul finire del 1998, la *Open Source Initiative (OSI)*, un movimento che si propone di promuovere il concetto di Open Source e di rappresentare anche le voci più moderate e meno radicali del variegato mondo del software libero. Il termine stesso di *Open Source* venne coperto da copyright per renderne esclusivo l'uso in questo contesto.

[18] Vedi sezione 2.4.1, pg. 50

Capitolo 2

Licenze e copyright nell'Open Source

"Ricavare denaro dagli utenti di un programma attraverso delle restrizioni al suo uso è distruttivo perché le restrizioni riducono il numero di modi in cui il programma può essere usato.
Ciò riduce la ricchezza che l'umanità può ricavare dal programma". [Richard M. Stallman, GNU Manifesto[1]]

2.1 Natura e proprietà del prodotto digitale

Il software è un prodotto costituito da un insieme coordinato e strutturato di istruzioni digitali. Ha delle caratteristiche economiche che lo accumunano alla risorsa informazione, quindi, è un bene immateriale che per essere utilizzato, prodotto e accumulato deve essere incorporato in un supporto. Tra supporto e prodotto c'è una stretta integrazione ma vanno tenuti logicamente distinti sia per comprendere i problemi inerenti alla sua produzione sia per quelli che sorgono per la sua tutela giuridica.

La struttura dei costi è molto particolare, a differenza dei comuni beni industriali. Il costo di produzione del software, infatti, è molto elevato e

[1] `http://www.gnu.org/manifesto`

all'aumentare della complessità del programma questo cresce in modo esponenziale, per contro ha un costo quasi nullo di duplicazione e trasmissione indipendente dal numero di acquirenti o utenti.

2.1.1 Tutela giuridica del software

La tutela giuridica del software incontra due problemi di fondo, uno legato alla riproducibilità e trasmissibilità a costo prossimo allo zero in un contesto globalmente interconnesso e l'altro alla disomogeneità delle normative nazionali ed internazionali di tutela. A livello mondiale tale tutela risulta in alcuni casi contraddittoria o inapplicabile con alti costi di misurazione ed impossibilità di ricorrere all'esecuzione coattiva.

Problema molto sentito è quello dei brevetti sul software e su concetti quali teorie matematiche ed algoritmi, consentiti dal sistema giuridico statunitense e spesso concessi con leggerezza su concetti di pubblico dominio. Attualmente in Europa viene usata solo la pratica del copyright, ma il dibattito in Commissione Europea è aperto e molto vivace e diventa così difficile, se non impossibile, controllare efficacemente il rispetto dei diritti di proprietà e dei contratti. Tutto ciò quindi crea incertezza e difficoltà, oltre a creare nuove forme di monopolio con possibili conseguenze negative per il mercato.

In generale queste difficoltà nascono dal fatto che le istituzioni economiche negli ultimi anni si stanno evolvendo ad una velocità mai sperimentata in 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 nuova legge italiana sul copyright ne è un esempio).

2.1.2 Licenze e Non-Disclosure Agreement

Nell'attesa di una discipilina certa sull'argomento brevetti, gli strumenti da sempre maggiormente usati nella tutela dei diritti sul software sono le *licenze* e i *Non-Disclosure Agreement* (NDA, ovvero gli impegni a non rivelare i segreti industriali, nel nostro caso i codici sorgenti).

Non-Disclosure Agreement

I Non-Disclosure Agreement sono promesse di mantenere segrete particolari informazioni o dati. Sono abbastanza usati nell'industria del software tradizionale quando un programma necessita di essere distribuito in forma di codice sorgente a dei partner industriali o commerciali.

Per la sua stessa natura un NDA non può essere accettabile per un Progetto Open Source, in quanto contrasterebbe con il fine di renderne pubblici e disponibili i risultati.

Licenze

La licenza è un contratto che ha come scopo quello di porre delle condizioni maggiori di 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.

In questo contesto il software libero ed open source si pongono in una posizione molto particolare. La licenza si è rivelata, infatti, lo strumento più adatto a tutelare e proteggere questo tipo di software da abusi ed appropriazioni non lecite. Il meccanismo, che sarà spiegato più in dettaglio nei prossimi paragrafi, è all'apparenza banale ma abbastanza efficace.

Come la licenza protegge il diritto del possessore di copyright di stabilire delle condizioni restrittive ai fruitori della propria opera, così può anche tutelare il diritto dello stesso soggetto di decidere di rendere l'opera liberamente disponibile per chiunque ne sia interessato e nella sua forma più accessibile (in questo caso in forma di codice sorgente). Chi utilizzasse soft-

ware libero od open source in modo diverso dalle condizioni prescritte da una *licenza libera o 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.

Tuttavia, mentre di licenze libere in senso stretto - cioè secondo i dettami della Free Software Foundation - c'è solo la GPL, di licenze open source ne esistono parecchie, spesso create appositamente per un particolare software. Per capire quali sono le più significative, cosa le accumuna e cosa le distingue, è necessario analizzare più a fondo le istanze dei movimenti - Free Software Foundation e Open Source Initiative - che hanno sentito la necessità di crearle a di svilupparle organicamente.

2.2 Le due correnti del software a sorgenti aperti

La Free Software Foundation, con il suo Progetto GNU, e la Open Source Initiative, fanno capo a due diversi movimenti. Sebbene il software libero e l'open source condividano alcune regole e concetti di base, i loro obiettivi sono abbastanza differenti.

L'idea fondamentale che sta alla base dell'open source è che *se* si può, tramite Internet o altri mezzi, leggere, distribuire e modificare liberamente del software, grazie alla libera disponibilità dei codici sorgenti, questo migliora e si evolve; viene trasformato e depurato dai cosiddetti bug, gli errori, grazie all'apporto di decine, centinaia e a volte migliaia di sviluppatori o appassionati e ciò avviene ad una velocità impensabile rispetto ai modi tradizionali di produrre software, producendone di migliore.

Il concetto di software libero pone invece l'accento sulla libertà, in particolare sulle quattro libertà fondamentali formalizzate nella Licenza GPL che vedremo più in dettaglio tra breve.

In sostanza si tratta di due visioni con priorità diverse: l'una - open source - più pragmatica e possibilista che ha come scopo principale quello di

produrre software di maggior qualità; l'altra - software libero - più radicale e intransigente che ha come imperativo assoluto la produzione di software libero al 100%, senza compromessi.

L'Open Source Definition lascia molte più libertà rispetto alla Licenza GPL, soprattutto nel far convivere software open source con altri software proprietari nello stesso prodotto, cosa che, come vedremo, risulta impossibile secondo la Licenza GPL. Ciò contribuì e contribuisce tuttora a far sì che molte imprese si potessero avvalere dei vantaggi e della spinta propulsiva della *liberazione dei codici sorgenti* senza per forza doversi attenere strettamente alle limitazioni della licenza GPL, da molti ritenute eccessive ed anacronistiche. Da molti ma non da tutti. È infatti altrettanto viva ed attiva la comunità di sostenitori della FSF e del concetto più radicale di free software.

Entrambe le comunità - Open Source Initiative e Free Software Foundation/Progetto GNU - sono molto attive e portano avanti le loro istanze con argomentazioni magari opinabili, ma difficilmente eccepibili.

A volte è difficile per chi si avvicina a questo mondo vedere, se c'è, la linea di demarcazione tra le due *filosofie*. Infatti, a parte le differenti priorità, queste due correnti attingono ad un patrimonio di conoscenze comune dato dalla prima fase delle esperienze Free Software Foundation e Progetto GNU, e spesso fanno riferimento l'una all'altra. Non è raro infatti, esplorando i siti che si riferiscono a queste organizzazioni, imbattersi in link che portano al sito della "rivale".

Addirittura, curiosando tra le numerosissime mailing list o le chat dedicate, è molto frequente trovare rappresantate tutte le anime e le variegate sfumature tra una posizione e l'altra.

È il caso comunque di analizzare meglio il contenuto dei documenti che più di ogni altra cosa identificano le linee guida delle realtà appena descritte e ne mettono in risalto le differenze e i punti in comune: la Free Software Philosophy e la Licenza GPL per Free Software Foundation/Progetto GNU e la Open Source Definition per Open Source Initiative.

2.3 Free Software Foundation e Progetto GNU

2.3.1 Free Software Philosophy

Si tratta di libertà, non di prezzo. Il termine inglese free è inteso nel suo significato di *libero*, non *gratis*, e ciò ha sempre generato di per sé una certa confusione. Il concetto di *software libero* si riferisce alla libertà dell'utente di eseguire, copiare, distribuire, studiare, cambiare e migliorare il software. Più in dettaglio un software può essere definito libero, secondo la filosofia del software libero, quando garantisce quattro tipi di libertà per gli utenti:

Libertà 0 Libertà di eseguire il programma, per qualsiasi scopo.

Libertà 1 Libertà di studiare il funzionamento del programma per adattarlo eventualmente alle proprie necessità. L'accesso al codice sorgente ne è una precondizione.

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

Libertà 3 Libertà di migliorare il programma e distribuirne pubblicamente i miglioramenti, in modo tale che tutta la comunità ne possa trarre beneficio. L'accesso al codice sorgente ne è una precondizione.

Inoltre gli utenti devono poter essere liberi di ridistribuire copie, con o senza modifiche, gratis o addebitando delle spese di distribuzione a chiunque ed ovunque, senza restrizioni. Essere liberi di fare queste cose significa, tra l'altro, che non bisogna chiedere o pagare nessun permesso. Bisogna anche avere la libertà di fare modifiche e usarle privatamente nel proprio lavoro o divertimento senza doverlo notificare a nessuno. Se si intendono pubblicare le proprie modifiche, non si deve essere tenuti a comunicarlo a qualcuno in particolare, o in qualche modo particolare.

Si possono otenere copie di software GNU a pagamento, od ottenerle senza nessuna spesa, ma indipendentemente da come si siano acquisite le copie, si ha sempre la libertà di copiare e cambiare il software. Queste libertà per essere reali devono essere irrevocabili fin tanto che non vi si contravviene.

Se lo sviluppatore del software si riserva il potere di revocare la licenza secondo il suo insindacabile giudizio, anche se non gli si è dato nessun motivo per farlo, il software non è libero. Tuttavia, certi tipi di regole sul come distribuire il software libero sono accettabili quando non entrano in conflitto con le sucitate quattro libertà principali. Per esempio il movimento del software libero ha coniato il termine *permesso d'autore*[2] che è normalmente inteso come la regola che quando il programma è ridistribuito, non è possibile aggiungere restrizioni per negare ad altre persone le libertà principali. Questa regola non entra in conflitto con le libertà principali anzi, secondo il Progetto GNU le protegge.

Sono accettabili regole su come preparare un pacchetto di una versione modificata, a meno che esse in pratica non blocchino in qualche modo la libertà di distribuire versioni modificate. Anche regole del tipo *se rendi disponibile il programma in questo modo, lo devi rendere disponibile anche in quell'altro modo* possono essere accettabili, con le stesse condizioni. È da notare, comunque, che tale regola lascia ancora aperta la possibilità di rendere il programma disponibile o meno.

Il Progetto GNU usa il permesso d'autore per i motivi sopra citati, ma esiste tuttavia anche software libero senza permesso d'autore.

Talvolta le leggi sul controllo delle esportazioni e le sanzioni sul commercio possono limitare la libertà di distribuire copie di programmi verso paesi esteri. I programmatori che credono nel software libero non hanno il potere di eliminare o di aggirare queste restrizioni, ma quello che possono e devono fare è rifiutare di imporle come condizioni di uso del programma. In tal modo, le restrizioni non influiranno sulle attività e sulle persone al di fuori della giurisdizione degli stati che applicano tali restrizioni.

I sostenitori del software libero tengono a precisare che quando si parla di questi argomenti è meglio evitare di usare espressioni come "senza spese"

[2]Modo in cui viene tradotto il termine *copyleft*, gioco di parole contrapposto a *copyright*. Come viene anche usata anche la dicitura *all rights reversed* in contrapposizione ad *all rights reserved*.

o "gratuito", perché essi pongono l'attenzione sul prezzo, e non sulla libertà. Allo stesso modo termini quali "pirateria" implicano una visione che, alla luce di quanto appena scritto, ha poco a che fare con il software libero. A tal proposito è curioso il fatto che nei siti web che parlano di software libero sono in genere disponibili traduzioni rigorose dei termini specifici in molte lingue.

2.3.2 GNU General Public License

Quello che segue è un riassunto dei *Termini e condizioni per la copia, distribuzione e modificazione* contenuti nella versione più recente, la 2, della GNU General Public License.

1. La Licenza si applica a qualsiasi programma od altro lavoro che contenga una nota in cui il legittimo possessore del copyright dichiara esplicitamente che può essere distribuito nei termini della GNU General Public License. La Licenza si applica non solo al programma o lavoro, ma anche a qualsiasi altro lavoro che lo contenga o ne contenga una porzione, anche testuale, modificata o tradotta.

   La Licenza si riferisce solamente ad attività come il copiare, il distribuire ed il modificare, altre attività sono al di fuori dello scopo della Licenza.

2. È possibile copiare a distribuire con ogni mezzo copie testuali del codice sorgente del programma così come lo si riceve, non senza però aver provveduto a pubblicare in ogni copia una appropriata nota di copyright e di esonero da garanzie. Vanno conservate intatte le note che si riferiscono alla Licenza e all'assenza di garanzia e va consegnata a chi riceve il programma una copia della Licenza.

   È possibile richiedere una somma in cambio dell'atto fisico di trasferire

una copia, nonchè per l'offerta di una garanzia che però resta una libera iniziativa e non rientra nei termini della Licenza.

3. Si possono modificare intere copie o porzioni del programma cui è stata applicata questa Licenza e distribuire le modifiche o il programma modificato, a patto che siano rispettati i termini di cui al punto 1 e che che contestualmente si verifichino le seguenti condizioni:

   (a) I file modificati devono contenere delle note che dichiarino le modifiche, l'autore e la data in chi sono avvenute.

   (b) Al lavoro che deriva in tutto o in parte da un programma sotto questa Licenza, deve essere applicata la Licenza stessa senza oneri per alcun terzo.

   (c) Se possibile il programma modificato deve indicare, all'avvio, una nota di copyright e di esonero da garanzie (a parte quelle prestate personalmente dall'autore e nel rispetto delle leggi vigenti), l'indicazione della possibilità di redistribuire il programma stesso sotto le presenti condizioni nonchè i modi per ottenere una copia della Licenza.

   La Licenza non si applica a parti di un lavoro che si possano chiaramente identificare come non derivate dal programma in questione, ma quando tali parti sono distribuite come parte integrante di un lavoro considerato nella sua interezza come derivato dal programma, la Licenza si applica all'intera distribuzione. L'intento in questo caso è di esercitare il diritto di controllo su distribuzioni di lavori che derivano da o contengono programmi sotto questa Licenza.

4. È possibile copiare e distribuire il programma cui è stata applicata questa Licenza, od il lavoro da esso derivato, in codice oggetto o in forma eseguibile nei termini di cui ai punti 1 e 2 se contestualmente si verificano le seguenti condizioni:

(a) Che sia allegato il codice sorgente corrispondente, anch'esso distribuito secondo i termini di cui ai punti 1 e 2; oppure,

(b) Che sia presente un'offerta scritta, valida tre anni, di rilasciare a qualunque terzo al prezzo massimo del costo effettivo il codice sorgente, sempre nei termini di cui ai punti 1 e 2; oppure,

(c) Che sia allegata l'offerta di cui sopra se se ne è stati a propria volta destinatari.

5. Non è possibile compiere azioni quali copiare, modificare, distribuire, in modi che non rispettino quanto dettato da questa Licenza. Chi contravviene ai termini dettati non potrà più godere dei diritti garantiti dalla Licenza. Ovviamente ciò non vale per chi avrà ricevuto in buona fede programmi sotto tale licenza.

6. Nessuno è obbligato ad accettare i termini di questa Licenza finchè non l'ha sottoscritta, tuttavia essa permette azioni quali copiare, modificare e distribuire che altrimenti sarebbero proibite dalla legge.

7. Ogni volta che viene distribuito un programma cui è stata applicata questa Licenza, od il lavoro da esso derivato, il ricevente acquisisce il permesso dal licenziatario originale di copiarlo, modificarlo o distribuirlo secondo i presenti termini e condizioni. Non è possibile apporre successive restrizioni a questi diritti né forzare terze persone ad agire in conformità con questa Licenza.

8. Il fatto che siano poste a qualcuno delle condizioni a causa di decisioni della magistratura o accordi privati, non lo esime dal seguire i termini di questa Licenza. Ciò nell'intento di proteggere il sistema di distribuzione del software libero.

9. Se per problemi di brevetti o copyright la distribuzione del programma sotto questa Licenza subisce restrizioni in alcune nazioni, l'autore deve aggiungere alla Licenza delle clausole che escludano la distribuzione del programma in quelle nazioni.

10. La Free Software Foundation può in ogni momento produrre delle revisioni a questa Licenza che saranno conformi allo spirito della presente versione, ma potranno differire in alcuni particolari per affrontare nuovi problemi o istanze.

11. Chi volesse includere parte di un programma sotto questa Licenza in altri programmi liberi ma sottoposti ad altre condizioni di distribuzione, deve chiedere il permesso all'autore od al possessore del copyright. Nel caso questo sia la Free Software Foundation, essa si impegna a decidere caso per caso seguendo due principï: preservare la libertà di cui gode il proprio software e promuovere la condivisione ed il riuso del software in generale.

2.4 Open Source Initiative

In Internet è ufficialmente disponibile una definizione di Open Source[3] il cui intento fondamentale è di fissare in modo inequivocabile dei criteri che catturino l'essenza di cosa significa *Open Source* per la comunità degli sviluppatori. Criteri che assicurino che il software distribuito sotto una licenza open source sarà disponibile per chiunque voglia studiarlo ed usarlo garantendo un continuo miglioramento ed una selezione per molti versi darwiniana che porta a livelli di affidabilità e potenza che difficilmente un prodotto proprietario può raggiungere se non con l'impiego di enormi risorse.

L'intento degli estensori di questi principï è proprio quello di favorire questo tipo di sviluppo del software cercando di evitare quelli che essi chiamano "incentivi di breve periodo" che portano gli sviluppatori a scegliere la soluzione proprietaria. Perché ciò non accada deve esistere una licenza i cui termini impediscano di porre dei limiti al software prodotto sotto di essa ma che, a differenza della GPL, non creino impedimenti al contrario vietando l'utilizzo di componenti proprietarie nella costruzione di un prodotto *aperto*.

[3] `http://www.opensource.org/osd.html`

2.4.1 Open Source Definition

Ovviamente open source non significa unicamente accesso al codice sorgente. I termini di distribuzione di un programma, perché sia open source, devono rispettare i precisi criteri di quella che Bruce Perens, fondatore di Debian, ha definito *"la carta dei diritti dell'utilizzatore di computer"*: l'Open Source Definition[4]. Essa non è di per sé una licenza ma specifica quali sono le caratteristiche che un software deve avere per essere definito open source. Tali caratteristiche, qui di seguito riportate, devono essere presenti in maniera simultanea ed in ogni caso.

1. Libera Distribuzione

La licenza non può limitare nessuna delle parti nella vendita o nella fornitura di software come componente di una distribuzione di software aggregati, contenente programmi provenienti da fonti diverse. La licenza non può richiedere il pagamento di una royalty o di diritti per tale rivendita.
La licenza, dunque, richiede esplicitamente la libera distribuzione. Ciò vuol dire che chiunque può fare delle copie del software open source e venderle o regalarle senza dover pagare nessuno per questo privilegio. In questo modo si elimina la tentazione di cercare un immediato riscontro economico mediante la mera vendita di licenze, spostando l'attenzione verso il lungo periodo e quindi sul confezionamento di prodotti di maggior qualità che renderanno soprattutto sul piano della vendita dei servizi. Anche nel caso di vendita di pacchetti di software open source, la licenza ne permette la libera duplicazione e la modifica grazie alla disponibilità del codice sorgente.

2. Codice sorgente

Il programma deve includere il codice sorgente e deve consentire la distribuzione sia sotto forma di codice che in forma compilata[5]. Nei casi in cui un

[4]Qui riprodotta nella versione 1.7

[5]Trasformato in *codice oggetto*, eseguibile dal computer

prodotto non venga distribuito con il codice sorgente deve esserci la possibilità, ben documentata, di ottenerlo ad un costo non maggiore di quello ragionevole di riproduzione - preferibilmente via Internet senza costi aggiuntivi. Il codice sorgente deve essere la forma privilegiata in cui il programmatore modificherà il programma. Non è ammesso codice sorgente deliberatamente nascosto. Forme intermedie come l'output di un preprocessore non sono ammesse.
Senza accesso al codice sorgente non è possibile modificare agevolmente il programma. Senza modificazione non c'è evoluzione e l'evoluzione, per essere efficace, non dovrebbe avere vincoli.

3. Prodotti derivati

La licenza deve consentire la creazione di modifiche e di prodotti derivati, consentendo inoltre la loro distribuzione sotto gli stessi termini di licenza del software originale.
Anche in questo modo, rendendo disponibili le modifiche e le migliorie si rende non solo possibile, ma anche più agevole e veloce l'evoluzione del software. Questo criterio non obbliga a porre in atto per il prodotto derivato la stessa licenza dell'originario, ma evita piuttosto il contrario, cioè che l'autore di modifiche non possa utilizzare la stessa licenza.

4. Integrità del codice sorgente dell'autore

La licenza può imporre limitazioni sulla distribuzione del codice sorgente in forma modificata solamente se la licenza consente la distribuzione di file *patch*[6] insieme al codice sorgente con lo scopo di modificare il programma durante la compilazione. La licenza deve consentire esplicitamente la distribuzione di software realizzato a partire dal codice sorgente modificato. La licenza può richiedere che i prodotti derivati portino un nome o un numero di versione diverso dal software originale.
Chi usa un programma ha il diritto di sapere chi lo ha prodotto e l'autore, al

[6]Vedi Glossario

pari dei successivi sviluppatori, ha il diritto non solo di vedere riconosciuto il proprio lavoro, ma anche di ottenere che le modifiche siano mantenute distinte pur consentendole. In questo modo il software può essere redistribuito anche con modifiche non autorizzate dall'autore purchè si mantenga separato il codice originario dalle cosiddette patch che ne costituiscono le modifiche. Così si preserva la reputazione e il lavoro di chiunque intervenga nel processo evolutivo del software.

5. Nessuna discriminazione verso singoli o gruppi

La licenza non deve porre discriminazioni verso qualsiasi persona o gruppo di persone.
In quanto si pensa che dalla diversità possa emergere il massimo beneficio per il processo evolutivo. Alcune licenze possono mettere in guardia sulle restrizioni poste da alcune nazioni, tra cui gli Stati Uniti, sull'esportazione di alcuni tipi di software, tuttavia non le possono incorporare.

6. Nessuna discriminazione verso campi di applicazione

La licenza non deve porre limitazioni sull'uso di un programma in un particolare campo di applicazione. Per esempio non può impedire l'uso del programma in una particolare azienda o per la ricerca genetica.
Questo è un punto fondamentale. Con questa affermazione si vuole escludere qualsiasi inibizione all'uso del software open source, in particolare all'uso commerciale. Si vuole con ciò rassicurare gli utilizzatori e gli sviluppatori di tali software sulla possibilità di poterli sfruttare commercialmente, pur nel rispetto dei principï qui descritti.

7. Distribuzione della licenza

I diritti legati al programma devono applicarsi a tutti coloro a cui viene ridistribuito, senza la necessità di applicare una licenza supplementare.
Così si prevengono limitazioni successive, in particolare metodi indiretti come i Non-Disclosure Agreement (NDA).

8. La licenza non deve essere specifica per un prodotto

I diritti legati ad un programma non devono dipendere dal fatto che il programma faccia parte di una distribuzione particolare. Se il programma viene estratto da tale distribuzione e usato o distribuito nei termini della licenza del programma, tutte le parti a cui il programma viene ridistribuito devono avere gli stessi diritti garantiti in occasione della distribuzione originale del software.
Ciò per evitare ulteriori metodi indiretti di chiusura del software. Nel senso che, per esempio, non si può considerare libero un prodotto Open Source solo nel caso in cui faccia parte di una particolare distribuzione, rimane libero e basta.

9. La licenza non deve contaminare gli altri programmi

La licenza non deve porre limitazioni su altro software che venga distribuito insieme con il software in licenza. Per esempio la licenza non deve asserire che tutti gli altri programmi distribuiti sullo stesso supporto devono essere software open source.

Punto fondamentale in quanto esprime una grossa frattura con i tradizionali concetti del mondo free software. La licenza GPL infatti, come abbiamo visto precedentemente, prevede che il software sotto di essa "contamini" qualunque altro software con cui viene a contatto, anche se appartiene solamente alla stessa distribuzione e non ha legami effettivi con esso. Gli autori di questa definizione invece, ritengono che sia da salvaguardare l'autore del software e le sue scelte riguardo il modo in cui intende distribuirlo, senza imposizioni.

Conformità della licenza e della certificazione

Qualsiasi programma che faccia uso di licenze certificate come conformi alla Open Source Definition può utilizzare il marchio registrato "Open Source" ed il codice sorgente può essere dichiarato esplicitamente di pubblico dominio. Nessun altro programma o licenza è certificato per fare uso del marchio

registrato "Open Source".
Le licenze attualmente considerate da Open Source Initiative conformi alla Open Source Definition sono le seguenti: GNU General Public License (GPL), GNU Library o "Lesser" Public License (LGPL), BSD License, MIT License, Artistic License, Mozzilla Public License (MPL), Qt Public License (QPL), IBM Public License, MITRE Collaborative Virtual Workspace License (CVW License), Ricoh Source Code Public License, Python License, zlib/libpng License, Apache Software License, Vovida Software License, Sun Internet Standards Source License (SISSL), Intel Open Source License, Jabber Open Source License.

Di seguito riporterò una breve descrizione solamente delle più significative o conosciute:

- *La GNU General Public License (GPL).* Già vista nei suoi aspetti fondamentali nel precedente paragrafo.

- *La GNU Library o "Lesser" Public License (LGPL)*[7]. È la General Public Licence adattata per le librerie sviluppate secondo le modalità del software libero.

- *La BSD License*[8]. Nata con la Berkeley Software Distribution permette, come accennato precedentemente, la redistribuzione e l'uso sia in forma di codice sorgente che in forma di codice oggetto del software licenziato con essa. Uniche condizioni sono la citazione delle note di copyright, delle condizioni della licenza e della clausola che solleva autore e contributori dall'obbligo di fornire qualsiasi garanzia.

- *La MIT License.* Cioè la licenza dell'X Consortium[9], un consorzio di aziende e organizzazioni distribuite a livello mondiale che mantiene e sviluppa in forma collaborativa la tecnologia X Window System, l'am-

[7] `http://www.gnu.org/copyleft/lesser.html`
[8] `http://www.opensource.org/licenses/bsd.html`
[9] `http://www.opensource.org/licenses/mit_license.html`

biente grafico utilizzato nei sistemi Unix, compreso GNU/Linux. La MIT Licence ha condizioni molto simili alla BSD Licence.

- *La Artistic License*[10]. Questa licenza, senza scendere nei dettagli, mira a stabilire le condizioni attraverso cui il possessore di copyright possa mantenere una sorta di "controllo artistico" sullo sviluppo del software da lui creato, dando nel contempo agli utenti la possibilità di usarlo, distribuirlo e apportarvi ragionevoli modifiche.

- *La Q Public License (QPL)*[11]. Scritta da Trolltech AS, proibisce lo sviluppo di qualsiasi software proprietario basato su software con licenza QPL. Sono permesse modifiche e la loro redistribuzione, ma in forma di patch distinte dall'originale. Tuttavia l'autore delle modifiche è costretto dalla licenza a lasciare al produttore originale la libertà di distribuire i cambiamenti anche sotto qualsiasi altra licenza, non escluse quelle proprietarie.

Si noti come la Open Source Definition prenda esplicitamente le distanze da alcuni assunti della Licenza GPL, in particolare per quanto riguarda la contaminazione di altri software. In sostanza però la ritiene comunque compatibile con i suoi principï al punto che questa definizione è stata recentemente corretta al punto 3 con l'aggiunta del termine *costo ragionevole di riproduzione* per ridurre in qualche modo la distanza che la separa dalla Licenza GPL.

[10] `http://www.opensource.org/licenses/artistic.html`
[11] `http://www.trolltech.com/products/download/freelicense/license.html`

Capitolo 3

Il Progetto Open Source

3.1 Un esempio di Progetto Open Source

È molto difficile attualmente definire in modo univoco un *Progetto Open Source*[1] standard per il semplice fatto che spesso i vari progetti in corso di svolgimento possono differire tra loro per vari aspetti non sempre irrilevanti, inerenti le modalità, il numero di persone coinvolte o altro.

È possibile tuttavia ricercare una sottile linea comune che unisca i Progetti Open Source e, con qualche semplificazione, ricondurli ad una serie di comuni denominatori, aspetti cioè sempre (o quasi) presenti nella nascita e nel naturale svolgimento di tali Progetti.

3.1.1 Fetchmail

Un esempio paradigmatico di come si sviluppa il classico Progetto Open Source è riportato da Raymond sempre nel suo *The Cathedral and the Bazaar* [Raymond1]. Egli infatti nel 1996 ebbe modo, un po' per scelta consapevole

[1] Ricordo che il termine *open source* usato in questo contesto si riferisce alla condizione imprescindibile della disponibilità del codice sorgente. Con il termine Progetto Open Source, quindi, si intende in generale l'insieme delle modalità con cui si svolge il processo di sviluppo sia del software libero che di quello open source, le cui definizioni sono state esposte nel Capitolo appena concluso.

ed un po' per necessità, di mettere in pratica ciò che aveva osservato nel modo in cui si era sviluppato Linux.

All'epoca Raymond era responsabile tecnico di un Internet Provider chiamato Chester Country InterLink. Una cosa che egli trovava abbastanza tediosa era il fatto che se voleva controllare la sua e-mail da casa doveva, una volta on line, accedere alla macchina del provider tramite il programma *Telnet*, avviare un check per vedere e scaricare la posta e poi disconnettersi dalla stessa macchina.

Quello che gli serviva era un programma che trasferisse la propria posta elettronica sulla sua macchina di casa evitandogli di andare ogni volta a cercarla su una macchina remota.

Il primo dilemma che affrontò fu se scrivere egli stesso ex-novo un programma adatto al suo bisogno oppure risparmiare un sacco di tempo cercandone qualcuno già esistente da adattare. In fondo, come egli stesso afferma, anche Linus Torvalds non scrisse il kernel Linux da zero, ma riutilizzò parte del codice di *Minix*, un piccolo sistema operativo di tipo Unix per PC.

Raymond quindi decise di cercare nella ormai già ampia varietà di codice open source disponibile in rete trovando più di un prodotto che facesse al caso suo. Tra questi, dopo attenta analisi, ne scelse uno: *Popclient*, un programma scritto da Carl Harris che tuttavia risultava un po' vecchio e mancante di alcune caratteristiche tecniche necessarie per essere al passo con i tempi.

Da buon sviluppatore open source, Raymond decise di mandare alcune modifiche in forma di patch a Carl Harris perché egli ne valutasse l'integrazione nel programma. Tuttavia egli si rese presto conto che l'autore originario di Popclient aveva già da un po' perso qualsiasi interesse nella sua creatura. Si trovò quindi di fronte ad un altro bivio: lasciar perdere e trovare qualche altro programma che facesse al caso suo, gettando tutto il lavoro fatto fino ad allora, o chiedere ad Harris di poter portare egli stesso avanti lo sviluppo di Popclient.

Harris, dopo un breve scambio di opinioni, fu ben felice di lasciare a Ray-

mond la responsabilità di mantenere Popclient. Con il programma Raymond scoprì di aver ereditato anche la sua consistente base di utilizzatori che, come vedremo, data la natura "aperta" del codice, sono anche spesso programmatori ed hanno tutto l'interesse a contribuire al miglioramento del programma che essi stessi utilizzano, diagnosticando problemi, suggerendo aggiustamenti e segnalando errori.

Come Mantainer di un programma open source Raymond notò che la base di utenti/sviluppatori, per essere veramente utile al miglioramento del codice, deve essere adeguatamente stimolata ed incentivata. Il fatto che Harris, autore e precedente mantainer di Popclient, si fosse da tempo disinteressato al progetto, aveva fatto sì che la piccola comunità di utenti/sviluppatori che in genere si crea attorno ad un progetto open source, si fosse atrofizzata e non fornisse più buoni spunti.

Per stimolare una ripresa di interesse da parte degli utenti/sviluppatori, Raymond decise di prendere spunto dalla sua osservazione della realtà Linux, cercando di replicare quelli che riteneva i punti di forza dei quel modello di sviluppo. Dopo aver cambiato il nome del programma da Popclient in *Fetchmail* per il fatto che vi aveva inserito anche altri protocolli di trasmissione della posta elettronica, oltre al POP (Post Office Protocol) su cui si basava inizialmente, iniziò a seguire delle regole di condotta che in seguito riassunse nelle seguenti:

- Rilasciare spesso (da ogni 10 giorni a giornalmente) e velocemente nuove versioni modificate con i contributi degli utilizzatori/sviluppatori. È una forma di ricompensa e uno stimolo allo sviluppo.

- Aggiungere alla *Beta List*[2] chiunque lo contattasse per avere informazioni riguardo il programma.

- Stimolare la Beta List, attraverso una mailing list, con inviti espliciti a partecipare e con tempestivi annunci al rilascio delle nuove versioni.

[2]La lista di coloro ai quali mandare in visione una versione Beta del programma

- Ascoltare i Beta Testers, chiedere loro consiglio sulle decisioni relative al design dei pacchetti e sottolineare il loro apporto in caso di invio di patch e consigli.

La Beta List, che al momento del passaggio di consegne da Harris a Raymond contava qualche decina di sviluppatori, cominciò ad aumentare ad un ritmo blando ma costante arrivando a circa 300 persone verso la fine di Maggio del 1997, quando si stabilizzò per il fatto che Fetchmail aveva raggiunto una eccellente stabilità e non necessitava che di ritocchi marginali o di mantenimento. Ciò fece comprendere a Raymond che anche un Progetto Open Source, portato avanti quindi nel tipico *Stile Bazaar*[3]di Linux aveva un suo ciclo vitale ben definito ed una sua maturità.

3.1.2 Spunti di analisi

Molti furono gli spunti di analisi che Raimond ottenne dall'esperienza Fetchmail, tanto da spingerlo a definire quelle che lui chiama le *Pre-condizioni per sviluppare un Progetto Open Source nello "Stile Bazaar"* [Raymond1].

Prima di tutto è difficile dare origine ad un Progetto ex-novo nello *Stile Bazaar*, nel senso che per costruire una comunità di sviluppatori attorno ad un Progetto è necessario avere già del codice eseguibile, anche se incompleto, acerbo, bacato e scarsamente documentato, ma pur sempre funzionante e testabile. Solamente in questo modo si possono convincere futuri co-sviluppatori della potenziale bontà di un Progetto Open Source.

In secondo luogo il coordinatore di un progetto di questo tipo non deve necessariamente essere un eccezionale designer di architetture software, ma deve assolutamente saper riconoscere le buone idee altrui a riguardo.

È abbastanza evidente la grossa influenza del contesto sociale[4] nel modo in cui questi Progetti si sviluppano. All'interno della comunità open source esiste un modello di mercato basato sulla reputazione che spinge le persone a

[3]Vedi paragrafo 1.3.3, pg. 28

[4]Vedi paragrafo 1.4, pg. 32

non lanciarsi in imprese per le quali non hanno le competenze, ma a chiedere appoggio o consiglio a chi ne ha, in uno scambio virtuoso che rimane invariato in ogni ambito di interazione: dai gruppi di sviluppo alle mailing list ai canali di discussione.

Dalla considerazione del contesto comunitario emerge un'altra condizione basilare che non è normalmente associata allo sviluppo del software. Un coordinatore di un Progetto Open Source deve avere una buona attitudine comunicativa e saper lavorare in gruppo, solo così potrà attirare altre persone interessandole a quello che sta facendo e gratificandole per i contributi che sapranno fornire.

3.1.3 Visioni alternative

Devo comunque rilevare che, nonostante quello appena presentato possa essere un esempio più immediato ed accessibile a molti, vi sono altri e più raffinati approcci al metodo di sviluppo open source. Tra gli altri Nikolai Bezroukov ad esempio considera la visione Raymondiana del modello "Stile Bazaar" troppo semplicistica, mentre lui vede nella ricerca accademica un paradigma migliore, considerando lo sviluppo di questi tipi di software come *...un caso speciale di ricerca accademica* [Bezroukov1], appunto.

L'affinità con il mondo scientifico si coglie dal procedimento seguito nello sviluppo dei programmi software a codici sorgenti aperti che in effetti funziona in modo conforme al metodo scientifico: la scienza si basa su un processo di scoperta ed uno di dimostrazione, i risultati debbono poter essere replicati. La segretezza lo impedirebbe rallentando od intralciando il progresso scientifico.

In realtà Bezroukov - come vedremo più approfonditamente nel capitolo conclusivo - pur apprezzando il metodo di sviluppo open source, avanza una serie di obiezioni e di correzioni al semplicismo ed all'eccessivo entusiasmo che animano talvolta, specialmente in questi anni di boom, i sostenitori di questo modello di sviluppo.

3.2 Caratteristiche di un Progetto Open Source

Grazie al fatto che attualmente Internet non solo ospita i Progetti Open Source, ma costituisce il mezzo principale di comunicazione tra sviluppatori di ogni singolo Progetto, è aumentata di molto la loro osservabilità e la possibilità di analizzarne non solo gli aspetti tecnici, ma anche le caratteristiche organizzative. In questo senso è possibile, entro certi limiti, cercare delle analogie o dei punti in comune almeno nei modi in cui tali Progetti nascono e si sviluppano a partire da certe premesse.

Su questi aspetti in particolare ho trovato nelle mie ricerche dei buoni spunti - che utilizzerò nella seguente esposizione - dalla recente Tesi di Diploma dello studente berlinese di Informatica Steffen Evers intitolata, nella sua più recente revisione, *An Introduction To Open Source Software Development* [Evers].

3.2.1 Definizione

Innanzitutto è necessario definire in maniera abbastanza univoca, se non rigorosa cos'è un progetto open source e da chi o cosa è composto ed in questo riprendo, in quanto mi sembra chiara pur se generalizzante e semplificativa, la definizione proposta da Evers:

> *"Ogni gruppo di persone che sviluppa software e rende disponibili i propri risultati al pubblico sotto una licenza open source, costituisce un Progetto Open Source."* [Evers].

3.2.2 Sviluppatori

Chi sviluppa software nei Progetti Open Source? Attualmente sono diverse le categorie di soggetti che sviluppano software secondo queste modalità, tra le principali si possono annoverare:

- **Istituzioni Scolastiche**. Le Università e altre Istituzioni Scolastiche producono parecchio software con scopi educativi e di ricerca. Sebbene in parte tale software diventi proprietario, molto viene distribuito in termini che si conformano alla Open Source Definition.

- **Istituzioni di Ricerca**. Di solito sono strettamente correlate alle Istituzioni Scolastiche e ne condividono personale e finanziamenti. A parte il fatto che in genere il prodotto di qualsiasi campo di ricerca scientifica è distribuito liberamente per sua natura, è abituale in questi ambiti rilasciare i risultati del lavoro di ricerca sotto licenze permissive, anche per permetterne l'uso alle diverse parti coinvolte. In più tali licenze possono essere il veicolo per finanziamenti da aziende interessate.

- **Distributori di Software**. Le aziende che producono distribuzioni di software open source[5] di norma partecipano a diversi Progetti Open Source. In questo modo, aumentando ed implementando le soluzioni contenute nei propri pacchetti, aumentano la loro base di utenti.

- **Imprese Commerciali**. Anche le imprese che intendono basare la loro attività sul software open source devono necessariamente partecipare in qualche modo a Progetti Open Source, è proprio la struttura stessa del modello produttivo che lo richiede. Tra le aziende più potenti e conosciute possiamo citare IBM, Hewlett Packard ed Intel.

- **Utenti Organizzati**. Se si considerano le enormi risorse finanziarie spese dalle aziende o dalle pubbliche amministrazioni per i propri sistemi informatici (milioni di dollari solo per le licenze), allora finanziare un Progetto Open Source può veramente essere più conveniente di pagare per le licenze di un software proprietario- una per ogni macchina installata...

- **Utenti Privati**. Chiunque usi software open source ha interesse - avendone le capacità - a migliorarlo in quanto ciò gli dona un beneficio

[5]Vedi sezione 1.3.5, pg. 30

diretto. È per questo che molti utenti privati partecipano attivamente ed in varie forme a Progetti Open Source.

- **Governi e Pubbliche Amministrazioni**. Oramai sempre più strutture governative, oltre all'economia in generale, dipendono dall'informatica. Ecco quindi che molti governi preferiscono guardarsi attorno nel panorama dell'offerta di soluzioni informatiche non solo per la discriminante dei costi, alla quale sono per definizione sensibili, ma anche con un ochio alla sicurezza dei dati e all'elasticità delle soluzioni adottate. Non ultimo pesa anche il fatto di non dover necessariamente dipendere da un'unica azienda monopolista straniera.
  Ecco quindi sorgere un po' ovunque esperienze di collaborazione tra open source e pubblica amministrazione; tra gli altri si possono citare il finanziamento del progetto GNU Privacy Guard (GPG) da parte del Ministero dell'Economia tedesco, l'interessamento del Sistema Bibliotecario Italiano a finanziare un progetto di software open source per gestire il proprio patrimonio librario e i contatti intrapresi dalla Cina con alcune grosse aziende statunitensi open source per sviluppare nei suoi vari distretti un sistema informatico basato su software open source.

3.2.3 Come inizia un Progetto Open Source

Non esiste ovviamente alcun documento a riguardo - nel tipico stile degli How-to[6] Linux, magari - né ci sono delle regole comunque formalizzate che stabiliscano una condotta tipo. È comunque possibile osservare dei punti in comune, dei paralleli nel modo in cui in genere inizia e si sviluppa nelle sue prime fasi un tipico Progetto Open Source. Si noterà come questa fase sia caratterizzata dall'essere svolta in prevalenza da singoli utenti o comunque da piccoli gruppi organizzati di utenti privati. Le aziende, soprattutto le grosse, entrano spesso solamente a livello di finanziamenti in un secondo momento

[6] Vedi Glossario

del processo, quando è chiaro che esso può portare ad una utilità dal punto di vista anche commerciale e che quindi gode di una certa stabilità e maturità.

Ecco a grandi linee un tipico inizio di un Progetto Open Source:

1. Un individuo ha un determinato problema o un'idea e cerca di trovare una buona soluzione che faccia al caso suo (si pensi al caso appena visto di Raymond ed al suo problema con la posta elettronica).

2. Può anche chiedere ad alcuni amici o colleghi cosa sanno sull'argomento. Alcuni di loro hanno lo stesso problema o problemi simili, ma nessuno ha una soluzione.

3. Le persone interessate cominciano a scambiarsi pareri e conoscenze sull'argomento creando così una vaga idea del problema su cui si concentra l'attenzione del gruppo.

4. Le persone interessate che intendono spendere delle risorse (a questo livello si tratta del proprio tempo libero) sulla soluzione del problema danno il via ad un progetto informale, mentre altre persone meno interessate lasciano il gruppo. In questo modo il problema centrale diviene oggetto dell'attenzione di tutti i partecipanti al progetto.

5. I membri del progetto lavorano al problema fino a che non raggiungono dei risultati presentabli.

6. Essi pongono i risultati del loro lavoro pubblicamente disponibili in un posto dove il maggior numero di persone possibile possa raggiungerli (siti ftp o siti http dedicati, come vedremo tra breve). Possono anche rendere noto il loro progetto presso mailing list, newsgroup o servizi di news in Internet.

7. Altre persone scoprono di condividere alcuni dei problemi o delle idee che hanno fatto sorgere il progetto e sono anch'essi quindi interessati a che venga portato a termine. Danno un'occhiata ai risultati del

progetto (magari usandoli...) e guardando il problema da una prospettiva diversa da quella che dominava il gruppo già costituito, hanno a volte dei buoni suggerimenti da dare e cominciano a comunicare con il progetto e spesso entrano a farvi parte.

8. Il progetto cresce e le interazioni tra i vecchi e i nuovi partecipanti aiutano a trovare dei modi per capire meglio il problema e quindi delle strategie per risolverlo.

9. Nuove informazioni e risorse sono continuamente integrate nel processo di ricerca.

10. Si chiude un altro ciclo di ricerca e si ritorna al punto 5 con la presentazione dei nuovi risultati.

11. Si è così fondata la comunità del progetto. Potrà crescere od estinguersi, ciò dipenderà dal progetto e dall'interesse che potrà suscitare nel suo sviluppo.

3.2.4 Vantaggi derivanti dal metodo di sviluppo Open Source

Il più grosso vantaggio derivante dal software libero ed open source sembra essere proprio la disponibilità dei codici sorgenti. Gli effetti immediati derivanti da questa fondamentale caratteristica sono, tra gli altri:

- Peer review del codice.
- Possibilità per l'utente non solo di segnalare errori (bug), ma anche di fornire modifiche (patch) per correggerli.
- Reale sviluppo distribuito.

Di seguito, con ordine, saranno spiegati meglio e più approfonditamente questi concetti.

Peer review del codice

> *"Se ci sono abbastanza occhi [che cercano errori], gli errori diventano di poco conto"* [Raymond1]

Con il termine *peer rewiew* si intende il processo di collaborazione e revisione del lavoro, svolto da parte di tutti o molti dei partecipanti, al processo di sviluppo del codice in un Progetto Open Source.

Ci sono spesso diversi modi di implementare una data funzionalità, alcuni più efficaci, altri più veloci, altri ancora di più facile manutenzione. Ci sarà sicuramente almeno una persona tra gli utenti/sviluppatori interessati che ha già affrontato il problema che mi assilla o che può darmi una mano a risolverlo, dicendomi dove ho sbagliato o addirittura mostrandomi una soluzione alternativa; questo è il peer rewiew.

In alcuni ambiti - per esempio la sicurezza dei sistemi - è necessario poter verificare a fondo l'effettivo funzionamento di un programma, l'efficacia dei suoi algoritmi e della loro implementazione. Il peer rewiew, tipico del mondo open source risulta essere uno strumento molto efficace in questo senso.

Segnalazione e correzione di errori *(Bug Report)*

> *"Se tratti i tuoi beta-tester come se fossero la tua risorsa più importante, essi risponderanno diventando la tua risorsa più importante."* [Raymond1].

Il fatto che il codice sia visibile a chiunque, permette di poterlo analizzare per poterne trovare gli errori, le imperfezioni, le debolezze o le soluzioni superficiali. Ovviamente una cosa del genere non è alla portata delle capacità tecniche di chiunque, ma le dimensioni[7] del fenomeno in molti suoi ambiti fa intuire come questo processo sia parecchio attivo.

[7] Vedi paragrafo 3.5 , pg. 83

Sviluppo distribuito

La disponibilità dei codici sorgenti permette a chiunque sia interessato e ne abbia le capacità di partecipare allo sviluppo di un programma.

La misura ed il modo in cui i nuovi sviluppatori possono collaborare dipende molto, oltre che da loro, dalla spinta data dall'autore del programma e dal core-team che guida lo sviluppo.

3.2.5 Alcuni esempi concreti di Progetti Open Source

I Progetti Open Source attualmente in fase di nascita, sviluppo o mantenimento sono alcune migliaia. Nell'impossibilità di citarli tutti, di seguito darò una breve descrizione dei più grossi o significativi escludendo solamente Linux, di cui ho parlato diffusamente in precedenza.

Apache. Apache[8] è un web server libero. Il Progetto Apache è nato nel 1995 per sviluppare un web server che fosse completo, affidabile, potente ed il cui codice sorgente potesse essere liberamente distribuibile. Il progetto è portato avanti da un gruppo di volontari, chiamati Apache Group, sparsi in tutto il mondo - uno di essi, David Welton, 23 anni, ha collaborato per Prosa s.r.l. e vive tuttora a Padova dove collabora con Innominate Italia - che usano Internet per comunicare tra loro, pianificare e sviluppare il server e la relativa documentazione.

Si contano inoltre centinaia di semplici utenti che hanno contribuito e contribuiscono con idee, codice e documentazione.

Statistiche ufficiali[9] mostrano come ad oggi Apache sia il web server più diffuso in Internet, superando tutti gli altri messi assieme.

Gnome. È, secondo la sua definizione inglese di difficile traduzione, un desktop environment, un ambiente applicativo che fornisce un'interfaccia grafica e un set di utility ed applicazioni per l'utente di sistemi di tipo Unix. Il

[8] `http://www.apache.org`

[9] `www.netcraft.com/survey`

fine del Progetto Gnome[10], infatti è di fornire all'utente, soprattutto a quello base, un set di applicazioni ed un ambiente grafico interattivo il più *user friendly* possibile, cercando quindi di superare quello che era sempre stato il grosso handicap dei sistemi Unix: la scarsa propensione alla facilità d'uso, all'intuitività e alle frivolezze grafiche.

Gnome fornisce anche il software e gli strumenti per lo sviluppo e una serie di spazi dedicati ai suoi sviluppatori, la Gnome Community, e a chi desidera contribuire al progetto in varie forme.

The Gimp. GNU Image Manipulator Program[11]. È un software per la manipolazione di immagini, il fotoritocco e compiti simili liberamente distribuito sotto Licenza GPL. Il sito dedicato al programma funge da fonte per scaricare liberamente le versioni aggiornate dello stesso o le varie patch di aggiornamento ed avere qualsiasi informazione a riguardo. Esiste una comunità di utenti/sviluppatori attiva sullo sviluppo del programma e di altri progetti correlati, di cui lo stesso sito dà notizie.

Mozilla. È il risultato della decisione di Netscape di rilasciare una versione libera del suo famoso browser *Netscape Communicator*. Il progetto nato da questa operazione ha preso il nome di *Mozilla*[12]. Mozilla è un browser open source il cui sviluppo è coordinato anche da Netscape mediante l'istituzione di forum di discussione, la diffusione di nuove versioni e lo stimolo alla ricerca di bug da parte degli utenti.

XFree86. È una implementazione liberamente redistribuibile[13] del Sistema X Window, sistema grafico a finestre in grado di operare sia su una macchina singola sia - indifferentemente - su rete, che funziona sui sistemi operativi Unix o simili. Sin dall'inizio il progetto XFree86 si è focalizzato su piattaforme basate su processori Intel x86, ma le sue più recenti versioni supportano

[10] `http://www.gnome.org`
[11] `http://www.gimp.org`
[12] `http://www.mozilla.org`
[13] `http://www.xfree86.org`

anche altre piattaforme e l'obiettivo attuale del progetto è appunto quello di supportarne sempre di più.

3.3 Aspetti organizzativi

Nell'analizzare gli aspetti organizzativi di un Progetto Open Source, bisogna considerare che per molti versi essi si discostano dal modo in cui è organizzato un normale progetto di software proprietario. Quest'ultimo in genere è promosso e sviluppato dal personale di una azienda o da collaboratori sfruttando una serie di risorse e strutture messe a disposizione dall'organizzazione, requisiti questi che assumono un aspetto radicalmente diverso nell'ambito di un Progetto Open Source.

3.3.1 Risorse

Spesso l'impressione che si ricava dalla superficiale osservazione di un Progetto Open Source è che esso funzioni essenzialmente senza utilizzare risorse concrete, o quasi. Soprattutto quando pensiamo al concetto di risorsa nel senso più tradizionale del termine, cioè come un quantitativo di denaro, scorte, materiali, personale e strutture disposte per il funzionamento di un'attività economica.

Se già di per sé il mondo del software presenta alcune sostanziali differenze rispetto all'industria tradizionale per quanto riguarda l'utilizzo di risorse, si pensi alla scarsa rilevanza dei fornitori, un Progetto Open Source attinge ad una serie di risorse che potremmo definire atipiche e che per questo sono inizialmente scarsamente individuabili.

Informazione e software

L'informazione, in generale, è una delle maggiori risorse per lo sviluppo del software libero, infatti il software non è altro che un particolare tipo di informazione.

L'uso dell'informazione è regolato dalle leggi sulla proprietà intellettuale che proteggono, come abbiamo visto[14], l'autore di un'opera di ingegno attribuendogli una serie di diritti su di esso. Per quanto riguarda le opere protette da licenze libere od open source l'autore, decidendo volontariamente di rimuovere i vincoli alla loro trasmissione, le rende di fatto "legalmente disponibili". Ed il fatto che non sia materiale deteriorabile, rende la risorsa informazione - non vincolata - una risorsa inesauribile.

Il software, come detto, è semplicemente un tipo di informazione. Esso è però usato in modo diverso da ogni altro tipo di informazione e per questo le leggi che lo regolano sono differenti. Tuttavia quando si parla di software libero ed open source, come per le informazioni disponibili senza vincoli, il problema non si pone. Si può porre invece parlando di informazione e software sottoposti a vincoli.

Nel caso di software proprietario o comunque non disponibile nelle forme e nei modi richiesti dalla comunità open source, i problemi dipendono da vari fattori come la natura del software stesso e l'uso per cui è richiesto.

Nel caso di strumenti di sviluppo di software, ogni programmatore può usare quello che più gli aggrada, ma se decide di usare uno strumento non libero od open source lo fa di propria iniziativa facendolo rientrare nella categoria delle risorse personali. Un Progetto Open Source infatti, per la propria natura non può disporre l'uso di strumenti non disponibili liberamente, in parte per problemi legati alla natura in parte volontaria dell'apporto di sviluppo ed in parte per evidenti vincoli finanziari all'adozione di strumenti che richiedono una licenza.

Uno sviluppatore può volere inserire un componente software proprietario in un prodotto open source, ad esempio per assecondare la volontà di un cliente. Non è impossibile in linea di massima, e talvolta succede, tuttavia ciò diventa problematico nel caso il software non proprietario sia sottoposto a Licenza GPL che, come abbiamo visto, tra le sue caratteristiche ha un "effetto virus" per cui in teoria si dovrebbe applicare a tutte le componenti

[14]Vedi paragrafo 2.1.2, pg. 41

software che vi entrano in contatto.

Esiste comunque anche il caso di software proprietario che può essere considerato una risorsa non limitata anche per un Progetto Open Source. È il caso per esempio di Java[15] di Sun Microsystems, di cui non è disponibile il codice sorgente, ma che è scaricabile gratuitamente da Internet.

L'informazione che non sia software - come manuali, commenti ed altro - può essere considerata non disponibile quando è sottoposta anch'essa ai vincoli sul copyright. Il problema è facilmente aggirabile nel caso si tratti di materiale pubblicato in Internet, mentre diviene limitante nel caso di supporti fisici come libri, riviste o CD-ROM. Il possesso di questi da parte dei singoli individui può essere considerato risorsa personale.

Risorse personali

In genere la maggior parte dei Progetti Open Source non dispone di proprie risorse da distribuire e condividere a parte il risultato del proprio lavoro, il proprio sito e una o più mailing list, il tutto spesso ospitato e/o sponsorizzato da istituzioni educative come le università o da altri tipi di organizzazione interessate al Progetto.

Esiste anche la possibilità di fondare persone legali come organizzazioni *non-profit* o associazioni di vario tipo a cui intestare la titolarità di eventuali risorse condivise. Tuttavia ciò comporterebbe un lavoro "amministrativo" poco gradito, ecco perché non sono molti gli esempi di questo tipo.

In genere le risorse materiali più diffuse sono quelle appartementi al singolo soggetto collaborante al Progetto. Esse possono essere di vario tipo: apparecchiature, pubblicazioni e documentazione, abilità.

Hardware

È una risorsa fondamentale addirittura per l'esistenza di un Progetto Open Source e la maggior parte di essa si sovrappone anche alla categoria delle risorse personali. È possibile distinguere tre gruppi di hardware - considerato

[15] Vedi Glossario

come risorsa - in base ai seguenti aspetti: Proprietà, Ubicazione, Accesso Fisico, Accesso remoto, Amministratore [Evers].

- *Hardware personale.* Non è posseduto dal Progetto e chi lo amministra permette solo a poche persone selezionate di usarlo. In genere è il computer di un partecipante al Progetto o il terminale in un ufficio. Fa parte delle risorse personali.

- *Hardware generalmente disponibile.* Deve essere raggiungibile da remoto, il sistema che lo ospita deve essere collegato permanentemente ad Internet e l'amministratore deve garantirne l'accesso a qualunque participante al Progetto.

- *Hardware parzialmente disponibile.* L'hardware che non può essere raggiungibile da remoto o che necessita di essere raggiunto fisicamente o non è permanentemente collegato ad Internet.

Risorse umane

La maggior parte dei partecipanti ad un Progetto Open Source, in genere, vi prende parte in modo volontario e senza essere pagato, anche se a volte alcuni di loro sono pagati da alcune Imprese interessate al Progetto per varie ragioni, come Ximian per il Progetto Gnome[16] o SuSE per il Progetto ALSA[17].

Chi sviluppa software per un'azienda, fornisce solo la sua professionalità. L'azienda fornisce l'attrezzatura, l'ambiente, le infrastrutture e tutto ciò che serve, compreso l'aggiornamento e benefici come l'assicurazione od altro. I partecipanti ad un Progetto Open Source in genere utilizzano le proprie risorse personali. Sul perché lo facciano si rimanda alle parti di questo lavoro che investigano in modo più approfondito le ragioni di una scelta del genere[18].

[16] Vedi paragrafo 4.6, pg. 125

[17] Advanced Linux Sound Architecture. `Http://www.alsa-project.org`

[18] Vedi paragrafi 1.4, pg. 32 e 4.3.4, pg. 104

Risorse finanziarie

In genere i tipici Progetti Open Source non hanno entrate o spese. Anche quando entrano in campo imprese interessate ad un Progetto, normalmente pagano degli sviluppatori per lavorarci - non il Progetto direttamente - o sponsorizzano delle attività connesse - come meeting, pubblicazioni, ecc. - o forniscono infrastrutture web.

Anche se non sono considerati una risorsa fondamentale, i soldi sono comunque ben accetti proprio per permettere lo svolgimento di quelle attività di coordinamento, aggregazione e rappresentanza che sarebbero impossibili altrimenti.

Altro

Oltre alle risorse appena esposte, i Progetti Open Source e chi ne fa parte hanno a disposizione tutta una serie di servizi, strumenti e mezzi di comunicazione che sono direttamente correlati ad Internet. Internet può essere considerato, a ragione, forse la principale risorsa del fenomento e l'infrastruttura *virtuale* su cui poggiano le loro basi sia le comunità che tutti i Progetti.

Per una analisi più dettagliata di questi elementi si rimanda al paragrafo 3.4 a pagina 80 sugli strumenti e sui mezzi di comunicazione tipici dei Progetti Open Source.

3.3.2 Coordinazione ed incentivi

Come ogni sistema complesso, anche per il Progetto Open Source, si pongono dei problemi di coordinazione per far sì che i vari elementi che lo conpongono funzionino in modo efficiente. La domanda è se possono essere identificati dei processi coordinativi in un Progetto Open Source e quanto essi sono consapevolmente attuati.

Per una risposta a questo quesito possono essere d'aiuto le osservazioni fatte da Thomas K. Malone e Kevin Crowston nel loro *The Interdisciplinary Study of Coordination* [Malone].

Essi innanzitutto definiscono la coordinazione come il *gestire le dipendenze tra le attività* [Malone], definendo in seguito quattro categorie di dipendenze collegate con altrettanti processi di coordinazione usati per gestirle.

Gestione delle risorse condivise

> *"Laddove diverse attività condividano delle risorse limitate [...], è necessario un processo di allocazione delle risorse per gestire le interdipendenze tra queste attività."* [Malone]

Abbiamo appena preso in considerazione le risorse coinvolte in un Progetto Open Source. Alcune di esse possono essere considerate illimitate o quasi, come l'informazione ed il software disponibili liberamente, e per questo non hanno bisogno di particolari processi di allocazione dal momento che chiunque ne può disporre come crede. Altro discorso si deve fare per quanto riguarda le risorse limitate.

- *Risorse umane.* Innanzitutto bisogna distinguere ancora una volta tra i partecipanti volontariamente al Progetto e coloro che sono in qualche modo pagati da altri soggetti commerciali per partecipare al Progetto. La coordinazione di questi ultimi è completamente estranea all'organizzazione del Progetto Open Source e riguarda esclusivamente i loro obblighi nei confronti dell'azienda che li paga.
  In un progetto Progetto Open Source, in genere, non esiste un'autorità centrale che ha il potere di allocare le risorse umane per particolari compiti, verrebbe a mancare l'incentivo fondamentale della motivazione. Tuttavia sono osservabili una serie di convenzioni sociali che fanno sì che i partecipanti al Progetto, anche grazie all'azione di alcune figure di riferimento[19], tendano a coordinarsi dandosi anche degli obiettivi e dei tempi attraverso un continuo feedback mediante i loro mezzi di comunicazione tipici.

[19] Vedi paragrafo 3.3.3, pg. 79

- *Risorse tecniche.* In questo tipo di risorsa sono comprese quelle tecniche personali e condivise - come hardware e documentazione - nonché le strutture e i servizi disponibili in Internet. Queste risorse sono da considerare limitate in quanto soggette a vincoli tecnici e di quantità. Anche qui non vi è in genere una responsabilità diretta da parte del/dei coordinatore/i del Progetto per la gestione di queste risorse, a parte il caso di risorse condivise gestite direttamente. In genere queste risorse sono gestite, quando non sono personali, da chi le mette a disposizione: università, aziende interessate, Internet service provider.

Gestione delle relazioni produttore/consumatore

> *"... una relazione produttore/consumatore [è] una situazione in cui una attività produce qualcosa che viene usato da un'altra attività."* [Malone]

La natura altamente modulare del software favorisce alti livelli di riuso. Questo fenomeno è molto accentuato nei Progetti Open Source per due motivi:

- Il software usato nel mondo open source è fortemente basato su architetture Unix, concepite per la massima modularità.
- La disponibilità dei codici sorgenti permette la *cannibalizzazione*, il riuso di parti di codice, altrimenti non disponibili né visibili.

Di relazioni produttore/consumatore nello sviluppo di Progetti Open Source ne troviamo fondalmentalmente di due tipi:

- *All'interno del Progetto.* Il lavoro all'interno di uno stesso Progetto è in genere diviso in compiti, a seconda dei vari moduli che compongono il codice. Spesso la realizzazione di uno di questi moduli necessita di parti di altri e quindi gli eventuali ritardi si ripercuotono in più moduli.

- *Tra diversi Progetti.* È molto comune che diversi Progetti utilizzino l'uno parti di codice dell'altro, questo anche per questioni di razionalità, praticità e risparmio di tempo, in sostanza per *... non dover inventare la ruota ogni volta* [Raymond1]. Ciò è favorito, oltre dalla libera disponibilità, anche dal fatto che spesso molti sviluppatori partecipano a più Progetti contemporaneamente.
  Spesso succede che il ritardo nello sviluppo di un particolare elemento di codice di un Progetto costringa chi ne ha bisogno in un altro Progetto a dover aspettare.

Gestione dei vincoli si simultaneità

> *"Un [...] tipo comune di dipendenza tra attività è quando devono svolgersi nello stesso momento (o non devono svolgersi nello stesso momento)."* [Malone]

Questi vincoli hanno in parte a che fare anche con quanto visto per le relazioni produttore/consumatore. Si pensi all'implementazione di una parte di codice che necessita di un'altra non ancora completata.

In genere comunque questi vincoli, per quanto riguarda un Progetto Open Source, riguardano soprattutto la comunicazione. La diversa dislocazione dei partecipanti al Progetto rende necessaria una certa coordinazione per poter comunicare in tempo reale utilizzando gli strumenti tipici che vedremo tra poco.

La modifica a parti di codice da parte dei molti sviluppatori che vi lavorano potrebbe dare dei problemi di sovrapposizione o di incongruenza se non organizzata. A questo proposito sono usati nei Progetti Open Source degli strumenti come il *Concurrent Versions System (CVS)*[20], che servono a gestire automaticamente l'uscita delle versioni e a tenere traccia delle modifiche e di chi le ha fatte.

[20] Vedi sezione 3.4.2, pg. 81

Gestione delle dipendenze tra compiti e sottocompiti

> *"Un tipo comune di dipendenza tra le attività è che un gruppo di attività sono [...] sottocompiti per il raggiungimento di un obiettivo superiore."* [Malone]

Malone e Crowston identificano due modi fondamentali in cui sono gestite di solito tali dipendenze:

- *Scomposizione degli obiettivi.* Si ha quando un gruppo decide di perseguire un obiettivo e per farlo lo scompone in attività - sotto-obiettivi - che svolte assieme faranno raggiungere l'obiettivo iniziale. Questo è un processo molto comune e diffuso in tutti i processi organizzativi umani, in particolare nell'ambito delle organizzazioni, soprattutto quelle caratterizzate da una struttura gerarchica.
  Nei Progetti Open Source questo modo di agire non è molto diffuso per vari motivi. A parte la generale mancanza di una struttura gerarchica e di comando e la scarsa propensione di un volontario a ricevere ordini, l'estrema volatilità e flessibilità dell'apporto in risorse umane renderebbe molto presto inadeguato un piano preciso di scomposizione degli obiettivi.

- *Unione degli obiettivi.* Si ha quando *diversi attori realizzano che ciò che stanno già facendo [...] può essere unito per il raggiungimento di un nuovo obiettivo* [Malone]. Questo è un fenomeno abbastanza comune nel mondo del software libero ed open source. Il fatto che il software dei vari Progetti sia liberamente disponibile e modificabile porta in modo naturale alla nascita di nuovi Progetti dalla combinazione delle risorse esistenti.

C'è un altro modello di gestione degli obiettivi tipico dei Progetti Open Source che non viene citato da Malone e Crowston ma a cui accenna Evers nel suo lavoro e che chiama Elaborazione Simultanea degli Obiettivi. In sostanza questo modello, simile a quello dell'unione degli obiettivi, vede il

lavoro contemporaneo di più soggetti sul medesimo obiettivo intervallato da momenti di discussione e confronto dei risultati per ottimizzare il lavoro di tutti.

In questo modo i membri del Progetto conserverebbero la libertà che contraddistingue il loro apporto prevalentemente volontario e aumenterebbero la loro produttività attraverso il confronto e la collaborazione.

3.3.3 Ruoli dei partecipanti ad un Progetto Open Source

Come ho già accennato sopra, i Progetti Open Source non sono caratterizzati da una autorità centrale che ha poteri organizzativi e dispositivi. Ci sono però dei ruoli ormai riconosciuti ed altri desumibili dal contesto, che permettono quel certo grado di organizzazione senza il quale un Progetto Open Source, come tutti i sistemi complessi, non potrebbe funzionare. È vero anche che questi ruoli sono maggiormente rilevabili e "isolabili" in specifiche sezioni del Progetto, piuttosto che nell'insieme; e sono:

- *Project Manager*. Se c'è dirige il Progetto. Coordina le varie fasi e scadenze e prende decisioni nel caso ce ne sia bisogno, il suo ruolo gli è comunque riconosciuto dai partecipanti al Progetto.

- *Mantainer*. È il responsabile per uno specifico componente del Progetto - per esempio una applicazione - e si occupa di tutto ciò che riguarda questo componente. Inoltre è l'interfaccia tra il componente ed il Progetto nel senso che il componente *comunica* attraverso di lui.

- *Amministratore*. È qualcosa di simile al mantainer, ma nello specifico è responsabile dell'amministrazione delle risorse come il sistema di supporto, il software di supporto e i servizi a disposizione del Progetto.

- *Sviluppatore*. Programma o lavora alla documentazione per un particolare problema. Come detto nessuno gli ha ordinato di farlo e nella

maggior parte dei casi lo fa volontariamente, mosso da una propria curiosità o necessità.

- *Utente attivo.* È colui che, non potendo o non volendo lavorare al Progetto, si rende utile in qualche modo segnalando errori o suggerendo nuove caratteristiche o miglioramenti.

È importante notare che spesso più d'uno tra i ruoli appena elencati possono ritrovarsi nella stessa persona come, al contrario, più persone possono rivestire lo stesso ruolo.

3.4 Strumenti e mezzi di comunicazione tipici

Chiaramente un Progetto Open Source, come qualsiasi forma di collaborazione organizzata, è caratterizzato da alcuni strumenti e da alcuni mezzi di comunicazione che permettono lo svolgimento e la coordinazione delle attività. Alcuni di questi strumenti sono inoltre diventati realmente indispensabili per un'efficace collaborazione tra sviluppatori spesso "remoti". Si ricordi che sono mezzi di comunicazione, anche se non li cito per semplicità, la normale posta elettronica, la documentazione - manuali ed *How-to* - e lo stesso codice sorgente, ricco di informazioni tecniche.

3.4.1 Internet

In un ambiente di sviluppo così distribuito - talvolta a livello mondiale - è necessario avere la possibilità di scambiare informazioni nel modo più rapido e meno costoso possibile. Queste informazioni possono essere:

- Discussioni.
- Modifiche al programma.
- Comunicazione di difetti o nuove idee.
- Informazioni relative all'organizzazione interna del progetto.

Il modello di sviluppo tipico del Progetto Open Source, pur già esistente in forme più primitive, ha avuto una enorme accelerazione ed ha raggiunto le dimensioni attuali in seguito alla diffusione su larga scala di Internet.

3.4.2 Concurrent Versioning System (CVS)

È un sistema di controllo delle versioni che permette di registrare la storia delle modifiche ad un codice sorgente. È uno strumento che organizza l'accesso e la modifica di una copia centralizzata del codice sorgente di un dato progetto, avendo cura di gestire in modo opportuno più modifiche tra loro incompatibili e tenendo traccia di tutti i cambiamenti.

CVS per le sue caratteristiche è fondamentale per l'organizzazione di un Progetto Open Source.

3.4.3 Mailing list, Usenet, IRC

Questi sono più strettamente strumenti di comunicazione, ognuno con delle caratteristiche peculiari che lo rende più adatto ad un particolare uso.

Mailing list. Sono un mezzo di comunicazione "*molti a molti*", molto usato per diffondere notizie generiche da parte di aziende od organizzazioni o per accendere discussioni su particolari argomenti tra un numero determinato di utenti "iscritti" interessati. Simili alle mailing list sono i Newsgroup[21].

Usenet. È un canale pubblico sul modello del bulletin board, perciò non ci sono limitazioni all'accesso e non si sa quindi chi leggerà. Viene utilizzato prevalentemente per discussioni generiche.

IRC È uno strumento molto simile ad una chat e suddivisa in *canali* dedicati - per esempio `#linux-it`, `#irc-prosa`, ecc - è lo strumento di discussione più interattivo, utilizzato per discussioni più complesse e su problemi contingenti, che via email e Mailing list durerebbero troppo tempo. Lo svantaggio è

[21] Vedi Glossario

che l'informazione raccolta in questo modo spesso non è registrata per poterla riutilizzare, anche se oggi alcuni programmi lo fanno.

3.4.4 Bug Tracking System

È un sistema attraverso il quale è possibile tenere traccia e gestire tutte le segnalazioni sui difetti di un software. È sia uno strumento, un database[22] dei bug, sia un mezzo di comunicazione per la segnalazione degli stessi.

Tutti i Progetti di una certa consistenza - come varie distribuzioni, Mozzilla, Samba, Evolution, ecc - hanno un bug tracking system gestito da sviluppatori che controllano le segnalazioni assegnandole a seconda dell'importanza alla correzione.

3.4.5 Altro

In rete sono presenti anche altre risorse più complesse per lo sviluppo dei Progetti Open Source. Tra i tanti siti che si occupano di fornire varie forme di servizi, ne ho scelto tre che in qualche modo presentano le varie possibili facce del fenomeno: *Sourgeforge*, *Freshmeat* e *Collabnet*.

Sourgeforge. Fornisce un servizio gratuito[23] per gli sviluppatori di software libero ed open source. Questo servizio fornisce a chi ha un Progetto Open Source ma manca dei mezzi, l'accesso ad un CVS, la possibilità di istituire una mailing list, un bug tracking, un sito ed un archivio completo dei file.

Freshmeat. Mantiene uno dei più grossi archivi[24] su Linux e software open source. Migliaia di applicazioni sono meticolosamente catalogate nel database di Freshmeat con aggiornamenti giornalieri. Sostanzialmente consiste

[22] Vedi Glossario

[23] `http://sourgeforge.net`

[24] `http://freshmeat.net`

in annunci del nuovo software con una descrizione ed i link da cui scaricarlo ed ottenere maggiori informazioni sugli autori e sulle precedenti uscite e variazioni.

Collabnet. È un servizio offerto[25] da un soggetto commerciale che si propone di fornire strumenti e servizi per aziende che intendano adottare sistemi di sviluppo del software di tipo collaborativo e distribuito sul modello del Progetto Open Source.

3.5 Dimensioni del fenomeno

Le comunità di sviluppo open source hanno creato, distribuito e continuano a sviluppare con successo molti importanti Progetti Open Source, tra i più significativi torniamo a ricordare:

- Il Progetto GNU
- Il Web server Apache
- Il sistemi operativi Linux e FreeBSD

Oltre anche a:

- I linguaggi di programmazione Perl e Tcl
- L'editor Emacs
- Il compilatore gcc

Abbiamo visto che concetti quali software libero ed open source significano molto più di semplice accesso al codice sorgente di un programma e che non c'è un unico modello condiviso di sviluppo di un Progetto Open Source. Tuttavia il principio alla base di tutto ciò che guida l'intero movimento open source è che condividendo i codici sorgenti gli sviluppatori possono cooperare

[25] `http://collab.net`

secondo un modello rigoroso di peer-rewiew e che possono trarre vantaggio dal lavoro di debugging parallelo per far guadagnare al software in innovazione e velocità di sviluppo. Il sistema di licenze, d'altra parte, assicura un mercato per l'integrazione ed il supporto dei prodotti open source.

Un problema fondamentale nel cercare di capire, stimare e trarre delle conclusioni sul fenomeno dello sviluppo collaborativo open source è la scarsezza di informazioni su chi sono, ma soprattutto quanti sono esattamente coloro che partecipano a questo sviluppo ed in che modo vi partecipano. Questa scarsezza di informazioni - paradossale se confrontata con la mole di righe di codice e di documentazione prodotti - genera una cronica sottostima delle reali dimensioni di un processo collaborativo che si presenta organico e distribuito.

I dati più recenti a riguardo provengono da uno studio [UNC] del 1999 compiuto dall'Open Source Research Team presso la University of North Carolina (UNC) sulla comunità di utenti/sviluppatori di Linux. Linux infatti è forse l'esempio più significativo di sviluppo di un progetto open source su larga scala. La sua comunità è molto attiva, geograficamente distribuita ed impegnata nella diffusione del software e della documentazione correlata.

3.5.1 Linux Repository e Linux Software Maps

Linux, nato nel 1991 dall'iniziativa personale di uno studente finlandese[26], è oggi un sistema operativo maturo che può funzionare su quasi tutte le piattaforme hardware. Sta assumendo un ruolo sempre più significativo nei *business plan* di alcune tra più grandi aziende informatiche su scala mondiale - IBM, Hewlett Pachard, Sun Microsistem, ecc -, nei laboratori di ricerca delle università e nella nascita di una serie di nuove imprese focalizzate su servizi di supporto e integrazione a Linux.

Le statistiche[27] parlano di un 31% di server in internet governati da Linux

[26] Vedi sezione 1.3.2, pg. 27

[27] `http://www.netcraft.com/survey`

e di un 6% di computer su cui è installato il sistema operativo libero. Cifre in continuo aumento ad un tasso elevato.

Il *Linux Kernel Project*[28], il progetto di sviluppo del kernel Linux, è tuttora diretto da Linus Torvalds che può contare su di un significativo e crescente numero di di co-sviluppatori di alto livello distribuiti su scala mondiale.

Le comunità di sviluppatori delle applicazioni per il sistema operativo e della documentazione relativa crescono nella stessa proporzione degli utenti e delle installazioni. Ed è proprio su queste comunità che si concentra lo studio dei ricercatori di UNC.

Gli sviluppatori delle applicazioni sono un gruppo molto esteso e variegato, che comprende tutti i livelli di contribuzione, a partire dai più significativi ed estesi fino ad arrivare a quelli più semplici e mirati a singoli aspetti delle utility. Un elemento di enorme aiuto per la raccolta e consultazione di questi dati è dato dall'esistenza del Linux Repository, un sito ftp[29] ed uno http[30] ospitati presso il *MetaLab* di UNC, che raccoglie praticamente tutto il materiale riguardante Linux disponibile in Internet, comprese le più inportanti distribuzioni del codice del kernel, il *Linux Documentation Project sulla documentazione*, tutto il software collegato e il materiale di supporto. Queste ultime due categorie sono appunto il frutto dello sforzo della comunità di sviluppatori delle applicazioni per Linux.

Non è infrequente che gli archivi Linux del MetaLab accessibili via ftp e http, superino i 100.000 accessi nell'arco di una giornata. In totale alla fine del 1999 i file presenti in questi archivi erano piu di 125.000.

Lo studio si è basato sull'analisi degli oltre 4500 *metadata*[31], chiamati *Linux Software Maps (LSM)*, associati dal 1994 ad ogni entrata all'archivio. I LSM, contenenti i dati relativi ai contributori ed al materiale depositato, danno un quadro preciso degli aspetti demografici della massa di individui coinvolti nello sviluppo degli strumenti applicativi e di utility.

[28] `http://www.kernel.org`
[29] `ftp://metalab.unc.edu`
[30] `http://metalab.unc.edu/pub/Linux`
[31] Vedi Glossario

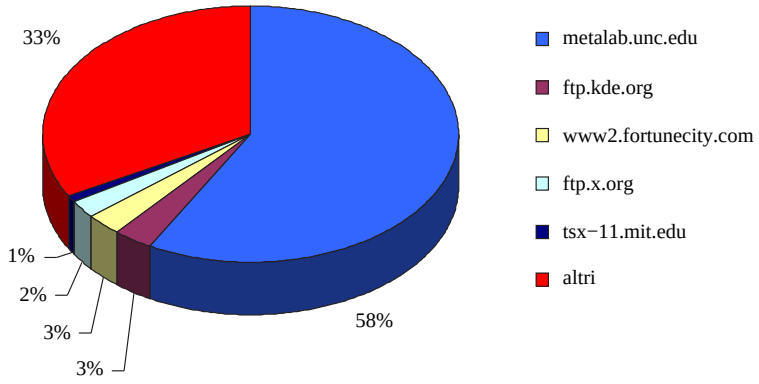

Figura 3.1: Siti primari indicati nei LSM

Il fatto che l'archivio al MetaLab contenga se non tutto, quasi tutto il il software documentato attraverso i LSM, consente di considerare i dati della ricerca come rappresentativi. Inoltre il suo sito funge da mirror[32] della maggior parte dei più famosi archivi su Linux e delle più importanti distribuzioni. Da quello che risulta dai dati contenuti nei file LSM quello del MetaLab è il server primario su cui deposita i propri contributi il 58% degli sviluppatori, come si può notare dalla Figura 3.1.

3.5.2 I dati

Come detto, ogni contributo inviato al MetaLab deve essere accompagnato dal file metadata LSM. È possibile quindi osservare come si siano distribuiti negli anni i contributi software. La Figura 3.2 mostra come dal 1993 al 1999 tali contributi siano cresciuti ogni anno. Il dato del 1999 risulta ridotto in quanto le rilevazioni si fermano al Settembre di quell'anno. Bisogna però considerare che la politica di archiviazione del MetaLab è quella di sostituire i vecchi LSM quando arriva una nuova versione di un pacchetto software. Perciò quella appena considerata non può essere una rappresentazione molto accurata di quanti contributi effettivi sono stati inviati per ogni anno, tuttavia può dare un'idea generale. I LSM contengono, tra le altre informazioni,

[32] Vedi glossario

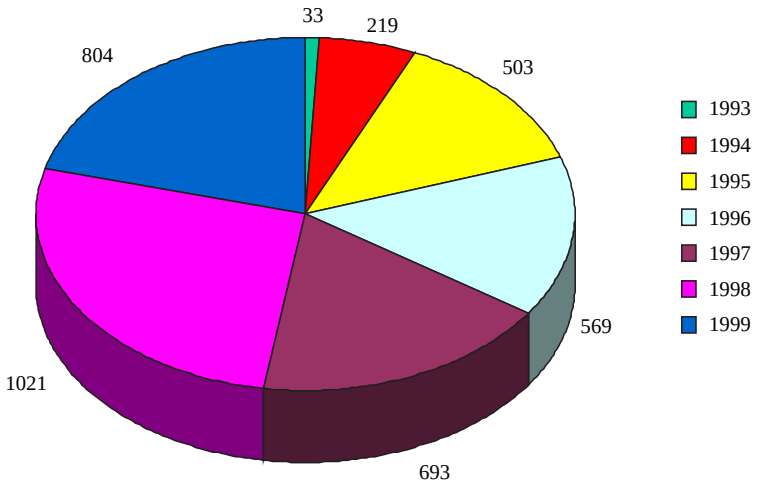

Figura 3.2: Distribuzione dei LSM per anno

anche l'indirizzo email dell'autore del contributo. È possibile quindi suddividere i contributi per provenienza, come illustrato dalle figure 3.3 e 3.4. Dai dati si nota come gli sviluppatori di applicazioni per Linux provengano in larga parte dal mondo commerciale (`.com`[33]) e da ambiti educativi (`.edu`).

Anche se i suffissi `.com`, `.edu` e `.net` non si possono con certezza definire geograficamente, appare forte la componente demografica europea, che è rappresentata in dettaglio nella Figura 3.4[34].

Un'altro dato interessante è quanto spesso il singolo sviluppatore contribuisce al software open source. Utilizzando i dati sull'identità degli autori contenuti negli LSM, risulta dalla Figura 3.5 che la stragrande maggioranza ha conferito uno o due contributi, mentre un numero molto basso ne ha dati più di cinque. Ciò sta a dimostrare il fatto che lo sviluppo del software open source non è dominato da pochi prolifici sviluppatori, ma piuttosto da molti portatori di contributi isolati. Un dato che emerge infatti dall'ultima analisi è che il numero totale di individui che hanno partecipato allo sviluppo

[33] Anche se oramai `.com` è un suffisso che non necessariamente è usato in via esclusiva da soggetti commerciali

[34] Questi dati sono chiaramente sottostimati dal momento che si presume che una buona parte dei suffissi `.com` siano localizzati in Europa

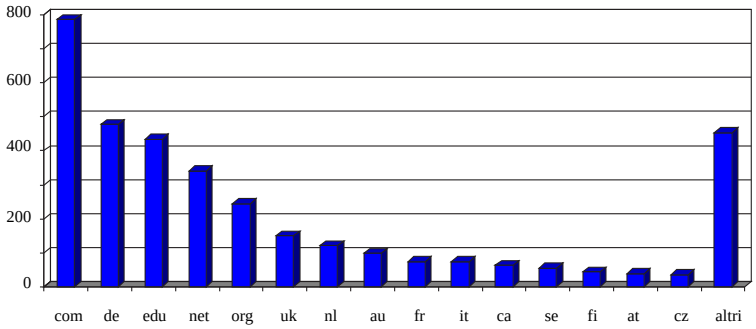

Figura 3.3: Suffissi email degli autori dei LSM

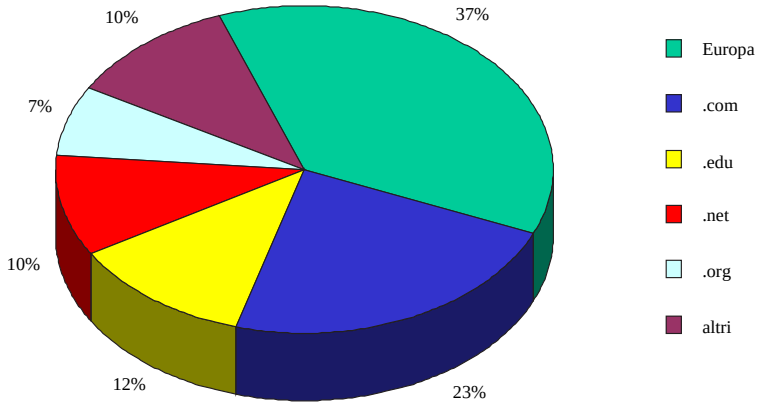

Figura 3.4: Componente europea dei LSM

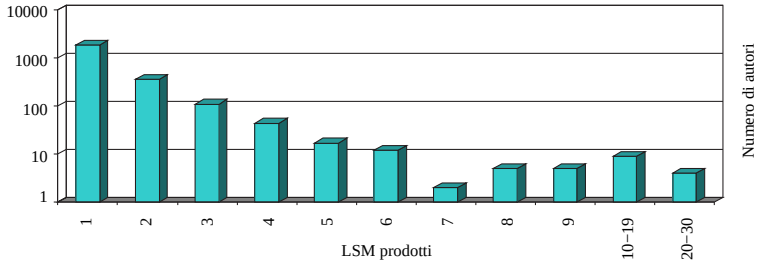

Figura 3.5: Numero di contribuzioni per autore

delle sole applicazioni per Linux è di 2429[35], considerando che ci si riferisce solamente ai contributi certificati ed ufficialmente inseriti nelle distribuzioni e nell'archivio del MetaLab. Nel corso dello studio, i ricercatori della UNC hanno anche svolto un esperimento per studiare la dinamica giornaliera del processo di contribuzione. L'osservazione, della durata di un mese, dal 16 Marzo al 16 Aprile del 1999, si è basata sul monitoraggio di alcune directory dell'archivio, quelle con la massima concentrazione di file LSM[36], in due momenti distinti della giornata: 8.30 e 20.30.

L'esperimento evidenzia come ci sia un costante traffico giornaliero di contributi, con dei picchi dovuti ad eventi esterni come per esempio importanti rilasci di nuovo software. È stato possibile inoltre osservare la tipologia dei contributi. La Figura 3.6 mostra che circa un terzo dell'attività osservata è consistita in cambiamenti o aggiornamenti di file già esistenti nell'archivio, mentre circa due terzi rappresentano aggiunte di nuovi file.

3.5.3 Osservazioni

I risultati di questo studio, pur datati, danno delle informazioni significative per la comprensione del fenomeno dello sviluppo dei Progetti Open Source. Si è presa in considerazione un'area, lo sviluppo di applicazioni per Linux, che può essere considerata rappresentativa del movimento di sviluppo di base

[35]Non sono ovviamente considerati i partecipanti ad altri Progetti Open Source
[36]In particolare applicazioni, utility e documentazione

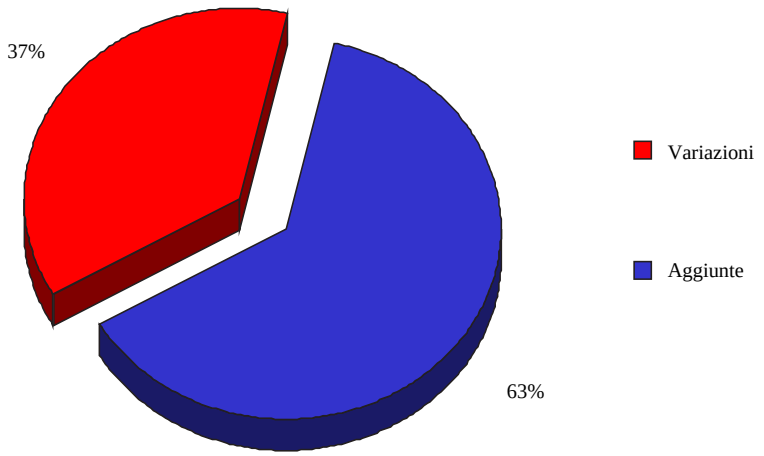

Figura 3.6: Nuovi contributi ed aggiornamenti

proprio perché non si riferisce ad ambiti più tecnici e ristretti come il Linux Kernel Project.

I dati confermano una larga partecipazione a questo livello di sviluppo del software libero ed open source con un'importante componente europea - riflettendo forse le origini geografiche e culturali di Linux - ed un contributo dalle istituzioni educative che rimane massiccio.

L'esperimento eseguito inoltre rivela una costante attività volta in gran parte all'archiviazione di nuovi file oltre all'aggiornamento ed al mantenimento dei vecchi.

Capitolo 4

Prosa s.r.l.
MixadLive s.r.l.
Ximian inc.
Tre realtà open source

4.1 Le dinamiche di Mercato

L'economia dei paesi avanzati si sta completamente ristrutturando e riorganizzando. La risorsa informazione è il cardine del cambiamento ed è sotto gli occhi di tutti la straordinaria accelerazione che la tecnologia ed il software in quanto strumento hanno impresso a questa evoluzione. Le imprese devono restare al passo o persino precedere il cambiamento per mantenere il vantaggio competitivo acquisito o per conquistarlo. In molti settori, soprattutto dell'area hi-tech, il cliente vuole prodotti demassificati, adattati alle sue specifiche esigenze ed in tempi sempre più brevi. Aziende estremamente flessibili ed in grado di coinvolgere il cliente nella progettazione tendono a lasciarsi alle spalle i concorrenti organizzati in modo tradizionale.

Molte *parole d'ordine* circolano già da alcuni anni: *Just-in-time*, *produzione flessibile*, *ottimizzazione organizzativa*, *qualità totale*, *produzione di massa*

a misura del cliente, ecc. Per reggere in questa economia della velocità, le linee di confine tra azienda, fornitore e cliente (singoli o altre aziende) devono essere sfumate e compenetrarsi ed occorre essere in grado di individuare e gestire beni o servizi già esistenti che formeranno il prodotto finale prima ancora che questo venga creato. L'*Azienda Virtuale* di Davidow e Malone [Davidow] congiunge tutte queste caratteristiche. L'aumento della tecnologia informatica ha fatto sì che la forza e la produttività di un'impresa, anche tradizionale, si basino sempre di più sulla risorsa informazione e sulla sua creazione. Internet ha favorito la nascita di un nuovo tipo di impresa che agisce in una realtà virtuale omogenea ed interconnessa, dove la competizione si svolge sul piano della velocità, della qualità e dell'efficienza in una corsa sfrenata alla soddisfazione dei bisogni e delle necessità, anche quelle più velleitarie.

4.1.1 L'Impresa Open Source

Le imprese che si dedicano all'open source rientrano solo in parte nel modello dell'Azienda Virtuale. Utilizzano gli stessi media e si basano sulla stessa risorsa principale: *l'informazione*; tuttavia differiscono dal modello di Davidow e Malone per le caratteristiche e la stessa natura del prodotto su cui si basano. Il software libero ed open source ed il loro modello di sviluppo, il Progetto Open Source, non obbediscono alle leggi ed ai tempi del mercato. Come vedremo in seguito, i concetti di *scadenza* e di *dovere aziendale* sono in parte estranei ai componenti delle comunità open source. Ciò non vuol dire che su questo tipo di prodotto sia impossibile creare un business, anzi, tuttavia è necessario tenere in considerazione tutti i suoi aspetti per trarre vantaggio dai suoi punti di forza ed evitare di essere danneggiati dai suoi punti deboli. Le Imprese Open Source hanno inoltre una caratteristica in più: si basano su un patto etico che tutte le parti devono rispettare considerando che la *materia prima* è donata all'azienda dalla comunità e che la logica del dono prevede che questo venga ricambiato.

La struttura dei costi e dei ricavi delle aziende fondate sul software libero o

comunque open source è completamente diversa rispetto a quelle tradizionali: la soglia di ingresso è estremamente bassa e basso il costo di produzione, ma il bene offerto può essere reperito facilmente e a bassissimi costi o quasi dai possibili clienti.

In realtà ciò che viene offerto sul mercato non sono beni ma servizi, il valore di vendita è indiretto. Ad esempio le distribuzioni Linux commerciali come SuSE, Red Hat, Caldera, ecc., non vendono i singoli bit del sistema operativo ma il valore aggiunto nell'assemblaggio dei diversi software che lo compongono e le garanzie di funzionamento e compatibilità con altri sistemi della stessa marca, oltre all'assistenza gratuita per un certo periodo di tempo ed opzioni per l'estensione dell'assistenza stessa. I distributori Linux, inoltre, sono costretti a competere con modalità che apportano benefici ai consumatori ed al mercato in generale: infatti, la Licenza GPL li obbliga a condividere le migliorie e le nuove caratteristiche via via sviluppate che prontamente vengono inserite nei propri pacchetti dalle altre distribuzioni ed evita le frammentazioni.

Alcune aziende fondate sull'open source vendono solo servizi (supporto, ricerca, sviluppo e formazione) come Linuxcare oppure documentazione, ad esempio i manuali sul software open source della casa editrice americana O'Reilly & Associates o della giovane ed italianissima Hops Libri.

Un'altra caratteristica dei software open source a vantaggio degli utenti, è che questi potranno essere sempre supportati anche se l'azienda che li ha distribuiti è scomparsa o se i programmatori che li hanno sviluppati all'interno di un'azienda non fanno più parte del suo organico: senza la disponibilità dei sorgenti ed il permesso di modifica ciò non sarebbe possibile. Anche la ripartizione del rischio, quindi, può far optare le aziende e gli utenti per questo tipo di software, non solo la ripartizione dei costi di sviluppo.

Non sempre, però, la scelta open source è la più conveniente per l'impresa. In assenza di alternative funzionali e dell'effetto rete è meglio tenere i propri bit propretari e scegliere la soluzione commerciale come accade ad esempio per un software innovativo che copre una nicchia di mercato non occupata

da altri.

Alla luce di tutto ciò le aziende tradizionali possono avere diversi tipi di adattamento a seconda del posizionamento sul mercato loro e dei loro prodotti. La Microsoft alla comparsa di Linux ha preferito rispondere con tattiche *FUD* (Fear, Uncertainty, Doubt) ovvero tendenti ad alzare i costi di transazione della concorrenza, oggi comincia a dare i primi segni di apertura rispetto ai metodi open in considerazione del cambiamento riscontrato nel mercato. Altre società si sono già *convertite*, distribuendo i propri prodotti completi dei sorgenti e con licenze open source, come è avvenuto per Netscape che nel '98 ha reso disponibili i sorgenti del suo web browser avviando il progetto *Mozilla* e riguadagnando fette di mercato sottratte da Internet Explorer di Microsoft. Lo stesso sta accadendo per Sun Microsystem che ha rilasciato StarOffice[1] completo dei sorgenti e sotto licenza GPL con l'ultima versione.

4.2 Redditività del software

Un quesito ricorrente che sorge spontaneo nell'approccio al mondo dell'open source è se questo modello, in genere efficace a livello di qualità del software prodotto, lo sia anche a livello economico, se produce dei profitti.

Generalmente si è portati a pensare al modello proprietario come all'unico in grado di garantire un ritorno economico, grazie alle royalties incassate dalle copie vendute, ma questo è vero solo in parte. Non sono effettivamente molte le aziende, in genere molto grandi, che possono basare le loro fortune quasi esclusivamente sulla vendita di copie di software proprietari da loro prodotti. Non sono molte proprio perchè la pratica di mantenere segreti i codici sorgenti dei loro prodotti ha portato ad un progressivo sfoltimento della concorrenza ed all'assunzione di posizioni monopolistiche non sempre corroborate da una effettiva maggiore qualità.

[1] Suite multipiattaforma per l'ufficio

Su questo aspetto, la concorrenza, il software open source può forse offrire una maggiore possibilità di confronto ad armi pari.

4.2.1 Quanto *costa* realmente il software

Anche se a prima vista ad un addetto ai lavori può non sembrare, il software - in generale - è un prodotto estremamente complesso, soprattutto nella fase dello sviluppo. Dietro la sua realizzazione esistono una lunga serie di costi che giustificano gli ingenti investimenti attuati in genere dalle tradizionali imprese che producono e mantengono software proprietario, costi che vengono ammortizzati in seguito con i ricavi della sua vendita.

Il Total Cost of Ownership (TCO)

Non irrilevante è il problema dei costi per chi tale software lo acquista. L'impresa che intende acquistare un prodotto di supporto alla sua attività deve fare i conti non solo con il suo prezzo d'acquisto, ma anche con tutti i costi futuri collegati a tale decisione, la cui somma, assieme all'esborso iniziale, va a comporre quello che è generalmente chiamato, *Total Cost of Ownership (TCO)*[TCO].

Nella teoria il Total Cost of Ownership è formalmente definito come *"la somma di tutte le spese ed i costi associati all'acquisto ed all'uso di equipaggiamenti, materiali e servizi"* [TCO]

Questo strumento, usato oggi da molte grosse imprese, dà la possibilità - se usato correttamente - di effettuare accurate misurazioni ex-ante ed ex-post del costo totale di un'operazione, identificando eventuali variazioni rispetto a quanto pianificato e permettendo di agire nel modo corretto per evitare futuri problemi.

Il Total Cost of Ownership, come detto, è un concetto che va oltre il semplice momento dell'acquisto. È visto dalla moderna teoria contabile come un'estensione dell'*Activity Based Costing (ABC)*[MioloVitali], un sistema contabile basato sui costi che attribuisce loro appropriati driver in base alla loro componente di valore aggiunto; uno dei tratti caratteristici di questo

sistema è che pone al centro del calcolo dei costi le *attività* e non i prodotti. In questo senso il Total Cost of Ownership estende l'Activity Based Costing a tutte le attività creatrici di valore per l'organizzazione, sia interne che esterne.

Questa visione può essere utile a livello decisionale, dando indicazioni nel momento in cui si studiano strategie di prezzo e di mercato; può permettere un approccio più corretto ed integrato alla gestione della supply chain ed alla pianificazione dell'attività produttiva che porti ad una riduzione del costo totale e influenzi così i profitti.

Il TCO nel software

Il prodotto software non solo non differisce da altri in questo senso ma addirittura, a causa della sua complessità, risulta semmai più oneroso. In particolare esiste tutta una serie di voci di costo connesse all'adozione degli strumenti informatici che fanno in molti casi lievitare drammaticamente il Total Cost of Ownership. Di seguito sono riportati alcuni tra i più significativi tra questi costi cui va aggiunto, ovviamente, il *costo d'acquisto*.

- *Pianificazione.* L'attività di pianificazione comporta costi connessi all'utilizzo di risorse, prevalentemente umane, sia interne che esterne. Al lavoro del personale interno che viene dedicato alla valutazione e definizione di quali sono gli effettivi bisogni dell'azienda, viene affiancato in genere il lavoro di consulenti esterni che poi saranno anche i fornitori del software.

- *Adeguamento dei sistemi.* Spesso per poter installare il nuovo software sono necessari ritocchi più o meno significativi ai sistemi esistenti, come ad esempio l'aggiunta di componenti hardware, nuove infrastrutture od altro software.

- *Efficienza operativa.* L'uso di un software, anche dopo la sua installazione, richiede nelle sue varie fasi una serie di risorse, da quelle umane

ad aggiornamenti hardware, che comportano dei costi. La scarsa efficienza operativa dei processi connessi ad un software sono dunque fonte di un deciso innalzamento dei costi; si pensi ad esempio alla presenza di errori nel software o ad una incompatibilità con gli standard usati per i dati o addirittura una scarsa intuitività o eccessiva complessità dell'interfaccia utente.

- *Training.* L'introduzione di nuovo software in una azienda spesso porta una certa spesa per l'aggiornamento o l'insegnamento ex-novo delle conoscenze necessarie agli utenti - impiegati - per usarlo. Ciò significa corsi e/o manuali d'uso.

- *Servizi.* Comprendono la manutenzione del software installato da parte dell'azienda fornitrice e l'intervento in casi d'emergenza come crash dei sistemi o malfunzionamento. Di solito tali servizi fanno parte del "pacchetto" completo acquistato con il software, ma sono comunque a pagamento.

- *Aggiornamenti del software.* In genere anche il software più testato contiene degli errori o comunque necessita di ritocchi dovuti alla sua normale evoluzione tecnica. Sono necessari dunque aggiornamenti, frequenti o meno, per poter disporre di un prodotto adeguato e funzionante e, normalmente, si pagano a parte.

- *Costi incidentali.* Il software installato opera in un ambiente complesso come quello aziendale e spesso il suo uso porta a dei costi non direttamente imputabili ad esso. Tali costi in genere hanno una loro giustificazione: il fattore umano. Ecco che allora una svista o un uso improprio del software possono portare all'accidentale cancellazione di dati anche importanti; o una errata valutazione soggettiva può scatenare delle conseguenze non imputabili al software in se, si pensi ad un macchinario che, nell'eseguire correttamente i movimenti dettati dal software che lo governa, provoca un danno materiale dovuto ad un suo errato posizionamento.

4.2.2 I servizi come reale fonte di reddito

Dalle valutazioni sul Total Cost of Ownership deriva un'importante osservazione sulla recente evoluzione del mercato del software: il business si sta spostando radicalmente dalla vendita dei programmi alla vendita dei *servizi correlati*.

Ciò non vuol dire che ci sia una drastica diminuzione della vendita di copie di software proprietario, si è ancora lontani da questo, ma i *costi correlati* per chi lo acquista stanno diventando sempre più onerosi, assumendo un'importanza sempre maggiore. Ciò porta a constatare la possibilità per le aziende di evitare di caricare il cliente del costo per copia del software venduto, nel momento in cui vengono generati abbastanza profitti dai soli servizi correlati. E questa possibilità è già una realtà osservata da alcuni anni negli Stati Uniti.

Il margine di profitto per copia, tolta la quota attribuita dei costi sostenuti per lo sviluppo, può essere distribuito sul ricavo derivante dai servizi, che in genere sono forniti durante un arco temporale anche pluriennale.

Se consideriamo inoltre che le aziende che utilizzano software libero o comunque open source devono sopportare costi di sviluppo molto minori in virtù di quanto già detto nei precedenti capitoli, capiamo il potenziale vantaggio competitivo a livello di prezzo di cui possono godere.

Ma forse è il caso di soffermarsi in modo un po' più approfondito sul concetto di Impresa Open Source: cos'è, come opera e come può essere redditizia e competitiva.

4.3 L'Open Source come business

Visto da vicino non c'è niente di strano o magico nel metodo di sviluppo open source dal punto di vista economico e non dovrebbe essere additato come impraticabile né visto come la panacea a tutti i mali dell'economia di mercato.

Abbiamo visto che non c'è un solo modello di sviluppo ben definito da

seguire pedissequamente. Il fenomeno open source, inteso come fenomeno generale comprendente tutte le sue varie anime, è semplicemente un nuovo modo di sviluppare, distribuire e licenziare il software. Alle imprese che comprendono i fattori economici, culturali e, volendo, politici che risiedono in una effettiva strategia open source, questo modello offre l'aspettativa di un business vitale con le forti premesse di un futuro caratterizzato da una domanda crescente.

Saranno rispettate le aspettative? Ci sono realmente tali premesse? Funziona davvero questo modello nel "freddo" mondo del business o è destinato a restare relegato tra le accoglienti e protettive braccia della comunità? Prima di analizzare dei casi reali di Impresa Open Source, vediamo più nel dettaglio quali sono gli elementi che la caratterizzano.

4.3.1 Perché creare un'Impresa Open Source

Perfino Richard M. Stallman, il fondatore della Free Software Foundation nel suo *GNU Manifesto*[2] non ha mai sostenuto che lo sviluppo del software non si dovesse pagare o comunque fosse un'attività non-profit. Semmai proponeva e propone l'abbandono del modello for-profit basato sul trattare il software come proprietà intellettuale, in favore di un modello di business che vedesse l'attività di sviluppo come una attività professionale.

Molte grosse imprese del settore hanno cominciato già da alcuni anni, e con un trend crescente, ad usare software libero ed open source come base per i propri prodotti commerciali e, in alcuni casi, hanno contribuito allo sviluppo di questo tipo di software atraverso donazioni in denaro alla Free Software Foundation o ai singoli progetti, hardware o tempo dei propri impiegati.

Eventi come il recente rilascio da parte di Netscape del codice sorgente del suo web browser *Communicator* hanno attirato l'attenzione generale sull'importanza del software libero ed open source sia per gli utenti che per i produttori, grazie in parte anche all'entrata in scena del concetto di *open*

[2] `http://www.gnu.org/manifesto`

source che ha contribuito a smorzare la diffidenza nata dalle incomprensioni sul concetto di *free software*[3].

Quali sono i motivi oggettivi, a parte le convinzioni personali sulla bontà del modello, che possono far decidere di abbracciare l'open source ad una impresa commerciale? Delle aree o necessità aziendali che possono trarre giovamento dall'adozione di una strategia open source ben applicata, tra le più significative si devono annoverare le seguenti [Hecker]:

- *Evoluzione del prodotto.* Il modello di sviluppo open source ben si adatta ad un'impresa che vuole ed ha bisogno di creare nuovi prodotti o migliorare gli esistenti. I costi connessi a queste attività, indispensabili nel mondo del software, potrebbero subire una drastica riduzione.

- *Qualità del prodotto.* Si collega al punto precedente l'osservazione della maggior qualità, già dalle prime versioni, di un software sviluppato correttamente con il modello open source che garantisce, tra l'altro, tempi di debugging minori.

- *Mantenimento del prodotto.* Un software sviluppato come Progetto Open Source può godere dell'appoggio della comunità dei suoi utilizzatori che, anche se piccola, può contribuire a tenerlo aggiornato.

- *Reclutamento e fidelizzazione dei collaboratori.* La comunità open source è ricca di individui dalle notevoli capacità tecniche e può fungere, con le dovute garanzie del rispetto degli ideali *free*, da capace serbatoio di giovani talenti o comunque di onesti tecnici.

Queste caratteristiche non solo possono giovare al prodotto in termini di qualità e funzionalità, ma possono anche aumentarne il valore, che è poi quello che i consumatori decidono di pagare se lo riconoscono.

[3]Vedi sezione 1.5.2, pg. 38

4.3.2 Il problema del codice sorgente

Il codice sorgente è visto oggi da molte delle imprese che producono software come qualcosa di simile ad una risorsa preziosa da proteggere ad ogni costo con licenze restrittive o altri mezzi[4] per impedirne la sottrazione per usi scorretti o quantomeno per non dare una mano alla concorrenza.

Penso che il problema del possesso esclusivo del codice sorgente vada ridimensionato in quanto, almeno in alcuni ambiti, irrilevante. Il codice sorgente acquisisce valore commerciale solo se lo si può utilizzare per creare prodotti e/o servizi da vendere, ma ciò si verifica con successo in genere nel caso di imprese dal marchio sufficientemente conosciuto e dalla buona posizione di mercato - come per esempio Microsoft.

Il fatto che un'impresa rilasci, sotto adeguata licenza, il codice sorgente di un suo prodotto, non vuol necessariamente dire che i concorrenti lo useranno con lo stesso successo o che influenzeranno negativamente il suoi affari. Il successo di un prodotto software è il risultato di molto più delle sole caratteristiche del codice, ma comprende fattori come, tra gli altri, la fiducia e la reputazione derivanti dalla qualità dei servizi forniti e l'efficacia dell'area commerciale e dei canali distributivi.

La disponibilità del codice sorgente di un prodotto software può aumentare direttamente il suo valore per i clienti a cui è diretto aiutandoli a risolvere una serie di problemi:

- I clienti possono trarvi una sorta di protezione del loro investimento nel caso chi ha loro fornito il software decida di non continuarne la produzione e l'assistenza o smetta semplicemente di esistere. In questo caso la disponibilità del codice sorgente del software permette di correggerlo e svilupparlo adeguandolo alle situazioni contingenti o di farlo fare ad altri.

- I clienti hanno la possibilità di capire meglio come funziona il software nel caso la documentazione fornita dal venditore risulti carente.

[4] Vedi sezione 2.1.2, pg. 41

- I clienti possono apportare o fare apportare al software le correzioni necessarie ai propri bisogni e che il venditore non è in grado o non intende eseguire.
- I clienti possono adattare il software a nuovi sistemi operativi o nuovi hardware non supportati dal venditore.
- I clienti possono creare versioni personalizzate del software o versioni rinnovate di esso. Possono anche creare intere nuove applicazioni utilizzando parte del codice in loro possesso, evitando di svilupparle ex-novo.

Ma la stessa disponibilità del codice sorgente può anche aumentare indirettamente il valore del software per i clienti:

- Chi compone manuali e chi insegna l'uso del software in questione può creare una documentazione più completa se per esempio quella fornita dal venditore risulta carente. E tale documentazione può essere di aiuto ai clienti stessi.
- Consulenti ed integratori di sistema possono acquisire familiarità con la tecnologia e le tecniche su cui è stato sviluppato il software divenendo esperti nell'implementare sistemi che si basino su di esso o che lo incorporino. Questo permette di creare una base di persone che possiedono le abilità per sviluppare applicazioni complesse basate sul software d'origine. Ciò significa che i clienti potranno avere la possibilità di sviluppare sistemi meno costosi o più funzionali sfruttando la competizione che si viene a creare tra chi ha acquisito le abilità necessarie.
- Terze persone interessate allo sviluppo del software in questione possono, in via più o meno autonoma, trovare degli errori nel codice o delle inconsistenze nella sicurezza, creando delle correzioni. Le correzioni possono in seguito essere inserite nel software originale migliorandolo ed aumentandone il valore per i clienti, che si trovano con un prodotto che genera meno costi di mantenimento.

- Altri produttori di software possono riutilizzare il codice del software in questione per creare modificazioni o aggiunte di caratteristiche non previste dal prodotto originale. Ciò incrementa certamente il valore del software per i clienti.

4.3.3 Il Prodotto Open Source

Nel precedente capitolo è stato descritto in generale il tipico Progetto Open Source basato sulla iniziativa e sulla adesione volontaria di *persone*. Tuttavia, nell'analizzare le caratteristiche di un'impresa che basa il suo business sul software libero e/o open source, è necessario prendere in considerazione il concetto di Prodotto Open Source.

Il Prodotto Open source può consistere in varie tipologie:

- Software prodotto da un'impresa che decide di renderne pubblico il codice sorgente secondo una licenza libera od open source.
- Software open source prodotto da un'impresa che riutilizza il codice sorgente reso disponibile da un'altra azienda.
- Software prodotto da un Progetto Open Source classico e riutilizzato da un'impresa per un proprio prodotto.
- Software prodotto da un Progetto Open Source promosso e coordinato da un'impresa anziché da singoli individui.

In linea di massima il Progetto Open Source promosso o sostenuto in via principale da un'impresa che intenda utilizzarlo come prodotto e come base per i suoi servizi, non si discosta molto dalle linee generali tracciate in precedenza per il Progetto Open Source tipico. La differenza sta nel fatto che è appunto l'impresa che lo fa partire o lo sostiene rendendo disponibile il codice sorgente del software di cui vuole stimolare lo sviluppo sulle basi open source. Se tale software risulta interessante e offre buone prospettive di sviluppo, e l'impresa offre le dovute garanzie di rispettare i principï della

comunità, non avrà difficoltà a trovare membri della comunità che affiancheranno il lavoro degli sviluppatori interni. È da notare inoltre che ciò offre l'opportunità all'impresa di reperire, all'occorrenza, buoni collaboratori da inserire nel suo organico.

4.3.4 Perché uno sviluppatore esterno contribuisce ad un prodotto open source

Se da un lato è comprensibile il contributo dato in via volontaria da uno sviluppatore ad un Progetto Open Source nato spontaneamente dalla libera iniziativa di singoli individui, non è così immediato e comprensibile lo stimolo che spinge lo stesso sviluppatore a contribuire ad un Prodotto Open Source: che vantaggi trae?

La risposta è in parte intuibile da quanto detto in precedenza sulle ragioni sociali ed antropologiche che stanno dietro l'aggregazione nella cosiddetta *comunità*.

I programmatori sono in gran parte portati, per loro inclinazione, a lavorare su software che si dimostri, in prospettiva, utile a risolvere problemi che essi considerano rilevanti. Se un'impresa produce del software che ha tali caratteristiche, almeno per alcuni sviluppatori, e ne rende disponibile il codice sorgente, facilmente troverà qualcuno che ci lavorerà sopra, almeno per ripulirlo dagli errori.

Detto questo, comunque, si possono osservare principlmente due diversi approcci collegati a questo fenomeno: uno più "commerciale", finalizzato ad un utilizzo anche remunerativo del proprio lavoro, e l'altro "non-commerciale", più vicino allo spirito ideale della *Cultura Hacker*.

Per quanto riguarda il primo approccio, quello definito "commerciale", lo sviluppatore può avere determinate finalità, tra cui:

- Sviluppare versioni personalizzate del software originale rispetto alle esigenze di un cliente o con specifiche caratteristiche per determinati ambiti di mercato.

- Creare e vendere applicazioni dedicate al software in questione.
- Essere assunti, come già accennato, dall'impresa che ha prodotto il software e ne ha rilasciato il codice sorgente.
- Creare a sua volta un'impresa che possa collaborare con l'impresa che ha lanciato il software open source.

Per quel che riguarda invece l'approccio definito "non-commerciale", come già detto nel primo capitolo[5], le motivazioni sono di carattere meno materiale. Tra queste la soddisfazione personale e un po' narcisistica di vedere che il proprio software è usato e richiesto da altri sviluppatori e, non ultima, l'idea di sostenere un ideale di libertà che per molti va ben oltre il mero riscontro economico.

4.3.5 Fonti di reddito delle aziende Open Source

È il momento dunque di rispondere ad una domanda fondamentale che aleggia su questo lavoro sin dall'introduzione: è possibile ricavare profitti dal software libero ed open source? L'incremento di valore di cui, come abbiamo visto, possono beneficiare i clienti, può portare denaro nelle casse delle Imprese Open Source?

Quali sono quindi le fonti di guadagno derivanti dallo sviluppo di software libero e open source, assumendo l'assenza del costo per licenza e della royalty, tipici del software tradizionale?

Principalmente si possono riassumere in alcune categorie fondamentali che corrispondono anche alle possibili aree di business per le imprese che si basano sull'open source:

- *Distribuzioni Software e supporto.* I distributori, come abbiamo già visto[6], vendono semplicemente copie di software libero e/o open source. Il software liberamente reperibile anche in rete spesso necessita di

[5] Vedi paragrafo 1.4, pg. 32
[6] Vedi sezione 1.3.5, pg. 30

conoscenze tecniche che, pur non dovendo essere eccezionali, non sono nel bagaglio culturale di chiunque. L'utente base quindi, spesso non ha le competenze per poter affrontare agevolmente e senza problemi l'installazione e la gestione di questi pacchetti software e preferisce pagare una cifra ragionevole per disporre di una distribuzione costruita apposta per poter essere accessibile anche ai meno esperti.
Accanto alle distribuzioni sono forniti a pagamento tutta una serie di prodotti di supporto post vendita e servizi del tipo che vedremo nella seguente categoria.

- *Servizi.* Alcune aziende si specializzano sui servizi. I servizi più diffusi tra quelli prestati sono: *supporto tecnico*, *customizzazione di software*, *training* e *bug-fixing* (correzione di bug) a pagamento.

- *Prodotti civetta.* La libera distribuzione del codice sorgente di un prodotto che si posiziona in un particolare ambito di mercato porta l'attenzione verso altri prodotti a pagamento, liberi o proprietari, che costituiscono la vera fonte di guadagno dell'azienda.

- *Software per il funzionamento di Hardware.* L'hardware non può funzionare senza il software appropriato (driver, compilatori e linker, applicazioni o interi sistemi operativi), ecco perché i produttori e i distributori di hardware investono somme considerevoli per sviluppare ed aggiornare tali driver. Normalmente questo software è reso liberamente disponibile ma senza i codici sorgenti, anche se ultimamente sempre più imprese tendono a rilasciarlo come software open source per stimolare la ricerca di maggior compatibilità e supporto ai lori prodotti.

- *Documentazione.* Manuali, libri, riviste e servizi di news, tutto materiale che diffonde informazioni e quant'altro sul software libero ed open source ai normali prezzi per questo tipo di supporti. La caratteristica di questa documentazione è che spesso viene resa anche disponibile liberamente in rete.

4.4 Prosa s.r.l. - ETLinux

Prosa s.r.l. è una realtà abbastanza particolare e merita un po' più d'attenzione. Nata, come vedremo, nel 1998 per produrre software libero, in capo a due anni ha visto la sua vita cambiare completamente per l'acquisizione da parte di una grossa company statunitense, Linuxcare inc.[7], per poi ritrovarsi in condizioni simili a quelle di partenza, a causa del ritiro di Linuxcare dall'Europa, alla fine del 2000.

Al momento in cui sto scrivendo Prosa formalmente non esiste, a dire la verità non esiste più dal momento dell'acquisizione. Tuttavia è sempre rimasta e rimane nei fatti un'entità ben definita, prima come Prosa Labs con Linuxcare, ora come unità produttiva di un nuovo soggetto, Ascensit s.r.l e domani, con la sua rifondazione ufficiale, come partner di quest'ultimo. Ma la cosa più singolare è la sopravvivenza dell'idea e dell'ideale di Prosa tra i membri della comunità italiana di utenti di Linux che, ben consapevoli delle vicissitudini della società padovana, continuano ad identificare il suo nome al gruppo dei suoi fondatori.

4.4.1 Nascita di Prosa s.r.l.

Prosa s.r.l è nata a Padova nel settembre del 1998 dall'iniziativa di sei amici che si erano conosciuti attraverso la comunità italiana di utenti Linux: *Pluto Linux User Group (PLUG)*. Tre di essi vivono nella provincia di Padova, uno in quella di Vicenza, uno si era stabilito per motivi di lavoro ad Helsinki in Finlandia ed il sesto vive a Pavia. Per quanto riguarda quest'ultimo in particolare si tratta di Alessandro Rubini, uno dei più noti sviluppatori ed attivisti della comunità Linux in Italia, autore di diverse pubblicazioni tra cui la più nota è il manuale *Linux Device Drivers*, edito nel 1998 da O'Reilly&Associates. A parte Rubini ed il socio "finlandese", entrambi attorno ai 35-40 anni, gli altri avevano, alla costituzione della società, un'età media di 22-23 anni.

[7] `http://www.linuxcare.com`

L'idea alla base della nascita di Prosa era di dimostrare la validità economica del modello open source. Per fare ciò si sono portate agli estremi le ipotesi iniziali, assumendo per statuto l'obbligo di usare, sviluppare e vendere *solo* ed esclusivamente software libero.

Le dimensioni iniziali di Prosa erano abbastanza piccole: 6 soci attivi, poi diventati 5 per l'uscita del socio vicentino e 4 collaboratori. Da tenere in considerazione il fatto che Prosa non ha avuto nessun investimento iniziale, a parte il capitale sociale minimo per una società a responsabilità limitata, quindi è cresciuta usando solo i propri mezzi e risorse. Il fatturato del 1999, alla vigilia dell'acquisizione da parte di Linuxcare inc., è stato di circa mezzo miliardo.

4.4.2 Attività svolte e mercato di riferimento

Nei piani dei soci Prosa doveva essere essenzialmente una società di consulenza, con la produzione di alcuni piccoli software, però *"nel mondo dell'open source è difficile "guidare" il mercato, in genere se ne è particolarmente guidati, specialmente se si è una società piccola e agli inizi come Prosa"* [Davide Barbieri, socio di Prosa]. È stato il caso anche di Prosa dunque: il "mercato", grazie alla distribuzione del software via internet come tutti i progetti opensource in canali dedicati[8], ha notato alcuni loro prodotti che si sono così autopubblicizzati.

In pratica, hanno usato i loro prodotti opensource come veicoli pubblicitari dato che il fatto che fossero disponibili a tutti su internet ne ha garantito una discreta diffusione, rendendo possibile usarli per la creazione di un *brand*.

4.4.3 Aspetti organizzativi

Dato il particolare settore e la provenienza dei fondatori e dei collaboratori dalla comunità Linux, ci sono alcuni aspetti peculiari su cui è il caso di soffermarsi e che sono comuni anche ad altre realtà simili da me osservate.

[8] Ad esempio Freshmeat: `http://freshmeat.net`

Una caratteristica di Prosa, almeno nella fase iniziale, è la pressoché completa assenza di ruoli formalizzati. Certo, vi era una suddivisione dei compiti e delle mansioni in base alle competenze personali ed in base al tipo di commessa svolta al momento, ma non erano il risultato di una determinazione formale dei ruoli, un po' sul modello dell'impresa artigiana. Volendo individuare dei ruoli precisi, si può dire che in sostanza vi era un *Project Manager* con compiti anche di area commerciale, come la gestione dei clienti, che in genere seguiva lo sviluppo; un altro socio assorbito quasi interamente da incombenze amministrative ed un terzo responsabile dell'area tecnica con compiti di definizione dei preventivi, controllo e gestione dei servizi post vendita. I restanti, compresi i collaboratori erano impegnati nello sviluppo

La struttura organizzativa di Prosa, almeno agli inizi, era quindi molto semplice. La suddivisione ideale a cui le intenzioni dei soci miravano era comunque quella di una normale *Piccola Media Impresa*, con la presenza di *business units* dedicate ai servizi offerti dall'azienda (consulenza, assistenza e corsi), ognuna organizzata al suo interno in modo autonomo e coordinata da un responsabile. Inoltre sarebbero state create un'unità amministrativa con compiti di segretariato e contabilità ed una commerciale per la gestione dei clienti.

4.4.4 Metodi di produzione e organizzazione del lavoro

Essendo Prosa nelle intenzioni dei suoi creatori una società completamente open source, gran parte dei progetti cui lavorava erano di dominio pubblico e venivano portati avanti esattamente secondo il classico modello di Progetto Open Source[9]. I lavori per i clienti, quindi, erano essenzialmente modifiche, sviluppi o adattamenti di questi progetti su cui gli sviluppatori di Prosa stavano lavorando.

Questi progetti consistevano essenzialmente nello sviluppo di alcuni tipi di driver, un programma gestionale ad uso interno ed in seguito anche di una

[9]Vedi capitolo 3, pg. 57

variante di Linux per sistemi embedded, che verrà descritta più diffusamente in seguito[10].

Modalità di acquisizione dei clienti

Non vi erano particolari modalità di acquisizione dei clienti. Questa è una delle peculiarità di Prosa che si può riscontrare diffusamente in molte società open source. Nella prima fase di Prosa, la società veniva contattata dai clienti, come detto, grazie alla presenza dei propri Progetti Open Source in rete. Il fatto che questi prodotti software fossero disponibili in rete nei circuiti dedicati[11], permetteva al cliente di venire a conoscenza di Prosa e di verificarne le capacità prima ancora di contattarli.

Il cliente tipo quindi in un periodo, 1998/99, in cui il concetto di software libero ed open source non era ancora molto diffuso, era l'azienda o l'ente che per qualche motivo - dirigenti o dipendenti vicini alla comunità o comunque interessati alle sue istanze - volessero impiegare software di questo tipo, principalmente per ragioni di contenimento dei costi. Il fatto che i clienti potessero essere pochi non era un problema in quanto Prosa è stata la prima azienda del suo genere in Italia e quindi era un riferimento abbastanza "obbligato" per chi avesse avuto bisogno di questo genere di prodotto.

Va detto che grazie alla fama nell'ambiente del software libero di Rubini e ad alcune sue conoscenze in ambito editoriale, Prosa beneficiò alla nascita di una certa attenzione nel settore dell'open source nonché di un paio di articoli sul quotidiano *Il Sole 24 ore*.

Valutazione dei bisogni del cliente e definizione dei preventivi

Anche in questo caso il processo di valutazione non differisce da quello di una normale azienda di piccole dimensioni, con un processo di valutazione pre-preventivo non particolarmente formalizzato in procedure rigide, ma aperto e flessibile.

[10] Vedi sezione 4.4.5, pg. 113

[11] Proprio sito (`http://www.prosa.it`), freshmeat (`http://freshmeat.net`) e altri

Un particolare però risulta importante nell'osservazione di questi aspetti. Nel mondo dell'open source è possibile riutilizzare il codice scritto da altri e messo a disposizione in rete dagli autori. Questo permette di avere a disposizione molti prodotti software già realizzati e testati che possono essere parzialmente o totalmente riutilizzati con una conseguente diminuzione del *tempo uomo* in fase di realizzazione, e quindi dei costi del servizio. In pratica, a parte lavori complessi in cui è necessaria una parte di sviluppo corposa, l'impresa open source mette a disposizione del cliente semplicemente la sua competenza da applicare a prodotti già sviluppati e testati.

Nella successiva fase di redazione del preventivo veniva definito dal responsabile dell'area tecnica nei dettagli il lavoro da svolgere dal punto di vista tecnico, suddividendolo in *milestone* o obiettivi e veniva stimato il tempo necessario a completare ogni obiettivo, ottenendo il tempo totale per la realizzazione del lavoro e i suoi costi. Successivamente questi dati venivano passati al responsabile dell'area amministrativa per un controllo di convenienza economica.

Allocazione delle risorse umane e definizione delle responsabilità

Naturalmente nella definizione del preventivo e della sua tempistica veniva presa in considerazione la disponibilità del personale, in questo caso gli sviluppatori, in base agli altri progetti a cui si stava lavorando ed al raggiungimento dei relativi milestone.

Una cosa importante da considerare quando si parla di sviluppatori open source è la loro partecipazione a Progetti Open Source estranei o meno all'attività aziendale. Spesso infatti questi soggetti sono coinvolti come semplici sviluppatori o anche come mantainer in vari Progetti Open Source, quindi svolgono una attività che non è direttamente collegata a quella aziendale. Nei compiti che svolgono per l'azienda, però, in genere utilizzano programmi o parti di codice che sono disponibili e testati proprio grazie al loro stesso lavoro o a quello di altri nei relativi Progetti Open Source. Ecco perché dunque nelle tipologie contrattuali degli sviluppatori che lavorano per imprese

open source, può trovarsi il diritto esplicito a disporre di una certa percentuale dell'orario lavorativo per la partecipazione a Progetti Open Source o comunque per una attività di documentazione ed aggiornamento personale.

Viste le dimensioni della società, in Prosa non vi era una particolare definizione delle responsabilità in relazione ai progetti, se non relativa ai compiti descritti poco sopra.

Produzione, controllo avanzamento, qualità

Anche se non formalmente definita c'era un organizzazione della produzione, dalla fase di progettazione fino alla consegna del lavoro, abbastanza tradizionale. Il lavoro veniva diviso in milestone e ognuna di queste assegnata ad una persona di competenza che, doveva a seconda dei casi, coordinare il lavoro o eseguirlo essa stessa secondo una scala temporale precisa.

Il controllo della produzione ed il coordinamento veniva affidato al Project Manager che tra l'altro teneva anche i contatti con il cliente. Non erano esclusi contatti di carattere tecnico con il cliente direttamente da parte degli sviluppatori qualora questi non comportassero un discostamento sensibile dal progetto originario.

Livello di formalizzazione del progetto e documentazione

La formalizzazione del progetto e delle sue fasi di attuazione avveniva in fase di preventivo. In alcuni casi era necessario uno studio approfondito o la realizzazione di un prototipo, che diventavano quindi un lavoro aggiuntivo pagato dal cliente. Raramente oltre all'accettazione firmata del preventivo si giungeva alla stipula di veri e propri contratti.

La documentazione veniva prodotta *ad-hoc* per il cliente solo nel caso in cui non esistesse qualcosa di già disponibile su Internet e limitatamente alle modifiche o al lavoro originale eseguito.

La produzione di ulteriore documentazione su parti di codice o su programmi liberi già esistenti consisteva in un ulteriore lavoro pagato dal cliente.

Livello di modularizzazione, standardizzazione delle strutture e dei programmi

La produzione di software libero, in particolare su Licenza GPL, permette e favorisce il riuso e l'integrazione di codice già esistente, quando non addirittura di interi programmi. Comunemente questo approccio permette di non *reinventare la ruota*, ossia di non impiegare risorse per fare del lavoro già fatto da altri

In informatica ad esempio esistono le librerie, che non sono altro che raccolte di pezzi di codice che è possibile riutilizzare. Questo è il primo concetto che Prosa ha sempre seguito nello sviluppo, cioè la possibilità di riutilizzare il codice.

Sistema di incentivi

L'incentivo principale dichiarato per i tecnici di Prosa è la garanzia che il software da loro prodotto fosse libero.

Non erano esclusi, quando le condizioni lo avessero permesso, premi in denaro legati alla produttività.

4.4.5 ETLinux

Il più significativo tra i Progetti Open Source a cui ha lavorato Prosa è stato sicuramente ETLinux[12] in quanto, oltre ad essere il progetto che ha portato più notorietà a Prosa nell'ambiente open source, è stato concepito, iniziato e promosso dalla stessa società padovana.

Nasce dalla collaborazione con EuroTech s.p.a.[13], società *tradizionale* udinese produttrice di hardware e PC industriali che non produce software ma che era interessata a vedere i propri prodotti supportati dal maggior numero di piattaforme. Un precedente contatto con uno dei collaboratori di Prosa

[12] `http://www.etlinux.org`
[13] `http://www.eurotech.it`

porta EuroTech a commissionare alla società un supporto software per le sue schede.

La società di Udine rimane subito impressionata dalla velocità con la quale tale supporto viene scritto, nonché dalla completezza del supporto di rete integrato e già funzionante. Inoltre è particolarmente soddisfatta del fatto di poter offrire ai propri clienti la possibilità di usare un software sulle schede fornite senza dover pagare una licenza per ogni pezzo.

Bisogna dire che EuroTech inizialmente rimane perplessa sul fatto di dover "regalare" il codice sorgente, perplessità che sfumano quando il suo presidente capisce i vantaggi di costo e di sviluppo derivanti dall'utilizzo di software libero e che i vincoli imposti dalla Licenza GPL sono a garanzia del mantenimento delle sue caratteristiche.

ETLinux è un sistema Linux completo progettato per funzionare su piccoli computer industriali e quindi per essere piccolo, modulare e flessibile. Rientra nella categoria dei sistemi operativi *embedded*, termine che identifica in generale i software destinati a piccoli apparati - come i telefoni cellulari, ad esempio - schede e micro PC industriali. I sistemi embedded negli ultimi anni sono stati protagonisti di una grossa crescita, grazie all'aumentata diffusione e ai livelli tecnologici raggiunti.

Il fatto che un sistema operativo embedded possa essere open source e quindi facilmente reperibile nelle sue versioni aggiornate, comporta certamente un valore aggiunto per chi sviluppa tali tecnologie, nel senso di non doversi affidare interamente nelle mani di un'altra azienda fornitrice di software proprietario dovendone condividere anche gli errori o le mancanze senza poter intervenire sul codice. Si pensi alle perdite patite da aziende dipendenti, per i loro apparati, da software proprietario afflitto dal baco dell'anno 2000 e prodotto magari da aziende già fallite, casi tutt'altro che rari.

L'enorme quantità di codice in forma di sorgente presente in rete (programmi, librerie, linguaggi, compilatori, debugger)[14] aumenta la produttività degli sviluppatori open source in tutti i settori, anche nell'embedded.

[14]Vedi Glossario

Oltre a tutto ciò la nota stabilità di Linux comporta un altro punto a favore fondamentale: si pensi alle conseguenze che potrebbe avere un *blue screen*[15] su un'attrezzatura elettromedicale. ETLinux è un sistema operativo completo di dimensioni ridotte e senza fronzoli per poter funzionare su piccoli apparati, con tutte le caratteristiche tipiche di un sistema Unix: multitasking[16] e multiutente, memoria condivisa e gestione della stessa separatamente ed in sicurezza per ogni utente, comunicazione inter-processo, networking, gestione uniforme delle periferiche, file system[17] multipli, ecc.

Caratteristiche tecniche di ETLinux

ETLinux utilizza un kernel linux modificato per ridurre al minimo la richiesta di risorse, adatto quindi alle schede ed ai PC industriali. Utilizza una versione molto ridotta dei tool Unix essenziali ed è dotato di tutte le funzionalità di rete. La versione base, la più ridotta, funziona su processori *386sx* con due soli mebabyte di memoria RAM e comprende un webserver ed un mailserver ridotto.

Sono inoltre in fase di sviluppo una micro-interfaccia grafica ed un web browser, un supporto CORBA che permette l'integrazione di parti di applicazioni anche con applicativi che funzionano su altre macchine collegate in rete, un migliore supporto per il real time (RTLinux e RTAI), un ambiente di sviluppo e un'implementazione Java.

Diffusione

Formalmente lo sviluppo e la distribuzione di ETLinux ha seguito i canoni tipici del Progetto Open Source visti precedentemente, tuttavia alcuni fattori contingenti ne hanno frenato, almeno inizialmente, la diffusione.

[15]Comunemente chiamato *Blue Screen Of Death* è la schermata blu che MS Windows mostra in caso di errori gravi che ne impediscono il buon funzionamento. Tale schermata contiene informazioni criptiche generalmente inutili per l'utilizzatore comune, e blocca il funzionamento del computer

[16]Vedi Glossario

[17]Vedi Glossario

Innanzitutto il fatto che Prosa non avesse fatto alcuna operazione di marketing e nessuna pubblicità, a parte qualche articolo apparso nelle riviste specializzate, non aiutò certo. D'altronde però questa era stata la politica di Prosa fin dall'inizio, un po' perché il lavoro non era mai mancato e un po' perché le risorse societarie in questa prima fase erano abbastanza limitate per il fatto che si era preferito partire senza grossi investimenti iniziali per ridurre i rischi.

Un altra limitazione alla diffusione di ETLinux fu l'effettiva reperibilità del codice dall'esterno, dovuta al fatto che al tempo Prosa usufruiva di collegamento ad Internet garantito solamente da una connessione molto lenta. Questo effettivamente rendeva abbastanza difficoltoso e lungo per chiunque cercare di scaricarsi il codice del programma che per quanto piccolo misurava comunque qualche decina di megabyte tra codice sorgente, documentazione e kit di sviluppo.

Inoltre il fatto che avesse, per forza di cose, *pochi fronzoli* non suscitava certo l'interesse di molti se non di quegli sviluppatori più interessati al mondo embedded.

Nonostante tutto ciò ETLinux viene indicato, in un sondaggio [Poll] promosso tra gli operatori del settore dalla rivista on-line LinuxDevices[18] e pubblicato nel febbraio 2000, come seconda migliore scelta per future applicazioni.

Una volta attivato un mirror presso il sito ftp dell'università di Napoli[19] i contatti si moltiplicano e con essi i segnali di interesse ed i feedback degli utilizzatori e degli sviluppatori. Vengono inoltre richiesti a Prosa diversi contributi ed articoli per la stampa (cartacea ed on-line).

Questo relativo successo di ETLinux porta Prosa ad essere abbastanza riconosciuta a livello internazionale nel settore. È questo è uno dei motivi che favorisce l'interessamento di Linuxcare inc., company americana fornitrice di servizi open source, all'acquisizione di Prosa[20].

[18] `http://linuxdevices.com` rivista di riferimento per Linux-Embedded

[19] `ftp://ftp.unina.it`

[20] Vedi sezione 4.4.6 pg. 117

Obiettivi mancati

Secondo i soci e gli sviluppatori intervistati, lo sviluppo di ETLinux non ha centrato alcuni degli obiettivi previsti. Ciò è stato dovuto, secondo tutti loro, al fatto che Linuxcare non ha permesso, come invece aveva promesso, di seguire nei modi dovuti il progetto. E dire che invece ETLinux era stato uno dei motivi per cui la company amenicana aveva individuato Prosa come soggetto ideale per garantire lo "sbarco" sul mercato italiano ed europeo.

Tra gli obiettivi mancati di ETLinux i più significativi sono sicuramente:

- Assenza di qualsiasi politica di marketing sul prodotto. Tra l'altro non si è mai sviluppato un prodotto *vendibile in scatola*.
- ETLinux è un prodotto ancora poco *user friendly*. Manca uno strumento semplice e magari grafico per la personalizzazione.
- Manca il supporto per alcune tecnologie più o meno secondarie molto di moda oggi.
- Non si è fatto praticamente nulla per favorire la nascita di una comunità vera e propria attorno ad ETLinux.

4.4.6 Prosa s.r.l diventa Linuxcare Italia S.p.A

Come accennato in percedenza, verso la fine del 1999 una grossa company americana, Linuxcare inc., contattò Prosa nel quadro della ricerca di adeguati partner per l'apertura di alcune filiali europee.

Linuxcare è una società fornitrice di consulenza professionale, supporto tecnico, sviluppo e didattica nell'ambito open source, nata nel 1998 a San Francisco nella famosa Silicon Valley con l'appoggio finanziario di alcune società di venture capital. Nel periodo dell'acquisizione di Prosa stava perseguendo una politica di espansione, aprendo filiali in Canada, Europa (Germania, Inghilterra, Svezia, Olanda ed Italia) ed Asia, in vista della sua prossima quotazione in borsa nell'indice americano dei titoli tecnologici, il *Nasdaq*.

La proposta per Prosa era di diventare Linuxcare Italia s.p.a.. La piccola società italiana chiedeva però una serie di garanzie che potessero conservare nella sostanza i propri principï ispiratori. Linuxcare vinse le perplessità dei soci di Prosa garantendo loro innanzitutto la conservazione ed il "congelamento" del marchio, la garanzia di poter sviluppare come in precedenza solamente software libero e, soprattutto, la garanzia di poter mantenere e sviluppare, dopo un breve periodo di assestamento, il loro prodotto di punta: ETLinux. I soci avrebbero inoltre percepito un adeguato compenso per la cessione delle loro quote di Prosa ed avrebbero fatto parte del consiglio di amministrazione della neonata società, in particolare Davide Barbieri ne sarebbe stato il presidente.

Linuxcare Italia s.p.a nacque dunque il 1 Gennaio 2000. L'organico di Prosa venne subito integrato con varie figure professionali, tra cui una segretaria, un responsabile commerciale ed uno amministrativo, raggiungendo un organico di una ventina di persone, compresi nuovi sviluppatori e tecnici. Due dei soci decisero di restare fuori dall'operazione. Uno, Alessandro Rubini, perché impegnato nel suo lavoro di programmatore *free-lance*, l'altro, quello stabilitosi in Finlandia, per l'effettiva impossibilità di conciliare i propri impegni lavorativi con la nuova realtà.

Erano previste diverse business unit per quanto riguarda i servizi offerti: supporto tecnico, consulenza professionale, didattica e sviluppo. Tuttavia per alcune di esse si verificò subito un certo ritardo nella partenza effettiva per motivi contingenti legati soprattutto al momento particolare di tensione dei mercati finanziari. I crolli dell'indice Nasdaq del Marzo 2000, infatti, fecero cambiare repentinamente i piani di Linuxcare inc., che decise di posticipare la quotazione, prevista inizialmente proprio in quel periodo. La necessità della casa madre di reperire dai propri finanziatori nuove risorse creò un certo clima di empasse, di cui risentirono, ovviamente, anche le sedi estere.

La sede italiana mantenne al suo interno una divisione chiamata *Prosa Labs* proprio perché, forte di un notevole numero di programmatori e tecnici nel suo organico, risultava il punto di riferimento tecnico per le altre sedi,

soprattutto quelle europee. In questa prolungata fase pre-quotazione, in cui la casa madre appariva soprattutto concentrata nell'acquisire nuove sedi per aumentare le proprie dimensioni piuttosto che svilupparne la produttività, Linuxcare Italia si trovò in una situazione di sottoutilizzo delle proprie risorse, soprattutto quelle adibite alla corsistica ed in parte allo sviluppo ed alle consulenze professionali.

La situazione, aggravata da un certo malumore della dirigenza italiana per effettive difficoltà di comunicazione con la casa madre ed il prolungato accantonamento di ETLinux, sfociò in un improvviso annuncio di *chiusura delle operazioni europee ed asiatiche* voluta dalla cordata di finanziatori senza alcun preavviso al management italiano e a parte di quello statunitense.

Linuxcare Italia S.p.a chiude ufficialmente il 31 Dicembre 2000.

4.4.7 La rinascita di Prosa - Ascensit

Nel periodo intercorso tra l'annuncio e la chiusura effettiva, tuttavia, non sono mancate le offerte da parte di altre aziende italiane e straniere del settore o di soggetti interessati alla società italiana.

Accantonata l'idea di finanziatori esterni per ragioni di indipendenza sulle scelte di sofware libero, nonché quella di far rivivere l'entità societaria per gli alti costi connessi, la dirigenza si è orientata alla valutazione delle proposte di altre società. Una volta scremate, le occasioni più vantaggiose rimasero quelle proposte da due società: Innominate A.G., tedesca e Ascensit s.r.l., italiana controllata da EuroTech partner nel progetto ETLinux, entrambe aziende open source.

Le due società erano interessate soprattutto al gruppo consistente di sviluppatori ed al fatto di avere la possibilità di rilevare una struttura già costituita ed affiatata dal punto di vista delle risorse umane.

Purtroppo però, il fatto che non fosse in ballo l'acquisizione di una società, ma la disponibilità di un gruppo di professionalità, ha messo la cosa su un piano di singole scelte personali, spaccando il gruppo quasi equamente tra le due offerte. La situazione rimase tale fino alla fine delle trattative.

La situazione odierna vede la costituzione, avvenuta ai primi di Febbraio a Padova, di una filiale di Ascensit e probabilmente a Marzo della fiiale italiana della tedesca Innominate AG.

I soci fondatori di Prosa rimasti, Paolo Didonè, Alessandro Gianelle e Davide Barbieri hanno deciso di aggregarsi ad Ascensit. Il progetto a breve termine prevede la ricostituzione di Prosa per avviare una partnership con Ascensit in cui Prosa rappresenti il settore *ricerca e sviluppo*.

4.5 MixadLive s.r.l.

4.5.1 Breve storia di Mixadlive s.r.l.

Mixadlive s.r.l. nasce a Torino nell'Agosto del 1999 e avrebbe dovuto diventare, secondo i suoi cinque fondatori, una piccola azienda di consulenza e *web design*. Ma due dei soci (Michele e Paolo Comitini) sono tecnici di prima qualità ed entrambi sono convinti della bontà della filosofia open source, decidendo di applicarla nel loro lavoro. Gli altri tre soci provengono da Mixad s.r.l., agenzia di pubblicità della quale Michele Comitini era consulente.

Mixadlive, grazie anche a Mixad, prende subito due grossi lavori: *punto.it*, un portale internet tra i più visitati tra i portali italiani e *tradinglab.com*, famoso servizio di trading on line di Unicredito. In questa fase iniziale Mixadlive - grazie alla relazione con Mixad - spinge, a differenza di quanto abbiamo visto per Prosa, sulla parte pubblicitaria, considerandola molto importante. A Dicembre, sempre del 1999, dopo un po' di consulenze entra in società anche Federico Di Gregorio[21] in parte perché è il miglior programmatore del gruppo, in parte perché, essendo lui sviluppatore Debian, gli altri soci ritengono di poter tenere forti contatti con la comunità Linux.

Mano a mano, più il tempo passa più Mixadlive si stacca da Mixad, diventando la parte grafica e pubblicitaria sempre meno importante. Ora eseguono consulenze e lavori di analisi e programmazione di alto livello lasciando

[21] Fonte principale per la raccolta di questi dati su MixadLive

progressivamente il campo web per indirizzarsi più verso il *Virtual Private Network (VPN)*, software da usare in ambito internet per la costruzione di reti virtuali protette da sistemi di criptaggio.

Le dimensioni attuali di Mixadlive sono di sei soci, dei quali tre lavorano a tempo pieno, e sette dipendenti. Il fatturato per il 2000 è stato di 1.7 miliardi e si prevede una ulteriore crescita per il 2001.

4.5.2 Aspetti organizzativi

Riguardo a Prosa, Mixadlive presenta delle differenze riguardo ad alcuni aspetti dell'organizzazione, dovute soprattutto al fatto che in quest'ultima tutto il personale - soci e collaboratori - almeno nella prima fase proveniva dalla comunità open source. Per quanto riguarda Mixadlive, invece, tra i dipendenti programmatori assunti solo alcuni provengono dal mondo del software libero, gli altri provengono da altre esperienze e sono "costretti" a produrre software libero per necessità aziendale.

La società si appoggia tuttora a Mixad per il settore commerciale, mentre per le mansioni di carattere amministrativo non ne ha in parte più bisogno in quanto ha di recente (Gennaio 2001) assunto in pianta stabile una segretaria.

Mixadlive mostra comunque una separazione delle funzioni tecniche da quelle di amministrazione e commerciali, anche per quel che riguarda le competenze del personale - segretaria - dove magari in Prosa questa divisione, che pure esisteva anche all'inizio, era però svolta per forza di cose da soci con competenze tecniche che venivano così in qualche modo sprecate.

L'area tecnica consta di tre reparti definiti almeno sulla carta, ognuno con un responsabile: reparto tecnico (sistemistica, hardware); reparto di sviluppo del software (tutta la programmazione di routine) e reparto di ricerca (analisi e nuove soluzioni). Anche qui, come abbiamo già visto in Prosa, i dipendenti passano da un reparto all'altro a seconda delle esigenze. In genere quando arriva un lavoro nuovo molti fanno ricerca, poi si passa alla programmazione vera e propria e tutti i programmatori sviluppano software.

4.5.3 Attività svolte e mercato di riferimento

L'attività svolta da Mixadlive si concentra essenzialmente sull'analisi delle esigenze del cliente e conseguente produzione degli strumenti informatici per la risoluzione dei problemi riscontrati. In sostanza quindi spazia dall'integrazione di piattaforme, alla gestione della documentazione e del personale, ai sistemi embedded.

Inoltre anche Mixadlive, come Prosa, sviluppa dei piccoli Progetti Open Source riguardanti soprattutto driver per hardware e periferiche.

In Piemonte sostanzialmente Mixadlive non ha concorrenti nel settore dell'open source, tanto che in uno dei colloqui avuti con Federico Di Gregorio per la raccolta di queste informazioni, egli ironizzava su questo affermando: *i clienti ci dicono "cercavamo qualcuno bravo con Linux ed abbiamo trovato solo voi."*

4.5.4 Metodi di produzione ed organizzazione del lavoro

Mixadlive non ha, in parte come Prosa, produzione di software in senso stretto. In sostanza il prodotto che vendono è la loro capacità di risolvere i problemi, possibilmente meglio e prima di altri, usando software libero. Cercano anche, in accordo con il cliente, di restituire sotto Licenza GPL alla comunità qualsiasi cosa prodotta, considerando questo punto fondamentale.

Proprio per questa precisa scelta Mixadlive tende a rifiutare commesse che richiedano di sviluppare software proprietario o di implementare direttamente sistemi con esso. Anche se tengono a precisare che questa scelta vuole essere sì radicale e coerente, ma non ottusa; se per esempio un cliente volesse un database, Mixadlive potrebbe offrirgli l'ottimo *Postgress*, un database open source. Ma se il cliente volesse invece installare il famoso Oracle, programma proprietario, Mixadlive rifiuterebbe la commessa tuttavia, se lo stesso cliente avesse già installato Oracle, Mixadlive potrebbe comunque offrirgli dei servizi sul database utilizzando Postgress in quanto compatibile con Oracle.

Modalità di acquisizione dei clienti

Vale il discorso fatto sopra per Prosa. Anche Mixadlive, essendo tra le poche aziende italiane che propongono software libero ed open source, risulta essere uno dei pochi riferimenti per chi, soprattutto in Piemonte dove è l'unica, vuole un prodotto di questo genere. Ancora una volta il veicolo principale per generare i contatti commerciali è la comunità attraverso la sua fitta rete di conoscenze, oltre all'appoggio in termini commerciali di Mixad.

Valutazione dei bisogni del cliente e definizione dei preventivi

In Mixadlive si può osservare una procedura di definizione del preventivo abbastanza strutturata che prevede due fasi a pagamento: *pre-analisi* ed *analisi* .

A seconda delle necessità del cliente, durante la fase definita di pre-analisi che il cliente paga anche se è una cifra simbolica, sono presenti i responsabili dei tre reparti: tecnico, sviluppo e ricerca. Questa fase, che dura in genere una o due giornate lavorative, porta a stilare un preventivo dettagliato con costi, tempistica e milestone con date precise.

La fase successiva, quella dell'analisi, sempre a pagamento, serve a fornire una consulenza specifica che porta a vagliare la necessità del cliente in modo più approfondito definendo con maggior dettaglio la situazione. La fase finale dell'implementazione di una soluzione ad hoc viene eseguita a discrezione del cliente, seguendo le indicazioni scaturite durante l'analisi.

Allocazione delle risorse umane e definizione delle responsabilità

Ad ogni progetto viene assegnato un capo progetto che assume *tutta* la responsabilità della sua esecuzione. Viene tenuta una *riunione-progetti* una volta la settimana in cui il capo progetto fa il punto della situazione in base al soddisfacimento delle milestone o al cambiamento delle condizioni o delle necessità del cliente e chiede le risorse di cui ha bisogno.

Produzione, controllo avanzamento, qualità

Nella fase di realizzazione della commessa, come detto, si seguono dei precisi obiettivi temporali: le milestone. Sono queste scadenze che offrono un mezzo di controllo degli stati di avanzamento del progetto.

Tuttavia un prodotto software su misura necessita spesso di alcuni aggiustamenti in base anche alle possibili mutate necessità del cliente. Ecco perché in Mixadlive c'è la regola di "spezzettare" il processo produttivo di ogni particolare prodotto in modo da poter considerare ogni troncone a sé stante. Ciò dà la possibilità di poter riprogrammare ogni scadenza in caso di variazioni nelle condizioni in modo di poter ottimizzare anche il lavoro del personale. Se per esempio un cliente ha bisogno di una variazione riguardo le scadenze del lavoro che ha commissionato, il capo progetto può riprogrammare la parte rimanente della commessa e ciò si riperquoterà sull'allocazione delle risorse umane. Questo permetterà di conoscere di settimana in settimana l'effettiva consistenza di risorse a disposizione di altri lavori in corso di svolgimento o in fase di acquisizione.

Livello di formalizzazione del progetto e documentazione

L'analisi produce un documento tecnico completo e questo è il primo prodotto che viene venduto al cliente. L'implementazione che segue, una volta eseguita l'analisi, produce un'ulteriore documentazione relativa al software prodotto. È da notare che l'analisi può anche produrre dei prototipi là dove i problemi lo richiedano.

Livello di modularizzazione, standardizzazione delle strutture e dei programmi.

Del tutto analogo a Prosa, d'altra parte l'attività di sviluppo dei prodotti software nelle due società è molto simile.

Sistema di incentivi

Il sistema di incentivi di Mixadlive prevede dei premi in denaro alla fine di ogni progetto portato a termine che dipenderanno di volta in volta da valutazioni contingenti.

4.6 Ximian inc. - Progetto Evolution

4.6.1 Breve storia di Ximian inc.

Ximian inc.[22] è stata fondata, con il nome di HelixCode inc, a Cambridge, Massachusets nell'Ottobre del 1999 da Miguel de Icaza[23] - messicano, fondatore e leader della Gnome[24] Foundation e membro dell'esecutivo della Free Software Foundation. Sviluppatore di Linux per Sparc, del file system[25] Midnight Commander, del foglio elettronico Gnumeric e ideatore di Bonobo[26] e Nat Friedman[27] - uno degli ispiratori della Gnome Foundation e sviluppatore di Bonobo.

Ximian si basa sul finanziamento da parte di alcune società di venture capital. Recentemente - 17 Gennaio 2001 - la società ha ricevuto un secondo giro di finanziamenti per 15 milioni di dollari. Alcune tra le più importanti aziende mondiali di software ed hardware, come Sun Microsystems, Compaq, IBM ed Hewlett Packard sono tra i membri del consiglio direttivo della Gnome Foundation ed in particolare Hewlett Packard ha appena siglato un accordo con Ximian per la distribuzione di *Ximian Gnome* assieme alle macchine HP-UX e per il supporto necessario. Sono inoltre già stati chiusi accordi con altri importanti Linux vendor.

[22] `http://www.ximian.com`
[23] Chief Technical Officer (CTO)
[24] Vedi sezione 3.2.5, pg. 68
[25] Vedi Glossario
[26] Vedi Glossario
[27] Chief Executive Officer (CEO)

4.6.2 Prodotti e mercato di riferimento

Entrambi i co-fondatori di Ximian provengono dal mondo open source, in particolare Miguel de Icaza è uno dei più attivi propugnatori dei concetti del software libero. Nelle loro intenzioni, con la fondazione della società vi è la precisa volontà di sviluppare, sfruttando l'interesse del mercato e degli investitori di professione, un business sul software libero. Un po' quello che più in piccolo hanno voluto fare sia MixadLive che, maggiormente, Prosa.

Obiettivo dichiarato di Ximian era ed è quello di far diventare Gnome il migliore e più diffuso ambiente applicativo, sfruttando Internet come media principale. Attraverso lo sviluppo di Gnome e di varie applicazioni collegate, il business su cui puntano è quello del supporto dei servizi via web connessi a questi prodotti Open Source. Per perseguire questo obiettivo attualmente la società è impegnata in un lavoro di sviluppo intenso delle applicazioni che formeranno la base dell'attività futura:

- *Ximian Gnome.* Una distribuzione testata e personalizzata di tutto il software del Progetto Gnome, pronta per una facile installazione su sistemi GNU/Linux o Unix. In questo momento si stima che Ximian Gnome conti più di 500.000 utenti in tutto il mondo[28].

- *Gnumeric.* Foglio elettronico compatibile con MS Excel.

- *Evolution.* Pacchetto applicativo che serve a gestire la propria posta elettronica, l'agenda ed i contatti. È stato costruito interamente con soluzioni all'avanguardia per le moderne esigenze comunicative. Somiglia nelle funzioni a MS Outlook Express, ma con l'aggiunta di una serie di strumenti in più. È alla sua versione beta 0.9, disponibile e quasi del tutto stabile[29] e a breve ne è prevista la prima versione non-beta, la 1.0.

[28] All'inizio dell'anno era stato scaricato da più di 450.000 utenti e ne erano state spedite più di 10.000 copie su CD-ROM

[29] Io la uso personalmente da mesi e non ha mai avuto problemi

- *Ximian setup tool*. Interfaccia grafica per compiere in modo semplice ed intuitivo il setup del sistema, gestire il file system ed impostare le configurazioni, tutte operazioni che normalmente, nei sistemi Unix, richiedono di intervenire manualmente sui file di configurazione. È chiaramente pensato per gli utilizzatori base e per garantire una interfaccia comune per più piattaforme.

- *Red Carpet*. È un *updater*, un aggiornatore automatico di software. Permette di installare le nuove versioni dei software che compongono Gnome rimuovendo nel contempo le vecchie. Dà la possibilità di "sfogliare" il software disponibile in rete e scegliere quello che interessa, risolvendo problemi di conflitti e dipendenze.

4.6.3 Aspetti organizzativi

Partita con dodici sviluppatori, Ximian ora ne conta una sessantina. Il lavoro è organizzato in team coordinati da responsabili, uno per ogni progetto seguito dalla società. Ogni team è costituito da più sottoteam, ognuno rappresentato da un leader. In pratica ogni team segue il proprio progetto secondo il modello del tipico Progetto Open Source, ufficializzando in qualche modo i ruoli visti nel precedente capitolo[30] e affiancando questa struttura "organizzata" ai collaboratori della comunità che vogliano contribuire volontariamente.

Essendo il business di Ximian rivolto maggiormente ai servizi ed al supporto per i soggetti commerciali, il singolo utente che voglia contribuire viene ripagato dei suoi sforzi con il prodotto risultante sia dal proprio impegno, sia dal lavoro di un team dedicato pagato dai soggetti di cui sopra.

4.6.4 Evolution come Progetto Open Source

Per meglio comprendere la struttura di un team e come viene organizzato il lavoro di Ximian è stato analizzato il team che sviluppa Evolution attraverso

[30] Vedi paragrafo 3.3.3, pg. 79

delle interviste al suo responsabile Ettore Perazzoli, programmatore milanese e membro del Progetto Gnome.

Evolution è stato pensato e strutturato dalla dirigenza di Ximian per essere in tutti i sensi un Progetto Open Source con una struttura, anche se organizzata con criteri aziendali, che riproponesse in tutto quella tradizionale già vista nel terzo capitolo di questo lavoro.

Project manager

Il team di Evolution è coordinato da un Project Manager, Ettore Perazzoli appunto, che si occupa della direzione tecnica, assegna i compiti ai vari programmatori, definisce gli obiettivi e le *deadline*. Allo stesso tempo programma attivamente ed è il Mantainer della shell[31] di Evolution.

Miguel de Icaza, il CTO, si occupa di definire la direzione tecnica di Ximian più in generale. Perazzoli, come coordinatore di Evolution si occupa di eseguire tali indicazioni nella pratica, per cui tutte le decisioni tecniche spettano a lui anche se il CTO, in linea di principio ha l'ultima parola.

Sottoteam e Mantainer

Il team è organizzato in sottoteam corrispondenti ai vari strumenti di cui si compone Evolution: calendario, indirizzario, mailer. Ogni sottoteam ha un suo leader/mantainer che assegna i compiti ai suoi sviluppatori. Come strumento per il project management viene utilizzato *Bugzilla*, un programma abbastanza versatile anche se non particolarmente sofisticato che funge da database per i bug. In Bugzilla in pratica viene tenuta la lista di tutti i bug e delle cose da fare e per ognuno di questi bug il corrispondente maintainer da una valutazione di massima del tempo richiesto e assegna il task a qualcuno del suo team. I bug emergono dal lavoro stesso degli sviluppatori di Ximian ma vengono anche in larga parte segnalati dagli utenti di Evolution attraverso un apposito Bug Tracking System.

[31] Vedi Glossario

Sviluppatori

La comunità di utenti di Evolution è già abbastanza estesa e vivace. Data la giovane età del programma ed un certo iniziale scetticismo di parte della comunità per la scelta commerciale, i contributi esterni in codice non hanno ancora raggiunto un'entità ragguardevole. Tuttavia sono molti i contributi in forma di bug report, suggerimenti, lamentele, piccole patch e traduzioni; sintomo di interesse per il prodotto e auspicio di futuri maggiori apporti. È interessante che i bug possono essere segnalati ad inseriti nel database, come detto, anche direttamente da singoli utenti esterni, creando un flusso di informazione immediato dall'utente allo sviluppatore.

Il team Ximian di Evolution è comunque composto attualmente da quattordici programmatori che nella maggior parte lavorano da remoto essendo sparsi per il globo (compreso Perazzoli). Ci sono inoltre una persona che si occupa del disegno dell'interfaccia utente e due artisti che disegnano le icone, gli splash screen ecc. Questi ultimi dipendono da Perazzoli per il lavoro su Evolution, ma sono condivisi con gli altri team (Ximian Setup Tools, Ximian GNOME e Red Carpet). Infine il team di Evolution ha una persona a tempo pieno che lavora alla documentazione per l'utente e che viene aiutato da un volontario che non viene pagato da Ximian.

Processi decisionali e procedure per l'assegnazione dei compiti

Come detto l'ultima parola su praticamente tutto in Evolution spetta al suo Project Manager. In generale tuttavia è prassi riconosciuta in Ximian arrivare ad un accordo su tutte le decisioni fondamentali. Fino ad ora non è mai capitato a Perazzoli o ad altri Project Manager di imporre qualcosa su cui il resto del team (o il CTO) non fosse d'accordo. Spesso per questo motivo ci vuole un po' di tempo per raggiungre una decisione condivisa, tuttavia è convinzione comune, dovuta alla provenienza di tutti gli sviluppatori dalla comunità open source, che quando l'obiettivo da raggiungere è condiviso da tutti si lavora meglio.

Il Project Manager redige un budget semestrale che viene sottoposto al-

l'approvazione del CEO e del Chief Financial Officer (CFO). Nel budget si stabiliscono gli obiettivi da raggiungere e di conseguenza quante persone si dovrebbero eventualmente assumere o prendere per collaborazione e per che compiti. Per casi straordinari approvati è possibile uscire dal budget, ma fino ad ora non è ancora successo.

Proprio per la provenienza delle persone con incarichi direttivi dal mondo del software libero ed open source non è molto facile per Ximian trovare le persone "giuste". Fino ad ora tutte le assunzioni che hanno funzionato, secondo Perazzoli, sono state quelle di persone che facevano già parte della comunità Gnome, e che erano quindi ben conosciute per quello che avevano fatto come volontari. *"Bisogna essere molto motivati per fare bene questo lavoro..."* [Perazzoli]

Produzione, controllo dell'avanzamento e qualità

Come detto, dall'assegnazione dei compiti scaturisce una definizione delle scadenze, con Bugzilla infatti si possono definire delle milestone, e specificare per ogni bug/task la target milestone. Tuttavia, non smentendo una delle caratteristiche dei Progetti Open Source, in Evolution ma anche in Ximian in genere c'è una cronica tendenza a mancarle, anche se di poco. Uno dei problemi con Evolution è stato il dover dipendere da librerie instabili (come Bonobo) che venivano sviluppate in parallelo e hanno causato delle volte dei rallentamenti notevoli.

Il controllo dell'avanzamento del lavoro è fatto ancora una volta con Bugzilla. In questo modo si riduce al minimo la necessità di dover parlare faccia a faccia automatizzando il più possibile la comunicazione perché sarebbe complicato fare dei meeting quando i programmatori sono sparsi per il globo.

La documentazione

Uno sviluppatore, Aaron Weber, sta scrivendo il manuale per l'utente di Evolution. Segue lo sviluppo di Evolution e lo usa come client di posta pri-

mario, aggiornando il manuale mano mano che nuove caratteristiche vengono implementate.

Sistema di incentivi

Il Project Manager ha la facoltà di assegnare premi per specifici obiettivi anche se fino ad ora, almeno per Evolution, non ha avuto bisogno di usare questo strumento. Per il resto Ximian funziona come una compagnia standard americana: ogni sei mesi viene svolta una *performance review* che consiste nel parlare singolarmente con i membri del proprio team, definendo un aumento dello stipendio basato sul lavoro dei sei mesi precedenti e definendo i goal per i successivi sei mesi.

Capitolo 5

Conclusioni

5.1 Critiche al modello di sviluppo open source

Nella prima parte di questo lavoro sono state illustrate le origini storiche e le caratteristiche di un fenomeno, quello della condivisione del codice nel software, che ha portato negli anni alla creazione di una variegata comunità di individui, fondata su una serie di precise relazioni sociali e strutturata su regole, comportamenti e sistemi di incentivi impliciti.

La componente motivazionale, legata alle forti istanze morali e quella narcisistica, fondata su un meccanismo di riconoscimento che si basa sulla reputazione, convivono in una "micro società" che appare governata da quella che i sociologi chiamano *cultura del dono*, basata su una risorsa non scarsa che in questo caso è il software, inteso anche come conoscenza.

Questa comunità ha prodotto, negli anni, i due movimenti che si dividono la scena del software non proprietario: *Free Software Foundation (FSF)* nata dal *Progetto GNU* ed *Open Source Initiative (OSI)*. Come si è visto lo strumento principale usato dai sostenitori del software libero e open source per proteggere il codice prodotto secondo questi principi è la Licenza che, attraverso l'istituto del copyright, garantisce la volontà dell'autore di mantenere, in questo caso, la libertà e riproducibilità del software da lui prodotto.

Tuttavia il prodotto più originale del movimento del software libero ed open source, quello che più interessa in questa analisi, è ciò che viene in genere definito *Progetto Open Source*, cioè il metodo aperto di sviluppo di questo tipo di software, di cui Linux è l'esempio più calzante e citato. Linux ha per primo proposto un modello funzionante di sviluppo caratterizzato dalla partecipazione diffusa, favorita dalla Rete, di utenti/sviluppatori alla costruzione ed al miglioramento del programma.

Se ogni sistema operativo o programma di una certa dimensione creato fino ad allora era stato concepito e sviluppato da singoli o gruppi relativamente piccoli di persone coordinati tra loro che lo rifinivano e mettevano a punto prima di rilasciarne una versione sufficientemente stabile, Linux invece ha sempre potuto contare sull'apporto di un gran numero di volontari coordinati attraverso Internet. Il continuo rilascio di versioni aggiornate, comprese le iniziali ed instabili versioni beta, ha stimolato il lavoro incessante di verifica e correzione da parte della comunità di sviluppatori e contributori. Ciò ha garantito, secondo molti e a dispetto delle previsioni più diffuse, una qualità impensabile per chiunque fosse abituato a lavorare in qualsiasi altra maniera.

Eric S. Raymond ha individuato, nel suo *The Cathedral and the Bazaar* [Raymond1] due teorie corrispondenti ad altrettanti fondamentali stili di sviluppo di software: quello *a Cattedrale*, proprio della maggior parte del mondo commerciale e caratterizzato, come detto, dal lavoro di un team relativamente ristretto e chiuso con poche occasioni di interazione con l'utenza finale, se non dopo il momento del primo rilascio definitivo; e quello *a Bazaar*, consacrato da Linux e tipico del mondo open source, dove ad un team abbastanza aperto di sviluppatori si affianca il contributo a volte di una moltitudine di utilizzatori che contribuiscono in modo critico alla crescita del programma, grazie alla sua precoce disponibilità e fruibilità sin dalle versioni più sperimentali.

Tuttavia non sono mancate, soprattutto negli ultimi tempi, opinioni discordanti su questo modello di sviluppo, sia sulla natura delle sue caratteristiche che sulla sua reale qualità. In questa sede, per una migliore com-

prensione del fenomeno, verranno illustrate lo opinioni di chi dall'interno e dall'esterno del movimento open source ha mosso delle critiche o rilevato delle incongruenze in un modello che, pur positivo nella sua generalità per la sua forza innovativa e propulsiva, mostra ancora molte sfaccettature e alcuni lati deboli.

5.1.1 Possibili punti deboli

Già Nikolai Bezroukov, in una celebre coppia di paper della fine del 1999 [Bezroukov1, 2], critica quello che lui chiama *raymondismo*, mettendo in luce le crepe e le contraddizioni di una descrizione del fenomeno open source, quella di Raymond appunto, un po' troppo incline alla celebrazione.

Bezroukov, professore di informatica alla Fairleigh Dickinson University, New Jersey, nonchè membro attivo della comunità open source[1] osserva, soprattutto grazie alla sua attività accademica, che si sta diffondendo una visione eccessivamente ottimistica e scarsamente realistica della pur positiva realtà open source.

Le ragioni di questa distorsione della realtà sono diverse secondo Bezroukov. Concetti come il software libero ed open source si stanno molto diffondendo, diventando abbastanza di moda e sollevando l'interesse della stampa, soprattutto di quella specializzata. Le notizie spesso mettono in risalto solamente i traguardi raggiunti ed i Progetti di successo, mancando di sottolineare i fallimenti, le difficoltà e i progetti abortiti. Tutto ciò genera l'impressione generalizzata che il modello di sviluppo open source sia una panacea e che possa essere la soluzione migliore in ogni caso.

Il modello di sviluppo open source è identificato con Linux e descritto come un fenomeno rivoluzionario da gran parte della letteratura, in particolare da Eric S. Raymond i cui scritti sono spesso presi come riferimento. Bezroukov cerca di ridimensionare i toni analizzando in modo critico le affer-

[1] È webmaster di `www.softpanorama.org` - Open Source Software University, un sito volontario collegato al programma SDNP delle Nazioni Unite per garantire connettività Internet e adeguata distribuzione di Linux alle nazioni in via di sviluppo

mazioni che Raymond fa nel suo *The Cathedral and the Bazaar*[Raymond1] per metterne in luce le contraddizioni.

Il punto di partenza di Bezroukov, come sottolineato nel primo dei due paper [Bezroukov1], è il fatto di considerare la complessiva comunità open source semplicemente come un'altro tipo di comunità scientifica. Lo sviluppo di Linux è visto dunque non come un nuovo modello rivoluzionario, ma come la logica continuazione del Progetto GNU, che all'inizio era in qualche modo legato al MIT[2]. Bezroukov è convinto che questo collegamento accademico sia stato cruciale per il successo del Progetto GNU, come lo fu quello con l'Università di Helsinky per i primi passi di Linux.

Nella sua critica Bezroukov prende spunto da alcune tra le affermazioni più importanti dello scritto di Raymond, che riassumono sostanzialmente quanto detto nei primi capitoli di questo lavoro:

- **La *Legge di Brooks* non si può applicare al modello di sviluppo distribuito basato su Internet**. In realtà Frederick Brooks nel suo famoso *The Mythical Man-month* affermava che la complessità ed i costi di comunicazione di un progetto aumentano in ragione del quadrato del numero degli sviluppatori, mentre i risultati aumentano solo linearmente. L'interpretazione che ne dà Raymond, cioè che se si aggiungono sviluppatori ad un progetto avanzato lo si fa solo ritardare, dà in realtà dell'affermazione originale una lettura parziale che chiaramente non trova riscontro nello sviluppo di Linux. Infatti successivi commentatori avevano già verificato il fatto che la Legge di Brooks non si poteva riferire a progetti già dotati di prototipi funzionanti in cui fossero già stati risolti gran parte dei problemi di architettura, come nel caso di Linux.

  Fondamentalmente Bezroukov trova quindi che non vi è una reale smentita dell'assunto di Brooks, il quale evidenzia invece una diminuzione

[2]Stallman infatti aveva formalmente lasciato l'AI Lab del MIT per non avere restrizioni su quanto prodotto personalmente, ma continuava ad usufruire delle strutture del laboratorio e dell'aiuto degli ex-colleghi

dell'output per sviluppatore con l'aumentare del loro numero al di là del tipo di connessione, fenomeno osservabile in molti Progetti Open Source.
Semmai Internet può favorire l'aumento della qualità media degli sviluppatori abbattendo le barriere geografiche.

- **Se ci sono abbastanza occhi che cercano errori, gli errori sono di poco conto**. Secondo Bezroukov questa non è una affermazione valida per ogni situazione e per ogni Progetto. Per molti Progetti complessi, per ogni due o tre errori corretti se ne può generare un altro. Ciò rende evidente quindi che l'azione di debugging parallelo di più sviluppatori non coordinati può generare dei problemi.
  Tuttavia bisogna notare anche che molti progetti ora tendono a dotarsi di sistemi di bug tracking che comunicano ad un gruppo di debugger coordinati gli interventi da compiere.

- **Linux appartiene al modello *a Bazaar***. La dicotomia tra modello *a Cattedrale* e modello *a Bazaar* proposta da Raymond appare troppo semplicistica a Bezroukov. Sostanzialmente è una metafora per modello di sviluppo *centralizzato* e *decentralizzato* che non tiene conto di vari aspetti come le dimensioni di un Progetto, la complessità, la tempistica, l'accesso alle risorse e la sua natura (elementi del kernel o applicazioni).
  Per esempio, mentre le applicazioni e le utility per Linux sono sviluppate effettivamente secondo il modello *a Bazaar*, il kernel invece presenta molti aspetti del modello *a Cattedrale*, con un gruppo abbastanza ristretto (per il mondo open source) di un centinaio di sviluppatori che lavorano sul kernel e che in più filtrano i contributi che arrivano dall'esterno.

- **Il modello di sviluppo del software open source porta automaticamente migliori risultat**i. Lo stesso Linus Torvalds, inventore del kernel Linux, afferma che ciò non è vero se non si applica il modello

nel modo adeguato e con le giuste condizioni. La qualità media del software libero ed open source appare effettivamente più alta del software proprietario. Tuttavia Bezroukov sottolinea come esistano diversi campi di applicazione in cui tale software non ha raggiunto livelli qualitativi elevati a causa del fatto che l'iniziativa è ancora molto legata alla necessità e alla voglia di singoli o gruppi e non a reali ragionamenti di opportunità e logiche di mercato.

Su questi temi torna anche Paul Jones [Jones], direttore del MetaLab presso l'University of North Carolina (UNC). Egli prende ancora una volta in considerazione la nota banalizzazione dell'assunto di Brooks meglio conosciuta come *Legge di Brooks*, cioè che aggiungendo sviluppatori ad un progetto avanzato lo si ritarda. La convinzione diffusa che tale legge non trovi applicazione nel metodo di sviluppo del software libero ed open source è, anche secondo Jones, priva in parte di fondamento per il fatto che l'osservazione dimostra che i progetti software sviluppati in questo modo raramente hanno una tempistica definita e delle scadenze precise, elementi che invece caratterizzano i progetti tradizionali.

Egli inoltre nota che nel tipico Progetto Open Source è molto facile riscontrare che spesso un gruppo abbastanza limitato di sviluppatori compie la gran parte del lavoro[3] mentre la restante massa di contributori collabora sostanzialmente con la segnalazione e/o correzione degli errori e con poco codice. Il Progetto Open Source quindi, secondo Jones, non è paragonabile con un progetto tradizionale per quanto riguarda l'applicazione della *Legge di Brooks*, in quanto l'aumento dei contributori generici al Progetto Open Source non è identificabile con l'aggiunta di sviluppatori al core team di un progetto tradizionale.

[3]Come risulta anche dallo studio compiuto dall'Open Source Research Team presso la University of North Carolina (UNC) sulla comunità di utenti/sviluppatori di Linux, riportato nel paragrafo 3.5, pg. 83

5.1.2 Conflitti e problemi di competizione all'interno della comunità

Sempre Bezroukov osserva [Bezroukov2] che, come tutte le comunità scientifiche, anche nella vasta comunità open source è presente una certa competizione basata sullo status, che coinvolge i membri in una valutazione di se stessi e degli altri basata su una scala di valori condivisi. In questo contesto lo status è dato dal contributo ad uno o più Progetti e ad attività non produttive di *codice* ma di promozione sociale. Esiste una sorta di potere politico, come esistono le "manovre" attuate dagli individui per acquisire od aumentare tale potere, anche se questo aspetto tende ad essere negato dagli interessati.

Tutto ciò porta alla creazione occasionale di strutture gerarchiche in corrispondenza della concentrazione di questo potere politico, con la conseguenza classica del fenomeno del favoritismo e della creazione di fazioni per la lotta per il potere, inquinando il processo naturale di acquisizione dello status.

In questo contesto possono crearsi delle distorsioni nel meccanismo del peer rewiew e problemi nell'agire in base al principio del *programmare senza ego* [Weimberger]. La paura dell'esclusione può apparire tra le spinte motivazionali assieme ad una accentuata attenzione nei confronti dei media.

5.1.3 Mancanza di incentivi a fare il *lavoro sporco*

L'impressione di Bezroukov [Bezroukov1], ma non solo, è che i Progetti Open Source tendono ad essere di successo e di qualità in campi che risultano direttamente od indirettamente più interessanti per gli stessi sviluppatori. La crescita di Internet rende disponibile un grande spettro di Progetti aperti ai contributi della comunità, tuttavia non tutti ricevono la stessa attenzione dagli sviluppatori o dagli utenti volenterosi. Com'è anche capibile, in un sistema di questo tipo, dove non ci sono coercizioni né una gerarchia fortemente centralizzata, chi è in grado di contribuire in modo effettivo e vuole mettere a disposizione le sue competenze in modo volontario tende a sce-

gliere progetti che lo possano interessare sia dal punto di vista della sfida tecnica, sia da quello del divertimento nel programmare. Ecco dunque che progetti pur meritevoli di sviluppo muoiono per la perdita di interesse da parte della comunità nei loro confronti in quanto considerati noiosi o anche poco portatori di prestigio e notorietà.

Per certi aspetti, come sottolinea Bezroukov, ciò può essere visto come qualcosa di positivo in quanto gli sforzi degli sviluppatori spinti da questo tipo di motivazione portano a risultati migliori. Tuttavia questa situazione può generare, come succede effettivamente, una serie di mancanze e di programmi perennemente allo stadio di versione beta che rimangono come sterili appendici o binari morti.

5.2 Convenienza del software libero ed open source

Nel quarto capitolo si è visto come il modello proprietario non sia l'unico in grado di garantire un ritorno economico. Non sono effettivamente molte le aziende, in genere molto grandi, che possono basare le loro fortune quasi esclusivamente sulla vendita di copie di software proprietari da loro prodotti. Non sono molte proprio perché la pratica di mantenere segreti i codici sorgenti dei loro prodotti ha portato ad un progressivo sfoltimento della concorrenza ed all'assunzione di posizioni monopolistiche non sempre corroborate da una effettiva maggiore qualità.

Su questo aspetto, la concorrenza, il software open source può forse offrire una maggiore possibilità di confronto ad armi pari.

5.2.1 Una questione di costi

Dietro la realizzazione nei modi tradizionali di un software esiste una lunga serie di costi che giustificano gli ingenti investimenti attuati in genere dalle

imprese che producono e mantengono software proprietario, costi che vengono ammortizzati in seguito con i ricavi della sua vendita.

Dall'altro lato c'è il problema dei costi per chi tale software lo acquista. L'impresa che intende munirsi di un prodotto di supporto alla sua attività deve fare i conti non solo con il suo prezzo d'acquisto, ma anche con tutti i costi futuri collegati a tale decisione, la cui somma, assieme all'esborso iniziale, va a comporre quello che abbiamo chiamato *Total Cost of Ownership (TCO)*[TCO].

Il Total Cost of Ownership è composto principalmente, come abbiamo visto, dai costi connessi a: pianificazione e adeguamento dei sistemi, efficienza operativa, training del personale, servizi collegati al software acquistato, aggiornamenti, costi incidentali oltre, ovviamente, al costo d'aquisto.

Come si può intuire da quanto detto per il Total Cost of Ownership, il business si sta spostando radicalmente dalla vendita dei programmi alla vendita dei servizi correlati, che sono veramente molti. D'altro canto, a parte le aziende strettamente o prevalentemente produttrici di software, la classica *softwarehouse* ha sempre ricavato una percentuale importante dei propri profitti dalla vendita di servizi.

Ciò non vuol dire che ci sia una drastica diminuzione della vendita di copie di software proprietario, si è ancora lontani da questo, ma i costi correlati per chi le acquista stanno diventando sempre più onerosi, acquisendo un'importanza sempre maggiore. Ciò porta a constatare la possibilità per le aziende di evitare di caricare il cliente del costo per copia del software venduto nel momento in cui vengano generati abbastanza profitti dai soli servizi correlati.

Inoltre le aziende che utilizzassero software libero o comunque open source per i loro prodotti, dovrebbero sopportare costi di sviluppo molto minori in virtù del fatto che sono già stati "virtualmente" sostenuti dalla comunità. Il fatto di non avere o avere dei bassi costi di produzione elimina la necessità di doverli ripagare mediante l'imposizione di un prezzo di vendita.

Ma anche nel caso in cui il prezzo di vendita sia necessario e giustificato

dal confezionamento del prodotto software ad hoc, viene a cadere la necessità di una royalty. E ciò non solo per la presenza di una Licenza GPL o di qualche altra licenza open surce, quanto per una effettiva mancanza di grossi investimenti iniziali da ripagare a forza di copie successive.

Il pericolo paventato da molte aziende di poter perdere il legame con il cliente per i servizi una volta "regalato" il software, appare francamente inconsistente dal momento che godono senza dubbio di un *vantaggio competitivo* dato dal fatto, se non altro, di conoscere meglio di altri il prodotto che hanno venduto. In ogni caso si stanno diffondendo tipologie di accordo sulle future forniture di servizi volte alla fidelizzazione del cliente.

5.2.2 Possibili punti a favore della scelta *libera*

La scelta per una impresa di utilizzare e/o sviluppare software libero e open source può giovare, come abbiamo visto, per diversi motivi:

- Maggiore spinta nell'evoluzione del prodotto con minori costi.
- Maggiore qualità del prodotto se vengono applicate in modo opportuno le metodologie di sviluppo open source.
- Più agevole ed efficace mantenimento del prodotto.
- Reclutamento e fidelizzazione di collaboratori motivati e capaci attingendo dall'enorme bacino di talenti della comunità.

La possibilità per l'impresa è di poter creare un prodotto con un maggior valore aggiunto per il cliente, non solo in termini di qualità ma anche dal punto di vista del prezzo.

Se un'impresa rilascia, sotto adeguata licenza, il codice sorgente di un suo prodotto, non vuol necessariamente dire che i concorrenti lo useranno con lo stesso successo o che influenzeranno negativamente il suoi affari. Il successo di un prodotto software è il risultato di molto più delle sole caratteristiche del codice, ma comprende fattori come, tra gli altri, la fiducia e la reputazione

derivanti dalla qualità dei servizi forniti e l'efficacia dell'area commerciale e dei canali distributivi.

5.2.3 Perché un cliente può accettare di *regalare* il codice sorgente di un prodotto che acquista

Tra i benefici diretti che la disponibilità del codice sorgente porta si possono ricordare, tra gli altri, la possibilità di continuare a fare esistere e progredire un prodotto software anche oltre la morte dell'azienda che l'ha prodotto; la possibilità di cercare di capire meglio il prodotto se la documentazione è carente e di apportarvi modifiche se cambiano l'ambiente operativo o le condizioni di utilizzo; la possibilità di creare nuovi strumenti ad hoc o di personalizzare quelli già esistenti.

Tra i benefici indiretti si possono citare tutti i possibili contributi esterni che possono essere di grosso aiuto anche al cliente che ha adottato il software, tra cui la documentazione e la manualistica, la consulenza e la creazione da parte di altri produttori di strumenti e applicazioni per il software in questione.

5.2.4 Un'Impresa Open Source avrà contributi dalla comunità?

I programmatori sono in gran parte portati per loro inclinazione a lavorare su software che si dimostri, in prospettiva, utile a risolvere problemi che essi considerano rilevanti. Se un'impresa produce del software che ha tali caratteristiche - almeno per alcuni sviluppatori - e ne rende disponibile il codice sorgente, facilmente troverà qualcuno che ci lavorerà sopra, almeno per ripulirlo dagli errori.

Ci sono vari motivi, a parte quelli morali, per cui uno sviluppatore esterno può trovare interesse a contribuire ad un Progetto Open Source promosso e/o mantenuto da un'impresa:

- Sviluppare versioni personalizzate del software originale rispetto alle esigenze di un proprio cliente o con specifiche caratteristiche per determinati ambiti di mercato.

- Creare e vendere applicazioni dedicate al software in questione.

- Essere assunti dall'impresa che ha prodotto il software e ne ha rilasciato il codice sorgente.

- Creare a sua volta un'impresa che possa collaborare con l'impresa che ha lanciato il progetto open source.

5.3 Modelli di business per software libero ed open source

I modelli di business derivanti dallo sviluppo di software libero ed open source che possono scaturire da questa analisi, derivano direttamente dall'osservazione fatta in precedenza delle possibili fonti di guadagno per una Impresa Open Source.

5.3.1 Distribuzioni Software e supporto

Il software liberamente reperibile anche in rete spesso necessita di conoscenze tecniche che, pur non dovendo essere eccezionali, non sono nel bagaglio culturale di chiunque. L'utente base, spesso non ha le competenze per poter affrontare agevolmente e senza problemi l'installazione e la gestione di questi pacchetti software e preferisce pagare una cifra ragionevole per disporre di una distribuzione costruita apposta per poter essere accessibile anche ai meno esperti.

Ecco quindi che aziende come le varie Red Hat, Turbo Linux, Mandrake ed altre fondano il loro business fondamentalmente sulla vendita al pubblico di un pacchetto completo pronto da installare contenente, oltre al kernel

Linux, una collezione di applicazioni, utility ed un software di installazione per rendere facile per l'utente l'uso e la gestione del sistema operativo.

Queste aziende, inoltre, pertecipano attivamente ai Progetti Open Source collegati al loro prodotto.

Generalmente queste distribuzioni differiscono tra loro per varie caratteristiche come i metodi di installazione, gli strumenti per l'amministrazione del sistema ed altro. Tra le differenze più significative ci sono i mercati di riferimento e l'esplicito orientamento verso determinati tipi di utente.

In questo modello il guadagno proviene da più fonti: la vendita di copie[4], la vendita di manualistica e materiale di supporto, lo sfruttamento del marchio, la vendita di training, consulenze, sviluppo e supporto tecnico post-vendita.

C'è dunque la vendita di due categorie generali di prodotto: beni fisici e supporto tecnico.

5.3.2 Vendita di servizi

A differenza della precedente categoria, le imprese che appartengono a questo modello basano il loro business prevalentemente od esclusivamente sul supporto tecnico, partecipando nel contempo allo sviluppo di alcuni Progetti Open Source, tipicamente quelli che più interessano al loro business. Esempi di grosse realtà di questo tipo sono le americane Linuxcare inc. e VA Linux e la tedesca Innominate A.G.

Come visto, i servizi più diffusi tra quelli prestati dalle aziende di questo tipo sono supporto tecnico, customizzazione di software, training, consulenza e bug-fixing (correzione di bug) a pagamento.

Tutti i casi aziendali proposti nel quarto Capitolo di questo lavoro rientrano in questa categoria. Tuttavia sono necessarie alcune distinzioni.

Innanzitutto le realtà ed i mercati sono differenti. Sia Prosa/Ascensit che Mixadlive sono aziende italiane fondate soprattutto sulla vendita di supporto, customizzazione e training riferiti alla fornitura di software libero. Il loro

[4]Che comunque possono essere riprodotte in quanto software open source

mercato di riferimento è soprattutto quello italiano e nella gran parte dei casi quello regionale, al massimo del nord Italia e con poche eccezioni al momento.

La loro struttura ricalca in parte quella della tradizionale softwarehouse nostrana. Entrambe sono partite con capitali abbastanza ridotti e con un piano di crescita graduale per i prossimi anni che seguirà i ritmi del mercato e l'ampliarsi della domanda di prodotti di software libero ed open source. Ciò che vendono non è tanto il software in sé, comunque disponibile tramite i normali canali open source, quanto la loro abilità nell'adattarlo alle esigenze del cliente o comunque la loro competenza.

Entrambe partecipano in modo per quanto possibile attivo allo sviluppo di alcuni Progetti Open Source, in particolare Prosa/Ascensit è un punto di riferimento per un Progetto, *ETLinux*, nato al suo interno e sviluppato dalla comunità, che sarà presumibilmente il principale generatore per il prossimo futuro di servizi di supporto tecnico.

Ximian è una impresa molto più grande fondata su ingenti investimenti di venture capital. Va inserita in questo modello anche se a prima vista potrebbe sembrare un *vendor*, un distributore, in quanto intende basare in modo esplicito il proprio business sui servizi di supporto tecnico per i prodotti che sviluppa, piuttosto che sulla vendita dei prodotti stessi.

La sua politica attuale è quella di sviluppare un "catalogo" originale di prodotti software open source sui quali basare la futura redditività. La sua struttura vuole ricalcare il modello di sviluppo del Progetto Open Source sulla scorta anche della provenienza dei suoi fondatori e di gran parte degli sviluppatori da esperienze simili nella comunità, a differenza di altre aziende open source che si basano invece sul tradizionale modello della Company.

Mentre per quanto riguarda piccole aziende come Prosa/Ascensit e Mixadlive, la situazione attuale mostra una realtà fatta di concretezza, fatturati - pur modesti ma in crescita -, mercati locali ricettivi e bilanci non stratosferici ma positivi, la realtà delle grosse company basate su ingenti investimenti appare in parte un'incognita.

Abbiamo accennato in precedenza alle difficoltà di Linuxcare su cui non ritorniamo. VA Linux sta vivendo un momento di crisi pesante delle proprie quotazioni sull'indice Nasdaq. Innominate, recentemente sbarcata in Italia, sta gestendo un momento di tensione in patria con la chiusura di due delle otto sedi tedesche e un ridimensionamento degli organici. Tutto ciò fa pensare che, se pur gli investitori di professione guardano con favore al mondo dell'open source, i mercati hanno ancora un po' di riserve verso un fenomeno ancora poco conosciuto che forse manca al momento della maturità e delle strutture per affrontare le grandi sfide.

Ciò non vuol dire che il software libero ed open source non diventeranno in un prossimo futuro una realtà affermata nell'industria del software anzi. Linux stesso è destinato secondo l'opinione di molti a crescere in popolarità e ad essere usato massicciamente per il funzionamento di reti, applicazioni Internet e altri strumenti che necessitano di elevata affidabilità, sicurezza e scalabilità. Tuttavia, come afferma Paul Stewart [Stewart] del Silycon Valley News Service, il prossimo futuro vedrà probabilmente l'assorbimento di questa serie di start-up idealistiche da parte di giganti come IBM, Hewlett Packard, Dell e Compaq che già vedono Linux come il miglior mezzo per un più favorevole posizionamento nell'eterna battaglia contro Sun Microsystem e Microsoft.

5.3.3 Software per il funzionamento di hardware

L'hardware non può funzionare senza il software appropriato (driver, compilatori e linker, applicazioni o interi sistemi operativi), ecco perché i produttori e i distributori di hardware investono somme considerevoli per sviluppare ed aggiornare tali drivers che normalmente sono resi liberamente disponibili ma senza i codici sorgenti.

Questo modello di business si applica alle imprese produttrici di hardware che usano il modello open source per la produzione del software per il funzionamento dei loro apparati, da distribuire gratuitamente unitamente ad essi. In questo caso il software open source, migliore di quello carente sviluppato

con i metodi tradizionali di questo settore, ha la funzione di attribuire un valore aggiunto all'hardware ottimizzandone la funzionalità, l'affidabilità e quindi l'utilità.

Esempi calzanti di imprese che applicano questo modello possono essere l'italiana Eurotech per le schede ed i PC industriali e Matrox per le schede periferiche per PC.

5.3.4 Documentazione

Questo modello è adatto alle imprese che vendono materiale come manuali, libri, riviste, servizi di news e quant'altro sul software libero ed open source ai normali prezzi per questi tipi di supporti. La peculiarità di questa documentazione è che spesso è resa anche disponibile liberamente in rete.

La caratteristica di queste imprese è che in genere non partecipano direttamente allo sviluppo del software libero ed open source, ma offrono un supporto documentativo e mediatico all'intero sistema.

L'esempio per antonomasia di applicazione di questo modello è la casa editrice americana O'Reilly & Associates. Per l'Italia va citata la promettente e molto attiva Hops Libri.

5.3.5 Prodotti civetta

In questo modello, un prodotto liberamente distribuito in forma di codice sorgente è usato come prodotto civetta per favorire la vendita di altri prodotti di software tradizionale a pagamento, libero o proprietario, che costituisce la vera fonte di guadagno dell'azienda.

Questo modello è quello usato, per esempio da Netscape con il rilascio del suo Communicator come software open source. Tuttavia, spesso, la forzata coesistenza di prodotti open source con altri proprietari impedisce l'uso di licenze restrittive come la Licenza GPL, in favore di altre più permissive come la BSD o la Mozzilla Public License.

In un certo senso, anche il software libero o open source per il funziona-

mento di hardware funge da prodotto civetta per la vendita dell'hardware stesso.

5.4 Futuro del software libero ed open source

Come abbiamo visto, il fenomeno del software libero ed open source è molto complesso e sfaccettato. Due sono le anime principali, quella corrispondente al movimento capitanato dalla Free Software Foundation e quella corrispondente alla Open Source Initiative, e tra queste sono riscontrabili molte sfumature.

Questo tipo di software è sviluppato secondo quello che è stato definito il modello del Progetto Open Source. Tuttavia, se l'applicazione di tale modello sembra giustificata nel mondo delle *comunità* open source, non si può dire con certezza che lo sia anche nel mondo del business; ma neanche il contrario.

La nascita di un interesse per lo sfruttamento commerciale su larga scala di Linux, il più rappresentativo tra i software liberi, è abbastanza recente e i dati sono ancora contraddittori. È vero che è molto diffuso a livello di web server in Internet, ma è altrettanto vero che è scarsamente usato come sistema operativo *desktop*. Chi utilizza Linux come sistema operativo per il proprio computer è solamente il 6% del totale, anche se in forte ascesa, mentre quasi l'80% usa MS Windows. Ciò è dovuto in gran parte, si è sempre detto, allo scarso *appeal* di un sistema operativo, Linux, senza fronzoli e poco *user friendly*. Tuttavia queste affermazioni potevano valere appieno fino ad un paio d'anni fa, quando Linux era effettivamente un prodotto *da hacker*, difficile da installare, da configurare e da utilizzare ma molto stabile ed efficace se paragonato ad altri sistemi operativi di larga distribuzione.

Con la diffusione dei concetti di software libero ed open source e con il crescente senso di sana competizione verso il *gigante proprietario*, negli ultimi anni si stanno sviluppando Progetti Open Source che mirano a rendere Linux molto più facile da usare pur senza privarlo delle caratteristiche positive.

Ecco allora Progetti come Gnome o KDE per l'interfaccia grafica o Gimp, Evolution, Mozzilla o Gnumeric per colmare il gap delle applicazioni e delle utility.

La strada da fare è ancora molta, se si pensa che i prodotti proprietari *concorrenti* possono contare su una anzianità in alcuni casi più che decennale, ma le prospettive sono incoraggianti, pensando alla giovane età di molti Progetti già a buon punto.

Questo non fa che confermare in parte gli aspetti positivi del modello del Progetto Open Source tracciati nel terzo capitolo. Tuttavia bisogna rilevare che le ombre non sono assenti. Proprio per le sue caratteristiche *volontarie* o quasi, il Progetto Open Source è portatore di risultati non sempre efficaci ed all'altezza della sua fama nella comunità o tra gli entusiasti dell'ultima ora. In linea con le affermazioni appena viste di Bezroukov, il mondo open source è disseminato di Progetti[5] lacunosi ed in forte ritardo di sviluppo a causa, spesso, dello scarso interesse da parte della comunità. Per la maggior parte questi Progetti sono di un'importanza irrilevante, tuttavia non è raro che alcuni facciano parte di Progetti ben più ampi ed importanti, provocando, con il loro ritardo, quello che si può definire un *collo di bottiglia*.

Alla luce di questo ci si può chiedere, quindi, quanto possa essere appetibile per il mondo commerciale un modello che presenta, accanto a dei grossi punti forti, altrettanto grosse debolezze. Si ha l'impressione che l'open source produca più incertezze che certezze, tanto care al business.

A mio avviso, per quanto ho potuto osservare, buona parte delle critiche a questo modello, pur fondate, appaiono poco rilevanti in un'ottica di mercato. L'attenzione, infatti, va posta sui principï che stanno alla base dei concetti di software libero ed open source. L'impresa può trarre un effettivo vantaggio dalla diversa struttura dei costi derivante sia dall'acquisizione[6] che

[5]Non molti per fortuna!

[6]Vedi Total Cost of Ownership

dalla produzione e distribuzione di questo tipo di software. Le licenze libere ed open source possono liberare sia l'impresa che il cliente da scomodi legacci che non fanno altro che tolgliere spazio alla vera risorsa principale del settore tecnologico e non solo: *i servizi*.

È un dato di fatto, come detto piu volte nel corso di questo lavoro, che le più grosse aziende nel campo del software stiano facendo grossi investimenti nell'open source per munirsi degli strumenti necessari per competere in questo campo.

Microsoft, dal canto suo, continua su una linea di accondiscendente disprezzo verso Linux e simili, forte della sua enorme quota di mercato. Recenti dichiarazioni di Bill Gates puntano il dito sulle difficoltà riscontrate da diverse *start-up* open source. Tuttavia questo sembra sempre di più un atteggiamento di aggressivo timore. Da diverse parti della dirigenza Microsoft provengono segnali di un forte avvicinamento al mondo della politica, in particolare nel tentativo di favorire l'adozione dei brevetti nel software e di cercare di bloccare il crescente interessamento da parte dei governi al mondo dell'open source. Ultimamente, inoltre, sono sorte all'interno del gigante di Redmond imbarazzanti divergenze di vedute, a partire dalla recente affermazione del CEO, Steve Ballmer, sulla pericolosità di Linux come concorrente e sulla sua qualità. Senza contare, tra l'altro, il famoso *Halloween Document*, un documento interno a Microsoft in cui veniva sottolineato il rischio portato dalla costante crescita di Linux e dell'open source in generale.

Non c'è dubbio, quindi, che le comunità continueranno ad esistere e a crescere, come hanno fatto dalla loro nascita, spinte da ideali forti e da un background radicato. Rimane da vedere se e quanto di tutto ciò può essere trasferito nel mondo del business. Come già detto in precedenza, il mondo delle grosse company probabilmente continuerà ad essere dominato dai soliti grossi nomi che si contenderanno il mercato mondiale con Microsoft. Mentre le imprese medio-piccole potranno forse ricevere, se lo capiranno, un giovamento dall'utilizzo di un prodotto software che le pone su un piano di

maggiore autonomia e libertà, senza onerosi e limitanti contratti e accordi di subordinazione conclusi ancora con le company di cui sopra.

Appendice A

Glossario

Nel ripercorrere la storia e le vicende che hanno portato all'affermarsi del tipo di software considerato in questo lavoro è necessario, per una maggiore comprensione, chiarire almeno alcuni dei termini che compongono lo sterminato bagaglio gergale che unisce coloro che appartengono alla cosiddetta *Cultura Hacker*. Linux, come anche gli altri software liberi, nasce infatti nell'ambito di questa cultura che nell'arco di quasi quarant'anni si è sviluppata generando una serie variegata di subculture collegate tra loro dalla condivisione di esperienze, radici e valori comuni.

Come tutte le comunità, quella degli hacker ha sviluppato negli anni un lessico - o per meglio dire un gergo - che li caratterizza e che prende a prestito alcuni termini comuni cambiandone il significato. Ecco perché nella compilazione di questo elenco di termini utili alla lettura di questa tesi, ho scelto di riferirmi in via principale ad una fonte autorevole quanto ufficiale presente in Internet: il *Jargon File* [Jargon].

Questo documento rappresenta un po' la memoria storica della cultura hacker, raccogliendone in una specie di vocabolario il lessico - arricchito da un po' di storia, miti e leggende - e registrando mano a mano le evoluzioni in un continuo lavoro di mantenimento da parte dei membri della comunità. Anche se ancora caratterizzato da quello spirito di scanzonata e goliardica ironia che lo ha fatto nascere nel 1975 come oggetto di svago tra i tecnici del

laboratorio di intelligenza artificiale di Stanford.

Ulteriore fonte per questa raccolta è stata l'enciclopedia virtuale disponibile in rete Webopedia [Webopedia]

BETA 1. La *beta* è una versione di un programma in gran parte funzionante ma ancora in fase di test. I programmi spesso attraversano due fasi di test: Alpha, all'interno della casa di produzione e Beta, di solito presso un gruppo di clienti selezionati e fidati. 2. qualcosa di nuovo e sperimentale. 3. Instabile, insicuro, sospetto (le versioni beta contengono notoriamente alcuni errori).

BUG Letteralmente cimice o piccolo insetto. Nel gergo informatico si riferisce ad una proprietà non desiderata e non prevista di un programma o di un componente hardware, che causa un malfunzionamento. Questo specifico significato si riferisce, come detto, all'uso del termine nel gergo informatico, tuttavia il suo uso negli ultimi anni si è esteso a molti ambiti della lingua parlata con significati simili.

CODICE SORGENTE Si riferisce alla originaria forma in cui un programma è leggibile e modificabile, fatta di istruzioni scritte nel linguaggio di programmazione, tipo il C o il FORTRAN per esempio. Generalmente poi il programma è distribuito in una forma interpretabile solo dal computer detta codice oggetto. Nel software proprietario costituisce segreto industriale e non viene allegato al software. Se distribuito in forma aperta, la disponibilità del codice sorgente consente la modifica del programma, quindi anche il suo sviluppo e la scoperta ed eliminazione di bug (vedi sopra).

CODICE OGGETTO È il codice prodotto dal compilatore a partire dal codice sorgente. Il codice oggetto è spesso simile - o esso stesso - al linguaggio macchina, il linguaggio di basso livello che il computer è in grado di eseguire. Se il codice oggetto non è già nella forma di linguaggio macchina, va trasformato con programmi detti *assembler*.

COMPILATORE È un programma che traduce il codice sorgente in codice oggetto. Il compilatore deve il suo nome al modo in cui funziona, prendendo cioè l'intero codice sorgente raggruppando e organizzando le istruzioni. È diverso dall'*interpreter*, che analizza ed esegue ogni linea del codice sorgente senza guardare all'intero programma. Ogni linguaggio di programmazione di alto livello ha il suo compilatore ed anzi, siccome il compilatore traduce il codice sorgente in codice oggetto, che è unico per ogni tipo di computer, ogni linguaggio di programmazione ha più compilatori.

DEBUGGER È un particolare tipo di programma usato per scovare gli errori - bug - in altri programmi. Il debugger permette al programmatore di fermare il programma in qualsiasi punto del codice ed esaminare e cambiare i valori delle variabili.

DISTRIBUZIONE Collezione di software e programmi confezionati allo scopo di costituire un sistema funzionante. Per esempio una distribuzione Linux comprende un kernel (vedi) ed una serie di applicativi che nel loro insieme formano un sistema operativo pronto per l'installazione.

DATABASE È una raccolta di informazioni organizzata in modo che un programma può selezionare velocemente i dati desiderati, è una specie di archivio elettronico. I tradizionali database sono organizzati per field, record e file. Un field è una singola informazione, un record è un set completo di field ed un file è una raccolta di record.

DRIVER È un programma che controlla e fa funzionare un componente. Ogni componente, come una stampante, un hard disk o una tastiera deve avere un driver. Un driver agisce come un interprete tra il componente ed il programma che usa tale componente.

FILE BINARIO È un file salvato in formato binario. Un file binario è leggibile solamente dal computer. Tutti i programmi eseguibili sono salvati in formato binario, come la maggior parte dei file di dati numerici.

FILE SYSTEM È il sistema che un sistema operativo od un programma usa per organizzare e salvare i file. Per esempio un file system gerarchico usa il sistema delle directory per organizzare i file in una struttura ad albero. Sebbene ogni sistema operativo fornisca un suo file system, è possibile installarne uno diverso a seconda delle preferenze e delle necessità dell'utente.

FTP Sta per *File Transfer Protocol*. È un protocollo di trasmissione che serve per trasmettere dei file attraverso Internet.

HACKER Letteralmente colui che produce manufatti di legno con un'ascia. 1. Una persona a cui piace esplorare i dettagli dei sistemi programmabili e i modi per estendere le loro caratteristiche, in contrasto con la maggioranza degli utenti che preferisce imparare solo il minimo necessario. 2. Qualcuno che programma appassionatamente (persino ossessivamente) o che ami programmare piuttosto che limitarsi a teorizzare sulla programmazione. 3. Una persona in grado di apprezzare i valori dell'hacking. 4. Una persona abile a programmare velocemente. 5. Un esperto in un particolare programma, o che lo usa con particolare frequenza o ci lavora su; come in "Unix hacker". (Le definizioni dalla 1. alla 5. sono correlate e le persone che le soddisfano sono assimilabili.) 6. Un esperto o un appassionato di qualunque tipo. Uno può essere un hacker dell'astronomia, per esempio. 7. Uno che ama la sfida intellettuale di superare o aggirare creativamente le limitazioni. 8. [deprecato] Un ficcanaso maligno che cerca di scoprire informazioni riservate infilandosi da tutte le parti. Da qui *hacker delle password*, *hacker di rete*. Il termine corretto in questo senso è "cracker" (e quindi non si riferisce a concetti presenti in questo lavoro).

Il termine hacker va inoltre a connotare l'appartenenza alla comunità mondiale definita da Internet. Ciò comporta che le persone descritte devono abbracciare consapevolmente una delle versioni dell'etica hacker. È meglio essere definiti hacker dagli altri piuttosto che descrivere

se stessi in questo modo. Gli hacker si considerano in qualche modo come un élite (una meritocrazia basata sull'abilità), nonostante ciò i nuovi membri sono accolti volentieri. [Jargon].

HOWTO È un termine gergale, ora diventato di uso comune, che sta ad indicare i documenti elettronici presenti nei sistemi Unix che spiegano come fare - How-to do, appunto - qualcosa, per esempio configurare una scheda o utilizzare un programma.

HTTP Sta per *Hyper Text Transfer Protocol* ed è il protocollo usato in Internet per la trasmissione dei dati. Definisce come i dati sono formattati e trasmessi e quali azioni devono eseguire i web server ed i web browser in risposta a vari comandi.

IRC Sta per *Internet Relay Chat*. Funziona in modo simile alle famose chat molto diffuse oggi in internet. Permette di conversare in tempo reale e quindi di scambiarsi informazioni in maniera molto più veloce ed efficace rispetto che con Usenet (vedi).
IRC è strutturata come un network di server in Internet, ognuno dei quali accetta una connessione per utente dai programmi client.

LIBRERIA In programmazione una libreria è una *collezione* di routine (vedi) precompilate che possono essere usate da un programma. Le routine, chiamate talvolta moduli, sono salvate in forma di codice oggetto e sono particolarmente utili per immagazzinare routine usate frequentemente perché non è necessario collegarle esplicitamente ad ogni programma che le usa. Il linker guarda automaticamente nelle librerie per routine che non trova altrove.

JAVA È un linguaggio di programmazione ad oggetti sviluppato originariamente da James Gosling della Sun Microsistems inc. con il nome di *Oak*. Nelle intenzioni di Gosling doveva essere il successore del linguaggio C++. Ha avuto una grossa espansione in seguito all'esplosione di Internet del 1993-1994.

È un linguaggio potente e pulito ed è stato adottato da molti programmatori anche se purtroppo non è ben supportato da tutti i web browser (Netscape e Galeon) ed ha alcuni problemi di performance. Nonostante ciò è comunque stato adottato da molti ambienti accademici, prendendo il posto del linguaggio Pascal come strumento preferito per l'insegnamento dei rudimenti della programmazione.

KERNEL È il nucleo essenziale di un sistema operativo responsabile della connessione tra le componenti fisiche di base e tutte le altre parti del sistema operativo.

MAILING LIST 1. Tecnicamente un indirizzo email che rappresenta un alias per altri indirizzi email. Alcune mailing list semplicemente redirigono la posta mandata loro a tutti i destinatari iscritti. Altre sono filtrate da persone (dette moderatori) o programmi secondo i criteri di partecipazione dei destinatari. 2. Le persone che ricevono la posta mandata a tale indirizzo email.
Le mailing list sono una delle prime forme di interazione tra hacker.

METADATA Sono *dati riguardo i dati*. Un file metadata descrive come, quando e da chi un particolare set di dati è stato raccolto ed in che formato sono trattati.

MIRROR Un sito mirror è una replica di un sito già esistente fatto per ridurre il traffico presso il server del sito originale o per aumentarne l'accessibilità. I siti mirror possono anche aumentare la velocità alla quale si può accedere ai file od ai siti: gli utenti possono scaricare più velocemente dei file da un server situato più vicino a loro.

MODULO Nel software un modulo è in pratica una parte di un programma. I programmi sono composti da uno o più moduli sviluppati indipendentemente che vengono combinati quando il programma è assemblato. Ogni singolo modulo può contenere una o più routine (vedi).

MULTITASKING La possibilità di eseguire più di un compito - *task* - contemporaneamente, essendo il compito un programma. Ci sono due tipi fondamentali di multitasking: cooperativo e *preemptive*. Nel cooperativo ogni programma può controllare il processore per il tempo che gli serve. Nel preemptive il sistema operativo assegna ad ogni programma delle porzioni di tempo in cui usare il processore.

NEWSGROUP Gruppi di discussione o forum presenti in Usenet (vedi). Questi gruppi possono essere moderati o meno, come le mailing list, ma si differenziano da queste per in fatto che per accedervi è nacessario l'accesso particolare ad Usenet. Per chi non dispone di questo accesso alcuni gruppi hanno mailing list parallele che utilizzano il canale Internet.

OPENSOURCE Significa sorgenti aperti e si riferisce ai codici sorgenti di un software. È un marchio registrato ed un software per poter essere definito Open Source deve fare uso di una delle licenze certificate come conformi; è un termine introdotto dall'ala più moderata della cultura hacker per renderlo accettabile al mercato e per evitare la valenza provocatoria e anticommerciale di Free Software (Software Libero) usato dagli hacker più intransigenti e radicali.

PATCH Una aggiunta temporanea ad un pezzo di codice, generalmente come rimedio veloce ed approssimativo ad un malfunzionamento. Può non essere incorporata permanentemente al programma. Si differenzia dalle modifiche vere e proprie al programma per il fatto che viene generata in maniera più primitiva e meno accurata rispetto ad esse.

REALTIME Termine che descrive un'applicazione che richiede un programma per rispondere alle sollecitazioni con un tempo di risposta massimo molto ridotto, tipicamente milli o microsecondi. Un esempio canonico sono i processi di controllo degli impianti chimici. Tali applicazioni spesso richiedono un sistema operativo dedicato (perché

ogni altro processo deve essere messo in secondo piano), e hardware particolare.

ROUTINE È una sezione di un programma che esegue un particolare compito. I programmi consistono in moduli ognuno dei quali contiene una o più routine. Il termine routine può essere sinonimo di procedura, funzione e subroutine.

SERVER È un computer od un componente di una rete che gestisce le risorse della rete. Per esempio un file server è un computer o componente dedicato a salvare i file del sistema e ogni utente della rete può salvare i suoi file sul server. Spesso i server sono dedicati, cioè sono adibiti esclusivamente alle funzioni di server.
Con i sistemi operativi multiprocesso, come Unix e Linux, un singolo computer può eseguire diversi programmi alla volta. In questo caso con il termine server ci si può riferire al programma che sta gestendo le risorse piuttosto che all'intero computer.

SHELL 1. Tecnicamente un'interprete di comando, usata per inviare i comandi al sistema operativo; in pratica è la parte del sistema operativo che si interfaccia con il mondo esterno. 2. Più genericamente è ogni programma di interfaccia che media l'accesso ad una speciale risorsa o server per ragioni di convenienza, efficienza o sicurezza; per questo è chiamata shell, cioè "guscio".

TCP/IP Sta per *Transmission Control Protocol/Internet Protocol*. È il protocollo di connessione che fa attualmente funzionare Internet, tanto si è diffuso.

USENET Deriva da "Users' Network". È un sistema distribuito di database di messaggi (Bulletin Board), supportato principalmente da macchine Unix. Implementato originariamente tra il 1979 e il 1980 da Steve Bellovin, Jim Ellis, Tom Truscott e Steve Daniel alla Duke University, si è

diffuso a livello planetario anche grazie ad Internet ed ora è probabilmente il più grande servizio pubblico di informazione decentralizzata esistente.

All'inizio del 1996 ospitava più di 10000 newsgroup (vedi) e mediamente più di 500 megabytes di articoli tecnici, notizie, discussioni.

WEB BROWSER Programma la cui funzione è di localizzare e mostrare le pagine web. I più famosi ed usati sono Netscape Navigator e Microsoft Internet Explorer, entrambi browser grafici, in grado cioè di visualizzare la grafica oltre al testo.

WEB SERVER È un computer che "recapita" le pagine web. Ogni web server ha un indirizzo IP e spesso un nome di dominio. Ogni computer può essere adibito a questa funzione installandovi il software adeguato e connettendolo ad Internet.

Bibliografia

[Apogeo] *Apogeonline OpenPress*. Casa editrice Apogeo: libri e notizie dedicate all'open Source.
`http://www.apogeonline.com/openpress/index.html`

[Ascensit] *Ascensit s.r.l.*
`http://www.ascensit.it`

[Bezroukov1] Nicolai Bezroukov, *Open Source Software Development as a Special Type of Academic Research (Chritique of Vulgar Raymo dism)*, First Monday, Ottobre 1999, vol. 4, n. 10.
`http://firstmonday.org/issues/issue4_10/bezroukov/index.html`

[Bezroukov2] Nikolai Bezroukov, *A Second Look at The Cathedral and the Bazaar*, Dicembre 1999, First Monday, vol. 4, n. 12.
`http://firstmonday.org/issues/issue4_12/bezroukov/index.html`

[Bourne] S.R. Bourne, *The Unix System*, Addison Wensley Publishing Company, 1983; trad. it. *Il Sistema Unix*, Masson Italia Editori, 1986.

[Brooks] Frederick P. Brooks, *The Mythical Man-Month*, Addison-Wesley, 1975, ISBN 0-201-00650-2.

[CVS] *Concurrent Versions System (CVS).*
`http://www.cvshome.org/docs/manual/cvs_1.htmlSEC1`

[Davidow] W.H. Davidow e M.S. Malone, *The Virtual Corporation*, Harper Collins, 1992; trad. it. L'azienda virtuale, Milano, Sperling & Kupfer, 1995.

[Debian] *The Debian Project.*
`http://www.debian.org`

[DiBona] C. DiBona, S. Ockman e M. Stone, (a cura di), *Open Sources. Voices from The Open Source Revolution*, Sebastopol, O'Reilly & Associates; trad. it. *Open Sources. Voci dalla rivoluzione Open Source*, Milano, Apogeo, 1999. Disponibile come documento elettronico in formato HTML nello spazio Web di Open Press all'indirizzo:
`http://www.apogeonline.com/openpress/libri/545/index.html`

[Dido] Paolo Didonè, *Modular Production and Product Customization in Software Development: The Case of ETLinux*. Materiale di un Talk tenuto il 5 Dicembre 2000 a Trento nell'ambito di: *R.O.C.K. 2000 seminar series on Modularity in Problem Solving and Artifact Production*, Dept. of Computer and Management Sciences, University of Trento.
`http://rock.cs.unitn.it/seminarseries/2000abstracts/didoneabstract.html`

[ETLinux] *Progetto ETLinux.*
`http://www.etlinux.org`

[Evers] Steffen Evers, *An Introduction To Open Source Software Development*, Giugno 2000, Tesi di diploma alla Technische Universität Berlin, Fachbereich Informatik, Fachgebiet Formale Modelle, Logik und Programmierung (FLP).
`http://user.cs.tu-berlin.de/ tron/opensource/`

[Freshmeat] *Freshmeat.*
`http://freshmeat.net`

[FSF] *Free Software Foundation*. Il sito ufficiale della FSF, collegata al Progetto GNU.
`http://www.fsf.org`

[GNU] *Progetto GNU*. Il sito ufficiale Progetto GNU creato da Richard Stallman:
`http://www.gnu.org`

[Hecker] Frank Hecker, *Setting Up Shop: The Business of Open-Source Software*, 20 Giugno 2000, versione 0.8.
`http://www.hecker.org/writings/setting-up-shop.html`

[Hops] *Hopslibri*. Casa editrice Hops: open source, internet, libri.
`http://www.hopslibri.com`

[Jargon] *Jargon File*, prima versione, 1, (iniziata da Raphael Finkel a Stanford; documento di pubblico dominio) qui versione 4.2.2, 2000 attualmente mantenuta da Raymond.
`http://www.tuxedo.org/~esr/jargon/jargon.html#`

[Jones] Paul Jones, *Brooks' Law and open source: The more the merrier? Does the open source evelopment method defy tha adage about cooks in the kitchen?*, Maggio 2000.
`http://www.ibm.com/developerworks/library/merrier.html`

[Lehrbaum1] Rick Lehrbaum, *The State of Embedded Linux*, Linuxdevices, Febbraio 2001.
`http://www.linuxdevices.com/articles/AT6573081217.html`

[Lehrbaum2] Rick Lehrbaum, *Prosa: EtLinux rise from the ashes*, Linuxdevices, Gennaio 2001.
`http://www.linuxdevices.com/articles/AT6337092142.html`

[Lerner] J. Lerner e J. Tirole, *The Simple Economics of Open Source*, Feb 2000.

[Levy] S. Levy, *Hackers. Heroes of the computer revolution*, Garden City, Doubleday; trad. it. *Hackers. Gli eroi della rivoluzione informatica*, Milano, Shake, 1996.

[Linux] *The Linux Home Page.*
`http://www.linux.org`

[North] D.C. North, *Institutions, Institutional Change and Economic Performance*, Cambridge, Cambridge University Press; trad. it. *Istituzioni, cambiamento istituzionale, evoluzione dell'economia*, Bologna, Il Mulino, 1994.

[Malone] Thomas W. Malone e Kevin Crowston, *The Interdisciplinary Study of Coordination*, 1993, ACM Computing Survey, vol. 26, n. 1, 1994.
`http://ccs.mit.edu/papers/CCSWP157.html`

[MioloVitali] Paola Miolo Vitali (a cura di), *Strumenti per l'analisi dei costi*. Vol II: *Il costing moderno per la comunicazione interna*, Giappichelli Editore, Torino, ISBN 88-348-6223-6.

[OSI] *Open Source Initiative*. Sito ufficiale della Open Source Initiative guidata da Eric S. Raymond:
`http://www.opensource.org`

[Panta] Marco Pantaleoni, *ETLinux: A New Competitor in the Embedded Arena*, Padova, 23 Giugno 2000.
`http://www.etlinux.org/papers/linuxandc.en.html`

[PLUG] *Pluto Linux User Group*. Comunità di appassionati di Linux. Documentazione in italiano (How-To) e gli articoli del Pluto.
`http://www.pluto.linux.it`

[Poll] AA.VV., *3rd Embedded Linux Poll results: "...and the winner is..."*, LinuxDevices.com, 19 Febbraio 2000.
`http://linuxdevices.com/news/NS3937502298.html`

[Prosa] *Prosa s.r.l.*
`http://www.prosa.it`

[Raymond1] Eric S. Raymond, *The Cathedral & the Bazaar. Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly & Associates, 1999.
`http://www.tuxedo.org/ esr/writings/cathedral-bazaar`

[Raymond2] Eric S. Raymond, *The New Hacker's Dictionary*, MIT Press, 1996, Third edition ISBN 0-262-68092-0.

[Stewart] Paul Stewart, *Will the Open Source Business Model Die?*, PCQuest, Novembre 2000.
`http://www.pcquest.com/content/techtrend/100112905.asp.`

[Schenk] Thomas Schenk, *Linux: It's history and current distributions*, IBM, 2001.
`http://www.developer.ibm.com/library/articles/schenk1.html`

[TCO] *Total Cost of Ownership*, North Carolina State University, Supply Chain Managemen Resource Center, materiale a cura di Ron Handfield.
`http://scm.ncsu.edu/scm/TotalCostofOwnership.html`

[UNC] UNC Open Source Research Team, *A Quantitative Profile of a Community of Open Source Linux Developers*, School of Information and Library Science University of North Carolina at Chapel Hill, Chapel Hill, North Carolina 27599-3360, USA, 6 Ottobre 1999.
`http://www.ibiblio.org/osrt/develpro.html`

[Webopedia] *Webopedia*. On line computer dictionary for Internet terms and technical support.
`http://webopedia.internet.com`

[Weimberg] G.M. Weimberg, *The psychology of computer programming*, NY, Van Nostrahnd Reinhold, 1971 .

[Ximian] *Ximian inc.*
`http://www.ximian.com`

Elenco delle figure