

# Free Software

Jesús M. González Barahona  
Joaquín Seoane Pascual  
Gregorio Robles

PID\_00000001



Universitat Oberta  
de Catalunya

[www.uoc.edu](http://www.uoc.edu)



# Indice

<b>1. Introduzione.....</b>	<b>9</b>
1.1. Il concetto di <i>libertà</i> del software .....	9
1.1.1. Definition .....	10
1.1.2. Termini correlati .....	11
1.2. Motivazioni .....	12
1.3. Le conseguenze della libertà del software .....	13
1.3.1. Per l'utente finale .....	14
1.3.2. Per la pubblica amministrazione .....	14
1.3.3. Per lo sviluppatore .....	15
1.3.4. Per l'integratore .....	15
1.3.5. Per i fornitori di servizi e manutenzione .....	15
1.4. Riassunto .....	16
<b>2. Un po' di storia.....</b>	<b>17</b>
2.1. Il software libero prima del software libero .....	17
2.1.1. E in principio era libero .....	18
2.1.2. Gli anni Settanta e i primi anni Ottanta .....	19
2.1.3. Gli inizi dello sviluppo di Unix .....	20
2.2. Gli inizi: BSD, GNU .....	21
2.2.1. Richard Stallman, GNU, FSF: nasce il movimento del software libero .....	21
2.2.2. Il CSRG di Berkeley .....	22
2.2.3. Le origini di Internet .....	24
2.2.4. Altri progetti .....	26
2.3. Tutti i sistemi vanno .....	26
2.3.1. La ricerca di un kernel .....	26
2.3.2. La famiglia *BSD .....	27
2.3.3. Entra in scena GNU/Linux .....	27
2.4. Un periodo di maturazione .....	29
2.4.1. Fine degli anni Novanta .....	29
2.4.2. Il decennio del 2000 .....	32
2.5. Il futuro: un percorso a ostacoli? .....	39
2.6. Riassunto .....	41
<b>3. Aspetti legali.....</b>	<b>42</b>
3.1. Breve introduzione alla proprietà intellettuale .....	42
3.1.1. Copyright .....	43
3.1.2. Segreto commerciale .....	45
3.1.3. Brevetti e modelli di utilità .....	46
3.1.4. Loghi e marchi registrati .....	47
3.2. Licenze per il software libero .....	47
3.2.1. Tipi di licenze .....	49

3.2.2.	Licenze permissive .....	49
3.2.3.	Licenze forti .....	52
3.2.4.	Distribuzione sotto varie licenze .....	56
3.2.5.	La documentazione di un programma .....	57
3.3.	Riassunto .....	58
<b>4.</b>	<b>Gli sviluppatori e le loro motivazioni.....</b>	<b>60</b>
4.1.	Introduzione .....	60
4.2.	Chi sono gli sviluppatori? .....	60
4.3.	Che cosa fanno gli sviluppatori? .....	61
4.4.	Distribuzione geografica .....	62
4.5.	Dedizione .....	64
4.6.	Motivazioni .....	65
4.7.	Leadership .....	66
4.8.	Riassunto e conclusioni .....	68
<b>5.</b>	<b>Economia.....</b>	<b>70</b>
5.1.	Finanziare i progetti di software libero .....	70
5.1.1.	Finanziamenti pubblici .....	71
5.1.2.	Finanziamenti privati non-profit .....	72
5.1.3.	Finanziamenti da chi ha bisogno di miglioramenti .....	73
5.1.4.	Finanziamento con vantaggi correlati .....	73
5.1.5.	Finanziamento come investimento interno .....	75
5.1.6.	Altre modalità di finanziamento .....	76
5.2.	Modelli di business basati sul software libero .....	78
5.2.1.	Maggiore conoscenza .....	79
5.2.2.	Maggiori conoscenze con restrizioni .....	80
5.2.3.	Sorgente di un prodotto libero .....	81
5.2.4.	Sorgente di un prodotto con restrizioni .....	82
5.2.5.	Licenze speciali .....	83
5.2.6.	Vendita basata sul marchio .....	84
5.3.	Altre classificazioni di modelli di business .....	84
5.3.1.	La classificazione di Hecker .....	84
5.4.	Impatto sulle situazioni di monopolio .....	85
5.4.1.	Elementi che favoriscono i prodotti dominanti .....	86
5.4.2.	Il mondo del software proprietario .....	87
5.4.3.	La situazione con il software libero .....	88
5.4.4.	Strategie per diventare monopolista con il software libero .....	88
<b>6.</b>	<b>Il software libero e le amministrazioni pubbliche.....</b>	<b>90</b>
6.1.	Impatto sulle pubbliche amministrazioni .....	90
6.1.1.	Vantaggi ed effetti positivi .....	91
6.1.2.	Difficoltà nell'adozione e altri problemi .....	94
6.2.	Le azioni delle pubbliche amministrazioni nel mondo del software libero .....	96

6.2.1.	Come soddisfare le esigenze delle amministrazioni pubbliche? .....	97
6.2.2.	Promuovere una società dell'informazione .....	99
6.2.3.	Promozione della ricerca .....	100
6.3.	Esempi di iniziative legislative .....	101
6.3.1.	Proposte di legge in Francia .....	101
6.3.2.	Proposta di legge in Brasile .....	102
6.3.3.	Proposte di legge in Perù .....	103
6.3.4.	Proposte di legge in Spagna .....	104
<b>7.</b>	<b>Ingegneria del software libero.....</b>	<b>106</b>
7.1.	Introduzione .....	106
7.2.	La cattedrale e il bazar .....	107
7.3.	Direzione e decisioni nel bazaar .....	109
7.4.	I processi del software libero .....	110
7.5.	Critica a "La cattedrale e il bazar" .....	112
7.6.	Studi quantitativi .....	113
7.7.	Sviluppi futuri .....	116
7.8.	Riassunto .....	117
<b>8.</b>	<b>Tecnologie e ambienti di sviluppo.....</b>	<b>119</b>
8.1.	Descrizione di ambienti, strumenti e sistemi .....	119
8.2.	Linguaggi e strumenti associati .....	120
8.3.	Ambienti di sviluppo integrati .....	121
8.4.	Meccanismi di collaborazione di base .....	122
8.5.	La gestione di sorgenti .....	123
8.5.1.	CVS .....	124
8.5.2.	Altri sistemi di gestione delle sorgenti .....	127
8.6.	Documentazione .....	129
8.6.1.	Convertitore DocBook .....	130
8.6.2.	Wikis.....	131
8.7.	Bug management e altri problemi .....	132
8.8.	Supporto per altre architetture .....	133
8.9.	Siti di supporto allo sviluppo .....	134
8.9.1.	SourceForge .....	134
8.9.2.	Gli eredi di SourceForge .....	136
8.9.3.	Altri siti e programmi .....	137
<b>9.</b>	<b>Casi di studio.....</b>	<b>138</b>
9.1.	Linux .....	139
9.1.1.	Storia di Linux .....	140
9.1.2.	La modalità di lavoro di Linux .....	141
9.1.3.	Lo stato attuale di Linux .....	142
9.2.	FreeBSD .....	144
9.2.1.	Storia di FreeBSD .....	145
9.2.2.	Lo sviluppo in FreeBSD .....	145
9.2.3.	Il processo decisionale in FreeBSD .....	146

9.2.4.	Aziende che lavorano intorno a FreeBSD .....	147
9.2.5.	Stato attuale di FreeBSD .....	147
9.2.6.	Radiografia di FreeBSD .....	148
9.2.7.	Studi Accademici su FreeBSD .....	150
9.3.	KDE .....	150
9.3.1.	Storia di KDE .....	151
9.3.2.	Sviluppo di KDE .....	151
9.3.3.	La lega KDE .....	152
9.3.4.	Stato attuale di KDE .....	154
9.3.5.	Radiografia di KDE .....	154
9.4.	GNOME .....	157
9.4.1.	Storia di GNOME .....	157
9.4.2.	La Fondazione GNOME .....	158
9.4.3.	Il settore che lavora attorno a GNOME .....	160
9.4.4.	Lo stato attuale di GNOME .....	161
9.4.5.	Radiografia di GNOME .....	162
9.4.6.	Studi Accademici su GNOME .....	164
9.5.	Apache .....	164
9.5.1.	Storia di Apache .....	164
9.5.2.	Lo sviluppo di Apache .....	165
9.5.3.	Radiografia di Apache .....	166
9.6.	Mozilla .....	167
9.6.1.	Storia di Mozilla .....	168
9.6.2.	Radiografia di Mozilla .....	170
9.7.	OpenOffice.org .....	172
9.7.1.	Storia di OpenOffice.org .....	172
9.7.2.	Organizzazione di OpenOffice.org .....	173
9.7.3.	Radiografia di OpenOffice.org .....	173
9.8.	Red Hat Linux .....	174
9.8.1.	Storia di Red Hat .....	175
9.8.2.	Stato attuale di Red Hat. ....	176
9.8.3.	Radiografia di Red Hat .....	177
9.9.	Debian GNU/Linux .....	179
9.9.1.	Radiografia di Debian .....	180
9.9.2.	Confronto con altri sistemi operativi .....	182
9.10.	Eclipse .....	184
9.10.1.	Storia di Eclipse .....	184
9.10.2.	Stato attuale di Eclipse .....	185
9.10.3.	X-ray of Eclipse .....	186
<b>10.</b>	<b>Altre risorse libere.....</b>	<b>188</b>
10.1.	Le più importanti risorse libere .....	188
10.1.1.	Articoli scientifici .....	188
10.1.2.	Leggi e standard. ....	189
10.1.3.	Enciclopedie .....	191
10.1.4.	Corsi .....	192
10.1.5.	Collezioni e databases .....	193

10.1.6. Hardware .....	193
10.1.7. Letteratura e arte .....	194
10.2. Licenze per altre risorse libere .....	194
10.2.1. Licenza di documentazione libera GNU .....	195
10.2.2. Licenze Creative Commons .....	196
<b>Bibliografia</b> .....	201





## 1. Introduzione

"Se tu hai una mela e io ho una mela e ci scambiamo le mele, tu ed io avremo ancora una mela ciascuno. Ma se tu hai un'idea e io ho un'idea e ci scambiamo queste idee, ognuno di noi avrà due idee."

Attribuita a Bernard Shaw

Che cos'è il software libero? Che cos'è la licenza di un programma libero e quali sono le sue implicazioni? Come viene sviluppato il software libero? Come sono finanziati i progetti di software libero e quali sono i modelli di business ad essi associati di cui stiamo facendo esperienza? Che cosa motiva gli sviluppatori, specialmente i volontari, a coinvolgersi in progetti di software libero? Come sono questi sviluppatori? Come vengono coordinati i loro progetti, e com'è il software che producono? In breve, qual è il panorama generale del software libero? Queste sono il genere di domande a cui cercheremo di dare risposta in questo documento. Poiché, nonostante il software libero sia sempre più presente nei media e nei dibattiti di chi lavora nelle tecnologie dell'informazione, e nonostante anche i cittadini in generale stiano iniziando a parlarne, esso è ancora per la maggior parte un'entità sconosciuta. E perfino coloro che lo conoscono, spesso ne conoscono solo alcuni aspetti, in genere ignorando totalmente gli altri.

Per cominciare, in questo capitolo presenteremo gli aspetti specifici del software libero, concentrandoci in particolare sulla spiegazione del suo contesto di formazione, per coloro che si accostano all'argomento per la prima volta, e sottolineandone l'importanza. Come parte del contesto, rifletteremo sulla definizione del termine (per sapere di che cosa stiamo parlando) e sulle principali conseguenze dell'usare il software libero - nonché della sua mera esistenza.

### 1.1. Il concetto di *libertà* del software

Fin dai primi anni Settanta ci siamo abituati al fatto che chiunque metta in commercio un programma possa imporre (e di fatto impone) le condizioni in base alle quali si può usare il programma stesso. Ad esempio, potrebbe essere vietato il prestito a terzi. Nonostante il software sia l'oggetto tecnologico più flessibile e adattabile a nostra disposizione, è possibile imporre il divieto (e spesso viene imposto) di adattarlo ad esigenze particolari, o di correggerne gli errori, senza il consenso esplicito del produttore, che in genere si riserva il diritto esclusivo a queste opportunità. Ma questa è soltanto una delle possibilità offerte dalla legislazione attuale: *il software libero*, d'altro canto, offre delle libertà che *il software proprietario* nega.

#### Software Proprietario

In questo testo useremo il termine *software proprietario* in riferimento a qualsiasi programma che non possa essere considerato software libero in base alla definizione che forniremo in seguito.

### 1.1.1. Definition

Perciò, il termine *software libero* (o *programmi liberi*), come concepito da Richard Stallman nella sua definizione (Free Software Foundation, "Definition of free software" <http://www.gnu.org/philosophy/free-sw.it.html> [120]), si riferisce alle libertà concesse a chi lo riceve, che sono sostanzialmente quattro:

- 1) Libertà di eseguire il programma in qualsiasi luogo, per qualsiasi scopo e senza limiti di tempo.
- 2) Libertà di studiare come funziona il programma e adattarlo alle proprie necessità. L'accesso al codice sorgente ne è un prerequisito.
- 3) Libertà di redistribuire copie, in modo da aiutare il prossimo.
- 4) Libertà di migliorare il programma e distribuire i miglioramenti al pubblico. Anche in questo caso è necessario l'accesso al codice sorgente.

Il meccanismo che garantisce queste libertà, in accordo con la legislazione corrente, è la distribuzione sotto una specifica licenza che vedremo più avanti (capitolo 3). Attraverso la licenza, l'autore permette a chi riceve il programma di esercitare queste libertà, con le eventuali limitazioni che l'autore desideri applicare (ad esempio l'obbligo di citare gli autori originali nel caso di una redistribuzione). Perché la licenza sia considerata libera, però, queste limitazioni non devono invalidare le libertà citate sopra.

#### L'ambiguità del termine *libero*

Il termine originale inglese per *programmi liberi* è *free software*. Tuttavia, in inglese, *free* significa sia 'libero' nel senso di 'libertà', che 'gratuito' o 'gratis', il che provoca una notevole confusione. Per questo, spesso gli inglesi prendono in prestito i termini spagnoli e parlano di *libre software*, in contrasto con *gratis software*, così come anche noi prendiamo in prestito il termine *software*.

Pertanto, le definizioni di software libero non fanno alcun riferimento al fatto che esso possa essere ottenuto gratuitamente: il software libero e il software gratuito sono due cose molto diverse. Tuttavia, chiarito questo punto, occorre anche spiegare che, a causa della terza libertà, chiunque può redistribuire un programma senza chiedere un compenso monetario o un permesso: questo rende praticamente impossibile ottenere grossi profitti semplicemente redistribuendo software libero: chiunque abbia ottenuto software libero può a sua volta redistribuirlo a prezzo più basso, o addirittura gratuitamente.

**Nota**

Nonostante il fatto che chiunque possa mettere in commercio un dato programma a qualsiasi prezzo, e che questo in teoria significhi far tendere i prezzi di redistribuzione verso il costo marginale del produrre copie del programma, esistono modelli di business basati proprio sulla vendita del software, perché in molte circostanze il consumatore è disposto a pagare per ottenere determinati benefici, come ad esempio una garanzia, benché soggettiva, per il software acquisito, o il valore aggiunto della scelta, aggiornamento e organizzazione di un insieme di programmi.

Da un punto di vista pratico, molti testi definiscono più precisamente quali condizioni una licenza debba soddisfare per poter essere considerata una licenza di software libero. Tra queste, desideriamo sottolineare per la loro importanza storica le definizioni di software libero della Fondazione Software Libero ( <http://www.gnu.org/philosophy/free-sw.it.html> ) [120], le direttive Debian per decider se un programma è libero ( [http://www.debian.org/social\\_contract.it.html#guidelines](http://www.debian.org/social_contract.it.html#guidelines) ) [104] e la definizione del termine *open source* della Open Source Initiative ( [http://www.opensource.org/docs/definition\\_plain.html](http://www.opensource.org/docs/definition_plain.html) ) [215], che è molto simile alle precedenti.

**Nota**

Ad esempio, le direttive Debian sono dettagliate al punto da permettere all'autore di esigere che i codici sorgente distribuiti non siano modificati direttamente, ma che l'originale sia accompagnato da aggiornamenti separati e che i programmi binari siano generati con nomi diversi da quelli originali. Esigono anche che le licenze non contaminino altri programmi distribuiti attraverso gli stessi mezzi.

**1.1.2. Termini correlati**

Il termine software open source ('programmi open source'), promosso da Eric Raymond e dalla Open Source Initiative, è equivalente al termine *software libero*. Rispetto alla filosofia, il termine è molto diverso perché sottolinea l'accessibilità del codice sorgente e non la sua libertà, ma la definizione è praticamente uguale a quella di Debian ("The open source definition", 1998 [http://www.opensource.org/docs/definition\\_plain.html](http://www.opensource.org/docs/definition_plain.html) ) [183]. Questo nome è politicamente più neutrale e sottolinea l'aspetto tecnico, che può portare a benefici tecnici, tra cui migliori modelli di sviluppo e di business, migliore sicurezza, eccetera. Fortemente criticato da Richard Stallman ("Perché *software libero* è meglio che *open source* ", o "Why *free software* is better than *open source* ") [204] e dalla Fondazione Software Libero, o Free Software Foundation ( <http://www.fsf.org> ) [27], ha funzionato molto meglio nella letteratura commerciale e nelle strategie aziendali che in un modo o nell'altro supportano il modello.

Altri termini associati in qualche modo al software libero sono i seguenti:

<b>Freeware</b>	Si tratta di programmi liberi. In genere sono distribuiti solo in formato binario e si possono ottenere gratuitamente. A volte è possibile ottenere il permesso di redistribuirli e a volte no: in questo caso il permesso può essere ottenuto solo attraverso il sito "ufficiale", mantenuto a questo scopo. In genere i programmi freeware vengono usati per promuovere altri programmi (solitamente con funzionalità più complete) o servizi. Esempi di questo tipo di programmi sono Skype, Google Earth e Microsoft Messenger.
<b>Shareware</b>	In questo caso non si tratta nemmeno di software gratuito, ma piuttosto di una modalità di distribuzione: solitamente questi programmi possono essere copiati liberamente, in genere senza codice sorgente, ma non si possono usare in continuazione per un lungo periodo senza pagare. La richiesta di pagare può essere motivata da una funzionalità limitata, dal ricevere messaggi fastidiosi o da un mero appello all'etica dell'utente. Inoltre, i termini legali della licenza possono essere usati contro i trasgressori.
<b>Charityware , careware</b>	In genere si tratta di <i>shareware</i> che richiede un pagamento da destinare ad un'associazione di <i>beneficenza</i> . In molti casi, anziché esigere un pagamento, si richiede un'offerta volontaria. Alcuni software liberi, come <b>Vim</b> , chiedono contributi volontari di questo tipo (Brian Molenaar, "Qual è il contesto del charityware?" o "What is the context of charityware?") [173].
<b>Dominio pubblico</b>	In questo caso, l'autore rinuncia totalmente a tutti i suoi diritti in favore del pubblico dominio. Questo deve essere esplicitamente dichiarato nel programma, altrimenti il programma viene ritenuto proprietario e non se ne può fare nulla. In questo caso, se viene messo a disposizione anche il codice sorgente, il programma è libero.
<b>Copyleft</b>	Questo è un caso particolare di software libero in cui la licenza richiede che tutte le eventuali modifiche distribuite siano anch'esse libere.
<b>Proprietario, locked-in, non-libero</b>	Questi termini sono usati per riferirsi a software né libero né open source.

## 1.2. Motivazioni

Come abbiamo visto, esistono due grandi famiglie di motivazioni per lo sviluppo del software libero, che a loro volta danno origine ai due nomi con i quali è conosciuto:

- La motivazione etica, rappresentata dalla Fondazione per il Software Libero ( <http://www.fsf.org> ) [27], che ha ereditato la cultura da *hacker* e promuove l'uso del termine *libero*, sostenendo che il software è conoscenza che dovrebbe essere condivisa senza ostacoli, che nasconderla è antisociale e che l'abilità di modificare i programmi è una forma di libertà di espressione. Per approfondimenti, leggere *Software libero, società libera. Saggi scelti di Richard M. Stallman* (Free software, free society. Selected essays of Ri-

chard M. Stallman ) [211] o l'analisi di Pekka Himanen *L'etica hacker e lo spirito dell'era dell'informazione* (The hacker ethic and the spirit of the information age. Random House, 2001) [144].

- La motivazione pragmatica, rappresentata dalla Open Source Initiative ( <http://www.opensource.org> ) [54] che promuove l'uso del termine *open source* e adduce l'argomento dei vantaggi tecnici e finanziari che discuteremo nella prossima sezione.

Oltre a queste due motivazioni principali, la gente che lavora sul software libero può farlo per molte altre ragioni, ad esempio per divertimento (Linus Torvalds e David Diamond, *Texere*, 2001) [217] o per denaro, possibilmente con *modelli di business* sostenibili. Il Capitolo 4 approfondisce queste motivazioni sulla base di analisi oggettive.

### 1.3. Le conseguenze della libertà del software

Il software libero offre molti vantaggi e, dei pochi svantaggi, molti sono stati esagerati (o inventati) dalla concorrenza del software proprietario. Lo svantaggio più fondato è quello finanziario, dal momento che, come abbiamo visto, non è possibile ricavare molti soldi dalla sua distribuzione, che può e tende ad essere realizzata da altri rispetto all'autore. Per questo motivo occorrono modelli di business e meccanismi di finanziamento diversi, che vedremo nel capitolo 5. Altri svantaggi, come la mancanza di assistenza al cliente o la scarsa qualità, sono legati all'aspetto economico, ma in molti casi sono falsi, perché perfino il software privo di forme di finanziamento tende ad offrire buoni livelli di assistenza, grazie ai forum di utenti e sviluppatori, e spesso la qualità è molto alta.

Tenendo presenti le considerazioni economiche, occorre però osservare che il modello di costo del software libero è molto diverso dal modello di costo del software proprietario, perché gran parte del software libero viene sviluppato al di fuori dell'economia monetaria formale, spesso utilizzando meccanismi di scambio/baratto: "Io ti dò un programma che ti interessa e tu lo adatti alla tua architettura e apporti i miglioramenti che ti servono". Il capitolo 7 discute i meccanismi giusti di ingegneria del software per far rendere al massimo queste risorse umane non pagate, con le loro particolari caratteristiche, mentre il capitolo 8 studia gli strumenti usati per rendere efficace questa collaborazione. Inoltre, una gran parte dei costi è ridotta dal fatto che il software è libero, e quindi i nuovi programmi non devono essere scritti da zero, perché si può riutilizzare software già esistente. Anche la distribuzione ha un costo molto minore, perché avviene attraverso Internet e con pubblicità gratuita all'interno di forum pubblici progettati a questo scopo.

Un altro effetto delle libertà è la qualità risultante dalla collaborazione di volontari che apportano contributi o che scoprono e segnalano bachi in ambienti o situazioni inimmaginabili dallo sviluppatore originale. Inoltre, se un pro-

gramma non offre qualità sufficiente, la concorrenza può prenderlo e migliorarlo sulla base di quel che esiste già. In questo modo la *collaborazione* e la *competizione*, due meccanismi potenti, si combinano per produrre migliore qualità.

Ora esaminiamo le conseguenze positive per l'utente.

### **1.3.1. Per l'utente finale**

L'utente finale, che sia un individuo o un'azienda, può trovare concorrenza vera in un mercato con tendenze monopolistiche. Per la precisione, non deve necessariamente dipendere dall'assistenza del produttore del software, perché potrebbero esistere diverse aziende, anche piccole, con il codice sorgente e le conoscenze necessarie per permettere loro di fare affari pur mantenendo liberi certi programmi.

Cercare di scoprire la qualità di un prodotto non dipende più tanto dall'*affidabilità* del produttore, quanto dalle indicazioni ricavate dal consenso della comunità e dalla disponibilità del codice sorgente. Inoltre, possiamo scordarci le *scatole nere*, di cui bisogna fidarsi "perché lo diciamo noi", e le strategie dei produttori che possono decidere unilateralmente se abbandonare o mantenere un particolare prodotto.

Valutare i prodotti prima della loro adozione è stato reso molto più semplice, perché adesso è sufficiente installare i prodotti alternativi nei nostri ambienti di lavoro reali e testarli, mentre per il software proprietario ci si deve affidare a relazioni esterne o negoziare con i fornitori per dei test, che non sempre sono possibili.

Grazie alla libertà di modificare il programma per i propri usi, gli utenti sono in grado di personalizzarlo o adattarlo ai propri requisiti, correggendo anche gli eventuali errori, se ce ne sono. Il processo di correggere gli errori trovati è per gli utenti di software proprietario in genere estremamente laborioso, se non addirittura impossibile, dal momento che, se riusciamo a far correggere gli errori, le correzioni saranno spesso incorporate nella versione successiva, che potrebbe essere rilasciata dopo diversi anni e per la quale - tra l'altro - dovremo di nuovo pagare. Con il software libero, d'altra parte, possiamo fare correzioni e riparazioni da soli, se ne siamo capaci, o altrimenti affidare il compito a esterni. Possiamo anche, direttamente o impiegando servizi esterni, integrare il programma con un altro o verificarne la qualità (per esempio in termini di sicurezza). Il controllo passa in larga parte dal fornitore all'utente.

### **1.3.2. Per la pubblica amministrazione**

La pubblica amministrazione è un utente di grandi dimensioni con caratteristiche particolari: essa infatti ha obblighi speciali nei confronti dei cittadini, si tratti di fornire servizi accessibili, di essere neutrale nei confronti dei produt-

tori, o di garantire l'integrità, l'utilità, la privacy e la sicurezza dei loro dati nel lungo periodo. Tutti gli aspetti sopra elencati obbligano la pubblica amministrazione ad essere più rispettosa degli standard rispetto alle aziende private, a mantenere i dati in formati aperti e a processarli con software indipendente dalle strategie di aziende spesso straniere, la cui sicurezza sia certificata da verifiche interne. L'adattamento agli standard è una caratteristica eminente del software libero, mentre il software proprietario non rispetta gli standard allo stesso modo, perché in genere preferisce creare mercati "prigionieri".

Inoltre, l'Amministrazione serve in un certo senso da vetrina e da guida per l'industria, nel senso che ha un forte impatto, che dovrebbe essere volto a intrecciare un tessuto tecnologico che generi ricchezza nazionale. Questa ricchezza si può creare promuovendo lo sviluppo di aziende dedicate a sviluppare nuovo software libero per l'Amministrazione, adattando o verificando il software esistente. Nel capitolo 6 approfondiremo questo argomento.

### **1.3.3. Per lo sviluppatore**

Per lo sviluppatore e il produttore di software, la libertà modifica significativamente le regole del gioco. Diventa più facile continuare a competere pur essendo una piccola impresa e acquisire tecnologia all'avanguardia. Ci permette di sfruttare il lavoro degli altri, perfino di fare concorrenza ad un altro prodotto modificando il suo stesso codice, anche se il concorrente da cui copiamo può a sua volta servirsi del nostro codice (se si tratta di *copyleft*). Se il progetto è gestito bene, è possibile ottenere la collaborazione gratuita di un gran numero di persone e, inoltre, di ottenere accesso a un sistema di distribuzione globale e virtualmente gratuito. Tuttavia, rimane il problema di come ottenere risorse finanziarie, se il software non è il prodotto di una commissione a pagamento. Il capitolo 5 tratta questo aspetto.

### **1.3.4. Per l'integratore**

Per gli integratori, il software libero è un paradiso. Significa che non ci sono più scatole nere da adattare una all'altra, spesso usando l'ingegneria inversa. Gli angoli aguzzi si possono smussare ed è possibile integrare insieme parti di programmi diversi per ottenere il prodotto integrato richiesto, perché esiste un vasto bacino condiviso di software libero da cui si possono estrarre le varie parti.

### **1.3.5. Per i fornitori di servizi e manutenzione**

Avere a disposizione il codice sorgente cambia tutto e ci mette nella stessa posizione del produttore. Se la posizione non è proprio la stessa, è perché ci manca la conoscenza profonda del programma che solo lo sviluppatore possiede, il che significa che, per chi lavora nella manutenzione, è consigliabile partecipare ai progetti per i quali poi si dovrà fare manutenzione. Il valore

aggiunto dei servizi è tanto più apprezzato in quanto il prezzo del programma è basso. Il software libero è attualmente il settore d'affari più trasparente e in cui è possibile maggiore concorrenza.

#### **1.4. Riassunto**

Questo primo capitolo è servito come incontro preliminare con il mondo del software libero. Il concetto definito da Richard Stallman si basa su quattro libertà (libertà di eseguire, libertà di studiare, libertà di redistribuire e libertà di migliorare), due delle quali richiedono accesso al codice sorgente. Tale accessibilità e i suoi vantaggi hanno motivato un altro punto di vista, meno etico e più pragmatico, difeso dalla Open Source Initiative, che ha dato origine ad un altro termine: *software open source*. Abbiamo citato anche altri termini, simili od opposti, che aiutano a chiarire vari concetti. Infine abbiamo discusso le conseguenze del software libero per i principali soggetti coinvolti.



## 2. Un po' di storia

"Quando ho iniziato a lavorare al Artificial Intelligence Lab del MIT nel 1971, sono entrato a far parte di una comunità di condivisione del software che esisteva già da vari anni. La condivisione del software non si limitava alla nostra particolare comunità; esiste da quando esistono i computer, come lo scambio di ricette esiste da quando si cucina. Ma noi lo facevamo più della maggior parte degli altri. [...] Non chiamavamo il nostro software *software libero*, perché quell termine ancora non esisteva, ma di software libero si trattava. Ogni volta che persone di un'altra università o di un'azienda volevano adattare e usare un programma, li lasciavamo fare con piacere. Se vedevi qualcuno usare un programma sconosciuto e interessante, potevi sempre chiedere di vedere il codice sorgente, in modo da poterlo leggere, cambiare, o sfruttarne dei pezzi per fare un nuovo programma."

Richard Stallman, "The GNU Project" (pubblicato per la prima volta nel libro *Open sources*) [208]

Benché la storia di tutte le tecnologie informatiche sia per forza breve, quella del software libero è una delle più lunghe. Si potrebbe addirittura dire che in principio quasi tutti i software sviluppati rientravano nella definizione di *software libero*, anche se il concetto ancora non esisteva. In seguito la situazione è cambiata completamente e, per un periodo piuttosto lungo, il software proprietario ha dominato la scena in modo quasi esclusivo. Proprio in quel periodo sono state gettate le basi per il software libero che conosciamo oggi: i programmi liberi hanno iniziato ad apparire, un pezzettino alla volta. Nel tempo, questi inizi si sono trasformati in una tendenza che è proseguita e maturata fino al giorno d'oggi, in cui il software libero è una possibilità degna di considerazione in quasi tutti i settori.

Questa storia è sconosciuta ai più, al punto che per molti professionisti delle tecnologie informatiche il software proprietario è software "allo stato naturale". In realtà la situazione è quella opposta; i semi di cambiamento che hanno iniziato a germogliare nella prima decade del XXI secolo erano già stati piantati all'inizio degli anni Ottanta.

### Bibliografia

Non esistono molte storie dettagliate del software libero, e quelle esistenti sono in genere articoli che si limitano a trattare il loro soggetto principale. In ogni caso, i lettori interessati possono approfondire quanto descritto in questo capitolo leggendo (in inglese) "Open Source Initiative. History of the OSI" [146] (<http://www.opensource.org/history>), che sottolinea l'impatto del software libero sulla comunità professionale negli anni 1998 e 1999; "A brief history of free/open source software movement" [190], di Chris Rasch, che narra la storia del software libero fino al 2000, o "The origins and future of open source software" (1999) [177], di Nathan Newman, che si concentra in particolare sulla promozione indiretta di software libero o sistemi simili da parte del Governo degli Stati Uniti negli anni Settanta e Ottanta.

### 2.1. Il software libero prima del software libero

Il concetto di software libero non comparve fino agli inizi degli anni Ottanta. Tuttavia la sua storia inizia molti anni prima.

### 2.1.1. E in principio era libero

Durante gli anni Settanta il panorama delle tecnologie informatiche era dominato da grossi calcolatori, in genere installati nelle aziende e nelle istituzioni governative. IBM era il principale produttore, molto più avanzato rispetto ai concorrenti. In questo periodo, quando si comprava un computer (l'hardware), il software era incluso. Finché il contratto per la manutenzione rimaneva valido, si aveva accesso a tutto il catalogo di software del produttore. Inoltre, l'idea di programmi come qualcosa di "separato" dal punto di vista commerciale era inusuale.

In questo periodo il software veniva normalmente distribuito insieme al suo codice sorgente (in molti casi era solo codice sorgente) e, in generale, senza restrizioni pratiche. Gruppi di utenti come SHARE (utenti dei sistemi IBM) o DECUS (utenti di DEC) partecipavano a questi scambi e, entro certi limiti, li organizzavano. La sezione "Algoritmi" della rivista *Communications of the ACM* era un altro buon esempio di forum di scambio. Si potrebbe dire che in questi primi anni dell'informatica il software era libero, almeno nel senso che chi aveva accesso al software poteva in genere accedere anche al codice sorgente, ed era abituato a condividerlo, modificandolo e condividendo anche le modifiche.

Il 30 giugno 1969, IBM annunciò che nel 1970 avrebbe iniziato a vendere parte del suo software separatamente (Burton Grad, 2002) [131]. Questo significava che i suoi clienti non avrebbero più potuto ottenere i programmi di cui avevano bisogno inclusi nel prezzo dell'hardware. Si cominciò a percepire il software come qualcosa con un valore intrinseco, e di conseguenza sempre più spesso l'accesso ai programmi veniva scrupolosamente ristretto; anche la possibilità di condividerli tra utenti, modificarli o studiare il software veniva limitata il più possibile (tecnicamente e legalmente). In altre parole, la situazione si trasformò in quella che continua tuttora nel mondo del software agli inizi del XXI secolo.

#### Bibliografia

I lettori interessati ad approfondire questo periodo di transizione possono leggere, per esempio, "How the ICP Directory began" [226] (1998), in cui Larry Welke racconta la nascita di uno dei primi cataloghi di software non associati a un produttore, e come durante questo processo si scoprì che le aziende erano disposte a pagare per programmi non sviluppati dai produttori dei loro computer.

A metà degli anni Settanta era diventato del tutto normale, nel settore informatico, trovare software proprietario. Questo portò ad un enorme cambio di mentalità tra i professionisti che lavoravano con il software e segnò l'inizio del fiorire di un gran numero di aziende dedicate a questo nuovo business. Doveva passare ancora un decennio prima che, in maniera organizzata e come reazione a questa situazione, iniziasse ad apparire quello che oggi chiamiamo *software libero*.

### 2.1.2. Gli anni Settanta e i primi anni Ottanta

Anche quando esplorare il modello del software proprietario era divenuta la tendenza di gran lunga prevalente, esistevano iniziative che mostravano già alcune delle caratteristiche di quello che più avanti sarebbe stato considerato software libero. Di fatto alcune di esse producevano proprio software libero come lo definiamo oggi. Tra queste ricordiamo SPICE, TeX e Unix, che è un caso molto più complesso.

SPICE (Simulation Program with Integrated Circuit Emphasis) è un programma sviluppato dall'Università della California di Berkeley per simulare le caratteristiche elettriche di un circuito integrato. Fu sviluppato e reso di pubblico dominio dal suo autore, Donald O. Pederson, nel 1973. SPICE era nato come programma didattico, e come tale si diffuse velocemente nelle università di tutto il mondo. Qui veniva usato da molti studenti di quella che era all'epoca una disciplina emergente: progettazione di circuiti integrati. Essendo di pubblico dominio, SPICE poteva essere distribuito, modificato, studiato. Poteva essere adeguato a requisiti particolari, e quella versione poteva essere venduta come un prodotto proprietario (il che è stato fatto da un gran numero di aziende dozzine di volte durante la loro storia). Con queste caratteristiche, SPICE aveva tutte le carte in regola per diventare lo standard industriale, con le sue varie versioni. E in effetti così fu. Si tratta probabilmente del primo programma con caratteristiche da software libero ad aver catturato per un certo periodo un mercato, quello dei simulatori di circuiti integrati - e probabilmente ci riuscì proprio grazie a queste caratteristiche (in aggiunta alle sue innegabili qualità tecniche).

#### Bibliografia

Ulteriori informazioni sulla storia di SPICE si possono consultare in "The life of SPICE", presentato durante il Bipolar Circuits and Technology Meeting, Minneapolis, MN, USA, nel settembre 1996 [175].

Il sito web di SPICE si può trovare al seguente indirizzo: <http://bwrc.eecs.berkeley.edu/Classes/IcBook/SPICE/>.

Donald Knuth iniziò a sviluppare TeX durante un anno sabbatico, nel 1978. TeX è un sistema di tipografia elettronica usato per produrre documenti di alta qualità. Fin dall'inizio, Knuth usò una licenza che oggi sarebbe considerata licenza da software libero. Quando il sistema venne considerato sufficientemente stabile, nel 1985, egli mantenne quella licenza. All'epoca, TeX era uno dei più grossi e più noti sistemi che si potessero considerare software libero.

#### Bibliografia

Alcune delle pietre miliari nella storia di TeX si possono trovare consultando online <http://www.math.utah.edu/software/plot79/tex/history.html> [39]. Per ulteriori dettagli, l'articolo corrispondente su Wikipedia è anch'esso estremamente utile, <http://www.wikipedia.org/wiki/TeX> [233].

### 2.1.3. Gli inizi dello sviluppo di Unix

Unix, uno dei primi sistemi operativi portabili, fu inizialmente creato da Thompson e Ritchie (tra gli altri) dei laboratori Bell di AT&T. Dalla sua nascita nel 1972 ha continuato a svilupparsi, dando origine a un'infinità di varianti, vendute (letteralmente) da decine di aziende.

Negli anni 1973 e 1974 Unix arrivò in molte università e centri di ricerca di tutto il mondo, con una licenza che ne permetteva l'uso per scopi accademici. Anche se alcune restrizioni ne proibivano la libera distribuzione, tra le organizzazioni che possedevano la licenza il funzionamento era molto simile a quello che si sarebbe visto più avanti in molte comunità di software libero. Coloro che avevano accesso al codice sorgente di Unix avevano a che fare con un sistema che potevano studiare, migliorare ed espandere. Attorno ad esso emerse una comunità di sviluppatori, che presto gravitarono verso il CSRG dell'Università della California di Berkeley. Questa comunità sviluppò la sua cultura, che, come vedremo più avanti, fu molto importante nella storia del software libero. In un certo senso, Unix era un primo esperimento di quello che avremmo visto molti anni dopo con GNU e Linux. Era limitato ad una comunità molto più ristretta, e la licenza di AT&T era necessaria, ma in tutti gli altri aspetti il suo sviluppo fu molto simile (in un mondo molto meno "comunicativo").

#### Metodi di sviluppo inerenti al software libero

In *Netizens. On the history and impact of Usenet and the Internet* (IEEE Computer Society Press, 1997 [139], pagina 139) si leggono alcune righe che potrebbero riferirsi a molti progetti di software libero: "Il fatto che il codice sorgente fosse accessibile e disponibile ha contribuito molto al valore di Unix nei primi tempi del suo sviluppo. Il codice poteva essere esaminato, migliorato e personalizzato".

A pagina 142 della stessa opera si trova la seguente affermazione: "Pionieri come Henry Spencer sono d'accordo su quanto fosse importante avere il codice sorgente per le persone della comunità di Unix. Egli osserva che avere i codici sorgente permetteva di identificare e correggere i bachi che si scoprivano. [...] Perfino alla fine degli anni Settanta e all'inizio degli Ottanta, quasi tutti i siti di Unix avevano i codici sorgente completi".

Il testo di Marc Rochkind "Interview with Dick Haight" è ancora più esplicito (*Unix Review*, Maggio 1986) [198]: "questa era una delle grandi cose di Unix nei primi tempi: la gente condivideva davvero le cose. [...] A quell'epoca non solo imparavamo tantissimo dal condividere il materiale, ma non dovevamo nemmeno preoccuparci di come le cose funzionassero realmente, perché potevamo sempre andare a leggere il codice sorgente."

Nel tempo, Unix divenne anche uno dei primi esempi dei problemi che possono emergere dai sistemi proprietari che a prima vista "avevano qualche caratteristica del software libero". Verso la fine degli anni Settanta e specialmente nel decennio degli anni Ottanta AT&T cambiò politica: l'accesso alle nuove versioni di Unix divenne difficile e costoso. La filosofia dei primi anni, che aveva reso Unix così apprezzato tra gli sviluppatori, cambiò radicalmente, al punto che nel 1991 AT&T tentò addirittura di fare causa all'Università di Berkeley per aver pubblicato il codice Unix BSD, creato dal CSRG di Berkeley. Ma questa è un'altra storia che riprenderemo più avanti.

## 2.2. Gli inizi: BSD, GNU

In tutti i casi discussi nella sezione precedente si trattava o di iniziative individuali o di sistemi che non rispondevano esattamente ai requisiti del software libero. Si dovette aspettare fino agli inizi degli anni Ottanta perché apparissero i primi progetti consapevoli e organizzati per creare sistemi comprendenti software libero. In quel periodo furono gettate le basi etiche, legali e perfino economiche di questi progetti (cosa probabilmente più importante), mentre questi venivano sviluppati e portati a termine man mano, fino ai giorni nostri. E siccome il nuovo fenomeno aveva bisogno di un nome, fu allora che venne coniato il termine *software libero*.

### 2.2.1. Richard Stallman, GNU, FSF: nasce il movimento del software libero

All'inizio del 1984 Richard Stallman, che all'epoca lavorava nei Laboratori di Intelligenza Artificiale del Massachusetts Institute of Technology (MIT), lasciò il suo lavoro per dedicarsi al progetto GNU. Stallman si considerava un *hacker* del tipo che condivide volentieri i suoi interessi tecnologici e il suo codice. Non gli andava giù il fatto che il suo rifiuto di firmare contratti di esclusività o di non-condivisione lo rendeva un emarginato nel suo stesso settore, e che l'uso di software proprietario nel suo ambiente di lavoro lo rendeva impotente di fronte a situazioni che prima avrebbero potuto essere risolte facilmente.

La sua idea al momento di lasciare il MIT era quella di costruire un sistema software completo, per uso generale, ma completamente libero ("The GNU Project", DiBona *et al.* ) [208]. Il sistema (e il progetto che si sarebbe occupato di realizzarlo) fu chiamato GNU (un acronimo ricorsivo: "GNU's not Unix", ossia "GNU non è Unix"). Benché fin dall'inizio il progetto GNU abbia incluso software nel suo sistema che era già disponibile (come TeX o, più avanti, il sistema X Window), c'era ancora molto da costruire. Richard Stallman iniziò scrivendo un compilatore in C (GCC) e un editore (Emacs), entrambi i quali sono usati (e molto apprezzati) ancora oggi.

Fin dall'inizio del progetto GNU, Richard Stallman si preoccupava di quali libertà avrebbero avuto gli utenti del software. Voleva che i diritti di modifica, redistribuzione, ecc. rimanessero validi non solo per coloro che ricevevano programmi direttamente dal progetto, ma anche per coloro che li avrebbero ricevuti dopo un qualsiasi numero di redistribuzioni e (potenziali) modifiche. Per questo tracciò una prima versione della licenza GPL, probabilmente la prima licenza software progettata esplicitamente allo scopo di garantire che un programma rimanesse libero in questo modo. Richard Stallman chiamò il meccanismo generico usato da questo tipo di licenze GPL per ottenere tali garanzie *copyleft*, che continua ad essere il nome di una numerosa famiglia di licenze di software libero (Free Software Foundation, GNU General Public Licence, versione 2, giugno 1991) [118].

Richard Stallman fondò anche la Fondazione Software Libero (Free Software Foundation, FSF) per ottenere fondi, che usa per sviluppare e proteggere software libero, e stabilì i suoi principi etici con il "Manifesto di GNU" ("The GNU Manifesto", Free Software Foundation, 1985) [117] e "Perché il software non dovrebbe avere proprietari" ("Why software should not have owners", Richard Stallman, 1998) [207].

Da un punto di vista tecnico, il progetto GNU fu concepito come un compito altamente strutturato, con obiettivi molto precisi. Il metodo più comune si basava su gruppi relativamente ridotti di persone (in genere volontari) che sviluppavano uno degli strumenti che poi si sarebbe inserito perfettamente nel puzzle completo (il sistema GNU). La modularità di Unix, a cui questo progetto si ispirava, corrispondeva perfettamente a questa idea. Il metodo di lavoro in genere implicava l'uso di Internet, ma siccome a quell'epoca la rete non era installata in modo così pervasivo, la Fondazione Software Libero vendeva anche nastri su cui registrare le applicazioni, il che significa che fu anche una delle prime organizzazioni ad ottenere compensi in denaro (benché in modo molto limitato) per la creazione di software libero.

Nei primi anni Novanta, circa sei anni dopo la fondazione del progetto, GNU era arrivato molto vicino ad avere un sistema completo, simile a Unix. Tuttavia, a quel punto non aveva ancora prodotto una delle parti più essenziali: la parte centrale del sistema (nota anche come *kernel*, o nucleo), ossia la parte del sistema operativo che si collega con l'hardware, lo astrae e permette alle applicazioni di condividere risorse e, in sostanza, di funzionare). Tuttavia, il software GNU era molto popolare tra gli utenti di molte varianti diverse di Unix, all'epoca il sistema operativo più comunemente usato nelle aziende. Inoltre il progetto GNU era riuscito a farsi conoscere relativamente bene tra i professionisti delle tecnologie informatiche, specialmente tra coloro che lavoravano nelle università. In quel periodo i suoi prodotti godevano già di una meritata fama di stabilità e buona qualità.

### 2.2.2. Il CSRG di Berkeley

Dal 1973 il CSRG (Computer Science Research Group) dell'Università della California di Berkeley era uno dei centri in cui era stata realizzata la maggior parte degli sviluppi legati a Unix, specialmente durante gli anni 1979 e 1980. Non solo le applicazioni erano state adattate (portate) e ne erano state costruite di nuove che funzionassero su Unix, ma erano stati fatti anche importanti miglioramenti al kernel ed erano state aggiunte un sacco di funzionalità. Ad esempio, durante gli anni Ottanta, molti contratti DARPA (sotto il Ministero della Difesa degli Stati Uniti) finanziarono l'implementazione del TCP/IP, che è stato considerato fino ad oggi il punto di riferimento per i protocolli che fanno funzionare Internet (collegando, nel processo, lo sviluppo di Internet e l'espansione di postazioni di lavoro Unix). Molte aziende usarono gli sviluppi del CSRG come basi per le loro versioni di Unix, dando origine a sistemi

all'epoca molto noti, come SunOS (Sun Microsystems) o Ultrix (Digital Equipment). In questo modo Berkeley divenne una delle due fonti fondamentali di Unix, insieme a quella "ufficiale": AT&T.

Per usare tutto il codice prodotto dal CSRG (e il codice dei collaboratori della comunità Unix che in qualche modo essi coordinavano), era necessario avere la licenza Unix di AT&T, che stava diventando sempre più difficile (e costosa) da ottenere, specialmente se si richiedeva accesso al codice sorgente del sistema. In parte per superare questo problema, nel giugno 1989 il CSRG liberò la parte di Unix associata al TCP/IP (l'implementazione dei protocolli nel kernel e le utilities), che non includevano codice di AT&T. Questo rilascio venne chiamato *Networking Release 1* (Net-1). La licenza con cui venne distribuito era la famosa *licenza BSD* che, a parte alcuni problemi con le clausole sugli obblighi di rivelazione, è sempre stata considerata un esempio di licenza libera minimalista (che, oltre a permettere la libera redistribuzione, permette anche l'incorporazione in prodotti proprietari). Inoltre il CSRG testò un nuovo modello economico (che la Fondazione Software Libero stava già sperimentando con successo): vendette nastri con la sua distribuzione per 1000 dollari americani. Nonostante il fatto che chiunque potesse redistribuire il contenuto dei nastri senza alcun problema (perché la licenza lo permetteva), il CSRG vendette nastri a migliaia di organizzazioni, ottenendo in questo modo fondi con i quali continuare a sviluppare.

Avendo osservato il successo della distribuzione Net-1, Keith Bostic propose di riscrivere tutto il codice che ancora rimaneva dell'iniziale Unix di AT&T. Nonostante lo scetticismo di alcuni membri del CSRG, pubblicò un annuncio chiedendo aiuto nell'impresa, e poco a poco le utilities (riscritte sulla base di specifiche) iniziarono ad arrivare a Berkeley. Nel frattempo lo stesso processo veniva ripetuto per il kernel, al punto che la maggior parte del codice che non era stato prodotto da Berkeley o da collaboratori volontari fu riscritto indipendentemente. Nel giugno 1991, dopo aver ottenuto il permesso da chi governava l'Università di Berkeley, venne distribuita la *Networking Release 2* (Net-2), con quasi tutto il codice del kernel e tutte le utilities di un sistema Unix completo. L'insieme venne ancora una volta distribuito sotto la licenza BSD e migliaia di nastri vennero venduti al costo di 1000 dollari americani ciascuno.

Appena sei mesi dopo il rilascio di Net-2, Bill Jolitz scrisse il codice che mancava per far funzionare il kernel sull'architettura i386, rilasciando 386BSD, che fu distribuito su Internet. Sulla base di quel codice in seguito emersero, uno dopo l'altro, tutti i sistemi della famiglia \*BSD: prima apparve NetBSD, una compilazione degli aggiornamenti che erano stati inviati attraverso la rete per migliorare 386BSD; più avanti emerse FreeBSD, in pratica un tentativo di supportare l'architettura i386; molti anni dopo fu formato il progetto OpenBSD, con attenzione particolare alla sicurezza. E ci fu anche una versione proprietaria basata su Net-2 (sebbene fosse certamente originale, in quanto offriva

ai suoi clienti tutto il codice sorgente incluso nel pacchetto base), che fu realizzata indipendentemente dall'ormai estinta azienda BSDI (Berkeley Software Design Inc.).

In parte come reazione alla distribuzione prodotta da BSDI, Unix System Laboratories (USL), la sussidiaria di AT&T che possedeva i diritti della licenza Unix, cercò di fare causa prima a BSDI, poi all'Università della California. L'accusa era che l'azienda aveva distribuito materiale di proprietà intellettuale di USL senza permesso. In seguito a diverse manovre legali (tra cui una contro-citazione dell'Università della California nei confronti di USL), Novell comprò i diritti di Unix da USL e nel gennaio 1994 raggiunse un accordo fuori dal tribunale con l'Università della California. In seguito a questo accordo il CSRG distribuì la versione 4.4BSD-Lite, che venne presto usata da tutti i progetti della famiglia \*BSD. Poco tempo dopo (in seguito all'uscita della versione 4.4BSD-Lite Release 2), il CSRG scomparve. A quel punto alcuni temevano che sarebbe stata la fine dei sistemi \*BSD, ma il tempo ha mostrato che sono ancora vivi e vegeti, sotto una nuova forma di gestione più tipica dei progetti di software libero. Anche nella prima decade dell'anno 2000 i progetti gestiti dalla famiglia \*BSD sono tra i più vecchi e consolidati nel mondo del software libero.

### **Bibliografia**

La storia di Unix BSD illustra un modo particolare di sviluppare software durante gli anni Settanta e Ottanta. Chi fosse interessato all'argomento può divertirsi a leggere "Vent'anni di Berkeley Unix" (Twenty years of Berkeley Unix, di Marshall Kirk McKusick, 1999) [170], che ne segue l'evoluzione dal nastro che Bob Fabry portò a Berkeley con l'idea di far funzionare una delle prime versioni del codice di Thompson e Ritchie su un PDP-11 (comprato in società dalle facoltà di informatica, statistica e matematica), fino alle cause intentate da AT&T e alle ultime uscite di codice che diedero origine alla famiglia di sistemi operativi liberi \*BSD.

### **2.2.3. Le origini di Internet**

Fin dalla sua creazione negli anni Settanta, Internet è stato strettamente associato al software libero. Da una parte, fin dall'inizio la comunità di sviluppatori che costruirono Internet aveva diversi principi molto chiari, che in seguito sarebbero diventati classici nel mondo del software libero; ad esempio, l'importanza di permettere agli utenti di correggere gli errori o condividere codice. L'importanza di BSD Unix nello sviluppo di Internet (avendo fornito negli anni Ottanta la più diffusa implementazione dei protocolli TCP/IP) facilitò il trasferimento di molte abitudini e modi di fare da una comunità - gli sviluppatori che ruotavano intorno al CSRG - a un'altra comunità - gli sviluppatori che stavano costruendo quello che all'epoca si chiamava NSFNet e che sarebbe poi diventato Internet - e viceversa. Molte delle applicazioni di base per lo sviluppo di Internet, come Sendmail (server di posta) o BIND (implementazione del servizio di nomi), erano libere e risultavano in gran parte dalla collaborazione tra queste due comunità.



Infine, verso la fine degli anni Ottanta e negli anni Novanta, la comunità del software libero fu una delle prime ad esplorare in profondità le nuove possibilità offerte da Internet per la collaborazione tra gruppi geograficamente dispersi. Si deve in gran parte a questa esplorazione la mera esistenza della comunità BSD, della Fondazione Software Libero o dello sviluppo di GNU/Linux.

Uno degli aspetti più interessanti dello sviluppo di Internet, dal punto di vista del software libero, era la gestione completamente aperta dei suoi documenti e delle sue regole. Sebbene possa sembrare normale al giorno d'oggi (perché è ormai un'abitudine, nel IETF o nel World Wide Web Consortium), all'epoca il libero accesso a tutte le sue specifiche e ai suoi documenti di progettazione, comprese le norme che definiscono i protocolli, era qualcosa di rivoluzionario e fondamentale per il suo sviluppo. In *Cittadini della rete. La storia e l'impatto di usenet e di Internet*. (Netizens. On the history and impact of Usenet and the Internet. Pagina 106) [139] si legge:

"Questo processo aperto incoraggiava e portava allo scambio di informazioni. Lo sviluppo tecnico ha successo solo quando l'informazione può circolare liberamente e facilmente tra i soggetti coinvolti. Incoraggiare la partecipazione è il principio chiave che ha reso possibile lo sviluppo della Rete."

Si capisce bene perché questo paragrafo verrebbe quasi sicuramente sottoscritto da qualsiasi sviluppatore in riferimento al progetto di software libero in cui si trovi coinvolto.

In un'altra citazione, da "Lo sviluppo dello scambio di pacchetti" (The evolution of packet switching, pagina 267) [195] si legge:

"Siccome ARPANET era un progetto pubblico che connetteva diverse grandi università e istituti di ricerca, i dettagli dell'implementazione e delle prestazioni furono pubblicati con ampia diffusione."

Ovviamente, questo è ciò che tende ad accadere nei progetti di software libero, in cui tutta l'informazione relativa a un progetto (e non solo alla sua implementazione) è normalmente pubblica.

In questa atmosfera e prima che Internet, nel pieno degli anni Novanta, diventasse un vero e proprio "business", la comunità degli utenti e la sua relazione con gli sviluppatori era essenziale. In quel periodo molte organizzazioni impararono a fidarsi non di un singolo fornitore di servizi di comunicazione dati, ma piuttosto di una complessa combinazione di aziende fornitrici di servizi, produttori di materiale, sviluppatori professionisti e volontari, eccetera. Le migliori implementazioni di molti programmi non erano quelle incluse nel sistema operativo acquistato insieme all'hardware, ma implementazioni libere che le avrebbero rapidamente sostituite. Gli sviluppi più innovativi non erano il risultato dei progetti di ricerca di grandi aziende, ma il prodotto di studenti o professionisti che mettevano alla prova idee e raccoglievano i commenti inviati loro dai vari utenti dei loro programmi liberi.

Come abbiamo già accennato, Internet offriva al software libero anche gli strumenti di base per la collaborazione a lunga distanza. La posta elettronica, i newsgroups, i servizi di FTP anonimo (che furono i primi grandi depositi del software libero) e, più avanti, i sistemi di sviluppo integrati basati su web, sono stati fondamentali (e indispensabili) per lo sviluppo della comunità del software libero come la conosciamo oggi, e in particolare per il funzionamento della stragrande maggioranza dei progetti di software libero. Fin dall'inizio progetti come GNU o BSD fecero ampio e intenso uso di tutti questi meccanismi, sviluppando, mentre li usavano, nuovi strumenti e sistemi che a loro volta migliorarono Internet.

#### 2.2.4. Altri progetti

Durante gli anni Ottanta molti altri importanti progetti di software libero videro la luce del giorno. Vorremmo sottolineare per la sua importanza e rilevanza futura X Window (windows system per sistemi tipo Unix), sviluppato al MIT, uno dei primi esempi di finanziamenti su larga scala per un progetto libero, finanziato da un consorzio di aziende. Vale la pena menzionare anche Ghostscript, un sistema di gestione documenti PostScript sviluppato da una compagnia chiamata Aladdin Software, che fu uno dei primi casi di ricerca di un modello di business basato sulla produzione di software libero.

Verso al fine degli anni Ottanta esisteva già un'intera costellazione di piccoli (e non tanto piccoli) progetti di software libero in corso. Tutti questi, insieme con i grandi progetti menzionati finora, stabilirono la base dei primi sistemi liberi completi, che apparvero agli inizi degli anni Novanta.

#### 2.3. Tutti i sistemi vanno

Verso il 1990 la maggior parte delle componenti di un sistema completo era disponibile come software libero. Da una parte il progetto GNU e le distribuzioni BSD avevano completato la maggior parte delle applicazioni che formano un sistema operativo. Dall'altra, progetti come X Window o GNU stesso avevano costruito da ambienti con finestre a compilatori, che erano spesso tra i migliori nella loro categoria (ad esempio, molti amministratori di sistemi SunOS o Ultrix sostituivano le applicazioni proprietarie dei loro sistemi con le versioni libere di GNU o BSD per i loro utenti). Per avere un sistema completo costruito esclusivamente con software libero, mancava soltanto un componente: il kernel. A colmare questa lacuna furono due sforzi separati e indipendenti: 386BSD e Linux.

##### 2.3.1. La ricerca di un kernel

Verso la fine degli anni Ottanta e l'inizio dei Novanta, il progetto GNU aveva una gamma di base di servizi e strumenti che rendevano possibile avere un sistema operativo completo. Anche allora molte applicazioni libere, tra cui il caso particolarmente interessante di X Window, erano le migliori nella loro

#### Bibliografia

I lettori interessati all'evoluzione di Internet possono consultare "Breve storia di Internet" (A brief history of the Internet, scritta da molti dei suoi protagonisti principali, pubblicata da ACM, 1997) [166].

categoria (servizi Unix, compilatori...). Tuttavia per completare il puzzle mancava un pezzo vitale: il kernel del sistema operativo. Il progetto GNU stava cercando quel pezzo mancante con un progetto noto come Hurd, che intendeva costruire un kernel utilizzando le tecnologie moderne.

### 2.3.2. La famiglia \*BSD

Praticamente in contemporanea, anche la comunità BSD era avviata sulla strada verso un kernel libero. Alla distribuzione Net-2 mancavano solo sei file per completarlo (il resto era già stato costruito dal CSRG o dai suoi collaboratori). All'inizio del 1992 Bill Jolitz completò quei file e distribuì 386BSD, un sistema che funzionava sull'architettura i386 e che nel tempo avrebbe dato origine ai progetti NetBSD, FreeBSD and OpenBSD. Nei mesi successivi i progressi furono rapidi e alla fine dell'anno era sufficientemente stabile per poter essere usato in ambienti di produzione non critici, tra cui ad esempio un ambiente a finestre grazie al progetto XFree (che aveva fornito X Window per l'architettura i386) o un compilatore di alta qualità, GCC. Sebbene vi fossero componenti che usavano altre licenze (come quelli dei progetti GNU, che usavano la licenza GPL), la maggior parte del sistema fu distribuita sotto la licenza BSD.

#### Bibliografia

Alcuni episodi di questo periodo illustrano la capacità dei modelli di sviluppo del software libero. C'è il noto caso di Linus Torvalds, che sviluppò Linux mentre frequentava il secondo anno all'Università di Helsinki. Ma questo non è l'unico esempio di uno studente che si fece strada grazie ai programmi liberi da lui sviluppati. Ad esempio, il tedesco Thomas Roel portò X11R4 (una versione del sistema X Window) su un PC basato su un 386. Questa impresa lo condusse a lavorare alla Dell, e in seguito a diventare il fondatore dei progetti X386 e XFree, che furono fondamentali nel fornire rapidamente a GNU/Linux e \*BSDs un ambiente a finestre. Per approfondire la storia di XFree e il ruolo di Roel in essa, leggere "La storia di xFree86" (The history of xFree86, *Linux Magazine*, Dicembre 1991) [135].

Quindi arrivò la causa intentata da USL, che fece temere a molti potenziali utenti di essere citati in tribunale a loro volta se l'Università della California avesse perso la causa o se, semplicemente, il progetto si fosse bloccato. Forse per questa ragione, più avanti la base di installazioni di GNU/Linux fu molto più vasta di tutte le \*BSD messe insieme. Ma non possiamo saperlo con certezza.

### 2.3.3. Entra in scena GNU/Linux

Nel luglio 1991 Linus Torvalds (uno studente finlandese di 21 anni) inviò il suo primo messaggio in cui accennava al suo (all'epoca) progetto di costruire un sistema libero simile a Minix. In settembre rilasciò la primissima versione (0.01), e nuove versioni sarebbero apparse ogni poche settimane. Nel marzo 1994 apparve la versione 1.0, la prima ad essere chiamata *stabile*, benché il kernel costruito da Linus fosse già stato usabile per diversi mesi. In questo periodo ci furono letteralmente centinaia di sviluppatori che passarono a Linux,

integrando tutto il software GNU attorno ad esso, così come XFree e molti altri programmi liberi. A differenza di \*BSDs, il kernel Linux e un gran numero di componenti integrate attorno ad esso vennero distribuiti con la licenza GPL.

### Bibliografia

La storia di Linux è probabilmente una delle più interessanti (e meglio conosciute) nel mondo del software libero. Si possono trovare molti link a informazioni su di essa sulle pagine che celebrano il decimo anniversario del suo annuncio, anche se probabilmente uno dei più interessanti è "La storia di Linux" (History of Linux), di Raghib Hasan [138]. Per curiosità, è possibile consultare la discussione in cui Linus Torvalds annunciò che stava iniziando a creare quello che in seguito sarebbe diventato Linux (nel newsgroup comp.os.minix) all'indirizzo <http://groups.google.com/groups?th=d161e94858c4c0b9> Qui spiega come abbia lavorato al suo kernel fin da aprile e come avesse già portato alcuni strumenti del progetto GNU su di esso (menzionando in particolare Bash e GCC).

Dei molti sviluppi emersi attorno a Linux, uno dei più interessanti è il concetto di *distribuzione*<sup>1</sup>. Le prime distribuzioni apparvero presto, nel 1992 (Interim Linux, della University of Manchester; TAMU, di Texas A&M e la più famosa, SLS, che più avanti diede origine a Slackware, ancora in distribuzione nel primo decennio del 2000), introducendo della concorrenza nel mondo dei sistemi assemblati attorno a Linux. Ogni distribuzione punta ad offrire un GNU/Linux pronto all'uso, e partendo dallo stesso software di base deve competere apportando miglioramenti considerati importanti dalla propria base di utenza. Oltre a fornire pacchetti pre-compilati, pronti all'uso, le distribuzioni tendono anche ad offrire i loro strumenti per gestire le funzioni di selezione, installazione, sostituzione e disinstallazione di questi pacchetti, in aggiunta all'installazione iniziale sul computer e alla gestione e amministrazione del sistema operativo.

<sup>(1)</sup>Questo concetto è spiegato in dettaglio nella pagina corrispondente di Wikipedia, [www.wikipedia.org/wiki/Linux\\_distribution](http://www.wikipedia.org/wiki/Linux_distribution)

Nel tempo le distribuzioni si sono succedute l'una all'altra, man mano che alcune di esse divenivano le più popolari. Tra tutte, desideriamo ricordare le seguenti:

- 1) Debian, sviluppata da una comunità di volontari.
- 2) Red Hat Linux, che è stata sviluppata inizialmente all'interno dell'azienda Red Hat, ma che ha poi adottato un approccio più basato sul modello comunitario, dando origine a Fedora Core.
- 3) Suse, che diede origine a OpenSUSE, seguendo un'evoluzione simile a quella di Red Hat.
- 4) Mandriva (successore di Mandrake Linux e Conectiva).
- 5) Ubuntu, derivata da Debian e prodotta sulla base di Debian dall'azienda Canonical.

## 2.4. Un periodo di maturazione

A metà del primo decennio del 2000, GNU/Linux, OpenOffice.org o Firefox erano presenti sui media piuttosto di frequente. La stragrande maggioranza delle aziende usa software libero per almeno alcuni dei suoi processi di tecnologia dell'informazione. E' raro essere uno studente di informatica e non usare grandi quantità di software libero. Il software libero non è più una nota a piè di pagina nella storia delle tecnologie informatiche ed è diventato estremamente importante per l'intero settore. Le aziende di prodotti informatici, quelle nel settore secondario che fanno un uso intensivo di software (anche se la loro attività principale è diversa) e le amministrazioni pubbliche stanno iniziando a considerarlo un elemento strategico. E, lentamente ma inesorabilmente, si sta facendo strada anche tra le mura domestiche. In generale, stiamo entrando in un periodo di maturazione.

E, alla fine di tutto, inizia ad emergere un'importante domanda, che riassume in qualche modo ciò che sta succedendo: "Siamo di fronte ad un nuovo modello di industria del software?". Forse potrebbe ancora succedere che il software libero si riveli niente più che una moda passeggera, da ricordare un giorno con nostalgia. Ma potrebbe anche trattarsi (e l'ipotesi sembra sempre più probabile) di un nuovo modello che è qui per rimanere, e forse per cambiare radicalmente una delle industrie più giovani, ma anche più influenti dei nostri tempi..

### 2.4.1. Fine degli anni Novanta

A metà degli anni Novanta il software libero offriva già ambienti completi (distribuzioni di GNU/Linux, sistemi \*BSD...) che supportavano il lavoro quotidiano di molte persone, specialmente sviluppatori di software. C'erano ancora molti compiti da completare (il primo dei quali era avere interfacce grafiche migliori, in un'epoca in cui Windows 95 era considerato lo standard), ma c'erano già diverse migliaia di persone al mondo che usavano esclusivamente software libero per il loro lavoro quotidiano. I progetti seguenti vennero annunciati in rapida successione e il software libero si imbarcò per il suo lungo viaggio verso le aziende, i media e l'opinione pubblica in generale.

Questo periodo è associato anche al decollo di Internet come rete per tutti, un decollo in molti casi guidato dai programmi liberi (specialmente nelle sue infrastrutture). L'arrivo della rete nelle case di milioni di utenti finali consolidò questa situazione, almeno in termini di server: i web server più diffusi (HTTP) sono sempre stati liberi (prima il server NCSA, seguito da Apache).

Forse l'inizio del cammino del software libero fino al suo rilascio completo al pubblico è descritto al meglio nel famoso saggio di Eric Raymond, "La cattedrale e il bazar" (The cathedral and the bazaar, by Eric S. Raymond, 2001) [192]. Benché molto di quanto vi viene descritto fosse già ben noto alla comunità di sviluppatori di software libero, raccontarlo in un articolo e diffonderlo

ampiamente l'ha reso uno strumento influente per promuovere il concetto di *software libero* come un meccanismo di sviluppo alternativo a quello in uso nell'industria del software. Un altro articolo importante di questo periodo fu "Metter su bottega. Il business del software open source" (Setting up shop. The Business of open source software) [141], di Frank Hecker, che per la prima volta descrisse i possibili modelli di business del software libero, e che fu scritto per influenzare la decisione di rilasciare il codice di Netscape Navigator.

Se l'articolo di Raymond fu un ottimo strumento per promuovere alcune delle caratteristiche fondamentali del software libero, il rilascio del codice di Netscape Navigator fu il primo caso in cui un'azienda relativamente grande, in un settore molto innovativo (l'allora nascente industria del web), abbia deciso di rilasciare uno dei suoi prodotti come software libero. A quell'epoca Netscape Navigator stava perdendo la battaglia dei browser per il web contro il prodotto di Microsoft (Internet Explorer), in parte a causa della strategia di Microsoft di combinarlo con il suo sistema operativo. Molti ritengono che Netscape abbia fatto l'unica cosa che poteva fare: cercare di cambiare le regole per riuscire a competere con un gigante. E da questo cambiamento di regole (nel tentativo di competere con un modello da software libero) nacque il progetto Mozilla. Questo progetto, che aveva i suoi problemi, ha condotto diversi anni dopo ad un browser che, se non ha recuperato l'enorme settore di mercato che aveva Netscape ai suoi tempi, sembra avere dal punto di vista tecnologico una qualità perlomeno equivalente a quella dei suoi concorrenti.

In ogni caso, indipendentemente dal suo successo futuro, l'annuncio di Netscape che avrebbe rilasciato il codice del suo browser ebbe un forte impatto sull'industria del software. Molte aziende iniziarono a reputare il software libero degno di considerazione.

Anche i mercati finanziari iniziarono a interessarsi del software libero. Nell'euforia del boom delle aziende "dotcom", molte aziende di software libero divennero bersaglio di investitori. Forse il caso più noto è quello di Red Hat, una delle prime aziende a riconoscere che vendere CDs con sistemi GNU/Linux pronti all'uso poteva essere un potenziale modello di business. Red Hat iniziò a distribuire il suo Red Hat Linux, enfatizzando enormemente (almeno rispetto agli standard di allora) la facilità d'uso e di manutenzione del sistema, anche da parte di persone prive di specifiche competenze informatiche. Nel tempo si diversificò, pur restando nell'orbita del software libero, e nel settembre 1998 annunciò che Intel e Netscape avevano investito su di esso. "Se va bene per Intel e Netscape, va bene anche per noi", devono aver pensato a quel punto molti investitori. Quando Red Hat venne quotata in borsa nell'estate 1999, tutte le sue azioni in vendita per la prima volta vennero acquistate e presto il loro valore salì in maniera spettacolare. Era la prima volta che un'azienda riusciva ad ottenere finanziamenti dal mercato azionario con un modello basato sul software libero. Ma non fu l'unica: in seguito altre, come VA Linux o Andover.net (che più tardi venne acquisita da VA Linux) fecero lo stesso.

**Nota**

Red Hat fornisce una lista delle tappe principali della storia della sua azienda all'indirizzo: <http://fedora.redhat.com/about/history/>.

Durante questo periodo nacquero anche molte altre aziende con modelli di business basati sul software libero. Anche senza andare in borsa e ottenere risultati incredibili in termini di crescita del valore delle azioni, furono comunque molto importanti per lo sviluppo del software libero. Per esempio, comparvero molte aziende che iniziarono a distribuire le loro versioni di GNU/Linux, come SuSE (Germania), Conectiva (Brasile) or Mandrake (Francia), che in seguito si sarebbe unita alla precedente per formare Mandriva. Altre offrivano servizi ad aziende che avevano bisogno di manutenzione o di adattare prodotti di software libero: LinuxCare (USA), Alcove (Francia), ID Pro (Germania); e molte altre.

Nel frattempo i giganti del settore iniziarono a prendere una posizione nei confronti del software libero. Alcune aziende, come IBM, lo incorporarono direttamente nella propria strategia. Altre, come Sun Microsystems, avevano con esso una strana relazione: a volte lo sostenevano, a volte si mostravano indifferenti e a volte lo attaccavano. La maggior parte (tra cui Apple, Oracle, HP, SGI, ecc.) esplorarono il modello del software libero con varie strategie, dal "liberare" selettivamente alcuni software al portare direttamente i propri prodotti su GNU/Linux. Tra questi due estremi c'erano diverse altre linee d'azione, come l'uso più o meno intensivo di software libero nei loro prodotti (come nel caso di Mac OS X) o l'esplorazione di modelli di business basati sulla manutenzione di prodotti liberi.

Dal punto di vista tecnico, l'evento più notevole di questo periodo fu probabilmente la comparsa di due ambiziosi progetti, pensati per portare il software libero nell'ambiente *desktop* per utenti non esperti di tecnologie: KDE e GNOME. Semplificando, l'obiettivo finale era quello di non dover usare righe testuali di comando per interagire con GNU/Linux o \*BSD, o con i programmi che funzionavano su quegli ambienti.

KDE fu annunciato nell'ottobre 1996. Usando le librerie grafiche Qt (all'epoca un prodotto proprietario che apparteneva all'azienda Trolltech, ma che si poteva usare gratis su GNU/Linux<sup>2</sup>), iniziò la costruzione di un insieme di applicazioni desktop che lavorassero in maniera integrata e avessero un aspetto uniforme. Nel luglio 1998 fu rilasciata la versione 1.0 del K Desktop Environment, presto seguita da nuove versioni sempre più complete e mature. Le distribuzioni GNU/Linux presto incorporarono KDE come desktop per i loro utenti (o perlomeno come uno degli ambienti di desktop tra cui gli utenti potevano scegliere).

Prevalentemente come reazione alla dipendenza di KDE dalla libreria proprietaria Qt, nell'agosto 1997 fu annunciato il progetto GNOME (Miguel de Icaza, "The story of the GNOME Project") [101], con obiettivi e caratteristiche simi-

<sup>(2)</sup>In seguito Qt iniziò ad essere distribuito sotto la licenza libera QPL (Licenza Pubblica Qt), non compatibile con GPL, il che causava alcuni problemi, perché la maggior parte di KDE veniva distribuito sotto la licenza GPL. Col tempo, Trolltech decise infine di distribuire Qt sotto la licenza GPL, mettendo fine a questi problemi.

li a quelli di KDE, ma dichiarando esplicitamente l'obiettivo che tutti i suoi componenti fossero liberi. Nel marzo 1999 fu rilasciato GNOME 1.0, che col tempo sarebbe anch'esso migliorato e diventato più stabile. Da quel momento la maggior parte delle distribuzioni di sistemi operativi liberi (e molti di quelli proprietari derivati da Unix) offrirono tra le loro opzioni il desktop GNOME o KDE e le applicazioni di entrambi gli ambienti.

Nel frattempo i principali progetti di software libero in corso rimanevano in buona salute, mentre nuovi progetti emergevano quasi ogni giorno. In molti mercati di nicchia si trovò che il software libero era la soluzione migliore (riconosciuta quasi in tutto il mondo). Ad esempio, dalla sua comparsa nell'aprile 1995, Apache ha mantenuto la più ampia quota di mercato per i server web; XFree86, il progetto libero che sviluppa X Window, è di gran lunga la versione più diffusa di X Window (e quindi il sistema a finestre più esteso per sistemi di tipo Unix); GCC è stato riconosciuto come il compilatore C più portatile e di migliore qualità; GNAT, il sistema di compilazione per Ada 95, ha conquistato la parte migliore del mercato dei compilatori Ada in pochi anni e così via.

Nel 1998 fu fondata l'Iniziativa Open Source (OSI), che decise di adottare il termine *software open source* come marchio per introdurre software libero nel mondo degli affari, evitando l'ambiguità del termine *libero* ("free", che in inglese significa sia libero da usare che gratuito). Questa decisione fece scoppiare uno dei più feroci dibattiti nel mondo del software libero (che continua tuttora), dal momento che la Fondazione Software Libero ed altri ritenevano molto più appropriato parlare di *software libero* (Richard Stallman, "Perché *software libero* è meglio di *open source*" - *Why free software is better than open source*, 1998) [206]. In ogni caso, l'OSI fece una grande campagna promozionale per il suo nuovo marchio, che è stato adottato da molti come il modo preferibile per parlare di software libero, specialmente nel mondo anglofono. Per definire il *software open source*, l'OSI usò una definizione derivata da quella usata dal progetto Debian per definire il software libero ("Linee guida Debian per il software libero", [http://www.debian.org/social\\_contract.html#guidelines](http://www.debian.org/social_contract.html#guidelines)) [104], che allo stesso tempo riflette piuttosto fedelmente l'idea della Fondazione Software Libero al riguardo ("Free software definition", <http://www.gnu.org/philosophy/free-sw.html>) [120], tanto che dal punto di vista pratico quasi tutti i programmi considerati software libero si possono considerare anche *open source* e viceversa. Tuttavia le comunità del software libero e del software open source (o almeno coloro che si identificano con esse) possono essere molto diverse.

#### **2.4.2. Il decennio del 2000**

Nei primi anni del 2000 il software libero era già un concorrente rispettabile nel segmento dei server e iniziava ad essere pronto per il desktop. Sistemi come GNOME, KDE, OpenOffice.org e Mozilla Firefox possono essere usati dagli utenti nelle loro case e bastano a soddisfare i bisogni di molte aziende, almeno per quanto riguarda le applicazioni da ufficio. I sistemi liberi (e spe-



cialmente quelli basati su Linux) sono facili da installare, e la complessità di mantenerli in funzione e aggiornarli è paragonabile a quella di altri sistemi proprietari.

Attualmente, ogni azienda nell'industria del software ha una strategia rispetto al software libero. La maggior parte delle multinazionali più importanti (IBM, HP, Sun, Novell, Apple, Oracle...) incorpora software libero in proporzioni più o meno grandi. Ad un estremo troviamo aziende come Oracle, che reagiscono semplicemente portando i loro prodotti su GNU/Linux. All'estremo opposto troviamo BM, che ha la strategia più decisiva e ha fatto le campagne pubblicitarie più grandi su GNU/Linux. Tra le aziende alla guida del mercato dell'informatica, solo Microsoft si è posizionata in aperta opposizione al software libero, in particolare al software distribuito sotto la licenza GPL.

Per quanto riguarda il mondo del software libero, nonostante i dibattiti che talvolta agitano la comunità, la sua crescita è massiccia. Ogni giorno si aggiungono nuovi sviluppatori, nuovi progetti attivi di software libero, nuovi utenti, eccetera. Ogni giorno che passa, il software libero si allontana dai margini e diventa una forza con cui si deve sempre più fare i conti.

Alla luce di questo, stanno emergendo nuove discipline che studiano specificamente il software libero, come l'ingegneria del software libero. Grazie alla ricerca, poco per volta stiamo iniziando a capire come il software libero opera nei suoi vari aspetti: modelli di sviluppo, modelli di business, meccanismi di coordinamento, gestione di progetti liberi, motivazioni degli sviluppatori, eccetera.

In questi anni stiamo anche iniziando a vedere i primi effetti dell'*offshoring* permesso dallo sviluppo del software libero: nazioni considerate "periferiche" stanno partecipando attivamente al mondo del software libero. Ad esempio, il numero di sviluppatori messicani o spagnoli (due nazioni con una limitata tradizione di industria del software) in progetti come GNOME è significativo (Lancashire, "Codice, cultura e denaro: l'altruismo che va svanendo nello sviluppo di open source" - Code, culture and cash: the fading altruism of open source development, 2001) [164]. Ed è sempre più interessante il ruolo del Brasile, con i suoi numerosi sviluppatori ed esperti in tecnologie del software libero, e il deciso supporto delle amministrazioni pubbliche. gnu-LinEx è un caso che merita attenzione speciale, in quanto esempio di come una regione con pochissima tradizione di sviluppo software possa cercare di cambiare la situazione attraverso una strategia aggressiva di installazione di software libero.

Dalla prospettiva di chi deve prendere decisioni quando si tratta di implementare soluzioni software, vorremmo sottolineare il fatto che in certi mercati (ad esempio i servizi internet o le applicazioni da ufficio), il software libero è una scelta naturale che non può essere trascurata quando si studia quale sistema usare.

Sul fronte negativo, in questi anni si è visto come l'ambiente legislativo in cui opera il software libero stia cambiando rapidamente in tutto il mondo. Da una parte vengono adottati sempre più brevetti di software (brevetti di programmazione) in sempre più nazioni. Dall'altra parte, le nuove leggi sul copyright rendono difficile o impossibile sviluppare applicazioni libere in certe sfere, la più nota delle quali è quella dei lettori DVD (a causa dell'algoritmo CSS di sfocamento delle immagini usato da questa tecnologia).

### **gnuLinEx**

All'inizio del 2002 il governo regionale dell'Estremadura annunciò pubblicamente il progetto gnuLinEx. L'idea era semplice: promuovere la creazione di una distribuzione basata su GNU/Linux, con l'obiettivo fondamentale di usarla sulle migliaia di computer da installare nelle scuole pubbliche di tutta la regione. L'Estremadura, situata nella Spagna occidentale, al confine col Portogallo, conta circa un milione di abitanti e non si è mai distinta per le sue iniziative tecnologiche. Di fatto la regione è praticamente priva di un'industria del software.

In questo contesto, gnuLinEx ha portato un contributo molto interessante al panorama del software libero su scala globale. Oltre ad essere semplicemente una nuova distribuzione di GNU/Linux basata su Debian (che è comunque un aneddoto degno d'attenzione), e oltre ad avere ottenuto un enorme impatto sui media (per la prima volta l'Estremadura è arrivata sulla prima pagina del *The Washington Post*, ed è stata una delle prime volte anche per un prodotto di software libero), l'aspetto straordinario è il solido (almeno apparentemente) sostegno di una pubblica amministrazione al software libero. Il governo regionale dell'Estremadura ha deciso di tentare un modello differente in merito al software per la didattica, e quindi di estendere questo modello a tutto il software usato nella sua sfera di influenza. Questo l'ha resa la prima pubblica amministrazione di un Paese sviluppato ad aver adottato questo approccio in modo deciso. L'iniziativa del governo regionale ha generato moltissimo di interesse, entro i confini dell'Estremadura e al di fuori di essi: ci sono università che insegnano informatica usando gnuLinEx; sono stati scritti libri di supporto a queste lezioni; si vendono computer con gnuLinEx pre-installato. In generale, si sta cercando di creare un tessuto educativo e aziendale attorno a questo esperimento, per sostenerlo. E l'esperimento è stato esportato. All'inizio del XXI secolo molte comunità autonome in Spagna hanno appoggiato il software libero nella didattica (in un modo o nell'altro) e, in generale, la sua importanza per le pubbliche amministrazioni è ampiamente riconosciuta.

## **Knoppix**

Dalla fine degli anni Novanta esistono distribuzioni di GNU/Linux che sono facili da installare, ma Knoppix, la cui prima versione è comparsa nel 2002, ha probabilmente permesso a questa idea di raggiungere la sua piena espressione. Si tratta di un CD che si carica su quasi tutti i PC, convertendoli (senza dover nemmeno formattare il disco, perché lo si può usare "in tempo reale") in una macchina GNU/Linux completamente funzionale, con una selezione degli strumenti più frequenti. Knoppix combina un buon rilevamento automatico di hardware con una buona scelta di programmi e funzionalità "in tempo reale". Ad esempio, permette una rapida esperienza diretta di che cosa vuol dire lavorare con GNU/Linux. E sta dando origine ad un'intera famiglia di distribuzioni dello stesso tipo, specializzate per i requisiti specifici di un profilo utente.

## **OpenOffice.org**

Nel 1999 Sun Microsystems acquistò un'azienda tedesca chiamata Stardivision, il cui prodotto più apprezzato era StarOffice, una suite di applicazioni da ufficio simili nelle funzionalità all'insieme di strumenti offerti da Microsoft Office. L'anno successivo Sun distribuì la maggior parte del codice di StarOffice sotto una licenza libera (la GPL), creando il progetto OpenOffice.org. Questo progetto rilasciò la versione 1.0 di OpenOffice.org nel maggio 2002. OpenOffice.org è diventato una suite di qualità di applicazioni per ufficio, con funzionalità simili a quelle di qualsiasi altro prodotto per ufficio, e - cosa più importante - è compatibile con i formati di dati di Microsoft Office, con i quali funziona molto bene. Queste caratteristiche l'hanno resa l'applicazione libera di riferimento nel mondo delle suite di strumenti da ufficio.

L'importanza di OpenOffice.org, dal punto di vista di estendere il software libero a un grande numero di utenti, è enorme. Finalmente è possibile passare, quasi senza traumi, dagli ambienti proprietari tipici delle suite da ufficio (senza dubbio l'applicazione più apprezzata nel mondo professionale) ad ambienti totalmente liberi (come GNU/Linux più GNOME e/o KDE più OpenOffice.org). Inoltre la transizione può avvenire in modo molto tranquillo: siccome OpenOffice.org funziona anche su Microsoft Windows, non occorre nemmeno cambiare sistema operativo per poter sperimentare fino in fondo che cosa significhi usare software libero.

## **Mozilla, Firefox e gli altri**

Praticamente dalla sua comparsa nel 1994 fino al 1996, Netscape Navigator fu leader di mercato incontrastato tra i web browser, con quote di mercato che arrivavano all'80%. La situazione iniziò a cambiare quando Microsoft incluse Internet Explorer in Windows 95, facendo gradualmente perdere quote di mercato a Netscape Navigator. All'inizio del 1998 Netscape annunciò che avrebbe distribuito gran parte del codice del suo browser come software libero, cosa che fece nel marzo di quello stesso anno, lanciando il progetto Mozilla.

Per molto tempo il progetto fu circondato da incertezza, o addirittura pessimismo (ad esempio quando il suo leader, Jamie Zawinski, lo abbandonò), perché il tempo passava e dal suo lancio non emergeva alcun prodotto.

Nel gennaio 2000 il progetto rilasciò Mozilla M13, che venne considerata la prima versione relativamente stabile. Nel maggio 2002 fu pubblicata infine la versione 1.0, la prima ufficialmente stabile, oltre quattro anni dopo il rilascio del codice del primo Netscape Navigator.

Mozilla era finalmente divenuto realtà, benché forse troppo tardi, considerando le quote di mercato che Internet Explorer deteneva nel 2002 o 2003 (quando era leader incontrastato, lasciando Mozilla ed altri in posizioni del tutto marginali). Ma, pur avendo richiesto moltissimo tempo, il progetto Mozilla aveva portato dei frutti; non solo quello atteso (il browser Mozilla), ma anche altri "collaterali", come ad esempio Firefox, un altro browser basato sullo stesso motore HTML, che è diventato il prodotto principale e che, dalla sua comparsa nel 2005, sta riuscendo a rosicchiare poco a poco le quote di mercato degli altri browser.

Il progetto Mozilla ha contribuito a colmare una vasta lacuna nel mondo del software libero. Prima della comparsa di Konqueror (il browser del progetto KDE), non esistevano molti browser liberi con interfaccia grafica. Dalla pubblicazione di Mozilla è emersa un'enorme quantità di progetti basati su di esso, che hanno prodotto un gran numero di browser. Allo stesso tempo, la combinazione di Mozilla Firefox e OpenOffice.org permette di usare software libero per le operazioni più comuni, anche in ambiente Windows (entrambi funzionano non solo su GNU/Linux, \*BSD e altri sistemi di tipo Unix, ma anche su Windows). Per la prima volta nella storia del software libero, il passaggio da software proprietario a software libero negli ambienti professionali è stato reso un'operazione semplice: si può cominciare ad usare queste due applicazioni su Windows (per chi lo usa normalmente) senza cambiare sistema operativo e poi, col tempo, eliminare la parte di software non libero e passare su GNU/Linux o FreeBSD.

## Il caso di SCO

All'inizio del 2003 la SCO corporation (in precedenza Caldera Systems e Caldera International) ha fatto causa a IBM, accusandola di aver violato i suoi diritti di proprietà intellettuale. Benché il caso fosse più complesso, l'accusa centrale era che IBM avesse contribuito al kernel di Linux usando codice che apparteneva a SCO. Nel maggio 2007 la questione non era ancora stata risolta ed era stata complicata da ulteriori azioni legali (querela di IBM e Red Hat contro SCO, di SCO contro AutoZone e DaimlerChrysler, due grossi utenti di tecnologia) e dalle campagne di SCO, che minacciavano di perseguire legalmente le grosse aziende che usavano Linux, ecc.

### Bibliografia

In "Netscape Navigator", di Brian Wilson, [234], si può consultare una lista dettagliata delle più importanti versioni di Netscape Navigator e Mozilla, con le loro caratteristiche principali.

Benché da questa enorme battaglia legale non sia ancora emerso un vincitore, il caso ha messo in luce alcuni aspetti legali che riguardano il software libero. In particolare, molte aziende hanno iniziato a prendere in considerazione i problemi a cui potrebbero andare incontro usando Linux e altri programmi liberi, e le garanzie che, facendolo, non si trovino in violazione dei diritti di proprietà intellettuale o industriale di terzi.

In un certo senso, questo ed altri casi (ad esempio quelli sulla validità delle licenze GPL, che sono stati risolti in Germania nel 2005) potrebbero essere interpretati anche come un segno della maturità del software libero. Ormai ha smesso di essere un estraneo nel mondo degli affari, per diventare parte di molte sue attività (comprese quelle relative a strategie legali).

### **Ubuntu, Canonical, Fedora e Red Hat**

Benché Canonical (l'azienda che produce e distribuisce Ubuntu) si possa considerare una nuova arrivata nel business delle distribuzioni GNU/Linux, le sue attività meritano la nostra attenzione. In un tempo relativamente breve Ubuntu si è imposta come una delle distribuzioni più note e più diffuse, con una fama di buona qualità e grande facilità di installazione ed uso. Ubuntu si distingue anche per la sua maggiore attenzione ad includere software fondamentalmente libero, più della maggior parte delle distribuzioni prodotte da aziende.

Tuttavia, probabilmente la caratteristica fondamentale di Ubuntu (e della strategia di Canonical) è stata quella di basarsi su Debian, una distribuzione creata e mantenuta da volontari. In realtà Ubuntu non è il primo caso di una distribuzione basata su Debian (un altro noto caso è gnuLinEx), ma è forse quella che ha ricevuto più finanziamenti. Ad esempio, Canonical ha assunto un gran numero di esperti di Debian (molti dei quali partecipano al progetto) e ha perseguito una strategia che cerca collaborazione con il progetto gestito dai volontari. In un certo senso, Canonical ha cercato di colmare quelle che considera le lacune di Debian per essere accettata dall'utente medio.

Red Hat, a sua volta, ha seguito un percorso diverso per ritrovarsi in una situazione piuttosto simile. Partendo da una distribuzione prodotta interamente con risorse proprie, ha deciso di collaborare con Fedora, un gruppo di volontari che stava già lavorando a distribuzioni basate su Red Hat, per produrre Fedora Core, la sua distribuzione "popolare". Red Hat mantiene la sua versione per le aziende, ma questa collaborazione è, in fin dei conti, molto simile a quella che ha prodotto Ubuntu.

Forse tutti questi movimenti non sono che il prodotto della forte concorrenza che caratterizza il mercato delle distribuzioni GNU/Linux e di un'altra significativa tendenza: la collaborazione delle aziende con volontari (con *la comunità*) per produrre software libero.

## Distribuzioni personalizzate

Dall'entrata in scena di Linux, molti gruppi e aziende si sono basati su di esso per creare le proprie distribuzioni. Ma in questi anni il fenomeno ha raggiunto molte organizzazioni e aziende che necessitano di versioni personalizzate per le loro esigenze specifiche. I casi di personalizzazione sono potuti aumentare perché il processo è diventato meno costoso e le competenze necessarie sono ampiamente disponibili, anche rendendo queste operazioni un mercato di nicchia per alcune aziende.

Forse uno dei casi più noti di distribuzioni personalizzate è quello per le comunità autonome della Spagna. Il Governo Regionale dell'Estremadura ha dato origine con il suo gnuLinEx a una tendenza seguita da molte altre comunità autonome. Il processo è così comune che molte di esse offrono regolarmente appalti per la creazione e la manutenzione di nuove versioni delle loro distribuzioni.

La creazione di distribuzioni personalizzate è la realizzazione di una tendenza su cui il mondo del software libero ha discusso a lungo: adattare i programmi alle esigenze specifiche degli utenti, senza che debbano essere per forza i produttori iniziali a fare gli adattamenti.

### Bibliografia

Alcune delle distribuzioni più note di GNU/Linux nelle comunità autonome comprendono:

- gnuLinEx: <http://linex.org> (Estremadura)
- Guadalinex: <http://guadalinex.org> (Andalusía)
- Lliurex: <http://lliurex.net> (Comunità di Valencia)
- Augustux: <http://www.zaralinux.org/proy/augustux/> (Aragona)
- MAX: [http://www.educa.madrid.org/web/madrid\\_linux/](http://www.educa.madrid.org/web/madrid_linux/) (Madrid)
- MoLinux: <http://molinux.info> (Castiglia-La Mancia)

## Collaborazioni tra aziende e tra volontari e aziende

Praticamente sin dall'inizio del software libero ci sono state aziende che hanno collaborato con volontari nello sviluppo di applicazioni. Tuttavia, in questi anni in cui sembra che si sia raggiunta la maturità del software libero, un numero crescente di aziende lo usa come parte della propria strategia di collaborazione con altre aziende, quando lo trova interessante. Due dei casi più significativi, organizzati espressamente a questo scopo, sono ObjectWeb (un'alleanza formata in Francia, che nel tempo è diventata chiaramente internazionale) e Morfeo (in Spagna). In entrambi i casi un gruppo di aziende si è accordato per sviluppare una serie di sistemi liberi a cui era interessato e ha deciso di distribuirlo come software libero.

In altri casi le aziende hanno attivamente cercato di collaborare a progetti liberi promossi da volontari, o cercato di far collaborare volontari ai loro progetti di software libero. La Fondazione GNOME o il già citato Ubuntu nei confronti di Debian sono esempi di questo primo scenario. Sun e OpenOffice.org e OpenSolaris, o Red Hat con Fedora Core, sono esempi del secondo.

### **Espansione ad altre sfere**

Il software libero ha dimostrato che nel settore della produzione di programmi esiste un altro modo di far le cose. In pratica si è visto che garantire libertà di distribuzione, modifica e uso può essere sostenibile, attraverso il lavoro di volontari o attraverso la generazione di business che permette alle aziende di sopravvivere.

Col passare del tempo questa stessa idea si sta trasferendo ad altre sfere di lavoro intellettuale. Le licenze Creative Commons hanno reso possibile liberare altre sfere, come la letteratura, la musica o il video. Wikipedia sta dimostrando che un settore particolare come quello della produzione di enciclopedie può prendere una strada molto interessante. E ci sono sempre più autori di letteratura, gruppi musicali e perfino produttori di film interessati in modelli di produzione e distribuzione libera.

In tutti questi settori c'è ancora molta strada da percorrere e in quasi tutti la sostenibilità della creazione con modelli liberi non è ancora stata dimostrata nei fatti. Tuttavia non si può negare che la sperimentazione con questi modelli stia raggiungendo il "punto di cottura".

### **Il software libero come oggetto di studio**

Benché alcune opere, come il noto libro "La cattedrale e il bazar", abbiano aperto la strada allo studio del software libero in quanto tale, si è dovuto attendere fino al 2001 perché la comunità accademica iniziasse a considerare il software libero come una materia degna di essere studiata. Nel tempo, l'enorme disponibilità di dati (quasi tutto nel mondo del software libero è pubblico e disponibile negli archivi pubblici) e le innovazioni prodotte dal software libero hanno attirato l'attenzione di molti gruppi. A metà del decennio del 2000 esistono già diversi congressi internazionali incentrati nello specifico sul software libero, riviste di altissimo livello scrivono spesso articoli su di esso e le agenzie che si occupano di fornire fondi alla ricerca stanno aprendo linee indirizzate espressamente ad esso.

### **2.5. Il futuro: un percorso a ostacoli?**

Certo, è difficile prevedere il futuro. E certamente non è il nostro obiettivo. Perciò, anziché cercare di descrivere il futuro del software libero, tenteremo di mostrare i problemi che prevediamo dovrà affrontare (e di fatto sta affron-

tando da molto tempo). Il modo in cui il mondo del software libero sarà in grado di superare questi ostacoli determinerà senza dubbio la sua situazione nei prossimi anni.

- FUD Technique (*paura, incertezza, dubbio*). Si tratta di una tecnica piuttosto comune nel mondo delle tecnologie informatiche, usata dai concorrenti del software libero per screditarlo, con giustificazioni più o meno valide e con gradi diversi di successo. In generale il software libero è rimasto piuttosto immune a queste tecniche, forse grazie alla sua complessità e ai suoi svariati modi di infiltrarsi all'interno delle aziende.
- Dissoluzione. Molte aziende stanno testando i limiti del software libero come modello: in particolare stanno cercando di offrire ai propri clienti modelli che presentano alcune caratteristiche simili al software libero. Il problema principale che può presentarsi con questo tipo di modello è che esso genera confusione tra clienti e sviluppatori, i quali dovrebbero leggere attentamente le clausole stampate in piccolo per rendersi conto che quanto viene loro offerto non ha i vantaggi offerti dal software libero. Il più noto modello di questo tipo è il programma Shared Source di Microsoft.
- Mancanza di conoscenza. In molti casi gli utenti si rivolgono al software libero semplicemente perché credono che sia gratuito, o perché pensano che vada di moda. Se non lo osservano più in profondità e non studiano sufficientemente nel dettaglio i vantaggi che offre il modello del software libero, corrono il rischio di non sfruttare appieno tali vantaggi. In molti casi i presupposti di base nel mondo del software libero sono talmente diversi da quelli tradizionali del mondo del software proprietario, che occorre un minimo di analisi per capire che ciò che è frequente in un caso potrebbe essere impossibile nell'altro, e viceversa. Perciò la mancanza di conoscenza può soltanto generare insoddisfazione e perdita di opportunità per le persone o le organizzazioni che si accostano al software libero.
- Ostacoli legali. Questo è senza dubbio il problema principale che il software libero dovrà affrontare nei prossimi anni. Anche se l'ambiente legislativo in cui il software libero si sviluppò negli anni Ottanta e nella prima metà degli anni Novanta non era l'ideale, lasciava almeno lo spazio per crescere liberamente. Da allora, l'estensione della copertura dei brevetti per il software (che si è verificata in molti Paesi sviluppati) e nuove leggi sul copyright (che limitano la libertà creativa dello sviluppatore di software) stanno producendo barriere sempre più alte all'ingresso del software libero in importanti segmenti di applicazione.



## 2.6. Riassunto

Questo capitolo presenta la storia del software libero. Gli anni Settanta furono un periodo dominato dai grossi calcolatori e da IBM, durante il quale il software veniva distribuito insieme all'hardware, di solito con anche il codice sorgente. Negli anni Settanta si cominciò anche a vendere il software separatamente e presto le distribuzioni proprietarie, che non includevano codice sorgente e non permettevano di modificare o redistribuire, divennero praticamente l'unica possibilità.

Negli anni Settanta iniziò presso i Laboratori Bell di ATT lo sviluppo del sistema operativo Unix, che in seguito diede origine a Unix BSD. La sua evoluzione, in parallelo con la nascita di Internet, servì come banco di prova per nuovi modi di sviluppare software in collaborazione, modi che in seguito divennero comuni nel mondo del software libero.

Nel 1984 Richard Stallman iniziò a lavorare al progetto GNU, fondando la Fondazione Software Libero (FSF), scrivendo la licenza GPL e in generale creando le basi del software libero come lo conosciamo oggi.

Negli anni Novanta Internet maturò, offrendo alle comunità di software libero nuovi canali di comunicazione e distribuzione. Nel 1991 Linus Torvalds iniziò a sviluppare un kernel libero (Linux) che contribuì a completare il sistema GNU, il quale aveva ormai quasi tutte le componenti necessarie a diventare un sistema completo, simile a Unix: compilatore C (GCC), editore (Emacs), sistema a finestre (X Window), ecc. Così nacquero i sistemi operativi GNU/Linux, che si ramificarono in diverse distribuzioni, come Red Hat Linux e Debian GNU/Linux. Verso la fine degli anni Novanta questi sistemi furono completati con due ambienti desktop: KDE e GNOME.

Nel decennio del 2000 il software libero è riuscito a diventare leader in certi settori (come i web server, dominati da Apache) e sono comparsi nuovi strumenti che rispondono a un ampio numero di requisiti informatici.

### Guarda anche

I lettori interessati troveranno nell'Appendice B una lista di alcune delle date più importanti nella storia del software libero.

### 3. Aspetti legali

"Le licenze della maggior parte del software sono progettata per toglierti la libertà di dividerlo e modificarlo."

Licenza GNU General Public, versione 2

Questo capitolo esplora i principali aspetti legali relativi al software libero. Per descriverli nel loro contesto, iniziamo introducendo brevemente i concetti più essenziali dei diritti di proprietà intellettuale e industriale, per poi offrire le definizioni dettagliate di *software libero*, *software open source* e altri concetti correlati. Vedremo anche nel dettaglio le licenze più usate per il software libero e il loro impatto sui modelli di business (trattati più estensivamente nel capitolo 5) e sui modelli di sviluppo.

#### 3.1. Breve introduzione alla proprietà intellettuale

Il termine *proprietà intellettuale* ha significati diversi a seconda del contesto e di chi lo usa. Attualmente è usato spesso in molti settori in riferimento a diversi privilegi assegnati per proprietà non tangibili con un valore economico. Comprende concetti come *copyright* e simili, che proteggono opere letterarie o artistiche dalla riproduzione non autorizzata, programmi per computer, collezioni di dati, progetti industriali, eccetera; i marchi registrati, che proteggono i simboli; i termini geografici, che proteggono le denominazioni d'origine; i segreti industriali, che permettono l'occultamento di informazioni e i brevetti, che concedono monopoli temporanei alle invenzioni in cambio della rivelazione di informazioni. Tuttavia in molte tradizioni legali, compresa quella ispanica, esiste una distinzione tra la *proprietà intellettuale*, che si riferisce esclusivamente al copyright, e la *proprietà industriale*, che comprende gli altri concetti.

In ogni caso la legislazione applicabile a tutti questi aspetti è tra quelle meglio coordinate a livello praticamente mondiale. Da un lato la WIPO (organizzazione mondiale per la proprietà internazionale, o Worldwide Intellectual Property Organisation) comprende entrambi i tipi di proprietà in tutti i loro aspetti. Dall'altro lato, l'accordo TRIPS (aspetti commerciali della proprietà intellettuale) stabilisce alcuni livelli minimi di protezione e obbliga tutte le nazioni che fanno parte del WTO (Organizzazione Mondiale del Commercio, o World Trade Organisation) a svilupparli entro determinate tempistiche, a seconda del livello di sviluppo della nazione.<sup>3</sup>

<sup>(3)</sup>L'accordo TRIPS fu firmato su insistenza delle nazioni industrializzate (in particolare gli Stati Uniti e il Giappone).

L'Articolo 27 della Dichiarazione dei Diritti Umani riconosce che ognuno ha diritto alla protezione degli interessi morali e materiali di cui è autore. Tuttavia in molti casi (e di frequente nel caso del software) questo diritto viene trasfe-

rito di fatto alle aziende che danno lavoro ai creatori del software, o che ne distribuiscono o vendono le creazioni. Ciononostante la proprietà intellettuale è giustificata non solo in termini morali, ma anche per ragioni pratiche, per rispettare un altro diritto: il diritto del pubblico di beneficiare di quella creazione, promuovendola con incentivi e proteggendo gli investimenti in creazione, ricerca e sviluppo. Per conciliare questi due diritti, la proprietà intellettuale è temporanea e scade una volta completata la sua funzione di promozione.

Ma la scadenza non è l'unica caratteristica che distingue la proprietà intellettuale da quella ordinaria. Al giorno d'oggi gli oggetti che essa protegge possono essere riprodotti facilmente e a basso costo, senza alcuna perdita di qualità. La riproduzione non danneggia il soggetto che sta già traendo benefici da quanto viene copiato, al contrario del furto, che invece priva il proprietario iniziale dell'oggetto rubato. La riproduzione può danneggiare il proprietario in quanto lo priva di potenziali ricavi di vendita. Controllare la riproduzione di beni intangibili è molto più complesso del controllare il furto di proprietà tangibili, e può portare a una situazione di stato di polizia, in cui si devono tenere sotto controllo tutte le copie di informazione, e di insicurezza legale, perché aumentano le possibilità di infrazioni accidentali. Inoltre la creatività è incrementale: creare significa sempre copiare qualcosa e c'è una linea sottile tra una cattiva imitazione e un'ispirazione.

Per analizzare questo argomento in maggiore profondità, le sezioni seguenti approfondiscono alcune delle categorie della proprietà intellettuale. In ogni caso possiamo già anticipare che il software libero propone un nuovo punto di equilibrio in questo ambito, difendendo i benefici della riproduzione e dell'innovazione incrementale contro il controllo esclusivo di un'opera da parte del suo autore.

### 3.1.1. Copyright

Il *Copyright* protegge la presentazione di un contenuto, non il contenuto in sé. Il Copyright è stato sviluppato per risarcire gli scrittori o gli artisti. Le opere protette possono esprimere idee, conoscenze o metodi che possono essere usati liberamente, ma è proibito riprodurli senza autorizzazione parziale o totale, con o senza modifiche. Questa protezione è molto semplice, perché entra automaticamente in funzione con una copertura quasi universale nel momento in cui l'opera è pubblicata / rilasciata. Attualmente è stata estesa ai programmi per computer e (in alcune aree geografiche) a compilazioni di dati.

La Legge sulla Proprietà Intellettuale (LPI) in Spagna e leggi simili in altri Paesi, sviluppate sulla base della Convenzione di Berna del 1886 per la protezione delle opere letterarie e artistiche, regolamenta il copyright. Tali diritti si suddividono in morali e intellettuali. I primi garantiscono all'autore il controllo sulla distribuzione dell'opera, sotto il suo nome o pseudonimo, il riconoscimento della paternità dell'opera, il rispetto dell'integrità dell'opera e il diritto di modificarla e di ritirarla. I secondi danno all'autore il diritto di sfruttamento

economico dell'opera e possono essere ceduti a terzi interamente o in parte, in modo esclusivo o non esclusivo. I diritti morali sono vitalizi o indefiniti, mentre i diritti intellettuali hanno una durata piuttosto lunga (nella legge spagnola, settant'anni dopo la morte dell'autore, nel caso sia una persona fisica).

La cessione di questi diritti è stabilita tramite un contratto noto col nome di *licenza*. Nel caso dei programmi proprietari, questi vengono in genere distribuiti sotto licenze d'uso "non esclusive", che si ritengono automaticamente accettate da chi apre o installa il prodotto. Perciò non occorre firmare un contratto, perché nel caso il ricevente non lo accetti, per legge perde automaticamente ogni diritto. Le licenze non possono limitare alcuni dei diritti garantiti dalla legislazione corrente, ad esempio il diritto di fare copie per uso privato di opere d'arte o musicali, che include il permesso di regalare una copia di una registrazione a un amico, ma questo diritto non si applica ai programmi. In base alla LPI del 1996 (Legge sulla Proprietà Intellettuale. Decreto Legislativo Reale 1/1996, del 12 aprile) [77], modificato nel 2006 (Legge sulla Proprietà Intellettuale. Legge 23/2006, del 7 luglio) [79], per quanto riguarda i programmi è sempre permesso fare una copia di backup, possono essere studiati per essere resi interoperabili con altri programmi e possono essere corretti e adattati alle proprie esigenze (il che è difficile, perché normalmente non si ha accesso al codice sorgente). Questi diritti non possono essere limitati attraverso le licenze, anche se le leggi sono in revisione, seguendo una tendenza apparentemente inarrestabile a limitare i diritti dell'utente. Anche le collezioni organizzate di opere o dati di terzi sono soggette a copyright, benché con termini diversi e una durata temporale più limitata.

Le nuove informazioni e specialmente le tecnologie web hanno trasformato profondamente la protezione del copyright, dal momento che le presentazioni di contenuto sono diventate molto più facili da copiare del contenuto stesso. E nel caso dei programmi e di alcuni tipi di opere d'arte (musica, immagini, film, e persino letteratura) le presentazioni "funzionano" automaticamente sul computer senza bisogno di alcun apprezzabile sforzo umano. Al contrario, i progetti o le invenzioni devono essere costruiti e a volte messi in produzione. Questa possibilità di generare ricchezza senza costo ha portato una vasta parte del pubblico, specialmente nelle nazioni povere, a duplicare programmi senza pagare la licenza, senza che l'opinione pubblica sia consapevole che si tratta di un'"azione maliziosa" (al contrario del caso ad esempio di furto di proprietà tangibili). Nel frattempo i produttori dei programmi, da soli o in coalizioni (ad esempio tramite la BSA, Business Software Alliance), esercitano enormi pressioni perché le licenze vengano pagate e perché i governi perseguano quella che viene chiamata *pirateria*.

### Nota

La parola *pirateria* è comunemente accettata come sinonimo di 'violazione di qualsiasi forma di proprietà intellettuale, specialmente nel caso della riproduzione illegale di programmi, musica e film'. Il termine appare esagerato e nel dizionario dell'Accademia Reale Spagnola del Linguaggio compare con questo significato in senso figurativo, dal momento che il vocabolo si riferiva originariamente a 'rapine con azioni violente commesse per mare'. Per questo motivo Richard Stallman consiglia di evitarlo ("Alcune parole ed espressioni ambigue o troppo cariche di significati che vale la pena evitare", o "Some confusing or loaded words and phrases that are worth avoiding", 2003) [212].

Proprio per proteggere il copyright di contenuti con licenze proprietarie sono nati i sistemi cosiddetti DRM (*gestione dei diritti digitali*, o *digital rights management*), progettati per controllare l'accesso e l'uso di dati in formato digitale o per limitarne l'uso a certi dispositivi. L'uso dei sistemi DRM è stato fortemente criticato in molti settori, perché il copyright viene protetto attraverso l'imposizione di restrizioni ben oltre quelle che sarebbero sufficienti - motivo per cui la Fondazione Software Libero raccomanda di interpretare l'acronimo come *gestione delle restrizioni digitali*, *digital restrictions management*, nel tentativo di evitare l'uso della parola *diritti*), perché ritiene che gli utenti vengano eccessivamente limitati nei loro diritti in favore della soddisfazione delle esigenze di copyright.

### 3.1.2. Segreto commerciale

Un'altra risorsa di cui le aziende si servono per trarre profitto dai loro investimenti è il segreto commerciale, protetto dalle leggi della proprietà industriale, a condizione che le aziende prendano misure sufficienti per nascondere le informazioni che non desiderano rivelare. Nel caso di prodotti chimici o farmaceutici che richiedono l'approvazione del governo, lo Stato si impegna a non rivelare i dati ad esso sottoposti che non è obbligato a rendere pubblici.

Una delle più note applicazioni del segreto commerciale è l'industria del software proprietario, che in generale vende programmi compilati senza accesso al codice sorgente, per evitare che vengano sviluppati programmi derivati.

A prima vista sembrerebbe che la protezione del segreto commerciale sia un meccanismo perverso, perché può privare la società di conoscenze utili per un tempo indefinito. Entro certi limiti anche alcune legislazioni la interpretano in questo modo e permettono l'ingegneria inversa per sviluppare prodotti di ricambio, benché le pressioni dell'industria siano riuscite a proibire questa attività in molte nazioni, e in altre nazioni è permessa solo per ragioni di compatibilità.

Che il segreto commerciale sia perverso o meno, in molti casi è meglio di un brevetto, perché offre un vantaggio competitivo alla persona che mette un prodotto sul mercato, mentre la concorrenza cerca di imitarlo attraverso l'ingegneria inversa. Più il prodotto è sofisticato, più la concorrenza dovrà spendere per riprodurlo, mentre se è banale verrà copiato rapidamente.

L'imitazione con miglioramenti è stata fondamentale per lo sviluppo delle superpotenze attuali (gli Stati Uniti e il Giappone) ed è molto importante per l'indipendenza finanziaria delle nazioni in via di sviluppo.

### 3.1.3. Brevetti e modelli di utilità

L'alternativa al segreto commerciale è il brevetto. In cambio di un monopolio della durata da 17 a 25 anni e uno specifico costo monetario, una *invenzione* viene rivelata al pubblico in modo da poter essere facilmente riprodotta. L'obiettivo è quello di promuovere la ricerca nel settore privato, senza costi per il contribuente e senza perderne i profitti. Chi possiede il brevetto può decidere se permettere ad altri di usarlo e quale prezzo far pagare per la licenza.

Secondo la dottrina ufficiale il sistema dei brevetti promuove l'innovazione, ma sempre più voci stanno facendosi sentire sostenendo che in realtà la ostacola, o perché il sistema funziona male o perché ritengono che sia perverso in sé (François-René Rideau, "I brevetti sono un'assurdità economica", o "Patents are an economic absurdity", 2000) [194].

Nel tempo è cambiata la concezione di che cosa sia un'invenzione e ci sono fortissime pressioni per ampliare la portata del sistema, in modo da includere algoritmi, programmi, modelli di business, sostanze naturali, geni e forme di vita, compresi piante e animali. TRIPS obbliga il sistema dei brevetti a non discriminare alcuna forma di conoscenza. Le pressioni dell'Organizzazione Mondiale per la Proprietà Intellettuale (WIPO o OMPI) mirano ad eliminare la necessità che un'invenzione abbia un'applicazione industriale e a ridurre anche gli standard di invenzione richiesti per un brevetto. Gli Stati Uniti sono in prima linea tra le nazioni con requisiti minimi su che cosa si possa brevettare, e tra le più battagliere perché anche le altre nazioni adottino questi standard, dimenticando che gli Stati Uniti rifiutavano di accettare i brevetti stranieri quando erano una nazione sottosviluppata.

Una volta ottenuto un brevetto, i diritti del proprietario sono indipendenti dalla qualità dell'invenzione e dagli sforzi investiti per ottenerla. Dato il costo di manutenzione di un brevetto, solo le grandi aziende sono in grado di mantenere - e di fatto mantengono - un ampio portafoglio di brevetti, che le mette in condizione di strangolare la concorrenza. Data la facilità di brevettare soluzioni banali o soluzioni con vasta applicabilità, esse riescono in questo modo a monopolizzare un settore esteso di attività economiche.

Con i brevetti molte attività, specialmente la programmazione, diventano estremamente rischiose, in quanto è molto facile che, sviluppando un programma complesso, si incorra nella violazione accidentale di qualche brevetto. Quando due o più compagnie stanno conducendo ricerche per risolvere un problema, è molto probabile che esse raggiungano soluzioni simili più o meno allo stesso tempo, ma solo una di esse (in genere quella con maggiori risorse) riuscirà a brevettare la sua invenzione, chiudendo alle altre ogni pos-

sibilità di recuperare i loro investimenti. Qualsiasi complesso sviluppo tecnologico diventa un incubo se, per risolvere ogni parte, si deve prima scoprire se la soluzione trovata è già stata brevettata (o in attesa di brevetto), in modo da ottenere la licenza o trovare un'altra soluzione. Questo problema è particolarmente grave per il software libero, in cui basta ispezionare il codice perché appaia evidente la violazione di brevetti sugli algoritmi.

Benché in Europa sia illegale brevettare un algoritmo, potrebbe diventare possibile in un prossimo futuro, forse già al momento in cui i lettori leggeranno queste righe.

### **3.1.4. Loghi e marchi registrati**

I loghi e i marchi registrati sono nomi e simboli che rappresentano una qualità riconosciuta (o un notevole investimento pubblicitario). Non sono molto importanti nel mondo del software libero, probabilmente perché registrarli costa. Perciò solo pochi nomi importanti sono stati registrati, come Open Source (dalla Open Source Foundation), Debian (da Software nell'Interesse Pubblico), GNOME (dalla GNOME Foundation), GNU (dalla Fondazione Software Libero) e OpenOffice.org (da SUN Microsystems), e sono registrati solo in alcune nazioni. Tuttavia, non registrare i nomi ha creato dei problemi. Ad esempio negli Stati Uniti (1996) e in Corea (1997) alcuni hanno registrato il nome Linux e preteso di essere pagati per il suo uso. Risolvere queste dispute comporta spese legali e la necessità di dimostrare l'uso del nome prima della data di registrazione.

## **3.2. Licenze per il software libero**

In termini legali, la situazione dei programmi liberi rispetto a quelli proprietari non è molto diversa: entrambi vengono distribuiti sotto una licenza. La differenza sta in quel che la licenza permette. Nel caso delle licenze dei programmi liberi, che non ne limitano in modo particolare uso, redistribuzione e modifica, ciò che può essere imposto sono condizioni che devono essere soddisfatte in modo esatto se si vuole redistribuire il programma. Ad esempio è possibile richiedere che vengano osservate indicazioni riguardo agli autori o che venga incluso il codice sorgente, se si vuole redistribuire il programma pronto a funzionare.

Benché la differenza essenziale tra il *software libero* e il *software proprietario* sia la licenza sotto la quale gli autori pubblicano i loro programmi, è importante sottolineare che questa distinzione si riflette in condizioni d'uso e di redistribuzione completamente diverse. Come si è visto negli ultimi anni, queste non solo hanno dato origine a modelli di sviluppo totalmente differenti, ma anche a modi praticamente opposti (sotto molti aspetti) di concepire le tecnologie informatiche.

Le leggi sulla proprietà intellettuale garantiscono che, in assenza di esplicita autorizzazione, non si possa fare praticamente nulla con un'opera (in questo caso un programma) ricevuta o acquistata. Solo l'autore (o chi detiene i diritti dell'opera) può dare l'autorizzazione. In ogni caso la proprietà dell'opera non cambia concedendo un'autorizzazione, poiché essa non implica alcun trasferimento di proprietà, ma solo il diritto di usare e, in alcuni casi, il diritto (obbligatorio per il software libero) di distribuire e modificare l'opera in questione. Le licenze per il software libero si distinguono da quelle per il software proprietario proprio perché, invece di limitare accuratamente quanto è permesso, concedono esplicitamente alcune libertà. Quando una persona riceve un programma libero può redistribuirlo oppure no, ma se lo redistribuisce può farlo solo perché la licenza lo permette. Per farlo, però, occorre osservare i termini della licenza stessa. La licenza contiene infatti le regole d'uso che gli utenti, distributori, integratori e tutti gli altri soggetti coinvolti nel mondo delle tecnologie informatiche devono osservare.

Per capire a fondo tutti i dettagli legali che emergono in questo capitolo (e che sono senza dubbio estremamente importanti per comprendere la natura del software libero) occorre anche tenere a mente che ogni nuova versione di un programma viene considerata una nuova opera. L'autore, ancora una volta, possiede pieni diritti per fare quello che vuole col suo programma, anche distribuirlo sotto termini e condizioni totalmente diversi (in altre parole, con una licenza totalmente diversa da quella precedente). In questo modo se il lettore è l'unico autore di un programma, può pubblicarne una versione sotto una licenza di software libero e, se lo desidera, una versione successiva sotto una licenza proprietaria. Se invece un programma ha vari autori e la nuova versione contiene codice che essi hanno prodotto, per pubblicarla sotto condizioni differenti è necessario che tutti gli autori approvino la modifica della licenza.

Una questione ancora relativamente aperta è la licenza che si applica ai contributi esterni. In genere si presuppone che chi contribuisce ad un progetto accetti *de facto* che il proprio contributo si adegui alle condizioni specificate dalla sua licenza, benché le basi legali di questa assunzione siano scarse. L'iniziativa della Fondazione Software Libero di chiedere per lettera (fisica) la cessione di tutti i diritti di *copyright* a chi contribuisce a un sotto-progetto di GNU con più di dieci linee di codice è una chiara indicazione che nel mondo del software libero esistono regole più severe nei riguardi di questi contributi.

In base a quanto descritto sopra, nel resto del capitolo ci concentreremo sull'analisi di diverse licenze. Per porre l'analisi in un contesto adeguato occorre tener presente che, da questo punto in poi, quando parliamo di una licenza di software libero intendiamo dire che essa rientra nelle definizioni di *software libero* presentate nella sezione 1.1.1.



### 3.2.1. Tipi di licenze

Esiste un'enorme varietà di licenze libere, anche se per ragioni pratiche la maggior parte dei progetti si serve di un gruppo ristretto di quattro o cinque. Da un lato, molti progetti non intendono o non hanno le risorse per progettare la loro licenza; dall'altro, la maggior parte degli utenti preferisce riferirsi ad una licenza già nota piuttosto che dover leggere e analizzare licenze complete.

#### Bibliografia

Esiste una lista con discussione delle licenze considerate non-libere o libere ma incompatibili con la GPL dal punto di vista della Fondazione Software Libero, in "Licenze libere" della FSF [121]. La prospettiva filosofica diversa della Open Source Initiative è descritta nella lista ("Licenze Open Source", della Open Source Initiative) [181]. Si possono notare incongruità in alcune licenze, come la Apple Public Source Licence Ver. 1.2, che la FSF considera non-libera per via dell'obbligo di pubblicare tutte le modifiche (anche quelle private), di notificare Apple di ogni redistribuzione, o per la possibilità di revoca unilaterale. Tuttavia la pressione di questa classificazione tra le licenze non libere ha spinto Apple a pubblicare la sua versione 2.0 nell'agosto 2003, versione che la FSF a quel punto ha classificato come libera.

E' possibile dividere le licenze di software libero in due grandi famiglie. La prima comprende le licenze che non impongono condizioni speciali sulla *seconda redistribuzione* (in altre parole, che specificano soltanto che il software può essere redistribuito o modificato, ma non impongono condizioni particolari su come farlo, il che permette, ad esempio, a chi riceve il programma di redistribuirlo come software proprietario): le chiameremo *licenze permissive*. La seconda famiglia, che chiameremo delle *licenze forti* (o *licenze copyleft*), include quelle che, seguendo lo stile della GPL di GNU, impongono condizioni sull'eventualità di voler redistribuire il software, mirando ad assicurare il rispetto delle condizioni della licenza anche dopo la *prima redistribuzione*. Mentre il primo gruppo enfatizza la libertà della persona che riceve il programma di poter fare praticamente tutto quello che vuole con esso (in termini delle condizioni per le redistribuzioni future), il secondo pone l'accento sulla libertà di chiunque potrebbe un giorno ricevere un'opera derivata da quel programma, imponendo che le modifiche e redistribuzioni successive rispettino i termini della licenza originale.

La differenza tra questi due tipi di licenze è stata (e rimane) una questione dibattuta nella comunità del software libero. In ogni caso occorre ricordare che si tratta in entrambi i casi di licenze libere.

### 3.2.2. Licenze permissive

le licenze permissive, note anche talvolta come licenze *liberali* o *minimali*, non impongono praticamente alcuna condizione alla persona che riceve il software e allo stesso tempo le concedono il permesso di usare, redistribuire e modificare. Da un certo punto di vista questo approccio può essere visto come una garanzia di massima libertà per la persona che riceve il programma. Da un altro punto di vista però lo si può intendere come un approccio di massima trascuratezza, in quanto non assicura che chi riceve il programma garantisca

#### Nota

Il termine *copyleft* applicato a una licenza, usato principalmente dalla Fondazione Software Libero in riferimento alle proprie licenze, ha implicazioni simili a quelle che in questo testo chiamiamo *licenze forti*.

le stesse libertà ai futuri utenti ai quali lo vorrà redistribuire. In pratica queste licenze generalmente permettono che il software che l'autore distribuisce sotto una licenza permissiva venga poi redistribuito sotto una licenza proprietaria. .

La più conosciuta tra queste licenze è la licenza BSD, al punto che spesso ci si riferisce alle licenze permissive come a licenze *di tipo BSD*. La licenza BSD (Berkeley Software Distribution) deriva dalla pubblicazione di versioni diverse di Unix prodotte dall'Università della California a Berkeley, negli Stati Uniti. L'unico obbligo che impone è quello di citare gli autori, mentre permette di redistribuire i programmi sia in formato binario che come codice sorgente, senza imporre in nessun caso alcuno dei due formati. Permette anche di apportare modifiche e di integrare il codice in altri programmi praticamente senza alcuna restrizione.

### **Nota**

Una delle conseguenze pratiche delle licenze di tipo BSD è stata quella di diffondere degli standard, dal momento che, sotto questo tipo di licenza, gli sviluppatori non hanno alcun problema a rendere i programmi compatibili con un'implementazione di riferimento. In effetti questa è una delle ragioni della rapida e straordinaria diffusione dei protocolli Internet e dell'interfaccia di programmazione basata su *sockets*, perché la maggior parte degli sviluppatori commerciali ha derivato i propri prodotti dai programmi dell'Università di Berkeley.

Le licenze permissive sono piuttosto diffuse ed esiste un'intera famiglia con caratteristiche simili alle BSD: X Window, Tcl/Tk, Apache, ecc. Storicamente queste licenze apparvero perché il software corrispondente veniva sviluppato dalle università all'interno di progetti di ricerca finanziati dal Governo degli Stati Uniti. Le università facevano a meno di vendere questi programmi, partendo dal presupposto che essi erano già stati pagati dal Governo, e quindi dal contribuente, il che significava che qualsiasi azienda o individuo poteva usare il software praticamente senza alcuna restrizione.

Come già accennato, sulla base di un programma distribuito sotto una licenza permissiva ne può essere creato un altro (in realtà una nuova versione), che può essere proprietario. I critici delle licenze BSD vedono un pericolo in questo aspetto, perché non garantisce la libertà delle versioni future del programma. I loro sostenitori, d'altro canto, le considerano la massima espressione di libertà e sostengono che, alla fine dei conti, con quel software si può fare (quasi) tutto quello che si vuole.

La maggior parte delle licenze permissive è una copia parola per parola dell'originale di Berkeley, con modifiche solo per quanto riguarda gli autori. In alcuni casi, come nella licenza del progetto Apache, si include una clausola aggiuntiva, ad esempio la proibizione di dare lo stesso nome dell'originale alle versioni redistribute. Tutte queste licenze in genere comprendono, come le BSD, la proibizione di usare il nome del detentore dei diritti per promuovere i prodotti derivati.

Allo stesso tempo tutte queste licenze, sia di tipo BSD che di altro genere, comprendono una *limitazione di garanzie* che è in realtà una *liberatoria*, necessaria per evitare rivendicazioni legali di garanzie implicite. Benché questa liberatoria nel software libero sia stata ampiamente criticata, è pratica comune nel software proprietario, che in genere garantisce solo la correttezza del supporto e il funzionamento del programma.

### **Schema riassuntivo della licenza BSD**

Copyright © *il proprietario*. Tutti i diritti riservati.

La redistribuzione e l'uso in formato binario o come codice sorgente, con o senza modifiche, sono autorizzate nel rispetto delle seguenti condizioni:

- 1) Le redistribuzioni di codice sorgente devono riprodurre la menzione di *copyright* ed elencare queste condizioni e la liberatoria.
- 2) Le redistribuzioni in formato binario devono riprodurre la menzione di *copyright* ed elencare queste condizioni e la liberatoria nella documentazione. .
- 3) Né il nome del *proprietario* né i nomi di coloro che hanno contribuito al programma possono essere usati per promuovere prodotti derivati da questo software senza autorizzazione.

Questo programma è offerto "così com'è". Non ci si assume alcuna responsabilità per qualunque garanzia, esplicita o implicita, della sua commerciabilità o adeguatezza ad uno scopo particolare. In nessun caso *il proprietario* è responsabile per alcun danno causato dal suo uso (compresa la perdita di dati, la perdita di profitti o l'interruzione di business).

Di seguito descriviamo brevemente alcune licenze permissive:

- Licenza X Window, versione 11 (X11)  
( [http://www.x.org/Downloads\\_terms.html](http://www.x.org/Downloads_terms.html) ) [73].  
Questa è la licenza usata per distribuire il sistema X Window, il sistema a finestre più largamente usato nel mondo di Unix e negli ambienti GNU/Linux. E' molto simile alla licenza BSD, che permette redistribuzione, uso e modifica praticamente senza restrizioni. A volte viene chiamata *licenza MIT* (con una pericolosa mancanza di precisione, dal momento che il MIT ha usato altri tipi di licenze). Anche molti programmi derivati da X Windows, come XFree86, sono distribuiti sotto questa licenza..
- Licenza Pubblica Zope 2.0 ( <http://www.zope.org/Resources/ZPL> ) [76].  
Questa licenza (comunemente nota come ZPL) è usata per la distribuzione di Zope (un server di applicazioni) e altri prodotti correlati. E' simile alla BSD, con la curiosa caratteristica che proibisce esplicitamente l'uso di marchi commerciali registrati dalla Zope Corporation.
- Licenza Apache.  
Questa è la licenza sotto cui viene distribuita la maggior parte dei programmi prodotti dal progetto Apache. E' simile alla licenza BSD.

Esistono alcuni programmi liberi che non sono distribuiti con una licenza specifica: l'autore dichiara esplicitamente che sono *di dominio pubblico*. Il risultato principale di questa dichiarazione è che l'autore rinuncia a tutti i diritti sul programma, che può quindi essere redistribuito, modificato, usato ecc. in qualsiasi modo. In termini pratici è molto simile a un programma distribuito sotto una licenza di tipo BDS.

### 3.2.3. Licenze forti

#### La Licenza Pubblica Generale GNU (GNU GPL)

La Licenza Pubblica Generale del progetto GNU (Fondazione Software Libero, 1991) [118] (meglio conosciuta con il suo acronimo inglese GPL), riportata nell'appendice C, è di gran lunga la più diffusa e meglio conosciuta tra tutte le licenze nel mondo del software libero. Fu creata dalla Fondazione Software Libero (promotrice del progetto GNU), e inizialmente fu progettata per diventare la licenza di tutto il software generato dalla FSF (Fondazione Software Libero, o Free Software Foundation). Tuttavia il suo uso si è esteso molto oltre, fino a renderla la licenza più usata (ad esempio, più del 70% dei progetti annunciati su Freshmeat hanno licenza GPL), anche dai progetti che sono il fiore all'occhiello del mondo del software libero, come il kernel di Linux.

La licenza GPL è interessante da un punto di vista legale perché fa un uso creativo della legislazione sul *copyright*, per ottenere praticamente l'effetto opposto a quello cui tale legislazione mira: anziché limitare i diritti degli utenti, li garantisce. Per questa ragione la manovra viene spesso chiamata *copyleft* (un gioco di parole in cui il termine inglese "right", che significa sia "diritto" sia "destra", viene sostituito con "left", ossia "sinistra"). Qualcuno con un discreto senso dell'umorismo ha inventato anche lo slogan "copyleft, all rights reversed", ossia "copyleft, tutti i diritti ribaltati".

In parole povere, la licenza GPL permette la redistribuzione in formato binario e come codice sorgente, anche se nel caso di una redistribuzione in formato binario è obbligatorio fornire accesso anche al codice sorgente. Inoltre permette di apportare modifiche senza alcuna limitazione. Tuttavia si può redistribuire codice con licenza GPL integrato con altro codice (ad esempio usando *links*) solo se ha una licenza compatibile. Questo è stato chiamato l'*effetto virale* (anche se molti considerano questo nome poco rispettoso) della GPL, dal momento che una volta che il codice è stato pubblicato sotto queste condizioni, esse non si possono più modificare.

### Nota

Una licenza è incompatibile con la GPL quando limita uno qualsiasi dei diritti garantiti dalla GPL, o esplicitamente contraddicendone qualche clausola, o implicitamente imponendo una nuova restrizione. Ad esempio la licenza corrente BSD è compatibile, ma la licenza Apache è incompatibile perché richiede che i materiali pubblicitari menzionino esplicitamente che il lavoro risultante dalla combinazione di diversi contributi contiene codice di ciascuno dei detentori di diritti. Questo non implica che non si possano usare simultaneamente, o addirittura integrare, programmi con entrambe le licenze. Significa solo che tali programmi integrati non possono essere distribuiti, perché è impossibile rispettare simultaneamente le condizioni di redistribuzione di entrambe le licenze.

La licenza GPL è progettata per garantire la libertà del codice sorgente in ogni momento, perché un programma pubblicato e distribuito sotto le condizioni di questa licenza non potrà mai essere proprietario. Inoltre, né quel programma né le sue modifiche possono essere pubblicati con una licenza diversa dalla GPL. Come già accennato, i sostenitori delle licenze tipo BSD vedono in questa clausola una limitazione di libertà, mentre i seguaci della GPL ritengono che sia una maniera per assicurare che il software rimanga sempre libero. Un modo di vedere la questione è considerare che la licenza GPL massimizza la libertà degli utenti, mentre la licenza BSD massimizza la libertà degli sviluppatori. Tuttavia occorre notare che nel secondo caso ci riferiamo agli sviluppatori in generale e non agli autori, dal momento che molti autori ritengono che la licenza GPL protegga meglio i loro interessi, perché obbliga i concorrenti a pubblicare le loro modifiche (miglioramenti, correzioni eccetera) nel caso redistribuiscono il loro software, mentre con una licenza di tipo BSD questo può anche non verificarsi.

Per quanto riguarda la natura opposta al *copyright* di questa licenza, è così perché la sua filosofia (e quella della Fondazione Software Libero) è che il software non dovrebbe avere padroni (Richard Stallman, "Perché il software non dovrebbe avere padroni" - Why software should not have owners, 1998) [207]. Benché sia vero che il software con licenza GPL ha un autore, che alla fine dei conti è la persona che permette che la legislazione sul *copyright* venga applicata al proprio lavoro, le condizioni con cui viene pubblicato conferiscono al software una tale natura che possiamo considerare la sua proprietà come una caratteristica della persona in possesso del software, e non della persona che l'ha creato.

Ovviamente anche questa licenza comprende *limitazioni di responsabilità* per proteggere gli autori. Allo stesso modo, per proteggere la buona reputazione degli autori originali, tutte le modifiche di un file di sorgente devono comprendere una nota che specifichi la data e l'autore della modifica.

La GPL tiene conto anche dei brevetti di software e richiede che, se la sorgente include algoritmi brevettati (come abbiamo detto, cosa comune e legale negli Stati Uniti, ma attualmente irregolare in Europa), occorre ottenere una licenza per uso gratuito del brevetto, altrimenti non si può distribuire il software sotto la licenza GPL.

La versione più recente della licenza GPL, la seconda, è stata pubblicata nel 1991 (anche se al momento in cui vengono scritte queste parole la terza versione si trova in avanzato stato di preparazione). Tenendo particolarmente presenti le versioni future, la licenza raccomanda di pubblicare il proprio software sotto le condizioni della seconda o di qualunque altra versione successiva pubblicata dalla Fondazione Software Libero, cosa che molti autori fanno. Altri tuttavia, tra cui in particolare Linus Torvalds (il creatore di Linux), pubblicano il loro software solo sotto le condizioni della seconda versione della GPL, scegliendo di distanziarsi da potenziali future evoluzioni della Fondazione Software Libero.

La terza versione della GPL (<http://gplv3.fsf.org>) [115] intende aggiornare la licenza in base allo scenario attuale del software, essenzialmente nei riguardi di aspetti come i brevetti, il DRM (*gestione dei diritti digitali, o digital rights management*) e altre limitazioni della libertà del software. Ad esempio, la bozza disponibile al momento di questo scritto (maggio 2007) non permette a un produttore di hardware di bloccare l'uso di certi moduli del software se non portano una firma digitale che accredita un determinato autore. Un esempio di questa pratica avviene con i videoregistratori digitali TiVo, che forniscono il codice sorgente di tutto il loro software (con licenza GPLv2), vietando allo stesso tempo qualsiasi modifica al codice da usare sull'hardware<sup>4</sup>.

<sup>(4)</sup> Questo caso ha addirittura suggerito l'uso del termine *tivoisation* (tivoizzazione) per altri casi simili che si sono verificati.

La licenza inoltre non permette che il software sia forzatamente messo in funzione su un ambiente predisposto, come avviene quando l'uso di kernel non firmati viene proibito su distribuzioni che lo considerano appropriato per motivi di sicurezza.

#### **Nota**

Diversi punti della licenza GPLv3 hanno generato un certo grado di opposizione. Uno dei gruppi oppositori è il gruppo degli sviluppatori del kernel di Linux (tra cui lo stesso Linus Torvalds). Essi ritengono che l'obbligo di usare componenti software firmate permetta di garantire certe caratteristiche di sicurezza che sarebbero altrimenti impossibili, mentre allo stesso tempo la loro esplicita proibizione estenderebbe la licenza al settore dell'hardware. Inoltre, la limitazione stabilita dal meccanismo delle firme interesserebbe solo le piattaforme hardware e software progettate in questo modo, nel senso che il software potrebbe essere modificato per essere usato su un hardware diverso. Rispetto a questo punto, la FSF è in favore dell'uso di meccanismi di firme che sconsiglino di usare componenti non firmate per motivi di sicurezza, ma ritiene che non proibire quei meccanismi di firme che impediscono l'uso di componenti non firmate potrebbe dare origine a scenari in cui non esisterebbero piattaforme hardware o software su cui far funzionare quei software modificati: ciò significherebbe che la libertà del software libero diventerebbe totalmente limitata in quanto a modifiche del codice.

### **La Licenza Pubblica Generale Minore GNU (GNU LGPL)**

La Licenza Pubblica Generale Minore GNU (Free Software Foundation, GNU Lesser General Public Licence o GNU LGPL, versione 2.1, febbraio 1999) [119], comunemente chiamata con le sue iniziali in inglese - LGPL - è un'altra licenza

della Fondazione Software Libero. Inizialmente progettata per il suo uso con le librerie (la L inizialmente stava per *library*), è stata recentemente modificata per essere considerata la sorella minore (*lesser*) della GPL.

La LGPL permette ai programmi liberi di essere usati con software proprietario. Il programma stesso viene redistribuito come se fosse sotto la licenza GPL, ma la sua integrazione a qualsiasi altro pacchetto software è autorizzata praticamente senza alcuna restrizione.

Come si può vedere, in origine questa licenza puntava alle librerie, per promuovere il loro uso e sviluppo senza imbattersi nei problemi di integrazione implicati dalla GPL. Tuttavia, quando ci si rese conto che l'obiettivo perseguito di diffondere le librerie libere non veniva compensato dalla generazione di programmi liberi, la Fondazione Software Libero decise di trasformare *library* in *lesser* e sconsigliò di usare questa licenza, eccetto in circostanze particolari e molto sporadiche. Attualmente esistono molti programmi, che non sono librerie, pubblicati sotto i termini della licenza LGPL. Ad esempio il browser di Mozilla o la *suite* per ufficio di OpenOffice.org hanno, tra gli altri, licenza LGPL.

#### **Nota**

Come per la GPL, l'ultima versione pubblicata della LGPL è la seconda, anche se esiste già un modello della terza versione ( <http://gplv3.fsf.org/pipermail/info-gplv3/2006-July/000008.html> ) [116]. Questa nuova versione è più corta della precedente, perché riferisce tutto il proprio testo alla GPLv3, limitandosi a sottolineare le differenze.

### **Altre licenze forti**

Altre licenze forti che meritano di essere citate:

- Licenza Sleepycat ( [www.sleepycat.com/download/oslicense.html](http://www.sleepycat.com/download/oslicense.html) ) [59]. - Sotto questa licenza l'azienda Sleepycat ( <http://www.sleepycat.com/> ) [60] distribuisce i suoi programmi (tra i quali ricordiamo il rinomato Berkeley DB). La licenza obbliga a rispettare certe condizioni ogni qualvolta il programma o sue derivazioni vengano redistribuiti. In particolare, obbliga ad offrire il codice sorgente (comprese le modifiche nel caso di un lavoro derivato) e ad imporre nella redistribuzione le stesse condizioni al ricevente. Benché sia molto più corta della GPL di GNU, le assomiglia molto nei principali effetti .
- eCos License 2.0 ( <http://www.gnu.org/licenses/ecos-license.html> ) [25]. Questa è la licenza sotto cui viene distribuito eCos ( <http://sources.redhat.com/ecos/> ) [24], un sistema operativo in tempo reale. Si tratta di una modifica della GPL di GNU, che non ritiene che il codice collegato ai programmi che protegge sia soggetto alle clausole della GPL di GNU qualora venga redistribuito. Da questo punto di vista, i suoi effetti sono simili a quelli della LGPL di GNU.

- Licenza Pubblica Generale Affero ( <http://www.affero.org/oagpl.html> ) [78].

E' un'interessante modifica della GPL di GNU, che prende in considerazione il caso dei programmi che offrono servizi via web, o in generale attraverso reti di computer. Questo tipo di programma rappresenta un problema dal punto di vista delle licenze forti. Siccome l'uso del programma non implica l'averlo ricevuto tramite una redistribuzione, anche se è pubblicato ad esempio sotto una licenza GPL di GNU, chiunque può modificarlo e offrire un servizio su web che si serve del programma modificato, senza tuttavia redistribuirlo in alcun modo, e perciò senza essere obbligato, ad esempio, a distribuire il suo codice sorgente. La Affero GPL ha una clausola tale per cui, se il programma ha modo di fornire il proprio codice sorgente via web a chiunque lo usi, questa caratteristica non può essere disabilitata. Questo significa che, se l'autore iniziale include questa funzionalità nel codice sorgente, qualsiasi utente può ottenerlo, e inoltre la *redistribuzione* è soggetta alle condizioni della licenza. La Fondazione Software Libero sta valutando se introdurre simili provvedimenti nella versione 3 della sua GNU GPL.

- Licenza Pubblica IBM 1.0 ( <http://oss.software.ibm.com/developer-works/opensource/license10.html> ) [40].

E' una licenza che autorizza una redistribuzione in formato binario di lavori derivati solo se (tra le altre condizioni) viene contemplato un meccanismo per la persona che riceve il programma in modo che possa ricevere anche il codice sorgente. La redistribuzione del codice sorgente deve avvenire sotto la medesima licenza. Questa licenza è interessante anche perchè obbliga il soggetto che redistribuisce il programma modificato a mettere sotto licenza, automaticamente e gratuitamente, qualsiasi brevetto che influenzi tali modifiche e che sia di proprietà del redistributore, per il soggetto che riceve il programma.

- Licenza Pubblica Mozilla 1.1 ( <http://www.mozilla.org/MPL/MPL-1.1.html> ) [49].

Questo è un esempio di licenza libera scritta da un'azienda. E' un'evoluzione della prima licenza libera di Netscape Navigator, che fu molto importante ai suoi tempi, perché rappresentava la prima volta in cui una nota azienda decise di distribuire un programma sotto la propria licenza libera.

### 3.2.4. Distribuzione sotto varie licenze

Finora abbiamo dato per scontato che ogni programma venga distribuito sotto una sola licenza, che ne specifica le condizioni d'uso e di redistribuzione. Tuttavia un autore può distribuire lavori sotto diverse licenze. Per comprendere questo aspetto occorre ricordare che ogni pubblicazione è una nuova opera, e che versioni diverse possono essere distribuite con la licenza come unica dif-



ferenza tra loro. Come vedremo, il più delle volte questo si traduce nel fatto che l'utente, a seconda di quello che vuole fare col software, dovrà osservare i termini dell'una o dell'altra licenza.

Uno degli esempi più noti di doppia licenza è quello della libreria Qt, su cui si fonda l'ambiente desktop KDE. Trolltech, un'azienda norvegese, distribuì Qt con una licenza proprietaria, anche se esonerò dal pagamento i programmi che non lo usavano a fini di lucro. Per questa ragione e a causa delle sue caratteristiche tecniche, venne scelto dal progetto KDE a metà degli anni Novanta. Questo diede origine ad un'intensa disputa con la Fondazione Software Libero, perchè a quel punto KDE smetteva di essere un software completamente libero, dal momento che dipendeva da una libreria proprietaria. In seguito ad un ampio dibattito (durante il quale apparve GNOME come concorrente libero di KDE tra gli ambienti desktop), Trolltech decise di usare il sistema della doppia licenza per il suo prodotto più famoso: i programmi sotto la GPL potevano usare una versione GPL di Qt, mentre chi intendeva integrarla in programmi che avevano licenze incompatibili con la GPL (ad esempio programmi proprietari) doveva acquistare da loro una licenza speciale. Questa soluzione riuscì a soddisfare tutti e attualmente KDE è considerato software libero.

Altri esempi molto noti di doppia licenza sono StarOffice e OpenOffice.org, o Netscape Communicator e Mozilla. In entrambi i casi, il primo prodotto è proprietario mentre il secondo è una versione libera (generalmente sotto le condizioni di varie licenze libere). Benché in origine i progetti liberi fossero versioni ridotte dei loro fratelli proprietari, nel tempo hanno seguito la loro strada, tanto che attualmente hanno acquisito un livello piuttosto alto di indipendenza.

### **3.2.5. La documentazione di un programma**

La documentazione che accompagna un programma è parte integrante di esso, così come i commenti nel codice sorgente, come riconosce ad esempio la Legge Spagnola sulla Proprietà Intellettuale. Dato il livello di integrazione, pare logico che le stesse libertà che si applicano al software valgano anche per la documentazione, e che questa evolva nello stesso modo in cui evolve il programma: qualsiasi modifica apportata al programma richiede una modifica simultanea e corrispondente nella sua documentazione.

La maggior parte di questo tipo di documentazione tende ad essere codificata in file di testo non formattati, con l'obiettivo di renderla universalmente accessibile con il minimo ambiente di strumenti, e (nel caso dei programmi liberi) normalmente comprende una breve introduzione al programma ( `README` ), indicazioni per l'installazione ( `INSTALL` ), una breve storia dell'evoluzione e del futuro del programma ( `CHANGELOG` e `ALL` ), autori e copyright ( `AUTHORS` e `COPYRIGHT` o `COPYING` ), insieme alle istruzioni per l'uso. Tutte queste voci, esclusi autori e copyright, devono essere liberamente modificabili man mano

che il programma evolve. Agli autori occorre solo aggiungere nomi e relative attribuzioni, senza eliminare niente, mentre il copyright può essere modificato solo se le condizioni lo permettono.

Le istruzioni per l'uso sono normalmente codificate in formati più complessi, perché tendono ad essere documenti più lunghi e ricchi. Il software libero richiede che questa documentazione possa essere modificata facilmente, cosa che a sua volta impone l'uso dei cosiddetti formati *trasparenti*, con specifiche note e in grado di essere processati da strumenti liberi, come, oltre al testo puro e semplice, il formato delle pagine del manuale di Unix, TexInfo, LaTeX o DocBook, senza impedire che sia possibile anche distribuire il risultato della trasformazione di questi documenti sorgente in formati più adatti alla visualizzazione o alla stampa, come HTML, PDF o RTF (in genere formati più *opachi*).

Tuttavia la documentazione di un programma è spesso preparata da terzi, che non sono stati coinvolti nello sviluppo. Talvolta la documentazione è di natura didattica, per facilitare l'installazione e l'uso di uno specifico programma (HOWTO); a volte si tratta di documentazione più estesa, che descrive diversi programmi e la loro integrazione, che confronta soluzioni, ecc., nella forma di un tutorial o di un manuale di consultazione; altre volte è semplicemente una lista di domande frequenti e relative risposte (FAQ). Un esempio degno di nota è il progetto di documentazione di Linux ( <http://www.tldp.org> ) [44]. In questa categoria potremmo includere anche altri documenti tecnici, non necessariamente riguardanti dei programmi, ad esempio le istruzioni per cablare una rete locale, fare un forno solare, riparare un motore o selezionare un fornitore di strumenti.

Questi documenti sono a metà strada tra semplice documentazione di un programma e articoli o libri estremamente tecnici e pratici. Senza pregiudicare la libertà di leggere, copiare, modificare e redistribuire, l'autore può esprimere opinioni che non desidera vengano distorte, o perlomeno non vuole che gli venga attribuita alcuna distorsione; oppure potrebbe voler mantenere inalterati certi paragrafi, ad esempio i ringraziamenti; o rendere obbligatoria la modifica di altri, ad esempio il titolo. Benché queste preoccupazioni spesso emergano anche con il software stesso, nel mondo del software libero non sono state espresse tanto energicamente quanto nel mondo della documentazione libera.

### 3.3. Riassunto

In questo capitolo abbiamo visto gli aspetti legali che governano o influenzano il mondo del software libero. Essi fanno parte della legislazione sulla proprietà intellettuale o industriale, concepita in origine per stimolare la creatività offrendo ricompense agli inventori per un determinato periodo. Tra le di-

verse tipologie, il cosiddetto *copyright* è quello che influenza maggiormente il software libero e, se adeguatamente applicato, può essere usato per garantirne l'esistenza attraverso le licenze libere.

Abbiamo visto l'importanza delle licenze nel mondo del software libero. Abbiamo inoltre presentato l'enorme varietà di licenze esistenti, le basi su cui si fondano, le loro ripercussioni, vantaggi e svantaggi. Di certo possiamo dire che la licenza GPL cerca di massimizzare la libertà degli utenti di software, sia che ricevano il software libero direttamente dal suo autore, sia che lo ottengano da altri, mentre le licenze di tipo BSD mirano a massimizzare la libertà di chi modifica o redistribuisce.

A partire da quanto osservato in questo capitolo, si può dedurre che è molto importante decidere in anticipo quale sarà la licenza di un determinato progetto ed essere ben consapevoli dei suoi vantaggi e svantaggi, perché eventuali modifiche successive tendono ad essere estremamente difficili, specialmente se il numero di contributi esterni è particolarmente alto.

In conclusione, vorremmo sottolineare il fatto che l'unica differenza tra il software libero e il software proprietario è solo ed esclusivamente la licenza sotto la quale i programmi sono pubblicati. Nei prossimi capitoli, tuttavia, vedremo che questa differenza puramente legale può influenzare o meno il modo in cui il software viene sviluppato, dando origine a un nuovo modelli di sviluppo, che può essere più o meno diverso, a seconda dei casi, dai metodi di sviluppo "tradizionali" usati nell'industria del software.

## 4. Gli sviluppatori e le loro motivazioni

"Essere un *hacker* è molto divertente, ma è un tipo di divertimento che richiede un notevole sforzo. Lo sforzo richiede motivazione. Gli atleti di successo traggono la propria motivazione da un tipo di piacere fisico che provano facendo lavorare il proprio corpo al meglio delle sue possibilità, spingendosi oltre i propri limiti fisici. Analogamente, per essere un *hacker* devi provare un certo brivido di piacere nel risolvere un problema, nell'aguzzare l'ingegno ed esercitare la tua intelligenza."

Eric Steven Raymond, "Come diventare un hacker"

### 4.1. Introduzione

Il modo parzialmente anonimo e distribuito in cui si è sviluppato il software libero ha implicato che per molti anni le risorse umane su cui si fonda sono rimaste generalmente sconosciute. Il risultato di questa mancanza di conoscenza è stato una mitizzazione, almeno in parte, del mondo del software libero e della vita di coloro che ne fanno parte, mitizzazione basata su stereotipi più o meno diffusi, riguardanti la cultura degli *hacker* e dei computer. Negli ultimi anni la comunità scientifica ha compiuto un enorme sforzo per cercare di rendere note le persone che partecipano ai progetti di software libero, le loro motivazioni, la loro formazione accademica e altri aspetti più o meno rilevanti. Da un punto di vista puramente pragmatico, sapere chi è coinvolto in questo tipo di progetti e perché può risultare estremamente utile quando si tratta di generare software libero. Alcuni studiosi, prevalentemente economisti, psicologi e sociologi, hanno voluto andare oltre e hanno visto in questa comunità il seme di comunità virtuali future, con le loro specifiche regole e gerarchie, in molti casi del tutto diverse da quelle che conosciamo della società "tradizionale". Uno dei più importanti misteri da risolvere era scoprire che cosa spinge gli sviluppatori di software a partecipare ad una comunità di questa natura, dal momento che i vantaggi economici, almeno diretti, sono praticamente inesistenti, mentre quelli indiretti sono difficili da quantificare.

### 4.2. Chi sono gli sviluppatori?

Questa sezione mira ad offrire una visione globale delle persone che spendono il loro tempo ed energie nella partecipazione a progetti di software libero. I dati che presentiamo derivano principalmente da ricerche scientifiche compiute negli ultimi anni, tra cui le più significative - ma assolutamente non le uniche - includono: *Software libero/libre e open source. Indagine e ricerca*, parte IV: "Indagine sugli sviluppatori", 2002 [126], e "Chi è che lo fa? Saperne di più sugli sviluppatori del software libero", 2001 [197].

In genere gli sviluppatori di software sono giovani. L'età media è intorno ai ventisette anni. Le differenze di età sono significative, dal momento che il gruppo dominante è nel segmento dai ventuno ai ventiquattro anni e il va-

lore più frequente è ventitrè. E' interessante notare che l'età a cui si aderisce al movimento del software libero ha un picco tra i diciotto e i venticinque, particolarmente pronunciato tra i ventuno e i ventitrè, che coincidono con gli anni dell'università. Questi dati sono in contrasto con l'idea che il software libero sia più che altro una cosa da adolescenti, anche se il coinvolgimento dei teenager è notevole (circa il 20% degli sviluppatori ha meno di vent'anni). Di sicuro possiamo vedere che la maggior parte degli sviluppatori (60%) è sulla ventina, mentre gli under-20 e gli over-30 si dividono equamente il rimanente 40%.

Dall'età dell'adesione possiamo intuire che l'università influenza enormemente il software libero. Ciò non deve sorprendere se si pensa che, come abbiamo visto nel capitolo sulla storia, prima ancora che il software libero fosse chiamato con questo nome era già strettamente legato all'istruzione universitaria. Anche al giorno d'oggi i gruppi di utenza formati da studenti e le università continuano a portare avanti l'uso e l'espansione del software libero. Non è perciò sorprendente che oltre il 70% degli sviluppatori abbia un'istruzione di livello universitario. Questo dato è ancora più significativo se si tiene presente che il rimanente 30% non ha frequentato l'università perchè va ancora a scuola. E tuttavia si tratta di un gruppo coinvolto e non meno apprezzato degli sviluppatori che non hanno avuto accesso ad un'istruzione universitaria, ma che sono appassionati di tecnologie.

Lo sviluppatore di software libero è generalmente maschio. Le cifre riportate da vari sondaggi sulla presenza di donne nella comunità variano tra l'1% e il 3%, con margini di errore comparabili. Allo stesso tempo la maggioranza (60%) dichiara di non essere single, mentre la percentuale di sviluppatori con figli è soltanto del 16%. Visto l'intervallo di età degli sviluppatori di software libero, questo dato riflette abbastanza precisamente un campione casuale, e può quindi essere considerato "normale". Il mito dello sviluppatore solitario la cui passione per l'informatica è l'unico interesse della sua vita risulta, come possiamo vedere, un'eccezione piuttosto che la regola.

### **4.3. Che cosa fanno gli sviluppatori?**

In termini di occupazione gli sviluppatori di software libero si descrivono come ingegneri del software (33%), studenti (21%), programmatori (11%), consulenti (10%), professori universitari (7%), ecc. Dall'altro lato della bilancia scopriamo che in genere non lavorano nelle sezioni marketing e vendite (circa 1%). E' interessante notare che molti si definiscono ingegneri del software anziché programmatori, in numero quasi tre volte superiore, se si considera che, come vedremo nel capitolo sull'ingegneria del software, l'applicazione delle tecniche classiche dell'ingegneria del software (e anche di alcune tecniche moderne) non è esattamente radicata nel mondo del software libero.

Il legame con l'università, che è già stato provato, torna ad alzare la testa in questa sezione. Circa uno sviluppatore su tre è studente o professore universitario, il che dimostra la formidabile collaborazione tra persone provenienti prevalentemente dall'industria del software (i rimanenti due terzi) e l'ambiente accademico.

Allo stesso tempo è stato possibile identificare un ambito molto ampio di discipline miste: uno sviluppatore su cinque viene da un settore che non è l'informatica. Questo, unito al fatto che un numero simile di sviluppatori non è all'università, riflette una varietà di interessi, origini e, sicuramente, composizioni nei team di sviluppo. E' molto difficile trovare un settore dell'industria moderna, se esiste, con un grado di eterogeneità paragonabile a quello che si può trovare nel software liberi.

Oltre a un 20% circa di studenti, il 64% degli sviluppatori ha in genere un impiego remunerato, mentre la percentuale di sviluppatori che lavorano in proprio è del 14%. Infine, solo il 3% dichiara di essere disoccupato, un dato significativo dal momento che il sondaggio è stato condotto dopo l'inizio della crisi delle dotcom.

#### **Nota**

Il fatto che il software libero, a differenza del software proprietario, non possa ottenere finanziamenti attraverso la vendita delle licenze, ha sempre provocato accesi dibattiti su come i programmatori dovrebbero guadagnarsi da vivere. Nei sondaggi a cui ci riferiamo in questo capitolo, il 50% degli sviluppatori afferma di aver ricevuto un compenso economico diretto o indiretto per la propria partecipazione a progetti di software libero. Molti altri, tuttavia, non ne sono così sicuri. Richard Stallman, il fondatore del progetto GNU, alla domanda su che cosa debba fare uno sviluppatore di software libero per far soldi, tende a rispondere che può andare a fare il cameriere.

#### **4.4. Distribuzione geografica**

Ottenere dati sulla posizione geografica degli sviluppatori è una questione che deve essere affrontata in modo più scientifico. Il problema delle ricerche presentate in questo capitolo è che, siccome si basano su sondaggi condotti su Internet, aperti a chiunque desideri partecipare, la partecipazione è dipesa in gran parte dai siti sui quali il sondaggio è stato pubblicato e dal modo in cui è stato presentato. Per essere precisi dobbiamo far presente che i sondaggi non miravano ad essere rappresentativi su questo aspetto, ma piuttosto ad ottenere le risposte e/o opinioni del numero più ampio possibile di sviluppatori di software libero.

In ogni caso possiamo arrischiare alcune affermazioni su questo aspetto, con la consapevolezza che questi dati non sono affidabili quanto i precedenti e che perciò il margine di errore è molto maggiore. Quel che sembra assodato è che la maggior parte degli sviluppatori di software libero proviene da nazioni industrializzate, mentre è rara la presenza di sviluppatori dai cosiddetti Paesi del Terzo Mondo. Di conseguenza non dovrebbe sorprendere che la mappa degli sviluppatori del progetto Debian (<http://www.debian.org/de->

vel/developers.loc ) [187], ad esempio, assomigli alle fotografie della Terra di notte: dove c'è luce - ossia "dove ci sono Paesi industrializzati" - là tendono a concentrarsi gli sviluppatori di software libero. Questo, che a prima vista può sembrare logico, è in contrasto con le potenziali opportunità che il software libero potrebbe offrire ai Paesi del Terzo Mondo.

Un chiaro esempio è la tabella seguente, che mostra i principali Paesi d'origine degli sviluppatori del progetto Debian negli ultimi quattro anni. Esiste un'evidente tendenza alla decentralizzazione del progetto, visibile dal fatto che la crescita nel numero di sviluppatori provenienti dagli Stati Uniti, la nazione che contribuisce maggiormente, è più bassa della media. In generale le nazioni sono riuscite a raddoppiare il proprio numero di volontari negli ultimi quattro anni; la Francia, che è riuscita a moltiplicare per cinque il proprio numero di presenze, è l'esempio più chiaro al riguardo. Se si considera che Debian ha mosso i primi passi sul continente americano (Canada e Stati Uniti in particolare), si può vedere che negli ultimi anni il progetto si è *uropeizzato*. Presumiamo che il prossimo passo sarà la tanto cercata globalizzazione, con il coinvolgimento di nazioni sudamericane, africane e asiatiche (ad eccezione di Giappone e Corea, che sono già ben rappresentati), benché i dati in nostro possesso (due collaboratori da Egitto, Cina o India, e uno in Messico, Turchia o Colombia a giugno 2003) non siano molto promettenti in questo senso.

Nel mondo del software libero (e non solo nel caso di Debian) esiste un intenso dibattito sulla supremazia dell'Europa o degli Stati Uniti. Quasi tutti gli studi hanno mostrato che la presenza di sviluppatori europei è leggermente maggiore rispetto ai nordamericani, un aspetto mitigato dal fatto che la popolazione europea è più numerosa di quella degli Stati Uniti. Abbiamo quindi a che fare con una guerra di numeri, dal momento che il numero di sviluppatori pro capite è superiore in Nord America, ma, se consideriamo il numero di persone con accesso a Internet anziché la popolazione totale, l'Europa passa di nuovo in vantaggio.

In termini di nazioni, le aree con i livelli più alti di radicamento (in numero di sviluppatori diviso per la popolazione) sono il Nord Europa (Finlandia, Svezia, Norvegia, Danimarca e Islanda) e il Centro Europa (Benelux, Germania e Repubblica Ceca), seguiti da Australia, Canada, Nuova Zelanda e Stati Uniti. Nonostante la sua importanza in termini assoluti (a causa dell'ampia popolazione di Francia, Italia e Spagna), però, l'area mediterranea è sotto la media.

**Tabella 1. Nazioni con il maggior numero di sviluppatori Debian**

Nazione	01/07/1999	01/07/2000	01/07/2001	01/07/2002	20/06/2003
USA	162	169	256	278	297
Germania	54	58	101	121	136
UK	34	34	55	63	75

Nazione	01/07/1999	01/07/2000	01/07/2001	01/07/2002	20/06/2003
Australia	23	26	41	49	52
Francia	11	11	24	44	51
Canada	20	22	41	47	49
Spagna	10	11	25	31	34
Giappone	15	15	27	33	33
Italia	9	9	22	26	31
Olanda	14	14	27	29	29
Svezia	13	13	20	24	27

#### 4.5. Dedizione

Il numero di ore che gli sviluppatori di software libero dedicano allo sviluppo di software libero è uno dei grandi sconosciuti. Occorre anche puntualizzare che questa è una delle principali differenze rispetto al software prodotto in azienda, in cui i membri del team e il tempo speso da ciascun membro su un determinato aspetto dello sviluppo sono ben noti. Il tempo che gli sviluppatori dedicano al software libero può essere preso come una misura indiretta del loro livello di professionalizzazione. Prima di presentare i dati attualmente disponibili è importante osservare che essi sono stati ricavati da stime fornite dagli sviluppatori in diversi sondaggi, perciò, oltre alle imprecisioni inevitabili in questo tipo di raccolta dati, occorre considerare anche un margine di errore associato al modo in cui ciascuno sviluppatore interpreta il tempo dedicato allo sviluppo. Ad esempio, sicuramente molti sviluppatori non includeranno il tempo impiegato per aprire e leggere le email (o forse sì) e indicheranno solo il tempo speso a programmare e ad eliminare i bachi. Occorre quindi considerare tutte le cifre presentate di seguito con le dovute riserve.

Le ricerche condotte fino ad oggi mostrano che in media ogni sviluppatore di software libero spende undici ore a settimana in questa attività ("Motivazione degli sviluppatori di software libero nei progetti open source: un sondaggio su Internet rivolto a coloro che contribuiscono al kernel di Linux", 2003) [143]. Tuttavia questo dato può essere ingannevole, perchè esiste un'enorme variazione nei tempi impiegati dai vari sviluppatori. Nella ricerca *Software libero/libre e open source. Indagine e ricerca*, parte IV: "Indagine sugli sviluppatori", 2002 [126], il 22.5% delle persone intervistate ha risposto che il proprio contributo era inferiore alle due ore settimanali, e la percentuale saliva a 26.5% per chi spendeva dalle due alle cinque ore a settimana; tra sei e dieci ore era il tempo speso dal 21.0%, mentre il 14.1% impiegava dalle undici alle venti ore



alla settimana; il 9.2% ha dichiarato che il tempo speso a sviluppare software libero era tra le venti e le quaranta ore a settimana, e il 7.1% addirittura più di quaranta ore settimanali.

**Tabella 2. Dedizione in ore alla settimana**

Ore alla settimana	Percentuale
Meno di 2 ore	22.5%
Tra 2 e 5 ore	26.1%
Tra 5 e 10 ore	21.0%
Tra 10 e 20 ore	14.1%
Tra 20 e 40 ore	9.2%
Più di 40 ore	7.1%

#### **Nota**

Oltre a mostrare il livello di professionalizzazione dei team di sviluppo del software libero, le ore di tempo spese sono un parametro rilevante quando si tratta di fare una stima dei costi e un confronto con i modelli di sviluppo del software proprietario nell'industria. Con il software libero, per ora, abbiamo solo il prodotto finale (nuove consegne di software, sincronizzazione di nuovo codice nei sistemi di versioni...) che non ci permette di sapere quanto tempo ha impiegato lo sviluppatore per portarlo a termine.

Un'analisi di queste cifre rivela che circa l'80% degli sviluppatori svolge questa attività nel proprio tempo libero, mentre solo uno su cinque ritiene di dedicare a questa attività la stessa quantità di tempo di un professionista. Più avanti, nel capitolo sull'ingegneria del software, vedremo come questi dati rispecchino i contributi degli sviluppatori, dal momento che entrambi sembrano seguire la legge di Pareto (*cf.* sezione 7.6).

## **4.6. Motivazioni**

Ci sono molte speculazioni, e tali rimangono, sulle motivazioni di chi sviluppa software libero, specialmente quando lo fa nel proprio tempo libero (il che, come abbiamo visto, corrisponde all'80% circa degli sviluppatori). Come nelle sezioni precedenti, abbiamo soltanto i dati dei sondaggi, che - è importante rendersi conto - sono le risposte fornite dagli sviluppatori stessi e possono essere quindi più o meno aderenti alla realtà. Le percentuali presentate di seguito superano il 100% perchè i partecipanti avevano la possibilità di selezionare diverse risposte.

In ogni caso sembra dalle loro risposte che la maggior parte voglia imparare e sviluppare nuove capacità (circa 80%) e che molti lo facciano per condividere conoscenze e competenze (50%) o per partecipare a nuove forme di collaborazione (circa un terzo). Il primo dato non sorprende, dal momento che un professionista con maggiori conoscenze sarà più richiesto di uno che ne ha meno. Tuttavia non è altrettanto facile spiegare il secondo dato, che sembra

addirittura contraddire l'opinione di Nikolai Bezroukov in "Un secondo sguardo alla cattedrale e al bazaar" (dicembre 1998) [91], secondo cui i leader dei progetti di software libero stanno attenti a non rivelare tutte le informazioni in loro possesso per perpetuare il loro potere. La terza risposta selezionata più di frequente, intanto, riflette senza dubbio l'entusiasmo degli sviluppatori per il modo in generale in cui viene creato il software libero; è difficile trovare un'industria in cui un gruppo di volontari con un'organizzazione molto lieve riesca, in termini tecnologici, a far fronte ai giganti del settore.

Benché la teoria "classica" per spiegare come mai gli sviluppatori di software libero dedichino tempo a contribuire a questo tipo di progetti sia la reputazione e i vantaggi economici indiretti nel medio e lungo periodo, sembrerebbe che gli sviluppatori stessi contraddicano queste idee. Solo il 5% degli intervistati ha risposto che sviluppa software libero per fare soldi, mentre il numero di coloro che lo fanno per costruirsi una reputazione arrivava al 9%, cifre molto lontane da quelle presentate nel paragrafo precedente. In ogni caso sembra che la ricerca sulle motivazioni che spingono gli sviluppatori ad entrare a far parte della comunità del software libero sia un compito fondamentale, che i sociologi e gli psicologi dovranno affrontare in un prossimo futuro.

#### 4.7. Leadership

La reputazione e la leadership sono due delle caratteristiche usate per spiegare il successo del software libero, specialmente il modello del bazaar, come vedremo nel capitolo sull'ingegneria del software. Come abbiamo visto in un altro capitolo, quello sulle licenze del software, esistono certe differenze tra le licenze di software libero e il loro equivalente nel settore della documentazione. Queste differenze derivano dal modo di mantenere la paternità di un'opera e dal fatto che l'opinione di un autore è più accentuata in un testo che in un programma.

In *Software libero/libre e open source. Indagine e ricerca*, parte IV: "Indagine sugli sviluppatori" (2002) [126] è stata inserita una domanda in cui si chiedeva agli sviluppatori di indicare quali persone conoscessero, anche non personalmente, in una lista di nomi. I risultati, evidenziati nella tabella 3, mostrano che queste persone si possono dividere in tre gruppi chiaramente distinti:

**Tabella 3. Quanto sono noti gli sviluppatori importanti**

Sviluppatore	Conosciuto da
Linus Torvalds	96.5%
Richard Stallman	93.3%
Miguel de Icaza	82.1%
Eric Raymond	81.1%
Bruce Perens	57.7%

Sviluppatore	Conosciuto da
Jamie Zawinski	35.8%
Mathias Ettrich	34.2%
Jörg Schilling	21.5%
Marco Pesenti Gritti	5.7%
Bryan Andrews	5.6%
Guenther Bartsch	3.5%
Arpad Gereoffy	3.3%
Martin Hoffstede	2.9%
Angelo Roulini	2.6%
Sal Valliger	1.2%

- Un primo gruppo di persone con chiare connotazioni filosofiche e storiche nel mondo del software libero (anche se, come sappiamo, possono anche essere dotati di notevoli competenze tecniche):
- 1) Linus Torvalds. Creatore del kernel di Linux, il sistema operativo libero più usato, e co-autore di *Solo per divertirsi: storia di un rivoluzionario per caso* [217].
  - 2) Richard Stallman. Ideologo e fondatore della Fondazione Software Libero e sviluppatore di vari progetti GNU. Autore di diversi importanti saggi sul mondo del software libero ("Perché *software libero* è meglio di *open source*", 1998 [206], "Copyleft: idealismo pragmatico", 1998 [205], "Il Progetto GNU" [208] e "Il Manifesto GNU", 1985 [117]).
  - 3) Miguel de Icaza. Co-fondatore del progetto GNOME e di Ximian Inc., e sviluppatore di parte di GNOME e di MONO.
  - 4) Eric Raymond. Promotore della Open Source Initiative, autore di "La cattedrale ed il Bazar" [192] e principale sviluppatore di Fetchmail.
  - 5) Bruce Perens. In passato leader del progetto Debian, promotore (convertito) della Open Source Initiative e sviluppatore dello strumento e-fence.
  - 6) Jamie Zawinsky. Ex sviluppatore di Mozilla e famoso per una lettera del 1999, in cui abbandonava il progetto Mozilla sostenendo che il modello che stavano seguendo non avrebbe mai portato frutti ("Rassegnazione e autopsia", 1999) [237].
  - 7) Mathias Ettrich. Fondatore di KDE e sviluppatore di LyX e altri.

- Un secondo gruppo che consiste di sviluppatori. Questo sondaggio ha preso i nomi degli sviluppatori principali dei sei progetti più famosi secondo l'indice di applicazioni di software libero FreshMeat. Si può vedere che (ad eccezione di Linus Torvalds, per ovvie ragioni, e di Jörg Schilling) il livello di notorietà di questi sviluppatori è ridotto:
  - 1) Jörg Schilling, creatore di cdrecord, tra altre applicazioni.
  - 2) Marco Pesenti Gritti, principale sviluppatore di Galeon.
  - 3) Bryan Andrews, sviluppatore di of Apache Toolbox.
  - 4) Guenther Bartsch, creatore di Xine.
  - 5) Arpad Gereoffy, sviluppatore di MPEGPlayer.
- Un terzo gruppo, che raccoglie i nomi delle ultime tre "persone" nella tabella. Questi nomi sono stati inventati dai creatori del sondaggio per controllare il margine di errore.

Possiamo trarre due conclusioni dai risultati: la prima è che il margine di errore si può considerare ridotto (meno del 3%), e la seconda è che la maggior parte degli sviluppatori delle più famose applicazioni di software libero sono noti quanto persone che non esistono. Questo dato dovrebbe far riflettere coloro che presumono che una delle principali ragioni per sviluppare software libero sia la ricerca della notorietà.

#### **4.8. Riassunto e conclusioni**

Questo capitolo ha cercato di far luce sul fenomeno largamente sconosciuto delle persone che sviluppano software libero. In generale possiamo dire che lo sviluppatore di software libero sia un giovane, maschio, con una qualifica universitaria (o in procinto di ottenerla). La relazione tra il mondo del software libero e le università (studenti e professori) è molto stretta, anche se gli sviluppatori che non hanno nulla a che spartire con l'ambiente accademico sono ancora la maggioranza.

In termini di ore dedicate allo sviluppo di software libero abbiamo mostrato che esiste un'enorme disparità, simile a quella postulata dalla legge di Pareto. Le motivazioni degli sviluppatori, secondo la loro stessa opinione, sono ben lontane dall'essere venali o egocentriche, come gli economisti e gli psicologi tendono a suggerire, ma sono molto più legate all'apprendere e condividere. Infine abbiamo mostrato una tabella delle personalità più significative nel mondo del software libero (inclusendo altri che non erano altrettanto noti,

come abbiamo visto) e si è dimostrato che la reputazione, nell'enorme comunità del software libero, dipende in genere da molto di più che semplicemente programmare un'applicazione di software libero di successo.

## 5. Economia

"Res publica non dominetur."

"Le cose pubbliche non hanno padrone." (traduzione libera)

Apparsa in un annuncio di IBM su Linux (2003)

Questo capitolo tratta alcuni degli aspetti economici legati al software libero. Inizieremo descrivendo come vengono finanziati (se vengono finanziati, dal momento che spesso si basano esclusivamente sugli sforzi e le risorse dei volontari) i progetti di software libero. Quindi vedremo i principali modelli di business attuati dalle aziende direttamente associate al software libero. Il capitolo si conclude con una breve analisi delle relazioni tra il software libero e i monopoli dell'industria del software.

### 5.1. Finanziare i progetti di software libero

Il software libero viene sviluppato in tanti modi, usando meccanismi per ottenere fondi che variano enormemente di caso in caso. Ogni progetto di software libero ha il suo modo di autofinanziarsi, da quello che si serve solo di sviluppatori volontari e fondi ceduti altruisticamente, al progetto portato avanti da un'azienda che fattura il 100% dei costi ad un'organizzazione interessata nel prodotto sviluppato.

In questa sezione ci concentriamo sui progetti in cui esistono finanziamenti esterni e non tutto il lavoro è volontario. In questi casi esiste normalmente un qualche genere di flusso monetario da una fonte esterna al progetto, responsabile di procurare i fondi per il suo sviluppo. In questo modo il software libero che viene costruito si può considerare, in un certo senso, il prodotto di questo finanziamento esterno. Per questo motivo la fonte esterna è solita decidere (almeno in parte) come e per che cosa vengano spesi i fondi.

In un certo senso questo finanziamento esterno dei progetti liberi si può intendere come una forma di sponsorizzazione, anche se lo sponsor non ha alcun motivo per essere disinteressato (e in genere non lo è). Nelle prossime sezioni discuteremo i tipi più comuni di finanziamenti esterni. Tuttavia, parlando di questi, occorre tener presente che si tratta di solo alcuni dei modi in cui i progetti di software libero ottengono risorse. Ne esistono altri, il più importante dei quali (come abbiamo visto nel capitolo 4) è il lavoro di molti sviluppatori volontari.

### 5.1.1. Finanziamenti pubblici

Un tipo molto speciale di finanziamento per i progetti liberi è quello pubblico. L'ente finanziatore può essere direttamente il governo (locale, regionale, nazionale o addirittura sovra-nazionale) o un'istituzione pubblica (ad esempio una fondazione). In questi casi il finanziamento tende ad essere simile a quello per i progetti di ricerca e sviluppo, e in effetti è normale che questo tipo di finanziamenti provenga da enti pubblici che promuovono ricerca e sviluppo. In genere l'ente finanziatore non cerca di recuperare l'investimento (o almeno non direttamente), anche se tende ad avere obiettivi ben definiti (ad esempio promuovere la creazione di un tessuto industriale o basato sulla ricerca, promuovere una certa tecnologia o un certo tipo di applicazione, eccetera).

Nella maggior parte di questi casi non esistono finanziamenti esplicitamente indirizzati a prodotti o servizi legati al software libero, ma questi tendono ad essere il sotto-prodotto di un contratto con altri obiettivi più generali. Ad esempio, tra i suoi programmi di ricerca l'Unione Europea finanzia progetti mirati a migliorare la competitività dell'Europa in certi settori. Alcuni di questi progetti hanno tra i loro obiettivi quello di usare, migliorare e creare software libero nell'ambito della ricerca (come strumento di ricerca o come suo prodotto derivato).

Le motivazioni dietro questo tipo di finanziamento sono molto variegate, ma si possono distinguere le seguenti:

- 1) Scientifica. E' la motivazione più frequente nel caso dei progetti di ricerca finanziati con fondi pubblici. Anche se l'obiettivo non è produrre software, ma investigare un determinato settore (che può essere legato o meno alle tecnologie informatiche), è probabile che occorra sviluppare dei programmi che serviranno da strumenti per ottenere gli obiettivi del progetto. In genere il progetto non è interessato a commercializzare tali strumenti, o potrebbe addirittura interessarsi attivamente a far sì che altri gruppi li usino e li migliorino. In tali casi è normale distribuirli come software libero. In questo modo il gruppo che conduce la ricerca ha impiegato parte dei fondi per produrre il software, perciò si può dire che esso sia stato sviluppato con fondi pubblici.
- 2) Promuovere degli standard. Avere un'implementazione di riferimento è il modo migliore per promuovere uno standard. In molti casi questo implica avere programmi che formino parte di tale implementazione (o, se lo standard si riferisce al settore del software, che siano essi stessi l'implementazione). Affinché l'implementazione di riferimento serva a promuovere lo standard, occorre che sia disponibile, almeno per controllarne l'interoperabilità per tutti coloro che desiderino sviluppare prodotti che applicano quello standard. E in molti casi è anche consigliabile che i produttori siano in grado di adattare direttamente l'implementazione di riferimento per usarla con i loro prodotti, se lo desiderano. In questo modo

sono stati sviluppati, per esempio, i protocolli Internet, che ormai sono diventati una norma universale. In questi casi, rilasciare l'implementazione di riferimento come software libero può contribuire enormemente a tale promozione. Ancora una volta, il software libero qui è un sotto-prodotto, nel caso della promozione di uno standard. E normalmente il soggetto responsabile di tale promozione è un ente pubblico (benché a volte possa essere un consorzio privato).

- 3) Sociale. Il software libero è uno strumento molto interessante per creare l'infrastruttura di base della società dell'informazione. Le organizzazioni interessate a usare software libero per promuovere l'accesso universale alla società dell'informazione possono finanziare progetti ad esso correlati (normalmente attraverso progetti mirati a sviluppare nuove applicazioni o adattare quelle già esistenti).

#### **Nota**

Un esempio di finanziamento pubblico per un obiettivo primariamente sociale è il caso di LinEx, creato dal Governo Regionale dell'Estremadura (Estremadura, Spagna) per promuovere la società dell'informazione essenzialmente in termini di alfabetizzazione informatica. Il Governo Regionale ha finanziato lo sviluppo di una distribuzione basata su Debian al fine di raggiungere questo obiettivo. Un altro caso analogo è il finanziamento da parte del governo tedesco di alcuni sviluppi GnuPG, mirati a rendere il sistema più facile da usare anche dagli utenti inesperti, con l'idea di promuovere l'uso della posta elettronica sicura tra i cittadini.

#### **Lo sviluppo di GNAT**

Un noto caso di finanziamento pubblico per lo sviluppo di software libero è quello del compilatore GNAT. GNAT, il compilatore Ada, fu finanziato dal progetto Ada 9X del Dipartimento della Difesa degli Stati Uniti, con l'idea di avere un compilatore della nuova versione del linguaggio di programmazione Ada (che in seguito sarebbe diventato Ada 95), che all'epoca si stava cercando di promuovere. Una delle cause identificate in merito all'adozione della prima versione di Ada (Ada 83) da parte delle aziende di software era il ritardo nella disponibilità di un compilatore del linguaggio e il suo alto costo quando finalmente fu rilasciato. Perciò cercarono di evitare che la stessa cosa si ripetesse con Ada 95, facendo in modo che il compilatore fosse pronto quasi in simultanea con l'uscita del nuovo standard del linguaggio.

A questo scopo Ada 9X stipulò un progetto con una squadra dell'Università di New York (NYU), per un valore approssimativo di un milione di dollari americani, per svolgere un "test concettuale" del compilatore Ada 95 compiler. Con questi finanziamenti, e sfruttando l'esistenza di GCC (il compilatore C di GNU, del quale fu usata la maggior parte del back end), la squadra della NYU riuscì in effetti a costruire il primo compilatore Ada 95, che rilasciò sotto la licenza GPL di GNU. Il compilatore ebbe un tale successo che, al termine del progetto, alcuni dei suoi creatori fondarono un'azienda (Ada Core Technologies), che da allora è diventata leader di mercato nei compilatori e negli strumenti che aiutano a costruire programmi in Ada.

In questo progetto vale la pena osservare la combinazione di elementi di ricerca (di fatto questo progetto fece avanzare le conoscenze sulla costruzione di front ends e sistemi run time per compilatori di tipo Ada) e promozione di standards (che era l'obiettivo più evidente dell'ente finanziatore).

### **5.1.2. Finanziamenti privati non-profit**

Questo tipo di finanziamenti ha molte caratteristiche simili al precedente, che normalmente è condotto da fondazioni o organizzazioni non governative. Le motivazioni dirette in questi casi tendono ad essere la produzione di software per l'uso in un campo ritenuto particolarmente rilevante dall'ente finanziato-



re, ma si può anche trovare la motivazione indiretta di contribuire alla soluzione di un problema (ad esempio, una fondazione che promuove la ricerca su una malattia potrebbe finanziare la costruzione di un programma di statistica che aiuti ad analizzare i gruppi sperimentali usati come parte della ricerca in quella malattia).

In generale, sia le motivazioni che i meccanismi per questo tipo di finanziamento sono molto simili a quelli del finanziamento pubblico, anche se naturalmente operano sempre nel contesto degli obiettivi dell'ente finanziatore.

#### **Nota**

Probabilmente l'archetipo della fondazione promotrice dello sviluppo di software libero è la Free Software Foundation (Fondazione Software Libero, o FSF). Fin dalla metà degli anni Ottanta questa fondazione si è dedicata a promuovere il progetto GNU e ad incoraggiare lo sviluppo del software libero in generale.

Un altro caso interessante, benché in un settore piuttosto differente, è la Open Bioinformatics Foundation. uno degli obiettivi di questa fondazione è promuovere dello sviluppo di programmi informatici di base per la ricerca in qualsiasi branca della bioinformatica. In generale la fondazione promuove questo obiettivo finanziando e contribuendo alla creazione di programmi liberi.

### **5.1.3. Finanziamenti da chi ha bisogno di miglioramenti**

Un altro tipo di finanziamento per lo sviluppo di software libero, non così altruista, avviene quando qualcuno ha bisogno di migliorare un prodotto libero. Ad esempio un'azienda potrebbe aver bisogno, per uso interno, di un certo programma con una particolare funzionalità, o di correggere determinati bachi. In questi casi è tipico per l'azienda in questione assumere qualcuno che sviluppi quanto richiesto. Spesso quanto sviluppato è software libero (o perché la licenza del programma modificato lo impone, oppure per decisione dell'azienda).

#### **Il caso di Corel e Wine**

Verso la fine degli anni Novanta Corel decise di portare i suoi prodotti su GNU/Linux. Durante questo processo scoprì che un programma libero, progettato per facilitare l'esecuzione di codici binari per Windows in ambienti Linux, avrebbe potuto aiutare a risparmiare notevolmente sullo sviluppo. Per far questo, però, il programma doveva essere migliorato, fondamentalmente aggiungendo le emulazioni di alcune funzionalità di Windows usate dai programmi Corel.

Corel incaricò di questo lavoro Macadamian, che contribuì i suoi miglioramenti al progetto Wine. In questo modo sia Corel che Wine ne furono avvantaggiati.

### **5.1.4. Finanziamento con vantaggi correlati**

Con questo tipo di finanziamento l'ente finanziatore mira ad ottenere vantaggi attraverso i prodotti legati al programma di cui finanzia lo sviluppo. Normalmente, in questi casi, i benefici ottenuti dai finanziatori non sono esclusivi, dal momento che altri possono entrare sul mercato e vendere i prodotti

correlati, ma o la quota di mercato catturata è sufficiente a eliminare le preoccupazioni relative al condividere la "torta" con altri, oppure il finanziatore ha un evidente vantaggio competitivo.

Alcuni esempi di prodotti legati ad un particolare software sono i seguenti :

- Libri. L'azienda in questione vende manuali, guide utente, materiale per corsi di formazione ecc., relativi al programma libero che aiuta a finanziare. E' chiaro che anche altre aziende possono vendere libri correlati, ma normalmente finanziare il progetto dà all'azienda accesso anticipato, prima della concorrenza, agli sviluppatori principali, o semplicemente fornisce un'immagine positiva dell'azienda nei confronti della comunità di utenti del programma in questione.
- Hardware. Se un'azienda finanzia lo sviluppo di sistemi liberi per un certo tipo di hardware, può dedicarsi con maggiore agio alla vendita di quel tipo di hardware. Anche in questo caso, siccome il software sviluppato è libero, potrebbero comparire concorrenti che vendono lo stesso tipo di dispositivi, usando gli stessi software senza aver contribuito al loro finanziamento. Ma anche così l'azienda in questione ha diversi vantaggi sulla concorrenza, uno dei quali è che, essendo nella posizione di chi finanzia il progetto, essa può influenzarlo in modo da far cadere la priorità sulle parti da sviluppare che le interessano maggiormente.
- CD con programmi. Probabilmente il modello più noto di questo tipo è quello delle aziende che finanziano certi sviluppi che poi applicano alle proprie distribuzioni di software. Ad esempio, avere un buon ambiente desktop può aiutare molto a vendere un CD con una certa distribuzione di GNU/Linux, perciò finanziarne lo sviluppo potrebbe essere un buon affare per chi vende i CD.

Occorre tener presente che sotto questo titolo raccogliamo i tipi di finanziamento fatti per ottenere un profitto, perciò l'ente finanziatore deve ottenere un potenziale vantaggio dal finanziamento. Tuttavia, in realtà si tratta spesso di una combinazione di motivi di lucro e altruistici, quando un'azienda fornisce finanziamenti per lo sviluppo di un programma libero dal quale si aspetta di trarre vantaggi indiretti.

**Note**

Un noto caso di contributi finanziari, benchè piuttosto indiretti, ad un progetto, è l'aiuto che la casa editrice O'Reilly dà allo sviluppo di Perl. Naturalmente non è un caso che O'Reilly sia anche tra i principali editori di pubblicazioni relative a Perl. In ogni caso è ovvio che O'Reilly non ha l'esclusiva sulla pubblicazione di libri di questo tipo e che altre case editrici gli fanno concorrenza in questo segmento di mercato, con gradi diversi di successo.

VA Software (in origine VA Research e più avanti VA Linux) ha collaborato attivamente allo sviluppo del kernel di Linux. In questo modo ha ottenuto, tra i vari vantaggi, la continuità assicurata, che è stata particolarmente critica per l'azienda nelle relazioni con i suoi clienti quando il suo business principale era la vendita di strumentazione con GNU/Linux pre-installato.

Red Hat ha finanziato lo sviluppo di diversi componenti di GNOME, ottenendo essenzialmente un ambiente desktop per la sua distribuzione, che ha contribuito ad aumentare le vendite. Come nei casi precedenti, altri produttori di distribuzioni hanno potuto beneficiare di questo sviluppo, anche se molti di essi non hanno collaborato al progetto GNOME quanto Red Hat (e un discreto numero non ha collaborato per niente). Nonostante questo, Red Hat trae benefici dall'aver contribuito a GNOME.

**5.1.5. Finanziamento come investimento interno**

Alcune aziende sviluppano software libero direttamente come parte del loro modello di business. Ad esempio un'azienda potrebbe decidere di iniziare un nuovo progetto libero in un campo in cui ritiene che ci siano opportunità di business, con l'idea di ottenere poi un ritorno dall'investimento. Questo modello si può considerare una variante del precedente (finanziamento indiretto), in cui i "benefici correlati" sarebbero i vantaggi che l'azienda ottiene producendo il programma libero. Ma siccome in questo caso è il prodotto libero stesso che dovrebbe generare vantaggi, ci è sembrato opportuno dedicargli una sezione apposita.

Questo tipo di finanziamento dà origine a diversi modelli di business. Nell'analizzarli (sezione 5.2) illustreremo anche i vantaggi che un'azienda di solito ottiene da questo tipo di investimento in un progetto e quali metodi tende ad usare in modo da renderlo vantaggioso. In ogni caso però occorre far presente che, talvolta, il software in questione può essere sviluppato semplicemente per rispondere ad un bisogno dell'azienda, e che solo in seguito essa potrebbe decidere di rilasciarlo e, magari, di iniziare una linea di prodotti basati su di esso.

**Nota**

Digital Creations (ora Zope Corporation) è tra i più noti casi di aziende che si dedicano a sviluppare software libero aspettandosi di ottenere un ritorno dal proprio investimento. Il progetto libero su cui Zope investe maggiormente è un server di applicazioni che sta godendo di un certo successo. La sua storia con il software libero è iniziata quando l'allora Digital Creations stava cercando capitale di rischio per sviluppare un server di applicazioni proprietario, intorno al 1998. Uno dei gruppi più interessati ad investire su di loro (Opticality Ventures) stabilì come condizione che il prodotto risultante fosse libero, perché in caso contrario non vedevano come avrebbero potuto ottenere una quota di mercato significativa. Digital Creations accettò questo approccio e pochi mesi dopo annunciò la prima versione di Zope (cambiando nome pochi anni dopo). Attualmente, Zope Corporation è specializzata in servizi di consulenza, formazione e assistenza su sistemi di gestione di contenuti basati su Zope, e altri prodotti di cui Zope è senza dubbio la testata d'angolo.

Ximian (in precedenza Helix Code) è un noto caso di sviluppo di applicazioni libere in un ambiente di mercato. Strettamente legata fin dalle sue origini al progetto GNOME, Ximian ha prodotto sistemi software come Evolution (un gestore di informazioni personali che include una funzionalità piuttosto simile Microsoft Outlook), Red Carpet (un sistema di facile uso per gestire pacchetti su un sistema operativo) e MONO (un'implementazione di una buona parte di .NET). L'azienda fu fondata nell'ottobre 1999 e attrasse molti sviluppatori da GNOME, che entrarono a far parte del suo team di sviluppo (pur continuando in molti casi a collaborare al progetto GNOME). Ximian si è posizionata come un'azienda ingegneristica specializzata in adattamenti di GNOME, nella creazione di applicazioni basate su GNOME e, in generale, nella fornitura di servizi basati sul software libero, in particolare strumenti legati all'ambiente desktop. Nell'agosto del 2003 Ximian è stata acquistata da Novell.

Cisco Enterprise Print System (CEPS) (<http://ceps.sourceforge.net/>) [17] è un sistema di gestione stampe per organizzazioni che usano un grande numero di stampanti. E' stato sviluppato all'interno di Cisco per rispondere alle sue esigenze e nel 2000 è stato liberato sotto la licenza GPL di GNU. E' difficile sapere con certezza quali fossero le ragioni di Cisco per fare questo, ma potrebbero essere legate a trovare contributi esterni (segnalazione di errori, nuovi controller, aggiornamenti, ecc.). In ogni caso è chiaro che, siccome Cisco non aveva alcuna intenzione di commercializzare il prodotto e il suo mercato potenziale era poco definito, nel prendere questa decisione non aveva molto da perdere.

### 5.1.6. Altre modalità di finanziamento

Esistono altre modalità di finanziamento che è difficile classificare sotto le categorie precedenti. Come esempi potremmo citare i seguenti:

- Uso del mercato per mettere in contatto sviluppatori e clienti. L'idea che sostiene questo tipo di finanziamento è che, specialmente per piccoli lavori di sviluppo, è difficile per un cliente che abbia bisogno di un determinato sviluppo entrare in contatto con uno sviluppatore in grado di protarlo a termine in maniera efficiente. Per migliorare questa situazione sono stati creati dei mercati di sviluppo del software libero, in cui gli sviluppatori possono pubblicizzare le loro competenze e i clienti possono annunciare i lavori di sviluppo di cui hanno bisogno. Uno sviluppatore e un cliente raggiungono un accordo; abbiamo una situazione simile a quella già descritta in "finanziamenti da chi ha bisogno di miglioramenti" (sezione 5.1.3).

#### SourceXchange

SourceXchange è un esempio di mercato che mette in contatto gli sviluppatori con i potenziali clienti. Per pubblicizzare un progetto, un cliente presentava una richiesta di proposta, o RFP (*request for proposal*), specificando il lavoro di cui aveva bisogno e le risorse che era disposto a impiegare per svilupparlo. Queste RFP venivano pubblicate sul sito. Quando uno sviluppatore ne leggeva una a cui era interessato, faceva un'offerta. Se uno sviluppatore e un cliente trovavano un accordo sui termini del lavoro, iniziava un progetto. Normalmente ogni progetto era supervisionato da un *peer reviewer*, ossia un supervisore incaricato di assicurare che lo sviluppatore aderisse alle specifiche - e che le specifiche fossero sensate, di dare consigli su come portare a termine il progetto, ecc. SourceXchange (di proprietà dell'azienda CollabNet) aveva il compito di mettere a disposizione il sito, garantire le competenze dei supervisori, assicurare il pagamento quando i progetti venivano completati e offrire strumenti di monitoraggio (servizi che fatturava al cliente). Il primo progetto mediato attraverso SourceXchange fu completato nel marzo del 2000, ma il sito chiuse i battenti poco più di un anno dopo, nell'aprile del 2001.

- Finanziamento di progetti attraverso la vendita di obbligazioni. L'idea alla base di questo tipo di finanziamento è simile a quella del normale mercato obbligazionario cui accedono le aziende, ma mirata a sviluppare software libero. Ha diverse varianti, ma una delle più note opera nel modo seguente. Quando uno sviluppatore (individuo o azienda) ha un'idea per

un nuovo programma, o per migliorare un programma esistente, la scrive sottoforma di specifiche, con una stima dei costi dello sviluppo, ed emette obbligazioni per la sua realizzazione. Il valore di queste obbligazioni è attuato solo se il progetto alla fine viene completato. Quando lo sviluppatore ha venduto un sufficiente numero di obbligazioni inizia lo sviluppo, finanziato con prestiti basati sulle obbligazioni stesse. Quando lo sviluppo viene completato e un terzo soggetto, indipendente, certifica che le specifiche sono state rispettate, lo sviluppatore "attua" le obbligazioni, paga i debiti, e quello che rimane è il profitto del lavoro di sviluppo.

Chi potrebbe essere interessato ad acquistare queste obbligazioni? Ovviamente, gli utenti che desiderano che il nuovo programma, o il miglioramento ad un programma esistente, venga realizzato. Questo sistema di obbligazioni permette ai soggetti interessati di stabilire entro certi limiti le priorità degli sviluppatori (almeno in parte), attraverso l'acquisto di obbligazioni. Questo vuol dire anche che i costi dello sviluppo non devono pesare interamente su di una sola azienda, ma possono essere divisi tra varie aziende (ed individui), che tra l'altro devono pagare soltanto se il progetto alla fine viene completato con successo. Un meccanismo simile viene proposto in maniera molto più dettagliata in "Il protocollo dell'artista di Wall Street. Usare obbligazioni di completamento di software per finanziare lo sviluppo di software open source" ("The Wall Street performer protocol. Using software completion bonds to fund open source software development"), di Chris Rasch (1991) [191].

## Bibliografia

Il sistema di obbligazioni descritto si basa sul *protocollo dell'artista di strada* ("The street performer protocol", in: *Third USENIX Workshop on Electronic Commerce Proceedings*, 1998 [152], e "Il protocollo dell'artista di strada e il copyright digitale" o "The street performer protocol and digital copyrights", 1999 [153]), un meccanismo basato su e-commerce progettato per facilitare finanziamenti privati a creazioni libere. In breve, chiunque sia interessato in un certo lavoro si impegnerebbe formalmente a pagare una determinata somma se il lavoro viene completato e rilasciato come libero. L'obiettivo è quello di trovare un modo per finanziare lavori relativamente piccoli che vengono messi a disposizione di tutti, ma che possono essere estesi in molti modi (uno dei quali è rappresentato dalle obbligazioni per la realizzazione di software libero). Possiamo vedere un piccolo caso in cui viene messa in pratica una derivazione di questo protocollo, il *protocollo dell'artista di strada razionale* (Paul Harrison, *The rational street performer protocol*, 2002, [137]), in cui [http://www.csse.monash.edu.au/~pjh/circle/funding\\_results.html](http://www.csse.monash.edu.au/~pjh/circle/funding_results.html) lo si applica per ottenere i fondi destinati a finanziare parte di The Circle, un progetto di software libero.

- Cooperative di sviluppatori. In questo caso gli sviluppatori di software libero, anziché lavorare da soli o per un'azienda, si uniscono in qualche tipo di associazione (in genere simile a una cooperativa). In tutti gli altri aspetti questa funziona come un'azienda, con una sfumatura di impegno etico per il software libero, che potrebbe far parte degli statuti dell'organizzazione (sebbene anche una normale azienda possa fare la stessa cosa). In questo tipo di organizzazioni possiamo vedere una varietà di combinazioni di lavoro volontario e lavoro remunerato. Un esempio è Free Developers.
- Sistema di donazioni. Questo implica creare un meccanismo che permetta di pagare l'autore di un determinato software, attraverso la pagina web che ospita il progetto. In questo modo gli utenti che desiderano che il progetto

continui a rilasciare nuove versioni possono sostenerlo economicamente, versando offerte volontarie destinate a finanziare gli sviluppatori.

## 5.2. Modelli di business basati sul software libero

Oltre ai meccanismi di finanziamento dei progetti descritti sinora, un ulteriore aspetto economico cui vale la pena fare cenno sono i modelli di business. Nel parlare dei meccanismi finanziari ne abbiamo già menzionati alcuni. In questa sezione li illustreremo in maniera più sistematica.

In generale possiamo dire che si stanno esplorando molti modelli di business attorno al software libero, alcuni più tradizionali ed altri più innovativi. Dobbiamo tener presente che, tra i modelli più diffusi nell'industria del software, non è facile servirsi di quelli basati sulla vendita di licenze, perché nel mondo del software libero questo modello è molto difficile da sfruttare. Tuttavia si possono usare quelli basati su servizi resi a terzi, con il vantaggio che è possibile offrire assistenza completa su un programma senza necessariamente esserne i produttori.

### Vendita di software libero per un tanto a copia

Nel mondo del software libero farsi pagare per le licenze di utilizzo è difficile, ma non impossibile. In generale non vi è nulla nelle definizioni di software libero che impedisca ad un'azienda di creare un prodotto e distribuirlo solo a chi paga una certa somma. Ad esempio un particolare produttore potrebbe decidere di distribuire il suo prodotto con una licenza libera, ma solo a chi paga 1000 Euro a copia (come nel classico mondo del software proprietario).

Tuttavia, benché teoricamente possibile, nella pratica è difficile che questo succeda, perché una volta che il produttore ha *venduto* la prima copia, chiunque la riceva potrebbe essere motivato a cercare di recuperare l'investimento rivendendo più copie a un prezzo minore (cosa che non può essere impedita dalla licenza del programma, se è libera). Nell'esempio precedente, uno potrebbe tentare di vendere dieci copie a 100 Euro l'una, il che significa che il prodotto potrebbe finire per diventare gratuito (tra l'altro questo renderebbe molto difficile, per il produttore iniziale, riuscire a vendere un'altra copia a 1000 Euro, dal momento che si potrebbe ottenere legalmente il prodotto a un decimo del prezzo). E' facile vedere come questo processo possa continuare a cascata fino a quando le copie verranno vendute ad un prezzo vicino al costo marginale della produzione di una copia, che con le tecnologie attuali è praticamente zero.

Nonostante questo, e tenendo presente che il meccanismo descritto significa che normalmente un produttore non possa apporre un prezzo (specialmente un prezzo alto) sulla mera redistribuzione di un programma, esistono modelli di business che implicitamente fanno proprio questo. Un esempio è il caso delle distribuzioni GNU/Linux, che vengono vendute ad un prezzo molto minore rispetto ai concorrenti proprietari, ma al di sopra (e di solito molto al di sopra) del costo della copia (anche quando è possibile scaricarle gratuitamente da Internet). Certamente in questi casi entrano in gioco altri fattori, come l'immagine del marchio o la comodità per il consumatore. Ma questo non è l'unico caso. Perciò, anziché dire che il software libero "non si può vendere un tanto a copia", dovremmo dire che farlo è più difficile, e che probabilmente genererà minori profitti, ma che possono esistere modelli basati proprio su questo.

Dati questi limiti (e questi vantaggi), per molti anni si sono tentate variazioni sui normali modelli di business dell'industria del software, mentre al tempo stesso si stanno cercando altre soluzioni più innovative per sfruttare le pos-

sibilità offerte dal software libero. Senza dubbio nei prossimi anni vedremo sempre più esperimenti in questo campo, e avremo anche maggiori informazioni su quali modelli possano funzionare e in quali situazioni.

In questa sezione offriamo un panorama dei modelli di business che si incontrano più di frequente al giorno d'oggi, divisi in gruppi in modo da mostrare al lettore ciò che hanno in comune e ciò che li distingue, con particolare attenzione a quelli basati sullo sviluppo e sui servizi che circondano un prodotto di software libero. I ricavi, in questo caso, provengono direttamente dalle attività di sviluppo e dai servizi per il prodotto, ma non implicano necessariamente lo sviluppo di nuovi prodotti. Quando avviene questo tipo di sviluppo, tali modelli considerano il finanziamento di prodotti di software libero come un *sottoprodotto*, il che significa che si tratta di modelli particolarmente interessanti, con un impatto potenzialmente molto vasto sul mondo del software libero in generale.

In ogni caso, e sebbene la classificazione offerta in queste pagine sia relativamente chiara, non bisogna dimenticare che quasi tutte le aziende in realtà si servono di varie combinazioni dei modelli che illustriamo, mescolati tra loro e ad altri più tradizionali.

### **5.2.1. Maggiore conoscenza**

L'azienda che segue questo modello di business cerca di far fruttare la propria conoscenza di un prodotto (o una serie di prodotti) di software libero. I suoi ricavi provengono dai clienti ai quali vende servizi legati a tale conoscenza: sviluppi basati sul prodotto, modifiche, adattamenti, installazioni e integrazioni con altri prodotti. Il vantaggio competitivo dell'azienda sarà strettamente legato alla sua maggiore conoscenza del prodotto: perciò l'azienda sarà posizionata particolarmente bene se ne è la produttrice, o se partecipa attivamente al progetto di produzione.

Questa è una delle ragioni per cui le aziende che usano questo modello tendono a partecipare attivamente ai progetti legati al software attorno al quale cercano di vendere servizi: è un modo molto efficiente per ottenere conoscenze su di esso e, cosa più importante, per fare in modo che tali conoscenze vengano riconosciute. E' chiaro che poter dire ad un cliente che tra gli impiegati dell'azienda vi sono vari sviluppatori impegnati nel progetto che produce il software di cui, ad esempio, il cliente richiede una modifica, tende ad essere una buona garanzia.

## Relazione con i progetti di sviluppo

Perciò le aziende di questo tipo hanno tutto l'interesse a trasmettere un'immagine di conoscenza approfondita di determinati prodotti liberi. Un risultato interessante è che il loro supporto ai progetti di software libero (ad esempio partecipandovi attivamente, o permettendo agli impiegati di contribuirvi anche durante l'orario di lavoro) non è quindi puramente filantropico. Al contrario, potrebbe essere uno dei più proficui punti di forza dell'azienda, dal momento che i clienti lo valuteranno molto positivamente come chiaro segno che l'azienda conosce bene il prodotto in questione. Inoltre, in questo modo l'azienda sarà in grado di seguire da vicino il processo di sviluppo, cercando di assicurarsi che, ad esempio, i miglioramenti richiesti dai suoi clienti divengano parte del prodotto sviluppato dal progetto.

Analizzando il fenomeno da un punto di vista più generale, questa è una situazione in cui entrambe le parti, l'azienda e il progetto, sono avvantaggiate dalla collaborazione. Il progetto trae vantaggi dal lavoro di sviluppo portato avanti dall'azienda, o dal fatto che alcuni dei suoi sviluppatori vengono remunerati (almeno in parte) per il loro contributo al progetto. L'azienda trae vantaggi in termini di conoscenza del prodotto, immagine nei confronti dei clienti e un certo grado di influenza sul progetto.

La gamma di servizi offerti da questo tipo di aziende può essere molto ampia, ma generalmente consiste nello sviluppo di personalizzazioni, adattamenti o integrazioni dei prodotti dei quali l'azienda è esperta, o in servizi di consulenza in cui clienti sono aiutati ad usare al meglio il prodotto in questione (specialmente se è un prodotto complesso, o se il suo corretto funzionamento è di importanza critica per il cliente).

## Esempi

Tra gli esempi di aziende che hanno usato entro certi limiti questo modello di business vi sono i seguenti:

- LinuxCare ( <http://www.linuxcare.com> ) [45]. Fondata nel 1996, inizialmente forniva servizi di consulenza e assistenza negli Stati Uniti per GNU/Linux e il software libero; il suo personale consisteva principalmente di esperti di GNU/Linux. Tuttavia alcuni anni dopo i suoi obiettivi sono cambiati: da allora si è specializzata nel fornire servizi quasi esclusivamente a GNU/Linux operante su virtual machines z/VM su grossi computer IBM. Anche il suo modello di business è cambiato, trasformandosi in "maggiori conoscenze con restrizioni", dal momento che, come parte fondamentale dei suoi servizi, offre un'applicazione non libera: Levanta.
- Alcôve ( <http://www.alcove.com> ) [3]. Fondata in Francia nel 1997, offre principalmente servizi di consulenza sul software libero, consulenza strategica, assistenza e sviluppo. Fin dalla sua fondazione, Alcôve ha mantenuto tra i suoi dipendenti gli sviluppatori di vari progetti liberi, cercando di ottenere da questo un ritorno di immagine. Ha cercato anche di costruirsi un'immagine, in generale, di azienda legata alla comunità del software libero, ad esempio collaborando con le associazioni di utenti e pubblicizzando le sue collaborazioni a progetti liberi (ad esempio tramite gli Alcôve-Labs <http://www.alcove-labs.org> [4]).

### 5.2.2. Maggiori conoscenze con restrizioni

Questi modelli sono simili a quelli descritti nella sezione precedente, ma tentano di limitare i potenziali concorrenti. Mentre nei modelli *puri* basati sulle maggiori conoscenze chiunque può, in linea di principio, diventare parte della concorrenza, dal momento che il software usato è lo stesso (ed è libero), in questo caso il tentativo è quello di evitare tale situazione ponendo barriere nei confronti della concorrenza. Le barriere tendenzialmente consistono in brevetti o licenze proprietarie, che normalmente influenzano una parte piccola



(ma fondamentale) del prodotto sviluppato. Per questo motivo tali modelli possono in realtà essere considerati misti, nel senso che si trovano a metà strada tra il software libero e il software proprietario.

In molti casi la comunità del software libero sviluppa la sua propria versione, il che significa che il vantaggio competitivo dell'azienda può scomparire, o addirittura rivoltarsi contro l'azienda stessa, se il *concorrente* libero diventa lo standard di mercato e viene richiesto anche dai clienti dell'azienda.

### Esempi

Esistono molti casi che usano questo modello di business, dal momento che viene considerato in genere meno rischioso del modello *puro* basato sulle maggiori conoscenze. Tuttavia le aziende che l'hanno usato si sono evolute in modi diversi. Tra di esse vi sono le seguenti:

- Caldera ( <http://www.sco.com> ) [16]. La storia di Caldera è complessa. Inizialmente essa creò la sua distribuzione di GNU/Linux, indirizzata alle imprese: Caldera OpenLinux. Nel 2001 acquistò la divisione di Unix da SCO, e nel 2002 cambiò nome per diventare SCO Group. La sua strategia di business è cambiata tanto frequentemente quanto il suo nome, dal suo totale sostegno a GNU/Linux, alle cause intentate contro IBM e Red Hat nel 2003, fino ad abbandonare la sua stessa distribuzione. Ma in relazione a questa sezione, il modello di business di Caldera, almeno fino al 2002, è un chiaro esempio di maggiori conoscenze con restrizioni. Caldera cercò di sfruttare la sua conoscenza della piattaforma GNU/Linux, limitando però la potenziale concorrenza da affrontare attraverso l'inclusione di software proprietario nella sua distribuzione. Questo rese difficile per i suoi clienti cambiare distribuzione dopo averla adottata, perché anche se le altre distribuzioni di GNU/Linux comprendevano la parte libera di Caldera OpenLinux, non includevano la parte proprietaria.
- Ximian ( <http://ximian.com/> ) [74]. Fondata nel 1999 con il nome di Helix Code da sviluppatori strettamente legati al progetto GNOME, nell'agosto 2003 fu acquisita da Novell. La maggior parte del software sviluppato da questa azienda è libero (e generalmente fa parte di GNOME). Tuttavia, in un ambito molto specifico, Ximian ha deciso di dare ad un componente una licenza di software proprietario: il Connector for Exchange. Questo modulo permette ad uno dei suoi prodotti di maggior successo, Evolution (un gestore di informazioni personali che comprende e-mail, agenda, calendario, ecc.), di interagire con i server di Microsoft Exchange, che sono generalmente usati dalle grandi organizzazioni. In questo modo Ximian ha cercato di competere godendo di un certo vantaggio sulle altre aziende che offrivano servizi basati su GNOME, magari con i prodotti sviluppati dalla stessa Ximian, che però non erano in grado di interagire così bene con Exchange. Ad eccezione di questo prodotto, il modello di Ximian è stato del tipo "maggiori conoscenze" e si è basato anche sull'essere la sorgente di un programma libero (come vedremo più avanti). In ogni caso questo componente è diventato libero nel 2005.

### 5.2.3. Sorgente di un prodotto libero

Questo modello è simile a quello basato su maggiori conoscenze, ma con una specializzazione, nel senso che l'azienda che lo usa è il produttore della quasi totalità di un prodotto libero. Naturalmente il vantaggio competitivo aumenta se si è gli sviluppatori del prodotto in questione, ne si controlla l'evoluzione e lo si ha a disposizione prima della concorrenza. Tutto ciò mette l'azienda in una posizione molto forte nei confronti dei clienti che stanno cercando servizi relativi a quel programma. Si tratta inoltre di un modello molto interessante in termini di immagine, dal momento che l'azienda ha dimostrato il suo potenziale di sviluppo creando e mantenendo l'applicazione in questione, il che può tornare molto utile quando si tratta di convincere un cliente delle capacità dell'azienda. Allo stesso tempo permette di costruire un'immagine

positiva anche nei confronti della comunità del software libero in generale, dal momento che questa riceve dall'azienda un nuovo prodotto libero, che diventa di dominio pubblico.

### Esempi

Molti prodotti liberi hanno iniziato ad essere sviluppati all'interno di un'azienda, che molto spesso ha continuato a dirigere i loro successivi sviluppi. Tra gli esempi vorremmo citare i seguenti casi:

- Ximian. Abbiamo già accennato a come abbia in parte usato il modello delle maggiori conoscenze conrestrizioni. In generale, però, Ximian ha seguito un chiaro modello basato sull'essere la sorgente di programmi liberi. I suoi programmi principali, come Evolution o Red Carpet, sono stati distribuiti sotto licenze GPL. Tuttavia altri non meno importanti, come MONO, sono distribuiti principalmente sotto licenze MIT X11 o LGPL. In ogni caso, Ximian ha sviluppato i suoi prodotti fin dall'inizio in modo quasi esclusivo. L'azienda ha cercato di ottenere un ritorno da questi lavori di sviluppo aggiudicandosi contratti per farli evolvere in determinati modi, adattandoli alle esigenze dei clienti e offrendo servizi di personalizzazione e manutenzione.
- Zope Corporation ( <http://www.zope.com/> ) [75]. Nel 1995 fu fondata Digital Creations, che sviluppò un prodotto proprietario per la gestione di piccoli annunci pubblicitari su web. Nel 1997 ricevette un'iniezione di capitale da parte di, tra le altre, un'impresa di venture capital chiamata Opticality Ventures. L'aspetto strano (per l'epoca) di questo investimento era la condizione che le fu imposta: distribuire come software libero il prodotto evoluto, che in seguito divenne Zope, uno dei sistemi di gestione contenuti di maggior successo su Internet. Da allora il modello di business dell'azienda è stato di produrre Zope e prodotti correlati, offrendo servizi di personalizzazione e manutenzione per tutti questi prodotti. Zope Corporation ha saputo anche creare attorno ai suoi prodotti una dinamica comunità di sviluppatori di software libero, con i quali collabora attivamente.

### 5.2.4. Sorgente di un prodotto con restrizioni

Questo modello è simile al precedente, ma implica il prendere alcune misure per limitare la concorrenza o per massimizzare i profitti. Tra le restrizioni più comuni troviamo le seguenti:

- Distribuzione proprietaria per un periodo limitato, quindi libera. Con o senza la promessa di una futura distribuzione libera, ogni nuova versione del prodotto viene venduta come software proprietario. Dopo un determinato periodo di tempo (di solito quando viene rilasciata una versione più recente, anch'essa proprietaria), la vecchia versione viene distribuita con una licenza libera. In questo modo l'azienda produttrice ottiene profitti dai clienti interessati alle nuove versioni, limitando allo stesso tempo la concorrenza, dal momento che ogni azienda che voglia competere usando quel prodotto può farlo solo con la versione libera (disponibile solo quando viene rilasciata una nuova versione, di solito migliorata e più completa).
- Distribuzione limitata per un determinato periodo. In questo caso il software è libero dal primo momento in cui viene distribuito. Siccome però nulla nella licenza libera obbliga a distribuire il programma a chiunque lo desideri (la persona in possesso del software può decidere se farlo oppure no), il produttore per un certo periodo lo distribuisce solo ai suoi clienti, che lo pagano (in genere attraverso un contratto di manutenzione). Dopo

un certo periodo il produttore lo distribuisce a tutti, ad esempio mettendolo in un file accessibile pubblicamente. In questo modo il produttore ottiene un ricavo dai suoi clienti, che percepiscono la disponibilità anticipata del programma come un valore aggiunto. E' chiaro che il modello funziona solo se i clienti a loro volta non rendono pubblico il programma subito dopo averlo ricevuto. Con certi tipi di clienti questo accade piuttosto raramente.

In generale in questi casi le aziende sviluppatrici ottengono i vantaggi accennati sopra, ma non a costo zero. A causa del ritardo con cui il prodotto viene reso disponibile alla comunità del software libero, è praticamente impossibile che i suoi membri possano contribuire allo sviluppo, perciò il produttore non trarrà molti benefici dai contributi esterni.

### Esempi

Alcune aziende che usano questo modello di business sono le seguenti:

- artofcode LLC ( <http://artofcode.com/> ) [9]. Dal 2000 artofcode vende Ghostscript in tre versioni (in precedenza Alladin Enterprises aveva fatto lo stesso con un modello simile). La versione più recente viene distribuita come AFPL Ghostscript, sotto una licenza proprietaria (che permette l'uso e la distribuzione non commerciale). La versione successiva (più o meno con un anno di ritardo) viene distribuita come GNU Ghostscript, sotto la licenza GPL di GNU. Ad esempio nell'estate 2003 la versione AFPL è la 8.11 (rilasciata il 16 agosto), mentre la versione GNU è la 7.07 (distribuita come tale il 17 marzo, ma la sua versione equivalente sotto licenza AFPL è datata 2002). artofcode offre inoltre una terza versione, con una licenza proprietaria che permette di integrarla a prodotti non compatibili con la GPL di GNU (in questo caso usa un modello doppio, che descriveremo più avanti).
- Ada Core Technologies ( <http://www.gnat.com/> ) [2]. Fu fondata nel 1994 dagli autori del primo compilatore Ada 95, il cui sviluppo fu parzialmente finanziato dal Governo degli Stati Uniti e si basava su GCC, il compilatore di GNU. Fin dall'inizio i suoi prodotti sono stati software libero. La maggior parte di essi però viene prima offerta ai suoi clienti, inclusa in un contratto di manutenzione. Ad esempio il suo compilatore, che continua ad essere basato su GCC ed è distribuito sotto la GPL di GNU, viene offerto ai suoi clienti come GNAT Pro. Ada Core Technologies non offre in alcun modo questo compilatore al pubblico in generale e normalmente non se ne trovano versioni in rete. Tuttavia, dopo un intervallo di tempo variabile (di circa un anno), Ada Core Technologies offre le versioni *pubbliche* del suo compilatore, molto simili ma senza alcun tipo di assistenza, in un anonimo file FTP.

### 5.2.5. Licenze speciali

In questi modelli l'azienda produce un prodotto che distribuisce sotto due o più licenze. Almeno una di queste è per software libero, ma le altre sono solitamente licenze proprietarie, che permettono di vendere il prodotto in modo più o meno tradizionale. In genere queste vendite sono completate con la vendita di servizi di consulenza e di sviluppo relativi al prodotto. Ad esempio un'azienda può distribuire un prodotto come software libero sotto la licenza GPL di GNU, ma offrire anche una versione proprietaria (simultaneamente e senza ritardo tra le due versioni) per coloro che non vogliono sottostare alle condizioni della GPL, ad esempio perché desiderano integrare il prodotto con un software proprietario (cosa che la GPL non permette).

Sleepycat Software ( <http://www.sleepycat.com/download/oslicense.html> ) [60]. Questa azienda fu fondata nel 1996 e ha annunciato di aver ottenuto profitti fin dall'inizio (il che è davvero notevole per un'azienda legata al software libero). I suoi prodotti, tra cui Berkeley DB (un gestore dati molto diffuso, perché lo si può incorporare facilmente in altre applicazioni), sono distribuiti sotto una licenza libera che specifica che, in caso di integrazione con un altro prodotto, occorre rendere accessibile il codice sorgente di entrambi. Sleepycat offre servizi di consulenza e sviluppo per i suoi prodotti, che offre anche sotto licenze che permettono di incorporarli senza dover distribuire il codice sorgente. Chiaramente lo fa sotto un contratto specifico e, in generale, in un regime di vendite di software proprietario. Nel 2005 Sleepycat Software è stata acquisita da Oracle.

### 5.2.6. Vendita basata sul marchio

Benché sia possibile ottenere prodotti molto simili a prezzi notevolmente inferiori, molti clienti sono disposti a pagare di più per acquistare un prodotto di *marca*. Questo principio viene adottato da aziende che investono per crearsi un marchio con un'immagine positiva e ben affermata, che permette loro di vendere prodotti liberi con un sufficientemente margine di guadagno. In molti casi non vendono solo i prodotti, ma anche servizi che il cliente accetterà come ulteriore valore aggiunto.

I casi più noti di questo modello di business sono le aziende che vendono distribuzioni GNU/Linux. Queste aziende cercano di vendere qualcosa che generalmente è possibile ottenere a prezzi molto più bassi in rete (o da altre fonti con marchi meno rinomati). Devono perciò fare in modo che i consumatori riconoscano il loro marchio e siano disposti a pagare i costi aggiuntivi. Per far questo non si limitano ad investire in pubblicità, ma offrono anche vantaggi oggettivi (ad esempio una distribuzione ben assemblata, o un canale di distribuzione che offre vicinanza al cliente). Inoltre tendono ad offrire un maggior numero di servizi attorno al prodotto (dal training a programmi di certificazione da parte di terzi), per sfruttare al massimo l'immagine del proprio marchio.

Red Hat ( <http://www.redhat.com> ) [56]. Red Hat Linux iniziò ad essere distribuito nel 1994 (l'azienda iniziò ad essere conosciuta con il suo nome attuale nel 1995). Per molto tempo Red Hat è riuscita ad affermare il proprio nome come la distribuzione GNU/Linux per eccellenza (anche se alla metà del decennio del 2000 condivide questa posizione con altre aziende, come OpenSUSE, Ubuntu e forse Debian). Per diversi anni Red Hat ha venduto e continua a vendere ogni tipo di servizi relativi alla distribuzione, a GNU/Linux e al software libero in generale.

## 5.3. Altre classificazioni di modelli di business

La letteratura sul software libero fornisce altre classificazioni dei modelli di business tradizionali. Eccone alcuni esempi.

### 5.3.1. La classificazione di Hecker

La classificazione proposta in "Metter su bottega: il business del software open source" ("Setting up shop: the business of open source software"; Frank Hecker, 1998) [141] fu usata specialmente per pubblicizzare la Open Source Initiative; fu anche uno dei primi tentativi di classificazione dei tipi di business che stavano emergendo a quell'epoca. Tuttavia include diversi modelli che hanno

poco a che vedere con il software libero (nei quali il software libero è poco più che un compagno del modello principale). In ogni caso i modelli che descrive sono i seguenti:

- *Venditore di assistenza* (vendita di servizi associati al prodotto). L'azienda produce un prodotto libero (che ha sviluppato o contribuisce attivamente a sviluppare) e vende servizi di consulenza o adattamento a requisiti specifici.
- *Leader della perdita* (vendita di altri prodotti proprietari). In questo caso il programma libero è usato per promuovere in qualche modo la vendita di altri prodotti proprietari ad esso collegati.
- *Glassa sui dispositivi* (vendita di hardware). Il business principale è la vendita di hardware e il software libero è considerato un'aggiunta che può aiutare l'azienda ad ottenere un vantaggio competitivo.
- *Accessori* (vendita di accessori). Si vendono prodotti legati al software libero, come libri, strumenti per il computer, ecc.
- *Abilitatore di servizi* (vendita di servizi). Il software libero serve per creare un servizio (generalmente accessibile online) dal quale l'azienda ottiene profitti.
- *Licenze sul marchio* (vendita del marchio). Un'azienda registra dei marchi che riesce ad associare a programmi liberi, probabilmente sviluppati dall'azienda stessa. Quindi ottiene ricavi facendosi pagare per autorizzare l'uso di tali marchi registrati.
- *Vendilo, liberalo*. Questo è un modello simile al *leader della perdita*, ma si svolge in modo ciclico. Prima un prodotto viene messo sul mercato come software libero. Se ottiene un relativo successo, la versione seguente viene distribuita come software proprietario per un certo periodo, trascorso il quale diventa libera. A quel punto è già iniziata la distribuzione di una nuova versione proprietaria, e così di seguito.
- *Software franchising*. Un'azienda concede in franchising l'uso dei suoi marchi in relazione a un particolare programma libero..

**Nota**

I lettori si saranno accorti che questa classificazione è piuttosto diversa da quella che abbiamo proposto, e che tuttavia alcune delle categorie rispecchiano quasi completamente alcune delle nostre. .

#### 5.4. Impatto sulle situazioni di monopolio

Il mercato del software tende al predominio di un prodotto in ciascuno dei suoi segmenti. Gli utenti vogliono far rendere al massimo lo sforzo profuso nell'imparare ad usare un programma, le aziende vogliono assumere dipendenti che conoscano già il loro software, e tutti vogliono che i dati che maneggiano siano utilizzabili anche dalle aziende e dalle persone con cui lavo-

rano. Per questo motivo un'iniziativa che progetti di rompere una situazione in cui *di fatto* un prodotto domina chiaramente il mercato è destinata a rafforzare tale situazione: se ha successo, il nuovo prodotto prenderà il posto del precedente e si avrà presto un nuovo prodotto dominante. Solo i cambiamenti tecnologici producono, per un breve periodo, un'instabilità sufficiente a fare in modo che nessuno abbia un netto perdominio.

Il fatto che esista un prodotto dominante però non conduce necessariamente alla creazione di un monopolio di mercato. Ad esempio il petrolio è un prodotto che domina quasi il mercato dei carburanti per auto private, ma (nel libero mercato del petrolio) esistono molte aziende che producono e molte aziende che distribuiscono il medesimo prodotto. In realtà, quando si parla di software, la situazione in cui ci si deve preoccupare è quando un prodotto riesce a dominare il mercato perché ha un solo possibile fornitore. Il software libero offre un'alternativa a questa situazione: i prodotti liberi possono essere promossi da una determinata azienda, ma quell'azienda non ne ha il controllo, o almeno non ha gli stessi gradi di controllo a cui siamo stati abituati dal software proprietario. Nel mondo del software libero un prodotto dominante non implica necessariamente il monopolio di una sola azienda. Al contrario, indipendentemente dal prodotto che domina il mercato, diverse aziende possono competere per fornirlo, migliorarlo, adattarlo ai bisogni dei clienti e offrire servizi ad esso correlati.

#### **5.4.1. Elementi che favoriscono i prodotti dominanti**

Nel software per computer è normale che esista un prodotto nettamente dominante in ciascun segmento di mercato. Ed è normale per diverse ragioni, tra cui vorremmo sottolineare le seguenti:

- **Formati di dati.** In molti casi il formato dei dati è strettamente legato ad un'applicazione. Quando esiste un numero sufficientemente grande di persone che lo usano, quel formato di dati diventa lo standard *de facto* e la pressione per la sua adozione (e quindi per l'adozione dell'applicazione corrispondente) è fortissima.
- **Catene di distribuzione.** In genere, quando si inizia ad usare un programma, uno dei problemi è ottenerne una copia. E di solito è difficile trovare programmi che non sono leader di mercato nel loro settore. Le catene di distribuzione sono costose da mantenere, il che significa che per i concorrenti minori è difficile raggiungere i negozi di elettronica in cui l'utente finale possa comprare i loro prodotti. Per il prodotto dominante invece è molto più facile: il primo interessato ad avere il prodotto sarà proprio il negozio stesso.
- **Marketing.** La pubblicità "gratuita" che un prodotto ottiene una volta usato da una parte significativa della popolazione è enorme. Anche il "passaparola" funziona molto bene quando chiediamo e scambiamo informa-

zioni con le persone che conosciamo. Ma soprattutto è enorme l'impatto dei media: le riviste di elettronica faranno riferimento a un prodotto più e più volte, se sembra essere il più usato; ci saranno corsi per imparare ad usarlo, libri che lo descrivono, interviste con gli utenti, ecc.

- Investimenti sul training. Una volta che tempo e denaro sono stati investiti per imparare ad usare uno strumento, c'è una forte motivazione a non cambiare strumento. Inoltre, spesso si tratta dello strumento che già domina il mercato, perché è più facile trovare persone e risorse che aiutino ad insegnare come si usa.
- Software pre-installato. Ricevere un calcolatore con un software già installato è sicuramente un forte incentivo ad usarlo, anche se deve essere pagato separatamente. E in genere il tipo di software che il venditore del computer è disposto a pre-installare sarà solo quello più usato.

#### **5.4.2. Il mondo del software proprietario**

Nel mondo del software proprietario, la comparsa di un prodotto dominante in qualsiasi segmento di mercato è equivalente ad un monopolio da parte dell'azienda che lo produce. Ad esempio abbiamo queste situazioni di monopolio *de facto* (o quasi) di un prodotto o di un'azienda nel mercato dei sistemi operativi, di desktop publishing, basi di dati, graphic design, programmi di videoscrittura, fogli di calcolo, ecc.

E questo accade perché l'azienda in questione ha un enorme potere sul prodotto dominante, tanto che solo loro possono definire la sua evoluzione, le linee fondamentali del suo sviluppo, la sua qualità, ecc. Gli utenti hanno pochissimo potere, perché (per le ragioni elencate sopra) sono decisamente poco motivati a provare altri prodotti. Per questo motivo non c'è molto che possano fare, a parte provare a sfidare la posizione del prodotto dominante cercando di migliorare i loro prodotti (nel tentativo di neutralizzare appunto le ragioni di cui sopra), di solito con scarso successo.

Questa situazione mette l'intero settore nelle mani della strategia dell'azienda dominante. Tutti gli attori dipendono da essa e perfino lo sviluppo della tecnologia software in quel campo sarà mediato dai miglioramenti che l'azienda sceglie di fare per il suo prodotto. In termini generali, questa è una situazione in cui emergono gli effetti peggiori di un monopolio, in particolare la mancanza di motivazione da parte dell'azienda dominante ad adattare i prodotti ai bisogni (sempre in evoluzione) dei clienti, dal momento che essi sono diventati un mercato "captive".

### 5.4.3. La situazione con il software libero

Nel caso del software libero, invece, un prodotto dominante non si traduce automaticamente in un monopolio di mercato. Se il prodotto è libero, qualsiasi azienda può lavorare su di esso, migliorarlo, adattarlo alle esigenze dei clienti e in generale aiutare ad evolverlo. Inoltre, proprio per via della sua posizione dominante, ci saranno molte aziende interessate a lavorare su di esso. Se il produttore "originale" (ossia l'azienda che inizialmente ha sviluppato il prodotto) vuole rimanere sul mercato, dovrà competere con tutte loro e sarà quindi fortemente motivato a far evolvere il proprio prodotto esattamente nella direzione voluta dagli utenti. Certo, avrà il vantaggio di conoscere meglio il programma, ma questo non è sufficiente. Dovrà competere con la concorrenza per ogni singolo cliente.

Perciò la comparsa di prodotti dominanti nel mercato del software libero si traduce in una maggiore concorrenza tra le aziende. E con essa gli utenti riacquistano potere: le aziende in competizione non possono fare altro che ascoltarli, se vogliono sopravvivere. Ed è esattamente questo che assicura che il prodotto venga migliorato.

#### **Prodotti liberi che dominano il loro settore**

Per molto tempo Apache è stato leader di mercato tra i web server. Esistono però diverse aziende dietro Apache, alcune molto grandi (come IBM) ed altre assai più piccole. E tutte queste non hanno altra scelta che competere cercando di migliorarlo, e in genere contribuendo al progetto con i loro miglioramenti. Nonostante Apache sia quasi monopolista in molti settori (ad esempio è praticamente l'unico web server considerato sulla piattaforma GNU/Linux o \*BSD), esso non dipende da una sola azienda, ma letteralmente da dozzine di aziende.

Anche le distribuzioni di GNU/Linux sono un caso interessante. GNU/Linux non è certamente un monopolio, ma è forse il secondo più scelto nel mercato dei sistemi operativi. E questo non ha forzato una situazione in cui un'azienda ha assoluto potere su di esso. Al contrario, esistono decine di distribuzioni fatte da aziende diverse, che competono liberamente sul mercato. Ciascuna cerca di offrire miglioramenti che i concorrenti devono adottare, a rischio di essere tagliati fuori. Inoltre non ci si può allontanare troppo da quello che è lo "standard GNU/Linux", perché questo verrebbe rifiutato dagli utenti come una "deviazione dalla norma". La situazione, dopo anni di quote di mercato in crescita per GNU/Linux, mostra decine di aziende che si fanno concorrenza e permettono così al sistema di evolvere. E, ancora una volta, tutte cercano di soddisfare i requisiti degli utenti. Questo infatti è l'unico modo che permette loro di rimanere sul mercato.

GCC è un prodotto dominante nel mondo dei compilatori C e C++ per il mercato GNU/Linux. E tuttavia questo non ha portato ad una situazione di monopolio di mercato, anche se Cygnus (ora Red Hat) è stata per molto tempo incaricata di coordinarne lo sviluppo. Esistono molte aziende che apportano miglioramenti al sistema e tutte competono, ciascuna nella sua specifica nicchia, per soddisfare le richieste dei loro utenti. Di fatto, quando una determinata compagnia ha fallito nel compito di coordinare (o alcuni utenti ritengono che l'abbia fatto), è stato possibile *biforcare* il progetto, con due prodotti che per un certo periodo si sono sviluppati in parallelo, fino a quando non sono stati riuniti (come sta succedendo adesso con GCC 3.x).

### 5.4.4. Strategie per diventare monopolista con il software libero

Sebbene il mondo del software libero sia molto più ostile ai monopoli rispetto al mondo del software proprietario, esistono strategie che un'azienda può impiegare per cercare di avvicinarsi ad una situazione di predominio monopolista sul mercato. Queste pratiche sono abituali in molti altri settori dell'economia



e per impedirle esistono enti che regolano la concorrenza, ragion per cui non entreremo troppo nel dettaglio su di esse. Tuttavia ne citiamo una che, entro certi limiti, è specifica del mercato del software e che è già stata sperimentata in alcune situazioni: accettare la certificazione di un prodotto da parte di terzi.

Quando un'azienda desidera distribuire un prodotto software (libero o proprietario) che funzioni in combinazione con altri, è normale "certificare" quel prodotto per una determinata combinazione. Il produttore si impegna ad offrire servizi (aggiornamenti, assistenza, soluzione di problemi, ecc.) solo se il cliente garantisce che il prodotto venga usato in un ambiente certificato. Ad esempio il produttore di un programma di gestione banche dati può certificare il proprio prodotto per una determinata distribuzione GNU/Linux e per nessun'altra. Ciò implica che i suoi clienti dovranno usare quella distribuzione GNU/Linux o scordarsi di ricevere assistenza dal produttore (cosa che, se il prodotto è proprietario, può essere impossibile nella pratica). Se un determinato produttore riesce ad ottenere una posizione di chiaro predominio come prodotto certificato da terzi, gli utenti non avranno altra scelta se non usare quel prodotto. Se in quel segmento la certificazione è importante, ci troveremo nuovamente di fronte ad una situazione di monopolio.

#### **Nota**

Entro certi limiti, nel mercato delle distribuzioni GNU/Linux si iniziano a intravedere alcuni casi di situazioni che tendono verso un monopolio *de facto* attraverso le certificazioni. Ad esempio molti produttori di prodotti proprietari certificano i loro prodotti solo per una certa distribuzione GNU/Linux (molto spesso Red Hat Linux). Al momento questo non porta al monopolio di alcuna azienda, il che potrebbe essere dovuto al fatto che, nel mercato delle distribuzioni GNU/Linux, la certificazione non è così importante per gli utenti. Ma solo il futuro può svelare se ad un certo punto questa situazione si avvicinerà ad un monopolio *de facto*.

In ogni caso è importante tener presenti due commenti rispetto a quanto sopra. Il primo è che queste posizioni di monopolio non sono facili da raggiungere, e in ogni caso si raggiungono attraverso meccanismi generalmente "non-software" (a differenza della situazione di prodotto dominante, che come abbiamo visto è relativamente frequente e si raggiunge tramite meccanismi puramente legati alle tecnologie informatiche e alle loro modalità di utilizzo). Il secondo è che, se tutto il software usato è libero, questa strategia ha scarse probabilità di successo (o addirittura nessuna). Un produttore potrebbe riuscire a fare in modo che molte aziende certifichino in favore dei suoi prodotti, ma i clienti avranno sempre la possibilità di cercare assistenza e servizi da altre aziende, diverse da quelle interessate dalla certificazione, se lo ritengono opportuno.

## 6. Il software libero e le amministrazioni pubbliche

"[...] per essere accettabile da parte dello Stato, il software non deve solo essere tecnicamente in grado di svolgere un'operazione, ma le condizioni del contratto devono rispettare una serie di requisiti rispetto alle licenze, senza i quali lo Stato non può garantire ai propri cittadini che i loro dati vengano elaborati in maniera adeguata, nel dovuto rispetto della privacy e in modo da rimanere accessibili nel tempo, poiché questi sono aspetti estremamente critici del suo regolare servizio."

Edgar David Villanueva Núñez (lettera di risposta al manager generale di Microsoft Peru, 2001)

Le istituzioni pubbliche, sia quelle in grado di emanare leggi che quelle impegnate ad amministrare lo Stato (le "pubbliche amministrazioni"), svolgono un ruolo molto importante quando si tratta di adottare e promuovere tecnologie. Benché nel 2000 queste istituzioni non mostrassero quasi alcun interesse per il fenomeno del software libero (ad eccezione di pochi casi isolati), da allora la situazione ha iniziato a cambiare. Da un lato, molte amministrazioni pubbliche hanno iniziato ad usare software libero come parte delle loro infrastrutture tecnologiche. Dall'altro lato, nel loro ruolo di promotrici della società dell'informazione, alcune hanno iniziato a promuovere direttamente o indirettamente lo sviluppo e l'uso del software libero. Inoltre alcuni enti legislativi hanno iniziato a prestare attenzione (poco a poco) al software libero, a volte favorendone lo sviluppo, a volte impedendolo, altre volte semplicemente prendendo nota della sua esistenza.

Prima di entrare nel dettaglio, è importante ricordare che per molto tempo il software libero è stato sviluppato senza esplicito sostegno (o addirittura interesse) da parte delle istituzioni pubbliche. Per questo motivo, l'attenzione che sta ottenendo di recente da molte di esse non è priva di controversie, equivoci e problemi. Inoltre, negli ultimi anni, le iniziative relative agli standard aperti stanno sempre più prendendo piede, il che porta a prendere misure (più o meno direttamente) associate al software libero.

In questo capitolo cercheremo di descrivere la situazione attuale e le peculiarità del software libero in relazione alla sfera "pubblica".

### 6.1. Impatto sulle pubbliche amministrazioni

Sono state svolte diverse ricerche sull'uso del software libero nelle pubbliche amministrazioni (ad esempio, "Software open source per le amministrazioni pubbliche", o "Open source software for the public administration", 2004 [159]; "Software open source nell'e-Government; analisi e raccomandazioni delineate da un gruppo di lavoro del comitato danese per le tecnologie", 2002 [180]; "Software libero / open source: opportunità per una società dell'informazione in Europa?", o "Free software / open source. Information society opportunities for Europe?", 1999 [132], e "Perché i governi dovrebbero

promuovere il software open source", o "The case for government promotion of open source software", 1999 [213]). In seguito discuteremo alcune delle ricerche più notevoli (sia in positivo che in negativo).

### **6.1.1. Vantaggi ed effetti positivi**

Alcuni dei vantaggi dell'usare il software libero nelle pubbliche amministrazioni e le principali nuove prospettive che esso offre sono:

#### **1) Sviluppare l'industria locale**

Uno dei maggiori vantaggi del software libero è la possibilità di sviluppare un'industria locale del software. Quando usiamo il software proprietario, tutto ciò che si spende in licenze finisce direttamente nelle mani del produttore, e l'acquisto ne rafforza la posizione; questo non è necessariamente un male, ma non è molto efficiente per la regione alla quale l'Amministrazione è associata, se si considera l'alternativa di usare un programma libero.

In questo caso, le aziende locali saranno in grado di competere nel fornire servizi (e i programmi stessi) all'Amministrazione, sotto condizioni molto simili a quelle di qualsiasi altra azienda. Si può dire che in qualche modo l'Amministrazione livella il campo di gioco e renda più facile a tutti partecipare alla competizione. Ovviamente, "tutti" comprende le aziende locali, che hanno l'opportunità di sfruttare i propri vantaggi competitivi (migliore conoscenza dei bisogni del cliente, vicinanza geografica, ecc.).

#### **2) Indipendenza da un solo fornitore e mercato concorrenziale**

E' chiaro che qualsiasi organizzazione preferirebbe dipendere da un mercato concorrenziale, piuttosto che da un unico fornitore in grado di imporre le condizioni alle quali intende fornire il prodotto. Tuttavia, nel mondo della Pubblica Amministrazione, questa preferenza diventa un requisito di base, e in alcuni casi addirittura un obbligo legale. In generale, l'Amministrazione non può decidere di dare l'appalto ad un determinato fornitore, ma deve specificare i propri requisiti in modo tale che qualsiasi azienda interessata, la quale soddisfi certe caratteristiche ed offra il prodotto o servizio richiesto, possa candidarsi per l'appalto.

Come già accennato, nel caso del software proprietario ogni prodotto ha un solo fornitore (anche se questo usa un certo numero di intermediari). Se si specifica un determinato prodotto, l'Amministrazione deciderà anche a quale fornitore appaltarlo. E in molti casi è praticamente impossibile evitare di specificare un determinato prodotto, quando si tratta di programmi per computer. Motivi di compatibilità con altre organizzazioni, o possibilità di risparmiare sul training e sull'amministrazione, o varie altre ragioni fanno sì che sia normale per un'amministrazione decidere di usare un determinato prodotto.

L'unica via di uscita da questa situazione è rendere libero il prodotto specificato. In questo modo qualsiasi azienda interessata sarà in grado di fornire sia il programma che ogni tipo di servizio correlato (a dipendenza solo delle capacità dell'azienda e della sua conoscenza del prodotto). Inoltre, nel caso di questo tipo di appalto, l'Amministrazione può in futuro cambiare fornitore se lo desidera, senza alcun problema tecnico, perché anche se cambia fornitore continuerà comunque ad usare lo stesso prodotto.

### 3) Flessibilità e adattamento a requisiti specifici

Benché l'adattamento a requisiti specifici sia qualcosa di cui ogni organizzazione che usa i computer ha bisogno, le particolarità dell'Amministrazione lo rendono un fattore molto importante per il successo dell'impianto di un sistema di software. Nel caso del software libero l'adattamento è molto facilitato e, cosa ancora più importante, può contare su un mercato concorrenziale nel caso sia necessario ricorrere ad appalti.

Quando l'Amministrazione acquista un prodotto proprietario, per modificarlo in genere occorre raggiungere un accordo col produttore, il quale è l'unico soggetto legalmente (e spesso tecnicamente) in grado di farlo. In queste circostanze è difficile contrattare, specialmente se il produttore non è particolarmente interessato al mercato offerto da quella specifica amministrazione. Al contrario, se si usa un prodotto libero, l'Amministrazione può modificarlo a suo piacimento, se ha persone competenti all'interno del suo personale, oppure può rivolgersi ad esterni per la modifica. In linea di principio qualsiasi azienda dotata delle capacità e conoscenze necessarie può offrire questo servizio, il che significa che ci si può aspettare una certa concorrenza tra diverse aziende. Naturalmente questo tende ad abbassare i costi e ad aumentare la qualità.

#### **Il caso delle distribuzioni GNU/Linux**

Negli ultimi anni in Spagna accade di frequente che alcune comunità autonome si creino le proprie distribuzioni GNU/Linux. Questa tendenza è iniziata con GNU/Linux, ma attualmente ce ne sono molti altri. Benché alcuni esperti abbiano criticato l'esistenza di queste distribuzioni, si tratta di un chiaro esempio della flessibilità offerta dal software libero. Qualsiasi amministrazione pubblica può, con una spesa relativamente piccola, farsi adattare una distribuzione GNU/Linux secondo le proprie esigenze e preferenze, praticamente senza limiti. Ad esempio può cambiare l'aspetto del desktop, scegliere l'insieme di applicazioni e linguaggi di default, migliorare la localizzazione delle applicazioni, ecc. In altre parole: volendo, il desktop (e qualsiasi altro elemento del software funzionante sul computer) può essere adattato ai requisiti richiesti.

E' chiaro che questo adattamento comporterà una certa spesa, ma l'esperienza dimostra che lo si può ottenere a costi relativamente contenuti e la tendenza sembra indicare che sarà sempre più facile (e meno costoso) produrre distribuzioni personalizzate.

### 4) Adozione facilitata di standard aperti

Per la loro stessa natura, i programmi liberi spesso usano standard aperti (open), o non proprietari. Infatti, quasi per definizione, ogni aspetto di un programma libero che ci potrebbe interessare può essere riprodotto facilmente e perciò non è proprietario. Ad esempio i protocolli usati da un programma

libero per interagire con altri programmi possono essere studiati e riprodotti, cioè non sono proprietari. Inoltre, molto di frequente e nell'interesse dei progetti stessi, si tende ad usare standard aperti.

In ogni caso, indipendentemente dal motivo, è un dato di fatto che i programmi liberi generalmente usano standard non proprietari per scambiarsi i dati. I vantaggi di questo aspetto sono più estesi per le pubbliche amministrazioni che per qualsiasi altra organizzazione, dal momento che la promozione di standard proprietari (anche indiretta, tramite il loro uso) è molto più preoccupante nel loro caso. E l'uso di standard non proprietari è fondamentale in almeno un aspetto, ossia quando si tratta di interagire con i cittadini, dal momento che essi non dovrebbero essere costretti ad acquistare alcun prodotto da alcuna azienda in particolare, per poter interagire con l'amministrazione.

### 5) Controllo pubblico della sicurezza

Per un'amministrazione pubblica, essere in grado di garantire che i sistemi informatici facciano soltanto ciò che devono fare è un dovere fondamentale e in molte nazioni un obbligo legale. Spesso questi sistemi gestiscono informazioni private che potrebbero essere di interesse per altri soggetti (ad esempio dati fiscali, penali, elettorali, di censo, ecc.). Se si usa un'applicazione proprietaria, è difficile garantire che l'applicazione elaborerà i dati come dovrebbe. Anche se si fornisce l'accesso al suo codice sorgente, è molto difficile per una pubblica istituzione garantire che esso non contenga alcun codice *esterno*. Solo se questo compito può essere abitualmente e periodicamente affidato a terzi, e se in più qualsiasi soggetto interessato ha la possibilità di fare una verifica, l'Amministrazione può essere ragionevolmente certa che sta ottemperando a questo suo dovere fondamentale, o almeno sta prendendo tutte le misure che sono in suo potere per farlo.

### 6) Disponibilità nel lungo periodo

Molti dei dati elaborati dalle amministrazioni, così come i programmi usati per elaborarli, devono essere disponibili per decenni e per i decenni successivi. E' molto difficile garantire che un programma proprietario sarà disponibile dopo tutto quel tempo, specialmente se l'idea è che continui a funzionare su quella che sarà la piattaforma più comune in quel periodo del futuro. Al contrario, è possibile che per quell'epoca il produttore abbia perso interesse in quel prodotto e non l'abbia reso portabile sulle nuove piattaforme, o che sia disposto a farlo solo per una grossa cifra. Ancora una volta, dobbiamo tenere presente che solo il produttore può portare il prodotto, il che comporta che le negoziazioni saranno difficili. Nel caso del software libero, invece, l'applicazione è sicuramente disponibile, perciò chiunque può portarla sulla nuova piattaforma e farla funzionare secondo i bisogni dell'Amministrazione. Se questo non avviene spontaneamente, l'Amministrazione può sempre rivol-

gersi a diverse aziende cercando l'offerta migliore per questo incarico. Questo garantisce che l'applicazione e i dati che processa saranno disponibili al momento del bisogno.

## 7) Impatto oltre l'uso da parte dell'Amministrazione

Molte applicazioni usate o promosse dalle amministrazioni pubbliche sono usate anche in molti altri settori della società. Per questa ragione, ogni investimento pubblico sullo sviluppo di un prodotto libero porta benefici non solo all'amministrazione stessa, ma anche a tutti i suoi cittadini, che potranno usare il prodotto per svolgere le loro operazioni col computer, magari utilizzando i miglioramenti fatti dall'Amministrazione.

### Nota

Un caso molto particolare, ma con impatto enorme, che dimostra questo uso più oculato delle risorse pubbliche, è la localizzazione di un programma (ossia il suo adattamento agli usi e costumi di una comunità). Sebbene l'aspetto più visibile della localizzazione sia la traduzione del programma e della sua documentazione, molti altri aspetti ne sono influenzati (dall'uso del simbolo della moneta locale al presentare le date e gli orari nei formati usati dalla comunità in questione, all'uso di esempi nella documentazione e di espressioni adeguate ai costumi locali).

In ogni caso è chiaro che, se una pubblica amministrazione usa dei fondi per localizzare una particolare applicazione adeguandola ai propri bisogni, è più che probabile che tali bisogni coincidano con quelli dei suoi cittadini, il che genererà non solo un programma che soddisfa i suoi requisiti, ma anche uno che può essere reso disponibile ad ogni cittadino in grado di sfruttarlo al meglio, senza alcun costo aggiuntivo. Ad esempio, quando un'amministrazione finanzia l'adattamento di un programma per computer ad un linguaggio usato all'interno della comunità, potrà non solo usarlo nei propri uffici, ma anche offrirlo ai cittadini, con tutto ciò che ne consegue in termini di sviluppo della società dell'informazione.

## 6.1.2. Difficoltà nell'adozione e altri problemi

Tuttavia, benché usare il software libero comporti molti vantaggi per le amministrazioni, ci sono anche diverse difficoltà da affrontare al momento di metterlo in pratica. Tra queste vorremmo menzionare in particolare le seguenti:

### 1) Mancanza di conoscenze e di impegno politico

Il primo problema incontrato dal software libero nel cercare di entrare nelle amministrazioni è lo stesso problema di altre organizzazioni: il software libero è ancora un'entità sconosciuta da chi prende decisioni.

Fortunatamente si tratta di un problema che si sta risolvendo a poco a poco, anche se in molte sfere dell'amministrazione il software libero è ancora percepito come qualcosa di *strano*, cosicché le decisioni che lo riguardano sembrano ancora implicare un certo rischio.

Inoltre tendiamo a scontrarci con un problema di decisioni politiche. Il vantaggio principale del software libero per l'Amministrazione non è il costo (che è alto in ogni caso, specialmente quando si tratta di installarlo su un gran numero di postazioni di lavoro), ma, come accennato in precedenza, i principali benefici sono di tipo strategico. Per questo la decisione è più politica che tecnica. Senza la volontà politica di cambiare si-

### Bibliografia

I lettori interessati in una relazione sui vantaggi del software libero per l'amministrazione, scritta nel contesto degli Stati Uniti del 1999, possono consultare "Perché i governi dovrebbero promuovere il software open source" ("The case for government promotion of open source software", Mitch Stoltz, 1999) [213].

stemi software e la filosofia con cui sono acquisiti, è difficile fare progressi nell'installazione di software libero nell'Amministrazione.

## 2) Scarsa adattabilità delle procedure di acquisizione

Le procedure di acquisizione che l'Amministrazione usa al giorno d'oggi, dai tipici modelli dell'appalto pubblico al prezzo al dettaglio, sono progettate fondamentalmente per l'acquisto di prodotti informatici e non si adattano altrettanto bene al pagamento di servizi relativi ai programmi. Tuttavia, quando si usa software libero, in genere non c'è alcun prodotto da acquistare, oppure il prezzo è trascurabile. Al contrario, per sfruttare appieno le opportunità offerte dal software libero, è bene essere in grado di acquistare servizi su di esso. E' quindi necessario, per poter fare un uso serio del software libero, progettare procedure burocratiche che facilitino l'acquisizione in questi casi.

## 3) Mancanza di una strategia di installazione

Spesso un'amministrazione inizia ad usare il software libero semplicemente perché il prezzo d'acquisto è inferiore. In questi casi capita di frequente che il prodotto in questione venga inserito nel sistema informatico senza ulteriore pianificazione e, in generale, senza una strategia globale su come usare e sfruttare al meglio il software libero. Questo fa sì che la maggior parte dei suoi vantaggi si perdano lungo la strada, perché tutto si riduce all'uso di un *prodotto più economico*, mentre abbiamo già visto che, in generale, i benefici principali sono di altra natura.

Se a questo si aggiunge una cattiva progettazione del passaggio al software libero, il suo uso può causare notevoli costi aggiuntivi e, come vedremo in alcuni casi isolati, al di fuori di una struttura ben pianificata l'uso del software libero nell'Amministrazione può essere un frustrante insuccesso.

## 4) Scarsità o mancanza di prodotti liberi in certi segmenti di mercato

Quando si installa software libero in qualsiasi organizzazione, può capitare di non trovare alternative libere di qualità a certi tipi di applicazione. In questi casi la soluzione è complessa: l'unica cosa che si può fare è promuovere la produzione del programma libero che ci serve.

Fortunatamente le pubbliche amministrazioni sono nella posizione di poter considerare seriamente se possa essere di loro interesse promuovere, o addirittura finanziare o co-finanziare lo sviluppo di quel prodotto. Occorre tenere a mente che uno degli obiettivi della pubblica amministrazione è di offrire ai propri cittadini un migliore accesso alla società dell'informazione, ad esempio, o di promuovere il tessuto industriale locale. Sicuramente la creazione di vari programmi liberi avrà un impatto positivo su entrambi gli obiettivi, il che significa che dovremmo aggiungere al mero calcolo dei costi/benefici i vantaggi indiretti di una simile decisione.

## 5) Interoperabilità con i sistemi preesistenti

Capita raramente che il passaggio al software libero sia compiuto sull'intero sistema tutto allo stesso tempo. E' importante perciò che la parte

di software che vogliamo inserire continui a funzionare correttamente nel contesto di tutto l'altro software con il quale dovrà interagire. Questo è un problema ben noto, che interessa qualsiasi migrazione di software (anche con un prodotto proprietario), ma che può avere un impatto particolare nel caso del software libero. In ogni caso sarà qualcosa di cui tenere conto quando si pianifica la transizione. Fortunatamente, spesso il software libero da installare può essere adattato per interagire adeguatamente con gli altri sistemi, ma questo punto, se necessario, dovrà essere preso in considerazione quando si pianifica il costo della migrazione.

#### 6) Migrazione dei dati

Questo è un problema generico che interessa qualsiasi migrazione a nuove applicazioni che utilizzano formati di dati differenti, anche se si tratta di applicazioni proprietarie. In realtà nel caso del software libero questo problema spesso è mitigato dal fatto che, con uno sforzo particolare, è di solito possibile prevedere tutti i possibili formati e standard di scambio dati. In genere però è necessario far migrare anche i dati. E i costi per far questo sono alti. Per tale motivo, quando si calcolano i costi di una possibile migrazione su software libero, è necessario prendere attentamente in considerazione questo aspetto.

### 6.2. Le azioni delle pubbliche amministrazioni nel mondo del software libero

Le pubbliche amministrazioni influenzano il mondo del software libero in almeno tre modi:

- Comprando programmi e servizi correlati. Le amministrazioni, grandi utenti di software, giocano un ruolo fondamentale nel mercato del software.
- Promuovendo modi diversi di usare (e acquistare) alcuni programmi nella società. A volte la promozione è attuata offrendo incentivi economici (esenzioni fiscali, incentivi diretti, ecc.), talvolta attraverso campagne di informazione e raccomandazione, talvolta attraverso un "seguite il mio esempio"...
- Finanziando (direttamente o indirettamente) progetti di ricerca e sviluppo che determinano il futuro del software.

In ciascuno di questi ambiti il software libero può offrire vantaggi specifici (oltre a quelli già descritti nelle sezioni precedenti) di notevole interesse, sia per l'Amministrazione che per la società in generale.



### 6.2.1. Come soddisfare le esigenze delle amministrazioni pubbliche?

Le pubbliche amministrazioni sono grandi consumatori di prodotti informatici. Quando si tratta di software, in genere tendono ad acquistare prodotti di consumo di massa (*off-the-shelf*), ma anche sistemi personalizzati. Da questo punto di vista si possono considerare fondamentalmente come grossi centri d'acquisto, simili alle grandi aziende, ma con caratteristiche particolari. Ad esempio, in molti ambiti, le decisioni sugli acquisti delle pubbliche amministrazioni dovrebbero tenere in considerazione non solo parametri di costo e funzionalità, ma anche altri, come l'impatto dell'acquisto sul tessuto industriale o sociale, o considerazioni strategiche di lungo periodo, che possono anch'esse avere grande importanza.

In ogni caso al giorno d'oggi chi compra software commerciale, o "off-the-shelf", tende ad usare i prodotti proprietari del leader di mercato. La quantità di denaro pubblico speso da municipalità, comunità autonome, amministrazioni centrali e amministrazioni europee per acquistare Windows, Office o licenze di altri prodotti simili è certamente considerevole. A poco a poco però anche le soluzioni libere stanno iniziando a penetrare il mercato. Sempre più di frequente si iniziano a prendere in considerazione soluzioni basate su software libero per i server, mentre prodotti come OpenOffice e GNU/Linux con GNOME o KDE si usano sempre più negli ambienti desktop.

Che cosa si guadagna da questa migrazione verso il software libero? Per illustrare la risposta, consideriamo il seguente scenario. Supponiamo che, con una frazione di quello che le amministrazioni europee (o probabilmente le amministrazioni di un qualsiasi Paese sviluppato di media grandezza) spendono per due o tre prodotti proprietari "famosi", si possa lanciare un appalto pubblico per un'azienda (o due, o tre, o quattro) che migliorasse e adattasse i programmi liberi attualmente esistenti, in modo tale che in uno o due anni potrebbero essere resi disponibili per un pubblico di massa, almeno per alcune operazioni standard (se non lo sono già). Immaginiamo per esempio uno sforzo coordinato, su scala nazionale o europea, in cui tutte le amministrazioni partecipassero ad un consorzio responsabile per la gestione di questi appalti. In un tempo molto breve nascerebbe un'industria "locale" specializzata nel fare questi adattamenti e miglioramenti. E le amministrazioni potrebbero scegliere fra tre o quattro distribuzioni prodotte da tale industria. Per promuovere la concorrenza, ogni azienda potrebbe essere pagata in base a quante amministrazioni scelgono di usare la sua distribuzione. E l'intero risultato di questa operazione, trattandosi di software libero, sarebbe disponibile anche ad altre aziende ed utenti individuali, che in molti casi avrebbero esigenze simili a quelle delle amministrazioni.

Nel caso del software personalizzato, il processo abituale attualmente implica acquisire i programmi necessari da un'azienda all'interno di un modello proprietario. Qualsiasi programma sviluppato su richiesta dell'Amministrazione

rimane di proprietà dell'azienda che lo sviluppa. E di solito l'amministrazione acquirente è legata al fornitore in tutto ciò che riguarda miglioramenti, aggiornamenti e assistenza, in un circolo vizioso che rende difficile la libera concorrenza e rallenta il processo di modernizzazione delle pubbliche amministrazioni. Peggiora le cose il fatto che spesso il medesimo programma viene venduto e rivenduto alle varie amministrazioni, applicando in ciascun caso il prezzo intero, come se ogni volta il programma dovesse essere sviluppato da zero.

Proviamo di nuovo a immaginare come potrebbe andare diversamente. Un consorzio di pubbliche amministrazioni che ha bisogno di un particolare tipo di software personalizzato potrebbe richiedere che il risultato ottenuto sia software libero. Questo permetterebbe ad altre amministrazioni di trarre vantaggi da questo lavoro e, nel medio termine, esse potrebbero divenire interessate a collaborare con il consorzio, in modo che i loro specifici requisiti vengano presi in considerazione. Dal momento che il software risultante sarebbe libero, non esisterebbe alcun obbligo di affidare miglioramenti e adattamenti allo stesso fornitore, perché la concorrenza entrerebbe sul mercato (che al momento è quasi un mercato captive). E in ogni caso il costo finale per ciascuna delle amministrazioni coinvolte non supererebbe mai il costo dell'adozione di un modello proprietario.

Si tratta di scenari fantascientifici? Come vedremo più avanti, esistono timide iniziative in direzioni simili a quelle descritte. Oltre a favorire la fondazione e lo sviluppo di un'industria nell'area della pubblica amministrazione acquirente, il software libero offre ulteriori specifici vantaggi per il dominio pubblico. Ad esempio, è il modo più efficiente di sviluppare software nelle lingue delle minoranze (un problema fondamentale di molte pubbliche amministrazioni). Può anche contribuire fortemente a mantenere un'indipendenza strategica nel lungo periodo e ad assicurare che i dati custoditi dalle pubbliche amministrazioni rimangano accessibili a lungo. Per tutte queste ragioni, gli enti pubblici sono sempre più interessati a divenire utenti di software libero.

#### **Alcuni casi relativi alle amministrazioni tedesche**

Nel luglio 2003 fu rilasciata la prima versione stabile di Kolab, un prodotto del progetto Kroupware. Kolab è un sistema informatico libero per lavoro di gruppo (*groupware*) basato su KDE. Il motivo per cui lo menzioniamo è che questo progetto inizialmente era un appalto del Bundesamt für Sicherheit in der Informationstechnik (BSI - ufficio federale per la sicurezza informatica) del governo tedesco. Questo appalto cercava una soluzione che interoperasse con Windows e Outlook da una parte e GNU/Linux e KDE dall'altra. Tra le varie offerte fu scelta la proposta comune di tre aziende, Erfrakon, Intevation e Klarälvdalens Datakonsult, che offrivano di fornire una soluzione libera in parte basata su software già sviluppato dal progetto KDE, completato dal software libero sviluppato da loro: il risultato finale fu Kolab.

Nel maggio 2003 il municipio di Monaco (Germania) approvò la migrazione su GNU/Linux e applicazioni libere di suite da ufficio per tutti i computer desktop, in totale circa quattordicimila. La decisione non fu puramente di tipo economico: secondo le autorità, furono presi in considerazione anche aspetti strategici e di qualità. Nell'analisi complessiva svolta prima di prendere questa decisione, la soluzione che fu scelta alla fine (GNU/Linux più OpenOffice, fondamentalmente) ottenne 6,218 punti (su un massimo di diecimila), mentre la soluzione tradizionale basata su software di Microsoft ottenne poco più di cinquemila punti.

Nel luglio 2003 la Koordinierungs-und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung (KBSt), sotto il Ministero degli Interni tedesco, pubblicò il documento *Leitfaden für die Migration von Basissoftwarekomponenten auf Server-und Arbeitsplatzsystemen* [107] ('Guida alla migrazione dei principali componenti software dei server e delle postazioni di lavoro'), che offre agli enti pubblici tedeschi una serie di linee guida per migrare su soluzioni basate su software libero. Queste linee guida sono progettate affinché i decisori possano valutare se sia opportuno far migrare il sistema su software libero e come condurre la migrazione se si decide in questo senso.

### 6.2.2. Promuovere una società dell'informazione

Gli enti pubblici spendono somme consistenti in incentivi per incoraggiare l'acquisto di prodotti informatici. Si tratta di uno strumento formidabile per aiutare la diffusione delle tecnologie nella società. Tuttavia, si tratta anche di uno strumento pericoloso. Ad esempio potrebbe non essere un'idea particolarmente buona quella di promuovere l'uso di Internet da parte della società raccomandando un particolare browser e incoraggiando in questo modo il monopolio *de facto* di un'azienda, perché nel lungo periodo questo potrebbe avere un impatto negativo sulla società che si sta cercando di beneficiare.

Ancora una volta il software libero può essere d'aiuto in queste situazioni. Innanzitutto è neutrale rispetto ai produttori, perché nessuno possiede l'esclusiva su alcun programma libero. Se un'amministrazione desidera promuovere l'uso di una famiglia di prodotti liberi, può lanciare un appalto al quale ogni azienda del settore ha la possibilità di rispondere, per gestirne la consegna ai cittadini, i miglioramenti, l'aggiunta di funzionalità, ecc. In secondo luogo, può essere di grande aiuto sugli aspetti economici. Ad esempio, in molti casi si può spendere lo stesso quantitativo di fondi per acquistare un certo numero di licenze di programmi proprietari, oppure per comprare una copia di un programma libero e pagare qualcuno per l'assistenza tecnica o eventuali modifiche; o addirittura per negoziare con un produttore di software proprietario i diritti per la conversione del suo prodotto a software libero.

In un settore differente, potremmo immaginare che una parte della somma allocata per l'informatizzazione delle scuole venga dedicata alla creazione di una distribuzione GNU/Linux adattata ai principali requisiti di insegnamento delle scuole. Con il resto dei fondi si potrebbe pagare per l'assistenza e la manutenzione del software nelle scuole, in modo che non si tratti meramente di software esposto "in vetrina", poiché ci sarebbero persone realmente responsabili di garantire che esso funzioni regolarmente. Questo non solo risponde ai requisiti didattici, ma genera anche un mercato per le aziende, specialmente locali, in grado di fornire servizi di manutenzione. E ovviamente lascia completamente aperta la strada del futuro: il software non diventerà obsoleto in pochi anni, costringendoci a ricominciare da capo, ma potrà essere aggiornato in modo incrementale, anno dopo anno, mantenendo i benefici del programma con un investimento assolutamente comparabile.

## Nota

I lettori a conoscenza di iniziative pubbliche riguardanti il software libero riconosceranno il caso di LinEx in questo esempio. Verso la fine del 2001 il Governo Regionale dell'Estremadura (Spagna) decise di usare una distribuzione GNU/Linux per informatizzare tutte le scuole pubbliche della regione. A questo scopo finanziò la costruzione di LinEx, una distribuzione GNU/Linux basata su Debian che fu annunciata nella primavera del 2002, assicurandosi che fosse obbligatoria in tutti gli appalti per l'acquisto di strumentazione informatica per le scuole. Inoltre diede inizio a corsi di addestramento per gli insegnanti, creando materiali didattici ed estendendo l'esperienza ad altri settori. A metà del 2003 l'esperienza sembrava un successo, dal momento che si diffuse per canali istituzionali ad altre regioni (ad esempio in Andalusia, sempre in Spagna, tramite il progetto Guadalinex).

### 6.2.3. Promozione della ricerca

Il software libero offre notevoli vantaggi anche quando si tratta di politiche per Ricerca e Sviluppo (R&D). Il denaro pubblico spesso viene speso per finanziare lo sviluppo di una grande quantità di software dal quale la società finisce per non trarre benefici, nemmeno indirettamente. In genere i programmi pubblici di ricerca e sviluppo finanziano, del tutto o in parte, progetti per creare software, senza preoccuparsi veramente dei diritti che il pubblico potrà avere su di essi. In molti casi i risultati, senza un adeguato piano di commercializzazione, sono semplicemente archiviati e lasciati a prendere polvere. In altri casi, le stesse persone che hanno finanziato un programma con i soldi dei contribuenti finiscono per dover pagare di nuovo se desiderano usarlo (dal momento che devono acquistare licenze per l'uso).

Il software libero offre una scelta interessante, che le autorità responsabili per le politiche di innovazione in varie amministrazioni stanno gradualmente iniziando a considerare con attenzione. Specialmente quando la ricerca è pre-competitiva (il caso più frequente quando si tratta di finanziamenti pubblici), il fatto che i programmi risultanti siano liberi permette all'industria nel suo insieme (e di conseguenza alla società) di trarre enormi vantaggi dal denaro pubblico investito in R&D nel settore del software. Dove un'azienda vede un risultato impossibile da vendere, un'altra potrebbe vederci un'opportunità di business. In questo modo, da una parte si massimizzano i risultati dei programmi di ricerca e dall'altra aumenta la concorrenza tra le aziende che desiderano usare i risultati di un progetto, dal momento che tutte competeranno sulla base degli stessi programmi che risultano dal progetto.

Questo modello non è una novità. In gran parte si tratta dello stesso modello che ha permesso ad Internet di svilupparsi. Se le amministrazioni pubbliche esigono che i risultati della ricerca portata avanti con i loro fondi siano distribuiti sotto forma di software libero, non sarebbe strano se si verificassero altri casi simili, a vari livelli. O il risultato della ricerca sarà di scarsa qualità e inutile (nel qual caso occorre rivedere i metodi di selezione dei progetti da finanziare), oppure le dinamiche generate dal lasciarli a disposizione di qualsiasi azienda in grado di convertirli in un prodotto aprirebbero la strada a sviluppi semplicemente imprevedibili.

### 6.3. Esempi di iniziative legislative

Nella sezione che segue vedremo alcune particolari iniziative legislative riguardanti l'uso e la promozione di software libero da parte delle pubbliche amministrazioni. Chiaramente la lista da noi fornita non intende essere esaustiva, ma si è concentrata sulle iniziative in qualche modo pionieristiche (anche se alcune alla fine non sono state approvate). I lettori interessati potranno completarla consultando "GrULIC. Legislation regarding the use of free software by the State" (GrULIC. Leggi sull'uso del software libero da parte dello Stato) [133], che cita molti casi simili. Inoltre, in una delle appendici (Appendice D) includiamo a scopo illustrativo il testo completo o le parti più rilevanti di varie tra queste iniziative.

#### 6.3.1. Proposte di legge in Francia

Nel 1999 e 2000 in Francia furono presentate due proposte di legge sul software libero, che aprirono la strada a una lunga serie di dibattiti legislativi sulla questione:

- Proposta di legge del 1999-495, proposta da Laffitte, Trégouet e Cabanel, fu resa disponibile sul web server del Senato della Repubblica francese nell'ottobre 1999. In seguito a un processo di dibattiti pubblici su Internet ( <http://www.senat.fr/consult/loglibre/index.htm> ) [102] che durò due mesi, la bozza fu modificata. Il risultato fu la Proposta di Legge 2000-117 (Laffitte, Trégouet and Cabanel, Proposition de Loi numéro 117, Senate of the French Republic, 2000) [162], che sosteneva l'uso obbligatorio di software libero da parte dell'Amministrazione, pur contemplando eccezioni e misure di transizione nei casi in cui non era ancora tecnicamente possibile, nel contesto più generale di esmpandere l'uso di Internet e del software libero in tutte le amministrazioni francesi.
- Nell'aprile del 2000 i parlamentari Jean-Yves Le Déaut, Christian Paul e Pierre Cohen proposero una nuova legge il cui obiettivo era simile a quello della proposta di Laffitte, Trégouet e Cabanel: rafforzare cioè le libertà e la sicurezza dei consumatori, oltre a migliorare l'uguaglianza di diritti nella società dell'informazione.

Tuttavia, a differenza della proposta di legge di Laffitte, Trégouet e Cabanel, questa seconda proposta non rendeva obbligatorio per l'Amministrazione l'uso del software libero. Essa si incentrava invece sul fatto che il software usato dall'Amministrazione dovesse avere il codice sorgente accessibile, senza però richiedere per forza che fosse distribuito sotto licenze di software libero.

Per raggiungere i loro obiettivi i legislatori miravano a garantire il "diritto di compatibilità" del software, fornendo meccanismi che mettessero in pratica il principio di interoperabilità descritto nella Direttiva EC relativa alla protezione legale dei programmi per computer (Direttiva del Consiglio

91/250/EEC, del 14 maggio 1991, sulla protezione legale dei programmi per computer, 1991) [111].

Nessuna delle due proposte francesi divenne legge, ma entrambe sono servite di ispirazione alla maggior parte delle iniziative successive in tutto il mondo, ragion per cui sono particolarmente interessanti da studiare. La seconda (proposta da Le Déaut, Paul e Cohen) mirava alla compatibilità e all'interoperabilità del software, sottolineando la disponibilità del codice sorgente per il software usato dall'Amministrazione. Tuttavia non richiedeva per forza che le applicazioni sviluppate fossero software libero, inteso come software distribuito sotto licenze che garantiscono la libertà di modificare, usare e redistribuire il programma.

Più avanti (sezioni D.1 e D.2 dell'appendice D) proponiamo quasi per intero gli articoli e i promemoria esplicativi di entrambe le proposte di legge. I promemoria esplicativi sono particolarmente interessanti, perché mettono in luce i problemi che minacciano attualmente le pubbliche amministrazioni riguardo l'uso di software in generale.

### **6.3.2. Proposta di legge in Brasile**

Nel 1999 il parlamentare Walter Pinheiro presentò una proposta di legge alla Camera Federale del Brasile (Proposição pl-2269/1999. Dispõe sobre a utilização de programas abertos pelos entes de direito público e de direito privado sob controle acionário da administração pública, Camera dei Deputati del Brasile, dicembre 1999) [185]. Questo progetto riguardava l'uso di software libero nella pubblica amministrazione e nelle aziende private di cui lo Stato era azionista di maggioranza.

Il progetto raccomanda l'uso di software libero da parte di questi enti senza restrizioni in termini di prestito, modifica o distribuzione. Gli articoli di legge descrivono nel dettaglio come viene definito il software libero e come dovrebbero essere le licenze che lo accompagnano. Le definizioni coincidono con la definizione classica di software libero del progetto GNU. Il promemoria esplicativo ripercorre la storia del progetto GNU, analizzando i suoi vantaggi e i suoi successi. Si riferisce anche alla situazione attuale del software libero, usando come esempio il sistema operativo GNU/Linux.

Una delle parti più interessanti dell'articolo uno chiarisce molto bene la sfera in cui viene proposto l'uso del software libero (tenendo presente che la definizione di "programma aperto" fornita negli articoli successivi è, come già accennato, la stessa di software libero):

"L'Amministrazione Pubblica a tutti i livelli, i poteri della Repubblica, le aziende statali e quelle miste pubblico-private, le aziende pubbliche e tutti gli altri enti pubblici o privati soggetti al controllo dello Stato del Brasile sono obbligati ad usare preferibilmente, nei loro sistemi e strumentazioni informatiche, dei programmi aperti, liberi da restrizioni proprietarie per quanto riguarda la loro cessione, modifica e distribuzione."

### 6.3.3. Proposte di legge in Perù

In Perù sono state presentate diverse proposte di legge sull'uso del software libero da parte della pubblica amministrazione ("GNU Perú. Proposte di legge sul software libero nella pubblica amministrazione del governo peruviano", Congresso della Repubblica) [184]. La prima e più conosciuta fu proposta dal membro del congresso Edgar Villanueva Núñez nel dicembre 2001 (Proposta di legge sul software libero numero 1609, dicembre 2001) [222]. Essa definisce il *software libero* secondo la definizione classica delle quattro libertà (aggiungendo forse una maggiore precisione legale, con una definizione che specifica sei caratteristiche necessarie perché un programma sia libero) e ne propone l'uso esclusivo nell'amministrazione peruviana:

"Articolo 2. Le autorità legislative, esecutive e giudiziarie, gli enti decentralizzati e le aziende di cui lo Stato è azionista di maggioranza, useranno esclusivamente programmi liberi nei loro sistemi e strumentazioni informatiche."

Tuttavia, più avanti gli articoli 4 e 5 comprendono alcune eccezioni a questa regola.

Ai suoi tempi questa proposta di legge ebbe risonanza mondiale. Da una parte, era la prima volta che veniva proposto l'uso esclusivo di software libero da parte dell'amministrazione. Ma una ripercussione ancora più importante di questa novità fu lo scambio epistolare tra il membro del congresso Villanueva e la rappresentanza di Microsoft in Perù, che lanciò accuse contro la proposta. Questa proposta di legge è interessante anche in relazione alla posizione adottata dall'ambasciata degli Stati Uniti, che attraverso canali ufficiali mandò addirittura una notifica (allegando una relazione preparata da Microsoft) al Congresso peruviano, esprimendo la sua "preoccupazione in merito a recenti proposte del Congresso della Repubblica di limitare gli acquisti del governo peruviano al software libero o open source" ("Lettera al presidente del Congresso della Repubblica", 2002) [147]. Tra i vari motivi, le accuse di Microsoft e dell'ambasciata americana cercavano di dimostrare che la proposta di legge avrebbe operato discriminazioni tra le diverse aziende, rendendo impossibili gli investimenti necessari a generare un'industria nazionale di produzione del software. Villanueva rispose che la proposta di legge non discriminava né favoriva alcuna azienda particolare in alcun modo, perché non specificava quale sarebbe stato il fornitore dell'Amministrazione, ma piuttosto come (in quali condizioni) il software avrebbe dovuto essere fornito. Questo ragionamento fa capire molto chiaramente come la promozione del software libero da parte dell'Amministrazione non pregiudichi in alcun modo la libera concorrenza tra i fornitori.

In seguito i membri del Congresso peruviano Edgar Villanueva Núñez e Jacques Rodrich Ackerman presentarono una nuova proposta di legge, numero 2485, dell'8 aprile 2002 (Proposta di Legge sull'Uso del Software Libero nella Pubblica Amministrazione numero 2485, 2002) [223], che nell'agosto 2003 era ancora sotto procedimento parlamentare. Questa proposta di legge era

un'evoluzione della Proposta di Legge 1609 [222], dalla quale aveva ricavato diversi commenti e apportato una serie di miglioramenti; la si può considerare un buon esempio di proposta di legge per l'uso esclusivo di software libero nelle pubbliche amministrazioni, tranne in alcuni casi eccezionali. Data la sua rilevanza, ne riportiamo il testo per intero (sezione D.3 dell'appendice D). In particolare, il suo promemoria esplicativo è un bel riassunto delle caratteristiche che dovrebbe avere il software usato dalle pubbliche amministrazioni e di come il software libero si conformi a tali caratteristiche meglio del software proprietario.

### 6.3.4. Proposte di legge in Spagna

In Spagna ci sono state diverse iniziative legislative in merito al software libero. Di seguito ne menzioniamo alcune:

- **Decreto sulle Misure per Promuovere la Società della Conoscenza in Andalusía**  
Una delle iniziative legislative più importanti in Spagna (perché è stata effettivamente applicata) è senza dubbio quella adottata dall'Andalusía. Il Decreto sulle Misure per Promuovere la Società della Conoscenza in Andalusía (Decreto 72/2003, del 18 marzo 2003) [99] tratta l'uso del software libero, essenzialmente (ma non solo) nel contesto dell'istruzione. Tra le altre cose, promuove l'uso preferenziale del software libero nei centri della pubblica istruzione, esigendo che tutta la strumentazione acquistata da tali centri sia compatibile con i sistemi operativi liberi, lo stesso vale per i centri del Governo della Regione che forniscono punti di accesso pubblico a Internet.
- **Proposta di Legge sul Software Libero nel contesto della Pubblica Amministrazione della Catalogna**  
Altre comunità hanno discusso proposte più ambiziose, ma senza ottenere il voto di maggioranza richiesto. La più nota tra queste è probabilmente quella dibattuta nel Parlamento di Catalogna (Proposició de llei de programari lliure en el marc de l'Administració pública de Catalunya, 2002) [221], molto simile a quella che lo stesso partito (Esquerra Republicana de Catalunya) presentò al Congresso dei Deputati, di cui parleremo nel prossimo paragrafo. Questa proposta non ebbe successo quando fu sottoposta a votazione.
- **Proposta di Legge di Puigercós Boixassa al Congresso dei Deputati**  
Ci fu anche un'iniziativa al Congresso dei Deputati proposta da Joan Puigercós Boixassa (Esquerra Republicana de Catalunya) (Proposta di Legge sulle Misure per l'Installazione di Software Libero nell'Amministrazione Statale, 2002) [188]. Questa iniziativa proponeva l'uso preferenziale di software libero da parte dell'Amministrazione dello Stato: in questo senso assomiglia ad altre iniziative che condividono lo stesso obiettivo. Tuttavia ha l'interessante particolarità di sottolineare la disponibilità di programmi



liberi localizzati nei linguaggi co-ufficiali (per le comunità autonome che li parlano). L'iniziativa non venne approvata nel corso dei procedimenti parlamentari.

## 7. Ingegneria del software libero

"Il modo migliore di avere una buona idea è averne tante."

Linus Pauling

Nei capitoli precedenti abbiamo mostrato i motivi per cui la sfida del software libero non è come quella di un concorrente che genera software più velocemente, a costi più bassi e con qualità migliore: il software libero è diverso dal software "tradizionale" in aspetti più fondamentali, a partire dalle ragioni e motivazioni filosofiche, proseguendo con nuovi modelli economici e di mercato, e concludendo con un modo diverso di generare software. L'ingegneria del software non può non essere influenzata da tutto ciò, e da poco più di cinque anni ha iniziato a condurre ricerche per approfondire maggiormente tutti questi aspetti. Il presente capitolo intende discutere le ricerche più significative e i dati che esse forniscono, con l'idea di offrire al lettore una visione dello stato dell'arte e delle prospettive future di ciò che abbiamo deciso di chiamare *ingegneria del software libero*.

### 7.1. Introduzione

Benché il software libero venga sviluppato ormai da diversi decenni, solo negli ultimi anni si è cominciato a prestare attenzione ai suoi modelli e processi di sviluppo dal punto di vista dell'ingegneria del software. Come non esiste un modello unico per lo sviluppo del software proprietario, allo stesso modo non esiste un modello unico per lo sviluppo del software libero<sup>5</sup>; ciononostante si possono individuare caratteristiche interessanti che risultano comuni alla maggior parte dei progetti analizzati e che potrebbero derivare dalle proprietà dei programmi liberi.

Nel 1997 Eric S. Raymond pubblicò il primo articolo ampiamente diffuso: *La cattedrale e il bazar: riflessioni di un rivoluzionario per caso su Linux e open source* ("The cathedral and the bazaar. Musings on Linux and open source by an accidental revolutionary", O'Reilly & Associates <http://www.ora.com>, 2001) [192], nel quale descrive alcune delle caratteristiche dei modelli di sviluppo del software libero, ponendo particolare enfasi su ciò che distingue tali modelli da quelli dello sviluppo proprietario. Da allora questo articolo è diventato uno dei più noti (e criticati) nel mondo del software libero e, fino a un certo punto, il segnale d'inizio dello sviluppo di un'ingegneria del software libero.

<sup>(5)</sup>L'articolo "Ecologia dello sviluppo di software open source" ("The ecology of open source software development", Kieran Healy and Alan Schussman, 2003) [140] mostra la grande varietà di progetti e la loro diversità in termini di numeri di sviluppatori, uso di strumenti e downloads.

## 7.2. La cattedrale e il bazar

Raymond opera un'analogia tra il modo di costruire le cattedrali medievali e la maniera classica di produrre software, sostenendo che in entrambi i casi esiste una chiara distribuzione di compiti e ruoli e sottolineando l'esistenza di un architetto che supervisiona il tutto e deve in ogni momento controllare lo sviluppo dell'attività. Allo stesso modo, la pianificazione è strettamente sotto controllo, il che dà origine a processi dettagliati in cui idealmente ogni partecipante all'attività ha un ruolo chiaramente definito.

Quello che Raymond presenta come modello della costruzione di cattedrali ha spazio non solo per i processi pesanti che si trovano nell'industria del software (il classico modello a cascata, i vari aspetti del Rational Unified Process, ecc.), ma anche per progetti di software libero come GNU e NetBSD. Per Raymond, questi progetti sono altamente centralizzati, dal momento che pochissime persone sono responsabili della progettazione e implementazione del software. I compiti svolti da queste persone, così come i loro ruoli, sono ben definiti, e chiunque desideri far parte del team di sviluppo deve essere incaricato di un compito e di un ruolo in base ai requisiti del progetto. Dall'altro lato, le distribuzioni di questo tipo di programmi sono distanziate nel tempo secondo un calendario piuttosto rigido. Questo comporta poche distribuzioni di software e cicli lunghi, che consistono di diverse fasi prima di ciascun lancio.

Il modello opposto alla cattedrale è quello del bazar. Secondo Raymond alcuni programmi di software libero, in particolare il kernel di Linux, sono stati sviluppati seguendo uno schema simile a quello di un bazar orientale. In un bazar non esiste una massima autorità che controlli i processi di sviluppo o che pianifichi rigidamente quanto deve accadere. Allo stesso tempo, i ruoli dei partecipanti cambiano continuamente (i venditori diventano clienti) e senza alcuna indicazione esteriore.

L'aspetto più innovativo di "La cattedrale e il bazaar" tuttavia è il modo in cui descrive il processo attraverso cui Linux è divenuto un successo nel mondo del software libero; è una lista di "buone pratiche" per sfruttare al meglio le opportunità offerte dalla disponibilità del codice sorgente e dell'interazione attraverso sistemi e strumenti telematici.

Un progetto di software libero tende ad apparire come il risultato di uno sforzo puramente personale; la causa può essere uno sviluppatore che si scopre limitato nella capacità di risolvere un problema. Lo sviluppatore ha bisogno almeno della conoscenza sufficiente per iniziare a progettare una soluzione. Una volta ottenuto qualcosa di usabile, con qualche funzionalità, semplice e, possibilmente, ben progettato o ben scritto, la cosa migliore che può fare è condividere la sua soluzione con il mondo del software libero. Questo è ciò che

viene chiamato *release early* (*rilasciare in anticipo*) e aiuta ad attirare l'attenzione di altre persone (in genere sviluppatori) che hanno incontrato lo stesso problema e che potrebbero essere interessate alla soluzione.

Uno dei principi di fondo di questo modello di sviluppo è il pensare agli utenti come co-sviluppatori. Devono essere trattati con attenzione, non solo perché possono fornire pubblicità per "passaparola", ma anche perché svolgeranno una delle operazioni più costose che esistano nella generazione di un software: il testing. A differenza dello sviluppo collaborativo, che è difficile portare su larga scala, testing e correzione di banchi possono essere tranquillamente svolti in parallelo da molte persone. Sarà l'utente a prendere il software e a provarlo sulla sua macchina sotto determinate condizioni (un'architettura, certi strumenti, ecc.), un'operazione che, moltiplicata per un alto numero di architetture ed ambienti, comporterebbe uno sforzo enorme per il team di sviluppo.

Se trattiamo gli utenti come co-sviluppatori, può succedere che uno di loro trovi un baco e lo risolva mandando un aggiornamento agli sviluppatori del progetto, così che il problema possa essere risolto nella versione successiva. Può anche capitare, ad esempio, che sia un'altra persona rispetto a chi ha scoperto il baco a capirlo e infine a correggerlo. In ogni caso, tutte queste circostanze sono vantaggiose per lo sviluppo del software libero, nel senso che è vantaggioso entrare in una dinamica di questo tipo.

Questa situazione diventa più efficace quando le nuove versioni vengono rilasciate con una certa frequenza, perché la motivazione a trovare, segnalare e correggere i banchi è più, alta dal momento che si può assumere che essi verranno presi in considerazione immediatamente. Inoltre si ottengono vantaggi secondari, ad esempio il fatto che integrazioni frequenti, idealmente una o più volte al giorno, non richiedono una fase finale di integrazione dei moduli che compongono il programma. Questo è stato definito *release often* (*rilasciare spesso*) e consente una notevole modularità (Alessandro Narduzzo e Alessandro Rossi, "Modularity in action: GNU/Linux and free/open source software development model unleashed", maggio 2003) [176], massimizzando nello stesso tempo l'effetto di propaganda creato dalla pubblicazione dell'ultima versione del software.

#### **Nota**

La gestione della nuova versione dipende, logicamente, dalle dimensioni del progetto, dal momento che i problemi da affrontare quando il team di sviluppo è composto da due o da centinaia di membri non sono gli stessi. Mentre in generale, per progetti piccoli, questo progetto avviene in modo più o meno informale, la gestione di versioni nei progetti grandi tende a seguire un processo definito, che non manca di un certo grado di complessità. L'articolo "Release management within open source projects" (Justin R. Ehrenkrantz, 2003) [110] descrive nel dettaglio la sequenza seguita nel caso del web server di Apache, del kernel di Linux e del sistema di versioni Subversion.

Per evitare che l'uscita frequente di nuove versioni spaventi gli utenti che danno priorità alla stabilità del software rispetto alla velocità a cui il software evolve, alcuni progetti di software libero hanno diversi rami di sviluppo che vengono portati avanti in parallelo. Il caso più famoso è quello del kernel di Linux,

che ha un ramo stabile indirizzato a coloro che apprezzano la sua affidabilità e un ramo più instabile progettato per gli sviluppatori, con le ultime novità e innovazioni.

### 7.3. Direzione e decisioni nel bazaar

Raymond suggerisce che tutti i progetti di software libero dovrebbero avere un *dittatore benevolo*, una specie di capo che normalmente è il fondatore del progetto, che lo dirige e che ha sempre l'ultima parola quando si tratta di prendere decisioni. Tra le abilità che questa persona deve avere vi è innanzitutto il saper motivare e coordinare un progetto, comprendere gli utenti e i co-sviluppatori, cercare consenso e integrare chiunque abbia un contributo da offrire. Come si può vedere, non abbiamo citato le competenze tecniche tra i requisiti più importanti, anche se non sono mai inutili.

Man mano che le dimensioni dei progetti e il numero degli sviluppatori crescevano, sono emersi nuovi modi di stabilire come vengano prese le decisioni. Linux, ad esempio, ha una struttura gerarchica basata su Linus Torvalds, il "dittatore benevolo", che delega le varie responsabilità a diverse persone. E come vedremo esistono parti di Linux che hanno i loro "dittatori benevoli", benché il loro potere sia limitato dal fatto che Linus Torvalds ha comunque l'ultima parola. Questo caso è un chiaro esempio di come un alto livello di modularità in un progetto di software libero abbia dato origine ad un modo specifico di organizzare le cose e di prendere decisioni (Alessandro Narduzzo e Alessandro Rossi, "Modularity in action: GNU/Linux and free/open source software development model unleashed", maggio 2003) [176].

#### Nota

Alcuni sostengono che il modo in cui sono organizzati i progetti di software libero assomigli a quello di una équipe chirurgica, come propone Harlan Mills (di IBM) nei primi anni Settanta, resi celebri da Brooks nel suo famoso libro *Il mitico mese-uomo* (The mythical man-month. Essays on software engineering, Frederick P. Brooks Jr., 1975) [150]. Benché esistano casi in cui il team di sviluppo di una particolare applicazione di software libero consista in un progettista / sviluppatore (il chirurgo) e diversi co-sviluppatori che svolgono funzioni ausiliarie (amministrazione di sistema, manutenzione, operazioni specializzate, documentazione), non esiste mai una separazione così chiara e netta come quella suggerita da Mills e Brooks. In ogni caso, come sottolinea Brooks nel caso della équipe chirurgica, nel software libero il numero di sviluppatori che devono comunicare tra loro per creare un sistema vasto e complesso è molto più basso del numero totale di sviluppatori.

Nel caso della Fondazione Apache abbiamo una *meritocrazia*, dal momento che questa istituzione ha un comitato direttivo composto da persone che hanno apportato contributi significativi al progetto. In realtà non è una meritocrazia in senso stretto, in cui governano coloro che hanno contribuito maggiormente, perché il comitato direttivo è eletto democraticamente e regolarmente dai membri della Fondazione (responsabili della gestione di diversi progetti di software libero, come Apache, Jakarta, ecc.). Per diventare membri della Fon-

dazione Apache occorre aver contribuito in maniera importante e continuativa ad uno o più dei vari progetti della Fondazione. Questo sistema è usato anche da altri progetti di grandi dimensioni, come FreeBSD o GNOME.

Un altro caso interessante di organizzazione formale è il GCC Steering Committee. Fu creato nel 1998 per evitare che una sola persona prendesse il controllo del progetto GCC (GNU Compiler Collection, il sistema compilatore di GNU) e pochi mesi dopo fu sostenuto anche dalla Fondazione Software Libero (FSF, promotrice del progetto GNU). In un certo senso questo comitato prosegue la tradizione di un organo simile che apparteneva al progetto EGCS (che per un certo tempo fu portato avanti in parallelo al progetto GCC, ma in seguito si fuse con esso). La sua missione fondamentale è di assicurare che il progetto GCC rispetti la *dichiarazione di intenti (mission statement)* del progetto. I membri del comitato sono membri privati, selezionati dal progetto stesso in modo da rappresentare fedelmente le diverse comunità che collaborano allo sviluppo del GCC (sviluppatori che offrono assistenza in diverse lingue, sviluppatori legati al kernel, gruppi interessati alla programmazione annidata, ecc.).

Non è necessario che il capo di un progetto di software sia sempre la stessa persona. Esistono fondamentalmente due circostanze in cui il capo del progetto cessa di essere tale. La prima è mancanza di interesse, tempo o motivazione a continuare. In questo caso il testimone passa ad un altro sviluppatore, che assume il ruolo di capo del progetto. Studi recenti (Jesús M. González Barahona e Gregorio Robles, 2003) [87] mostrano che, in generale, la direzione di un progetto passa di mano frequentemente, tanto che nel tempo si possono vedere diverse *generazioni* di sviluppatori. Il secondo caso è più problematico: implica una divisione (biforcazione) del progetto. Le licenze di software libero permettono a chiunque di prendere, modificare e redistribuire il codice senza richiedere l'approvazione del capo del progetto. In genere però questo non succede, tranne nei casi in cui l'idea è proprio di evitare deliberatamente il capo del progetto (e il suo potenziale veto su un determinato contributo). Da una parte questo assomiglia a una sorta di colpo di stato, ma dall'altra è totalmente lecito e legittimo. Per questa ragione, uno degli obiettivi del capo di un progetto per fare in modo che i suoi co-sviluppatori rimangano soddisfatti è quello di evitare la possibilità di una divisione.

#### **7.4. I processi del software libero**

Benché il software libero non sia necessariamente legato ad uno specifico processo di sviluppo del software, esiste un vasto consenso rispetto ai processi più comunemente usati. Questo non significa che nessun progetto di software libero sia stato creato usando le procedure classiche, come ad esempio il modello a cascata. In generale il modello di sviluppo dei progetti liberi tende ad

essere più informale, soprattutto per via del fatto che gran parte del team di sviluppo svolge questi compiti volontariamente e senza alcun compenso economico, perlomeno diretto.

Il metodo per catturare i requisiti nel mondo del software libero dipende tanto dall'"età" quanto dalle dimensioni del progetto. Nelle fasi iniziali, il fondatore del progetto e l'utente tendono ad essere la stessa persona. In seguito, e se il progetto si espande, la raccolta di requisiti tende ad avvenire attraverso mailing lists elettroniche e si tende a raggiungere una distinzione più netta tra il team di sviluppo, o almeno gli sviluppatori più attivi, e gli utenti. Per i progetti di grandi dimensioni, con molti utenti e molti sviluppatori, i requisiti si catturano usando gli stessi strumenti che si usano per gestire i banchi del progetto. In questo caso, invece di avere a che fare con banchi, ci si riferisce ad attività, benché il meccanismo usato per gestirle sia identico a quello per la correzione dei banchi (vengono classificate in ordine di importanza, dipendenza, ecc. ed è possibile monitorare se siano state risolte o meno). L'uso di questo strumento di pianificazione è piuttosto recente, perciò possiamo vedere come il mondo del software libero si sia evoluto in qualche modo da una mancanza totale ad un sistema centralizzato per gestire queste attività in termini ingegneristici, anche se certamente in modo più limitato. In ogni caso, non è raro trovare un documento che raccoglie i requisiti, come avviene solitamente nel modello a cascata.

Per quanto riguarda l'architettura globale del sistema, solo i progetti di grandi dimensioni tendono a documentarla in maniera esaustiva e dettagliata. Per il resto, lo sviluppatore principale o gli sviluppatori sono in tutta probabilità gli unici ad averla in mente, o a vederla prendere forma man mano che vengono rilasciate le versioni successive del software. La mancanza di una progettazione dettagliata non solo pone limiti al possibile riutilizzo dei moduli, ma è anche un grosso ostacolo quando si tratta di dare accesso a nuovi sviluppatori, che dovranno per questo affrontare un lento e costoso processo di apprendimento. Anche un progetto dettagliato non è molto comune. La sua mancanza significa che molte opportunità di riutilizzo del codice andranno perdute.

L'implementazione è la fase su cui gli sviluppatori di software libero concentrano la maggior parte dei propri sforzi, tra le altre ragioni perché dal loro punto di vista è chiaramente la più divertente. Per far questo, in genere si osserva il classico modello di programmazione per tentativi fino a quando il risultato desiderato, dal punto di vista dello sviluppatore, non viene raggiunto. Storicamente, è raro che insieme al codice vengano inseriti dei test unitari, anche quando faciliterebbero la modifica o l'integrazione successiva di codice da parte di altri sviluppatori. Nel caso di alcuni grossi progetti, come ad esempio Mozilla, esistono macchine dedicate esclusivamente a scaricare depositi contenenti il codice più recente per compilarlo per diverse architetture ("Una panoramica del progresso di ingegneria del software e degli strumenti nel pro-

getto Mozilla", o "An overview of the software engineering process and tools in the Mozilla Project", 2002) [193]. I bachi rilevati vengono segnalati ad una mailing list di sviluppatori.

Tuttavia i test automatici non sono una pratica radicata. In generale sono gli utenti stessi, con la loro enorme varietà di usi, architetture e combinazioni, a svolgere i test. Il vantaggio è che vengono svolti in parallelo a costi minimi per il team di sviluppo. Il problema di questo modello è come ottenere feedback dagli utenti e organizzarlo nel modo più efficiente possibile.

Per quanto riguarda la manutenzione nel mondo del software libero, intesa come la manutenzione di versioni precedenti, dipende da progetto a progetto se questa operazione venga considerata o no. Per i progetti che hanno bisogno di stabilità, come i kernel di sistemi operativi, le versioni precedenti devono essere mantenute in efficienza, perché il passaggio ad una nuova versione può essere traumatico. Ma in generale, nella maggior parte dei progetti di software libero, se qualcuno trova un baco in una versione precedente gli sviluppatori in genere lo ignorano e consigliano di usare la versione più recente, nella speranza che il baco sia scomparso durante l'evoluzione del software.

### 7.5. Critica a "La cattedrale e il bazar"

"La cattedrale e il bazar" soffre di una mancanza di sistematicità e di rigore, data la sua natura giornalistica più che scientifica. Le critiche più frequenti si riferiscono al fatto che, fondamentalmente, descrive il caso particolare dell'esperienza di Linux e mira ad estendere le proprie conclusioni a tutti i progetti di software libero. In questo senso, in "Caverna o comunità? Un'analisi empirica di 100 progetti open source maturi", o "Cave or community? An empirical examination of 100 mature open source projects" [160] possiamo vedere che l'esistenza di una comunità vasta come quella del kernel di Linux è un'eccezione piuttosto che la regola.

Ancora più critici sono coloro secondo i quali Linux è un esempio del modello di sviluppo della cattedrale. Essi sostengono che esiste chiaramente una forza motrice, o almeno una persona con massima autorità, e un sistema gerarchico che delega le varie responsabilità ai lavoratori / programmatori. Inoltre vi è una divisione di ruoli, benché implicita. "Un secondo sguardo alla cattedrale ed il bazar", o "A second look at the cathedral and the bazaar" [91] va oltre e sostiene, non senza un certo astio e arroganza nel ragionamento, che la metafora del bazar è intrinsecamente contraddittoria.

Un altro tra gli aspetti più criticati di "La cattedrale ed il bazar" è l'asserzione che la legge di Brooks, secondo cui "aggiungere sviluppatori ad un progetto software già in ritardo lo rallenta ancora di più" (*The mythical man-month. Essays on software engineering*, 1975) [150], non è valida nel mondo del software



libero. In [148] si può leggere come ciò che realmente accade è che i contesti sono differenti e che quanto sembra in principio contraddire la legge di Brooks, in seguito ad un'analisi più esaustiva, risulta soltanto un miraggio.

## 7.6. Studi quantitativi

Il software libero permette di approfondire lo studio del codice e di altri parametri che intervengono nella sua generazione, grazie all'accessibilità del codice stesso. Questo permette di portare avanti aree tradizionali dell'ingegneria del software come l'ingegneria empirica del software, grazie all'esistenza di una grande quantità di informazioni accessibili senza bisogno di intromettersi pesantemente nel processo di sviluppo del software libero. Gli autori sono convinti che questa visione possa contribuire enormemente all'analisi e alla comprensione dei fenomeni associati al software libero (e al software in generale) e che possa addirittura, tra altre possibilità, riuscire a produrre modelli predittivi del software con feedback in tempo reale.

L'idea che vi sta dietro è molto semplice: "Dato che abbiamo la possibilità di studiare un'immenso numero di programmi di software libero, facciamolo." Oltre allo stato attuale di un progetto, anche la sua evoluzione nel passato è pubblica, il che significa che tutte queste informazioni, adeguatamente estratte, analizzate e confezionate, possono servire come base di conoscenza per permetterci di valutare lo stato di *salute* di un progetto, aiutando a prendere decisioni e a prevedere complicazioni presenti e future.

Il primo studio quantitativo di qualche importanza nel mondo del software libero risale al 1998, benché sia stato pubblicato all'inizio del 2000 ("Sondaggio Orbiten sul software libero", o "The Orbiten free software survey") [127]. Lo scopo era di scoprire in termini empirici la partecipazione degli sviluppatori al mondo del software libero. Per far questo, vennero elaborate statisticamente le assegnazioni di autore che gli autori tendono a inserire nel titolo dei file di codice sorgente. I risultati mostrarono che la partecipazione era coerente con la legge di Pareto ("Course of Political Economy", Losanna, 1896) [182]: 80% del codice viene prodotto dal 20% degli sviluppatori, i più attivi, mentre il rimanente 80% degli sviluppatori contribuisce al 20% del codice totale. Molti studi successivi hanno confermato ed esteso la validità di questi risultati a vari modi di partecipazione per contribuire codice sorgente (mailing lists, segnalazione di bachi o anche il numero di downloads, come possiamo vedere in <http://www-mmd.eng.cam.ac.uk/people/fhh10/Sourceforge/Sourceforge%20paper.pdf> [145]).

### Nota

Il fatto che nello studio del software libero appaiano molti termini di economia è un risultato dell'interesse mostrato da alcuni economisti per imparare e comprendere che cosa spinge dei volontari a produrre beni di notevole valore senza generalmente ottenere in cambio un beneficio diretto. L'articolo più noto è "I mercati del pentolame: un modello economico per il commercio di beni e servizi liberi su Internet", o "Cooking pot markets: an economic model for the trade in free goods and services on the Internet" [125], che introduce l'idea di *economia del regalo* su Internet. Su <http://www.wikipedia.org/wi>

ki/Pareto [232] si possono trovare ulteriori dettagli sul principio di Pareto e la sua generalizzazione alla distribuzione di Pareto. E' interessante anche la curva di Lorenz ( [http://www.wikipedia.org/wiki/Lorenz\\_curve](http://www.wikipedia.org/wiki/Lorenz_curve) ) [231], che mostra graficamente la partecipazione degli sviluppatori ad un progetto, così come il coefficiente di Gini ( [http://www.wikipedia.org/wiki/Gini\\_coefficient](http://www.wikipedia.org/wiki/Gini_coefficient) ) [230], calcolato sulla base della curva di Lorenz, che produce un numero che rivela la disegualianza del sistema.

Lo strumento usato per svolgere questa ricerca è stato pubblicato dai suoi autori sotto una licenza libera, perciò è possibile riprodurne i risultati e riutilizzarlo per condurre nuove ricerche.

In uno studio successivo, Koch ("Risultati da una ricerca di ingegneria del software sullo sviluppo di progetti open source usando dati pubblici", o "Results from software engineering research into open source development projects using public data", 2000) [158] andò oltre e analizzò anche le interazioni in un processo di software libero. Le fonti di dati erano le mailing list e il repository di versioni del progetto GNOME. L'aspetto più interessante della ricerca di Koch tuttavia è la sua analisi economica. Koch si concentra sul controllo della validità delle classiche previsioni di costi (punti funzione, modello CO-COMO...) e mostra i problemi insiti nella loro applicazione, pur ammettendo che i risultati ottenuti, presi con le dovute riserve, in parte corrispondono alla realtà. Egli conclude che il software libero ha bisogno dei propri modelli e metodi di studio specifici, perché quelli noti non sono adeguati alla sua natura. Tuttavia, dal momento che gran parte dei dati relativi allo sviluppo del software libero sono ovviamente accessibili al pubblico, possiamo essere ottimisti sul raggiungimento di tali obiettivi in un prossimo futuro. Quella di Koch si può considerare la prima analisi quantitativa completa, benché manchi certamente di una metodologia chiara e, soprattutto, di alcuni strumenti *ad hoc* che avrebbero reso possibile verificarne i risultati e studiare altri progetti.

Nell'anno 2000, Mockus *et al.* presentarono la prima ricerca su progetti di software libero che comprendesse una descrizione completa del processo di sviluppo e delle strutture organizzative, con dati sia qualitativi che quantitativi ("Qual è il contesto del 'software in beneficenza'?", o "What is the context of charityware?") [172]. Per farlo si servirono del software change log (che documenta le modifiche al software) e delle segnalazioni di bachi, per quantificare aspetti relativi a: partecipazione degli sviluppatori, dimensioni del kernel, autori del codice, produttività, densità di difetti, intervalli tra problema e soluzione. In un certo senso, questa ricerca è ancora parte dell'ingegneria del software classica, tranne per il fatto che tutti i dati sono stati ottenuti tramite ispezione semi-automatica dei dati che i progetti offrono pubblicamente in rete. Come nel caso di "Risultati da una ricerca di ingegneria del software sullo sviluppo di progetti open source usando dati pubblici", o "Results from software engineering research into open source development projects using public data", 2000 [158], questo articolo non forniva alcuno strumento o processo automatizzato che potesse essere riutilizzato in futuro da altri gruppi di ricerca.

In "Una stima delle dimensioni di Linux", o "Estimating Linux's size", 2000 [227], e "Più di un gigadollaro: una stima di GNU/Linux", o "More than a gigabuck: estimating GNU/Linux's" [228] si trova un'analisi quantitativa delle linee di codice e linguaggi di programmazione usati nella distribuzione Red Hat. González Barahona *et al.* hanno seguito questi passi in una serie di articoli sulla distribuzione Debian (*cf.* ad esempio "Anatomia di due distribuzioni GNU/Linux", o "Anatomy of two GNU/Linux distributions" [88]). Tutti questi forniscono una sorta di *raggi-X* di queste distribuzioni GNU/Linux sulla base dei dati forniti da uno strumento che conta il numero fisico di libee di un programma (linee di codice che non sono righe vuote o commenti). A parte lo spettacolare risultato del totale di linee di codice (Debian versione 3.0, noto come Woody, ha oltre cento milioni di righe), si può vedere la distribuzione del numero di linee per ciascun linguaggio di programmazione. La possibilità di studiare l'evoluzione delle diverse versioni Debian nel tempo ha portato ad alcuni risultati interessanti [88]. Vale la pena tener presente che negli ultimi cinque anni la dimensione media di un pacchetto è rimasta praticamente costante, nel senso che la tendenza naturale alla crescita è stata neutralizzata dall'inclusione di pacchetti più piccoli. Allo stesso tempo, si può vedere che l'importanza del linguaggio di programmazione C, benché ancora predominante, nel tempo stia andando in declino, mentre linguaggi di script (Python, PHP e Perl) e Java sono in un momento di crescita esplosiva. I linguaggi compilati "classici" (Pascal, Ada, Modula...) si stanno chiaramente ritirando. Infine, questi articoli comprendono una sezione che mostra i risultati raggiunti se applichiamo il classico modello COCOMO di stima dello sforzo produttivo, risalente ai primi anni Ottanta (*Software Engineering Economics*, 1981) [93] e usato dal software proprietario per fare una stima di costi, sforzo produttivo e tempistiche del progetto.

Benché si tratti di precursori, molti degli studi presentati in questa sezione sono piuttosto limitati ai progetti che analizzano. La metodologia impiegata è stata adattata al progetto in analisi, è in parte manuale e solo talvolta la parte automatizzata può essere usata in generale con altri progetti di software libero. Questo significa che lo sforzo necessario per analizzare un nuovo progetto è molto maggiore, perché occorre riadattare il metodo e ripetere le operazioni manuali.

Per questo motivo, i lavori più recenti ("Studiare l'evoluzione dei progetti di software libero usando dati pubblicamente disponibili", o "Studying the evolution of libre software projects using publicly available data", in: *Proceedings* del terzo Workshop su Open Source Software Engineering, 25th International Conference on Software Engineering, Portland, EE.UU. [196] e "Automatizzare le misure nei progetti open source", o "Automating the measurement of open source projects", 2003 [124]) si concentrano sulla creazione di un'infrastruttura per l'analisi che integri diversi strumenti, in modo che il processo possa essere automatizzato il più possibile. Esistono due ragioni piuttosto ovvie per far questo: la prima è che, una volta che si è investita una grande quantità di tempo e di energie per creare uno strumento di analisi per un progetto, ponendo

speciale attenzione nel renderlo generalizzabile, lo sforzo necessario per usarlo con altri progetti liberi diventa minimo. La seconda è che fare ricerca usando una serie di strumenti che analizzano i programmi da punti di vista diversi, a volte complementari, talvolta non permette di ottenere una visione più ampia del progetto. Nel Sito Web dell'Ingegneria del Software Libero (Libre Software Engineering Web Site [86]) si possono seguire queste iniziative in maggiore dettaglio.

### 7.7. Sviluppi futuri

Dopo aver descritto la storia, breve ma intensa, dell'ingegneria del software applicata al software libero, possiamo dire che essa sta ancora muovendo i primi passi. Molti aspetti importanti attendono ancora di essere analizzati ed esaminati nel dettaglio, prima di poter trovare un modello che spieghi almeno in parte come venga generato il software libero. Le questioni che dovranno essere affrontate in un prossimo futuro sono, tra le altre: la classificazione dei progetti di software libero, la creazione di una metodologia il più possibile basata su analisi automatizzate e l'uso delle conoscenze acquisite per costruire modelli che aiutino a comprendere come viene sviluppato il software libero, facilitando allo stesso tempo le decisioni da prendere sulla base dell'esperienza acquisita.

Un altro aspetto che non può essere trascurato e che si sta iniziando ora a prendere in considerazione è la validità dei metodi classici dell'ingegneria del software nel campo del software libero, attraverso tutte le intensificazioni dell'ingegneria del software. Così, ad esempio, le leggi sull'evoluzione del software postulate da Lehman ("Metriche e leggi dell'evoluzione del software - la visione degli anni Novanta", o "Metrics and laws of software evolution - the nineties view" [165]) agli inizi degli anni Settanta e aggiornata ed estesa negli anni Ottanta e Novanta, non sembrano essersi compiute in maniera incondizionata nello sviluppo di alcuni progetti liberi ("Comprendere l'evoluzione del software open source: applicare, rompere e ripensare le leggi sull'evoluzione del software", o "Understanding open source software evolution: applying, breaking and rethinking the laws of software evolution", 2003 [199]).

Attualmente, una delle lacune più gravi è la mancanza di una classificazione precisa, in modo che i progetti di software libero possano essere assegnati a diverse categorie. Al momento i criteri di classificazione sono troppo ampi, cosicché progetti con caratteristiche organizzative, tecniche o di altro genere molto differenti sono messi tutti nella stessa categoria. L'argomentazione secondo cui Linux, con una comunità così vasta e un grande numero di sviluppatori, sia di natura diversa e non si comporti allo stesso modo dei progetti con un numero di sviluppatori e di utenti assai più limitato, è senz'altro vera. In ogni caso, una classificazione più dettagliata permetterebbe di riutilizzare l'esperienza acquisita in altri progetti simili (in altre parole, con caratteristiche simili), facilitando le previsioni e rendendo possibile prevedere i rischi, ecc.

Il secondo aspetto importante che l'ingegneria del software libero deve affrontare, legato al punto precedente ed alle tendenze attuali, è la creazione di una metodologia e di strumenti per supportarla. Una metodologia chiara e concisa permetterebbe di studiare tutti i progetti sullo stesso piano, scoprire il loro stato attuale, imparare come si sono evoluti e, ovviamente, classificarli. Gli strumenti sono fondamentali quando si tratta di affrontare questo problema perché, una volta creati, permettono di analizzare migliaia di progetti con un minimo sforzo aggiuntivo. Uno degli obiettivi dell'ingegneria del software libero è di rendere possibile l'analisi approfondita di un progetto sulla base di un insieme limitato di parametri, mostrando dove si possono trovare in rete i dati sul progetto (l'indirizzo del repository delle versioni del software, il luogo dove sono immagazzinati i file delle mailing list, dove si trova il sistema di gestione dei banchi, e un brevissimo sondaggio). I manager di progetto si troverebbero a quel punto soltanto a un pulsante di distanza da un'analisi completa, una specie di *analisi clinica* che aiuterebbe a diagnosticare lo stato di *salute* di un progetto, fornendo allo stesso tempo indicazioni sulle aree da migliorare.

Una volta acquisiti metodi, modelli e una classificazione, le opportunità che emergerebbero dall'uso di simulazioni e, per la precisione, di agenti intelligenti, potrebbero essere immense. Tenendo presente che il nostro punto di partenza è un sistema notoriamente complesso, sarebbe interessante creare modelli dinamici su cui si potessero modellare le diverse entità che partecipano alla generazione del software. Ovviamente, più cose sappiamo sui vari elementi, più il nostro modello sarebbe adeguato alla realtà. Benché si conoscano varie proposte di simulazione del software libero, sono piuttosto semplici e incomplete. Questo è dovuto in parte al fatto che esiste ancora un'enorme mancanza di conoscenza delle interazioni che hanno luogo nella generazione del software libero. Se riusciamo a confezionare ed elaborare correttamente i dati dei progetti lungo tutta la loro storia, gli agenti potrebbero diventare cruciali per scoprire le loro future evoluzioni. Benché esistano diverse proposte su come accostare questo problema, una delle più avanzate per ora si può trovare a <http://wwwai.wu-wien.ac.at/~koch/oss-book/> [82].

## 7.8. Riassunto

In breve, abbiamo cercato di mostrare in questo capitolo che l'ingegneria del software libero è ancora un settore giovane e inesplorato. I suoi primi passi si devono a saggi giornalistici che proponevano, non senza una certa mancanza di rigore scientifico, un modello di sviluppo più efficiente, ma gradualmente sono stati fatti dei progressi verso uno studio sistematico del software libero da un punto di vista ingegneristico. Attualmente, dopo diversi anni di relazioni e analisi quantitative e qualitative di progetti liberi, si sta compiendo uno sforzo enorme per ottenere un'infrastruttura globale che permetta di classificare, analizzare e modellare il progetto all'interno di un limitato periodo di tempo e in maniera parzialmente automatizzata. Quando l'analisi dei progetti di software libero smetterà di essere così costosa in termini di tempo e di energie,

probabilmente inizierà un nuovo stadio dell'ingegneria del software, con la comparsa sulla scena di un tipo diverso di tecniche, progettate principalmente per predire l'evoluzione del software e prevedere potenziali complicazioni.

## 8. Tecnologie e ambienti di sviluppo

"Gli strumenti che utilizziamo hanno una profonda (e subdola!) influenza sulle nostre abitudini di pensiero, e di conseguenza, sulla nostra capacità di pensare".

Edsger W. Dijkstra, "Come facciamo a dire verità che possono ferire?"

Col passare degli anni i progetti di software libero hanno creato i loro propri strumenti e sistemi (liberi) per contribuire al processo di sviluppo. Anche se ogni progetto segue le proprie regole e usa il proprio insieme di strumenti, ci sono certe pratiche, ambienti e tecnologie che possono essere considerati comuni all'interno del mondo dello sviluppo del software libero. In questo capitolo vedremo i più conosciuti e ne discuteremo l'impatto sulla gestione ed evoluzione dei progetti.

### 8.1. Descrizione di ambienti, strumenti e sistemi

Prima di analizzare nello specifico gli strumenti, definiremo le caratteristiche e le proprietà generali a seconda del compito da svolgere e del modo in cui gli sviluppatori sono organizzati.

In primo luogo, anche se non è necessariamente un fattore decisivo, è comune per l'ambiente di sviluppo che gli strumenti siano *liberi* (compresa la macchina virtuale di destinazione, nel caso ve ne sia una). Questo non è sempre stato così. Ad esempio, il progetto GNU, che aveva lo scopo di sostituire Unix, doveva essere sviluppato sulla base di e per sistemi proprietari Unix fino a quando non apparirono Linux e free BSDs. Al giorno d'oggi, specialmente quando il software libero viene sviluppato come parte di un modello commerciale, la tendenza è che la macchina di destinazione può anche essere un sistema proprietario, spesso attraverso macchine virtuali interposte (Java, Python, PHP, ecc.). In ogni caso l'ambiente e la macchina virtuale devono essere sufficientemente diffusi e convenienti da raggruppare abbastanza sviluppatori con gli stessi strumenti.

In secondo luogo, al fine di attirare il maggior numero possibile di sviluppatori, gli strumenti devono essere *semplici*, *conosciuti* e in grado di funzionare su macchine *a buon mercato*. Forse è per questi motivi che il mondo del software libero è piuttosto conservativo quando si tratta di linguaggi, strumenti e ambienti di sviluppo.

In terzo luogo, il modello di sviluppo del software libero tende ad essere distribuito in modo notevole, con molti potenziali collaboratori sparsi in tutto il mondo. Per questo motivo sono necessari strumenti di collaborazione gene-

ralmente asincroni, che allo stesso tempo permettono di progredire facilmente, a prescindere dalla quantità e dal ritmo di lavoro di ciascun collaboratore, senza far aspettare nessuno.

Infine, è opportuno fornire agli sviluppatori diverse architetture sulle quali possono compilare e testare i loro programmi.

## 8.2. Linguaggi e strumenti associati

La maggior parte dei software liberi è scritta in C, non solo perché è il linguaggio naturale di qualsiasi variante Unix (tradizionale piattaforma di software libero), ma anche perché è diffuso sia nelle menti delle persone che nelle macchine (GCC è un compilatore standard installato di default in quasi ogni distribuzione). Proprio per questi motivi e per la sua efficienza, Stallman raccomanda il suo impiego nei progetti GNU ("GNU coding standards") [203]. Altri linguaggi piuttosto simili sono C++, supportato inoltre come impostazione predefinita da GCC, e Java, che ha qualche somiglianza ed è ben conosciuto perché permette di sviluppare codice per macchine virtuali, disponibili in una vasta gamma di piattaforme. In generale, gli standard dell'ingegneria del software non sono spesso presi in considerazione: in SourceForge (vedi paragrafo 8.9.1), nel 2004, su centosessanta progetti in C, ce n'era uno in Ada, anche se quest'ultimo si suppone sia un linguaggio più appropriato per lo sviluppo di programmi di qualità. Allo stesso tempo, l'inglese è la *lingua franca* degli sviluppatori di software libero, nonostante il fatto che Esperanto è un linguaggio molto più facile da imparare, con una struttura molto più logica. Sono altresì noti i linguaggi interpretati progettati per la realizzazione rapida di prototipi delle normali applicazioni e servizi web come Perl, Python e PHP.

Proprio come C è il linguaggio standard, *make* è lo strumento standard di costruzione del programma, dati i suoi codici sorgente. Un programmatore di software libero userà di solito la versione GNU (GNU make) [36], piuttosto che la versione incompatibile di BSD (Adam de Boor, "PMake - a tutorial") [100]. Queste ultime due possono essere usate per indicare i legami di dipendenza tra i file e le regole per la generazione di file dipendenti da essi. In questo modo, si può precisare che un file oggetto `x.o` dipende dai file sorgente `xc` e `xh`, e che per costruirlo abbiamo bisogno di eseguire `gcc -c x.c`. Oppure si può affermare che l'eseguibile del nostro programma dipende da un insieme di oggetti ed è assemblato in un certo modo. Quando si modifica il codice sorgente e poi si esegue *make*, saranno ricompilati solo i moduli interessati e l'oggetto finale sarà assemblato nuovamente. Questo è uno strumento di livello molto basso, poiché, ad esempio, non è in grado di scoprire da sé quando un modulo deve essere ricompilato in C, nonostante il fatto che potrebbe farlo esaminando le catene di *includes*. È anche molto potente perché può combinare tutti gli strumenti di trasformazione di file disponibili per creare target molto complessi di un progetto multi-lingua. Tuttavia è molto complicato e dipende in larga misura dagli ambienti di tipo Unix. Altre alternative presumibilmente



migliori, come *jam* (Jam Product Information) [41], *aap* (Aap Project) [1] o *ant* (The Apache Ant Project) [7] vengono usate raramente (quest'ultimo sta guadagnando popolarità soprattutto nel mondo Java).

Data l'eterogeneità di sistemi esistenti persino nel mondo Unix, si usano anche strumenti che contribuiscono a rendere portabili i nostri programmi. Gli strumenti GNU *autoconf* (<http://www.gnu.org/software/autoconf>) [10], *automake* (<http://www.gnu.org/software/automake>) [32] e *libtool* (<http://www.gnu.org/software/libtool>) [35] rendono più facili questi compiti in C e negli ambienti Unix. Data la diversità dei linguaggi, degli insiemi di caratteri e dei contesti culturali, i programmatori C (e i programmatori che utilizzano altri linguaggi) usano spesso *gettext* (<http://www.gnu.org/software/gettext>) [31] e le opzioni di internazionalizzazione della libreria standard C (<http://www.gnu.org/software/libc>) [34] per la programmazione di applicazioni che possono essere utilizzate con facilità in qualsiasi ambiente culturale e nel tempo di esecuzione.

Perciò, quando riceviamo un pacchetto sorgente, è più probabile che sia scritto in C, suddiviso in pacchetti con *tar*, compresso con *gzip*, reso portabile con *autoconf* e gli strumenti relativi e può essere costruito e installato con *make*. La sua installazione sarà eseguita in un processo molto simile al seguente:

```
tar xzvf package-1.3.5.tar.gz cd package-1.3.5./configure make make install
```

### 8.3. Ambienti di sviluppo integrati

Un IDE (*integrated development environment*) è un sistema che facilita il lavoro dello sviluppatore di software integrando in modo stabile la revisione orientata al linguaggio, la compilazione o l'interpretazione, l'individuazione e la correzione degli errori (debugging), le misure di performance, l'incorporazione di un codice sorgente ad un sistema di controllo, etc., solitamente in maniera modulare.

Non a tutti gli sviluppatori di software libero interessano questi strumenti, anche se il loro uso si è gradualmente esteso. In questo campo, il primo ad essere stato utilizzato in larga misura è stato GNU Emacs (<http://www.gnu.org/software/emacs/>) [33], l'opera migliore di Richard Stallman, scritta ed estesa a Emacs Lisp, per la quale esistono innumerevoli contributi.

Eclipse (Eclipse - An Open Development Platform) [23] può essere considerato il riferimento IDE di oggi nel campo del software libero, con lo svantaggio che funziona meglio (ad oggi maggio 2007) su un macchina virtuale Java proprietaria (quella di Sun, che comunque si spera diventi presto libera). Altri ambienti noti di sviluppo sono Kdevelop per KDE (<http://www.kdevelop.org>) [42], Anjuta per GNOME (<http://www.anjuta.org>) [6], Netbeans di Sun per Java (<http://www.netbeans.org>) [51] e Code::Blocks per le applicazioni C++ (<http://www.codeblocks.org>) [18].

## 8.4. Meccanismi di collaborazione di base

Il software libero è un fenomeno reso possibile grazie alla collaborazione di comunità distribuite e che, pertanto, ha bisogno di strumenti per rendere efficace questa partecipazione. Anche se per lungo tempo i nastri magnetici venivano fisicamente trascritti, il rapido sviluppo del software libero iniziò una volta che fu possibile comunicare in modo veloce con molte persone e distribuire codici sorgente o rispondere con commenti e correzioni. Per comodità, piuttosto che inviare codice, si potevano usare messaggi per inviare informazioni sul sito da cui poteva essere preso il codice. Infatti, proprio agli inizi degli anni settanta, l'e-mail era un'estensione del protocollo ARPANET per il trasferimento di file.

Nel mondo Unix, a metà degli anni Settanta, *uucp*, il protocollo di trasferimento del file Unix, veniva sviluppato per macchine che comunicavano attraverso il dial-up oltre alle macchine dedicate, sulle quali era assemblata la posta elettronica; nel 1979, avvenne il primo collegamento USENET con il supporto di UUCP. USENET *news*, un sistema di forum gerarchicamente strutturato distribuito attraverso l'invio massiccio di informazioni relative ai siti sottoscritti alla specifica sezione, ha svolto un ruolo fondamentale nello sviluppo del software libero, inviando programmi completi ai gruppi dello stesso interesse `comp.sources`.

Contemporaneamente venivano sviluppate le mailing list, tra le quali BITNET (1981) i cui gestori meritano di essere menzionati. Oggi la tendenza è quella di preferire le mailing list rispetto a newsgroup tipo USENET. Il motivo principale è stato l'abuso a fini commerciali e l'intrusione di persone "distratte" che interferivano in modo confusionario nelle discussioni. Inoltre, le mailing list forniscono un maggiore controllo e possono raggiungere più persone. I destinatari devono sottoscrivere ed è valido qualsiasi indirizzo e-mail, anche se non vi è l'accesso diretto a internet. L'amministratore della mailing list può scegliere di sapere chi si abbona o decidere di cancellare qualcuno. I contributi possono essere limitati ai soli membri o il programmatore può scegliere di controllare gli articoli prima che appaiano <sup>6</sup>.

<sup>(6)</sup>Ci sono anche i newsgroup moderati.

L'amministrazione della mailing list è stata tradizionalmente fatta via e-mail, tramite messaggi speciali con una password che consentono all'amministratore di non avere un accesso permanente a internet, anche se questo sta diventando un fenomeno sempre più raro; ciò significa che il gestore di mailing list più famoso al giorno d'oggi (Mailman, the GNU Mailing List Manager) [46] non può essere amministrato via e-mail, ma necessariamente attraverso il web. Le mailing list svolgono un ruolo cruciale nel mondo del software libero e in molti casi possono essere l'unico modo con cui partecipare

Attualmente, data la notorietà del Web, molti forum sono puri web forum o *weblog*, senza alcuna interfaccia se non quella fornita dal navigatore. Possono essere generici, come il famoso Slashdot (Slashdot: News for Nerds) [58] (<http://barrapunto.com>) [11], dove viene annunciato il nuovo software libero o si discutono novità ad esso legate oppure si trattano argomenti specializzati in un programma particolare, che sono normalmente integrati in siti di collaborazione con vari strumenti supplementari (*vedi* paragrafo 8.6.2). Vi sono anche interfacce web a newsgroup e mailing list tradizionali.

(7)Ad esempio, i contributi di Linux devono essere resi sotto forma di correzioni di testo alla mailing list `linux-kernel@vger.kernel.org`.

Un altro meccanismo di collaborazione che è diventato contemporaneamente famoso è quello che si fonda su *wikis*, soprattutto quando l'idea è di costruire un documento condiviso, come ad esempio le specifiche per un programma, un modulo o un sistema. Ne parleremo nel paragrafo 8.6.2.

Infine, occorre citare i meccanismi d'interazione utilizzati dagli sviluppatori per conversare in tempo reale. Per il software libero non è tendenzialmente un meccanismo pratico perché, col fatto che gli sviluppatori sono sparsi in tutto il mondo non è facile trovare un momento adatto a tutti. Tuttavia, ci sono diversi progetti che utilizzano strumenti di chat di testo, sia regolarmente sia in occasione di conferenze virtuali in date stabilite. Lo strumento più comunemente usato è l'IRC (Internet Relay Chat, <http://www.ietf.org/rfc/rfc2810.txt>) [151], che di solito permette di comunicare attraverso "canali" a tema, stabiliti sulla base di una serie di server collaboratori. Gli strumenti multimediali non sono normalmente usati sull'IRC (suono, immagine...), probabilmente perché sono richieste connessioni di qualità di cui non tutti dispongono e questo può comportare problemi relativi al software libero disponibile e difficoltà di registrazione e di revisione dei risultati delle conversazioni ai fini della documentazione.

## 8.5. La gestione di sorgenti

È consigliabile salvare la cronologia di qualsiasi progetto di sviluppo di ogni programma perché, ad esempio, una modifica potrebbe produrre un errore nascosto che viene scoperto in un secondo momento e pertanto è necessario recuperare l'originale, almeno per analizzare il problema. Se il progetto è sviluppato da diverse persone, bisognerà inoltre registrare l'autore di ogni modifica, per le stesse ragioni qui sopra illustrate. Se vengono rilasciate nuove versioni di un progetto, è necessario sapere esattamente quali versioni di ogni modulo formano parte di ciascun rilascio. Spesso, un progetto manterrà una versione stabile e una versione sperimentale: entrambe devono essere oggetto di manutenzione ed essere corrette; gli errori sistemati devono essere trasferiti da una versione all'altra. Tutto questo può essere fatto attraverso il salvataggio e l'etichettatura di tutte le versioni dei file in modo corretto; generalmente questo è sempre stato considerato un costo eccessivo, anche se con i drive attuali ciò sta diventando sempre meno vero. Quello che fa di solito un *sistema di controllo delle sorgenti*, noto anche come *sistema di gestione delle versioni*, è

quello di salvare i file della cronologia come un insieme di disuguaglianze rispetto ad un modello, di solito il più recente, per efficienza, etichettando ogni differenza con i metadata necessari.

D'altra parte però, si vuole anche un sistema con queste caratteristiche che possa servire a molti programmatori per collaborare efficacemente senza pestarsi i piedi, ma anche senza ostacolare il progresso dell'altro. Pertanto, è necessario consentire ai programmatori di lavorare simultaneamente ma con un certo livello di controllo. Questo controllo può essere ottimista o pessimista. Con un controllo pessimista, un programmatore può tenere da parte alcuni file da migliorare per un certo periodo, durante il quale nessun altro può accedere a questi file. Questa modalità è molto sicura, ma limiterà gli altri programmatori e perciò può ritardare il progetto, soprattutto se il programmatore che ha bloccato i file è impegnato in altre cose o addirittura si è dimenticato. Consentire ad altri di partecipare allo sviluppo è un aspetto più dinamico ma più pericoloso in quanto possono verificarsi modifiche incompatibili. Un sistema ottimista permette di compiere progressi ma ci avverte quando ci sono stati dei conflitti e ci fornisce gli strumenti per risolverli.

### 8.5.1. CVS

Il CVS (Concurrent Version System) è un sistema di gestione ottimale delle sorgenti progettato verso la fine degli anni Ottanta e utilizzato dalla stragrande maggioranza dei progetti di software libero (Concurrent Version System [20], *Open source code development with CVS*, seconda edizione) [113], "The Internet standards process", terza edizione [95]). Esso utilizza un repository centrale al quale si accede tramite un sistema client/server. L'amministratore del sito decide chi può avere accesso al repository, o a quali parti, anche se di norma, una volta che uno sviluppatore è stato ammesso all'interno del cerchio di fiducia, avrà accesso a tutti i file. L'accesso anonimo, in modalità di sola lettura, può essere autorizzato a chiunque.

#### Il collaboratore anonimo

Il CVS *anonimo* è uno strumento essenziale per la realizzazione del concetto "rilascio frequente e anticipato", sostenuto da Eric Raymond. Ogni utente che sia ansioso di provare l'ultima versione di un programma può estrarlo da CVS, trovare i bug e comunicarli anche sotto forma di patches con la correzione. Può inoltre esaminare l'intera cronologia dello sviluppo.

Vediamo un po' di meccanica. Un utente esperto desidera ottenere la versione più recente del modulo `mod` da un repository accessibile in modo anonimo su `progs.org`, `directory /var/lib/cvs` e il protocollo `pserver`. La prima volta dichiarerà la sua intenzione di entrare:

```
-d:pserver:anonymous@progs.org:/var/lib/cvs login
```

Nel caso in cui venga richiesta una password, sarà l'utente anonimo (semplicemente premendo il tasto invio) che verrà registrato in un file locale (questa operazione non è strettamente necessaria per l'accesso anonimo ma se il file con la password non esiste, il programma segnalerà errore). Di seguito, la cosa importante è ottenere la prima copia del modulo:

```
cvcs
-d:pserver:anonymous@progs.org:/var/lib/cvs co mod
```

Questo creerà una directory `mod` con tutti i file e le directory del modulo e alcuni metadati (contenuti in sottodirectory chiamati `CVS`), che permetterà, tra le altre cose, di non dover ripetere le informazioni già fornite. Il nostro utente avanzato entrerà nella directory creata, creerà il pacchetto ed eseguirà il test:

```
cd mod ./configure make make install ...
```

Nel momento in cui desidera ottenere una nuova versione, l'utente aggiornerà semplicemente la sua copia all'interno di `mod`.

```
cd mod cvs update ./configure make make install ...
```

Se trova un errore, l'utente può correggerlo sul sito e poi inviare una correzione via e-mail al responsabile della manutenzione del programma (mail individuale o mailing list):

```
cvs diff -ubB | mail -s "My patches" mod-maint@progs.org
```

## Lo sviluppatore ordinario

Lo sviluppatore ordinario avrà un account sul server. Egli potrà utilizzare lo stesso meccanismo e lo stesso protocollo come l'utente anonimo, sostituendo però `anonimo` con il proprio account.

Una volta che l'utente ha una copia di lavoro del modulo, può apportare le modifiche necessarie e quando ritiene che siano state stabilizzate, *salva* le modifiche nel repository. Ad esempio, se modifica i file `part.h` e `part.c`, le manderà in questo modo:

```
cvs ci part.h part.c
```

### Nota

Per ragioni di sicurezza, si tende ad usare `ssh` per account con permessi di scrittura perché fornisce un canale autentico e cifrato.

Prima di completare l'operazione, il CVS chiederà spiegazioni di ciò che ha fatto, che sarà allegata al registro di entrambi i file. Anche il *numero di revisione* di ogni file sarà incrementato di una unità. Questo numero identifica ogni momento importante nella cronologia di un file e può essere utilizzato per recuperare ciascuno di quei momenti.

Quando uno sviluppatore deve effettuare una conferma, questo è una questione di metodo su cui i membri del progetto devono essere d'accordo ma è ovvio che le modifiche che non sono elencate non dovrebbero essere mandate. Tuttavia è preferibile inoltre, far passare alle modifiche una serie minima di test. In molti progetti viene anche richiesta l'approvazione di un supervisore di un progetto o sotto-progetto, che esamini la modifica.

Durante lo sviluppo della modifica, qualcuno potrebbe aver alterato dei file o addirittura gli stessi. Quindi si consiglia agli sviluppatori di fare un aggiornamento piuttosto frequente della loro copia ( *cvs update*). Se sono stati modificati altri file, può darsi che sia cambiato l'ambiente e perciò i test che avevano precedentemente passato, questa volta potrebbero fallirli. Se gli stessi file sono stati modificati, è possibile che questi cambiamenti si siano verificati in luoghi o routine che non sono stati toccati o in codici che sono stati modificati. Nel primo caso non vi è alcun conflitto (almeno non apparente) e l'operazione di modifica "unisce" la nostra versione con quella del repository, generando file combinati con tutte le modifiche. Nel caso in cui ci sia un conflitto, bisogna discutere con lo sviluppatore che ha fatto le altre modifiche e definire una versione finale.

Per una migliore identificazione di ogni componente del progetto, è consigliabile eseguire direttamente le informazioni associate alla revisione. CVS può segnare automaticamente i codici sorgente e gli oggetti, a condizione di seguire una certa disciplina. Ad esempio, se in un commento di un codice sorgente si scrive la parola chiave `$Id$` ogni volta che il file è inviato al repository, la parola verrà sostituita con una catena di identificazione che mostrerà il nome del file, il numero di revisione <sup>8</sup>, la data e il tempo dell'invio e il suo autore:

<sup>(8)</sup>Nel CVS, i numeri di revisione hanno normalmente due componenti (maggiore e minore) ma possono averne anche quattro, cinque, sei, etc.

```
$Id: part.c,v 1.7 2003/07/11 08:20:47 joaquin Exp $
```

Se introduciamo questa key word in una modifica di caratteri del programma, una volta stabilita, la catena apparirà nell'oggetto e nell'eseguibile, rendendo possibile identificarlo con uno strumento (*ident*).

## L'amministratore

Ovviamente, gli amministratori sono responsabili per la parte più complicata della manutenzione del repository. Per esempio, essi devono registrare il programma, emettere i permessi per gli sviluppatori e coordinarli, etichettare le versioni consegnate etc.

È prassi comune per tutti i progetti avere una versione stabile e una versione sperimentale. Per fare questo dobbiamo creare rami. Mentre quelli dedicati alla manutenzione correggono gli errori sul ramo stabile, sul ramo sperimentale vengono fatti nuovi sviluppi. Quando il ramo sperimentale si stabilizza, viene trasferito a quello stabile, ma non senza aver precedentemente applicato le correzioni apportate al precedente ramo stabile. Questa operazione delicata si chiama *merging*, ed è supportata da CVS, anche se in modo un po' primitivo. Quest'idea può essere estesa al concetto di rami sperimentali che evolvono in direzioni diverse, che possono o meno giungere a buon fine, e che in ogni caso, a meno che non siano binari morti, sarà parzialmente o totalmente integrata nel prodotto stabile con le giuste combinazioni.

Un diritto che il software libero garantisce è quello di modificare un programma ad uso privato. Anche se si desidera comunicare tutti i miglioramenti all'esterno, spesso le modifiche che vogliamo fare sono troppo specifiche e poco interessanti per il pubblico generale. Ma noi siamo interessati a integrare l'evoluzione nel programma originale. Questo può essere fatto con uno speciale tipo di divisione e unione (*seller branches*).

L'amministratore può anche facilitare il coordinamento del team attraverso meccanismi automatici, come ad esempio generando messaggi e-mail quando si verificano determinati eventi, come i commit, oppure imponendo alcune azioni automatiche che devono essere eseguite prima di un commit, come i controlli automatici di stile, compilazioni o test.

### 8.5.2. Altri sistemi di gestione delle sorgenti

Nonostante sia il sistema di controllo di versione maggiormente usato, CVS presenta alcuni svantaggi:

- 1) CVS non supporta sia le rinominazioni sia le modifiche ai file directory, i metadati (proprietary, permessi, ecc) o collegamenti simbolici.
- 2) Poiché si tratta di un'evoluzione di un sistema di controllo di versione per singoli file, esso ovviamente non supporta il controllo della versione per gruppi completi.
- 3) CVS non supporta set di modifiche coerenti. Aggiungere una caratteristica o correggere un errore possono davvero comportare il cambiamento di numerosi file. Queste modifiche dovrebbero essere indivisibili.
- 4) In CVS l'uso di rami e unioni è piuttosto complicato. In realtà, se creiamo una divisione sperimentale di un progetto e desideriamo includere le correzioni apportate alla versione stabile, abbiamo bisogno di sapere nel dettaglio quali correzioni sono state già fatte e quali no, in modo da non doverle rifare più volte.

#### Note

Nota: Nel 2007 Subversion è già un chiaro successore di CVS e molti sviluppi di software libero si sono spostati verso di esso.

- 5) CVS dipende da un server centralizzato e, sebbene sia possibile lavorare senza una connessione, ne abbiamo davvero bisogno una se vogliamo generare versioni, confrontarle e unirle.
- 6) CVS non genera, senza l'ausilio di strumenti separati, il file `changelog`, che mostra la cronologia globale delle modifiche di un progetto.
- 7) CVS non supporta correttamente i progetti che hanno un numero elevato di file, come nel caso del kernel di Linux.

Eppure ci sono altri sistemi liberi che risolvono molti di questi problemi. Vorremmo ricordare il già citato successore di CVS, Subversion (<http://subversion.tigris.org>) [62], (<http://svnbook.red-bean.com/>) [96], che risolve rigorosamente i problemi base di CVS e può utilizzare le estensioni HTTP (Web-DAV) al fine di aggirare le aggressive politiche di sicurezza.

Il modello di sviluppo basato su un repository centralizzato, anche se adatto per il lavoro collaborativo, non soddisfa tutte le aspettative in quanto la capacità di creare i propri rami di sviluppo dipende, da un lato, dall'accessibilità del server e dal buon funzionamento e, dall'altro lato, dagli amministratori di quel server. Ai repository distribuiti viene richiesto a volte di permettere a chiunque di avere un repository con una sezione pubblica o privata che possa essere poi unita o meno a quella ufficiale. Così è come funzionano *GNU arch* (Arch Revision Control System) [8] o *bazar* (Bazar GPL Distributed Version Control Software) [12], e il sistema proprietario BitKeeper (BitKeeper Source Management) [14], scelto da Linus Torvalds per mantenere Linux a partire dal febbraio 2002, in quanto secondo lui non c'era alcun strumento libero adeguato. Si dice che l'utilizzo di BitKeeper raddoppiasse il ritmo di sviluppo di Linux. Ciononostante, la decisione fu oggetto di pesanti critiche perché era proprietario, con una licenza che permetteva ai progetti liberi di ottenerlo gratuitamente; a condizione però che tutte le modifiche commit, con i metadati relativi, venivano registrate su un server pubblico, progettato dai proprietari e accessibile a tutti, e sempre a condizione che il licenziatario non cercava di sviluppare un altro sistema di controllo della sorgente per fargli concorrenza. È stato proprio il tentativo di sviluppare un prodotto compatibile libero da parte di un dipendente della stessa società dove lavorava Linus Torvalds, che provocò il cambiamento nel sistema di gestione della sorgente. Linus sviluppò rapidamente un sostituto provvisorio, *git* ("Git manual page") [218], che divenne ben presto definitivo, condensando tutta l'esperienza dello sviluppo di cooperazione e decentrato di Linux: esso supporta progetti di grandi dimensioni in modo decentrato, facilitando in larga misura lo sviluppo delle sezioni sperimentali e la loro unione con altri o con quello principale, con meccanismi di sicurezza crittografata che impediscono l'alterazione del registro. Ad oggi aprile 2005, Linux è gestito tramite *git* o attraverso le sue versioni wraps (per esempio, *cogito* "Cogito manual page") [90].



## 8.6. Documentazione

Nel mondo del software libero, i processori di testo WYSIWYG e altri strumenti software di produttività personale office che hanno avuto tanto successo in altri ambienti, qui sono poco utilizzati anche se ci sono già strumenti liberi come OpenOffice.org. Ciò è dovuto a diversi importanti fattori:

- Si consiglia di mantenere la documentazione sotto controllo dei file sorgenti e i sistemi di controllo del codice sorgente come CVS che, sebbene ammettono formati binari, preferiscono formati di testo trasparenti che possono essere modificati con un normale editor di testo e processati con strumenti sviluppati per programmi che permettono di vedere facilmente le differenze tra le versioni, per elaborare e applicare le correzioni sulla base di tali differenze ed effettuare le dovute unioni.
- Alcune licenze libere, come ad esempio la GFDL (*vedi* paragrafo 10.2.1) richiedono formati trasparenti soprattutto perché agevolano il lavoro di quelli che preparano i documenti derivati.
- Gli strumenti WYSIWYG ("What You See Is What You Get", quello che vedi è quello che ottieni), generalmente non contengono altre informazioni se non quelle strettamente visualizzate, rendendo difficile, se non impossibile, identificare autori, titoli o conversioni negli altri formati. Anche se tali strumenti permettono le conversioni ad altri formati, questo tende ad essere fatto in forma interattiva ed è spesso impossibile automatizzare (usando *make* ad esempio).
- In generale, le applicazioni office generano formati di file piuttosto considerevoli, che non è una caratteristica ben accettata sia dagli sviluppatori sia dai repository.

### Nota

In Unix gli strumenti più comuni per queste operazioni sono *diff*, *diff3*, *patch* e *merg*.

Per tutto quanto detto sopra, i programmatori di software libero, abituati alla programmazione e alla compilazione, preferiscono i formati trasparenti dei documenti: in molti casi il testo puro e semplice e in molti altri casi i formati processabili di documenti.

I formati processabili in uso non sono molti. Tradizionalmente, nel mondo di Unix, i programmi sono stati documentati nei formati previsti dalla famiglia dei processori *roff*, con la versione libera (*GNU troff*) [37] da parte di Norman Walsh. Tuttavia, questa pratica è stata gradualmente abbandonata, fatta eccezione delle pagine di manuale tradizionali, poiché è obbligatorio preparare le pagine di manuale per gli strumenti base del sistema. Dato che molte pagine di manuale sono aumentate talmente tanto che non è più tanto appropriato chiamarle *pagine*, è stato necessario preparare un formato di ipertesto alternativo che permetteva di seguire facilmente documenti strutturati con indici e riferimenti incrociati. Il progetto GNU ha progettato il formato *texinfo* (Texinfo - Gnu Documentation System) [63] e ne ha fatto il suo standard. Questo

formato consente di ottenere documenti navigabili con lo strumento *info* o all'interno dell'editor di testo *emacs*, e, a sua volta, permette di ottenere ottime stampe di documenti grazie al processore di testo TeX di Donald Knuth (The TeXbook) [156].

Il formato *texinfo* può essere tradotto, se richiesto, in formato HTML multipagina e molte persone preferiscono visualizzare le informazioni con un navigatore web anche se si è persa la funzione di ricerca delle parole in un documento. Questo è uno dei risultati indesiderati della popolarità del formato HTML dal momento che i navigatori non implementano il concetto di *documento multipagina*, anche se esistono i *links* che permettono collegare le parti.

Vi è una grossa richiesta di poter essere in grado di visualizzare documenti complessi con la stessa facilità con cui si naviga nel web a più pagine. Ci sono persone che scrivono documenti in LaTeX (*LaTeX user's guide and reference manual*) [163], un'applicazione TeX molto popolare tra gli scienziati, che è più espressiva di Texinfo e convertibile nel formato HTML multipagina con determinati strumenti (The LaTeX Web Companion) [130] a condizione che venga mantenuta una certa disciplina. Infatti, le applicazioni TeX sono insiemi di macro che raggruppano operatori tipografici di basso livello per trasformarli in linguaggi astratti che lavorano con concetti di alto livello (autore, titolo, sommario, capitolo, paragrafo, ecc.) Se si usano solo le macro di base, la conversione è semplice. Ma, dal momento che nessuno impedisce l'uso di operatori di basso livello, è difficile ottenere una buona conversione e inoltre esistono enormi quantità di pacchetti macro al di là della capacità dei responsabili di curare gli strumenti di conversione.

### 8.6.1. Convertitore DocBook

Il problema deriva dal fatto che non vi è distinzione tra contenuto e presentazione, in TeX o nelle applicazioni *nroff*, dal momento che l'astrazione viene costruita a strati. Questa distinzione è fatta secondo le applicazioni SGML (*standard generalized markup language*) [81] e XML (*extensible markup language*) [224], dove la presentazione viene specificata con *style sheets* completamente separati. Poco dopo, cominciarono ad essere utilizzate applicazioni SGML molto semplici, come *linuxdoc* e *debiandoc*, ma a causa delle loro limitate capacità espressive venne scelto DocBook. (*DocBook: The Definitive Guide*) [225].

DocBook è un'applicazione SGML originariamente sviluppata per documentazione tecnica IT e ora ha una variante XML. Attualmente DocBook è il formato standard e libero di documentazione per molti progetti (Linux Documentation Project, KDE, GNOME, Mandriva Linux, ecc) e un traguardo ancora da raggiungere per altri (Linux, \*BSD, Debian, ecc).

Tuttavia, DocBook è un linguaggio complesso, pieno di tag, il che significa che è utile disporre di strumenti che agevolino l'editing, anche se sono elementari e rari. Uno degli strumenti più conosciuti di questo tipo è il `psgml` modalità di *emacs*. È molto pesante da processare e i processori liberi non generano ancora documenti di qualità accettabile.

### 8.6.2. Wikis

Molte persone trovano difficile scrivere documentazione con un linguaggio così complesso come DocBook e con dei meccanismi di collaborazione come CVS. È per questo motivo che è diventato famoso un nuovo meccanismo di collaborazione per la preparazione di documenti on-line tramite il web, chiamato *wiki*, inventato da Ward Cunningham ("Wiki design principles") [97]. La prima volta che è stato messo in servizio fu nel 1995 e oggi è ampiamente utilizzato in un vasto spettro di varianti per la preparazione di documenti molto dinamici, non progettati per la stampa e spesso a breve scadenza (per esempio per le organizzazione di conferenze).

A differenza di DocBook, un *wiki* ha un linguaggio tag molto semplice e conciso che ricorda la presentazione finale, senza che sia esattamente così. Per esempio, i paragrafi sono separati da una riga vuota, gli elementi di un elenco, se non sono numerati, sono preceduti da un trattino e, se sono numerati, da uno zero e le celle della tabella sono separate da barre verticali e orizzontali.

Non esiste nemmeno il concetto di un "documento completo": *wiki* è piuttosto un insieme di piccoli documenti interconnessi, creati quando diventa necessario spiegare un nuovo concetto o tema. I documenti vengono creati quasi automaticamente mentre lo strumento di editing mostra molto chiaramente che abbiamo inserito un concetto (attraverso un `NameWiki` quasi sempre sono due parole unite con la prima lettera maiuscola). Quasi nessuna *wiki* permettere collegamenti ipertestuali all'interno della stessa pagina.

A differenza di CVS, chiunque può scrivere su un *wiki*, anche si consiglia all'autore di identificarsi alla prima registrazione. Quando si visita un *wiki*, si può vedere che tutte le pagine hanno un pulsante che permette di modificarle. Se premuto, il navigatore mostrerà un modulo con il codice sorgente del documento, che potremo modificare. Questo non è un edit WYSIWYG che scoraggia chiunque voglia anche solo interferire, ma è abbastanza semplice per chi sia interessato a modificare i documenti senza troppa fatica.

I *Wiki* eseguono il loro controllo di versione del documento in modo tale che tutte le loro versioni siano generalmente accessibili, indicando chi li ha fatti e quando. Essi possono anche essere facilmente messi a confronto. Inoltre, tendono ad includere meccanismi di ricerca, almeno per nome della pagina e per contenuto di parola.

Normalmente, l'autore originale di una pagina vuole sapere quali modifiche vengono apportate. A tal fine, può sottoscrivere i cambiamenti e ricevere notifiche via e-mail. A volte una persona che vede un documento, non osa cambiare niente ma magari lascia un commento. Tutte le pagine *wiki* hanno di solito un forum relativo di commenti, posto in fondo al documento; l'autore originale o chiunque assume il ruolo di editor può usare il forum per modificare il testo originale, possibilmente spostando le frasi in punti più visibili.

### Consiglio

Il miglior modo per comprendere il concetto di *wiki* è di accedere ad un file di questo tipo e fare un esperimento su una pagina creata a questo scopo, che di solito viene chiamata *SandBox*.

## 8.7. Bug management e altri problemi

Uno dei punti di forza del modello di sviluppo libero è che la comunità contribuisce alle segnalazioni di bug e crede che tali report o soluzioni vengano prese in considerazione. Ciò richiede un semplice meccanismo di segnalazione dei bug così che gli sviluppatori possono ricevere un buon numero di informazioni, in modo sistematico e che contengono tutti i dettagli necessari, attraverso il collaboratore che fornisce una spiegazione di ciò che sta accadendo, il livello di importanza e la possibile soluzione, oppure attraverso un meccanismo automatico che determina, per esempio, la versione del programma e l'ambiente in cui funziona. Gli errori dovrebbero anche essere salvati in un database che si può essere consultare, per vedere se un bug è già stato comunicato, corretto, il suo livello di importanza, ecc.

Esistono parecchi sistemi di questo tipo con filosofie diverse. Alcuni sono via web, altri via e-mail, attraverso qualche programma intermediario. Tutti hanno un'interfaccia web per la consultazione. Alcuni permettono segnalazioni anonime mentre altri richiedono l'identificazione (un valido indirizzo e-mail) per evitare confusione. Anche se le procedure via web sembrano essere le più semplici, non ottengono facilmente informazioni automatiche sull'ambiente dei bug. Ad esempio, il sistema Debian fornisce programmi simili al *reportbug*, il quale, dopo aver chiesto il nome del pacchetto, consulta il server di errore per i bug che sono stati segnalati. Se nessuno di questi riguarda il problema esistente, ci verrà chiesta una descrizione, il suo livello di importanza ("fondamentale", "grave", "serio", "importante", "non può essere rigenerata da codici sorgente", "normale", "minore" o "suggerimento") e le etichette sulla sua categoria (ad esempio "sicurezza"). Seguendo questa procedura, se confermiamo la richiesta, il programma troverà automaticamente la versione del pacchetto e quelli da cui dipende, oltre alla versione e architettura del kernel. Ovviamente, il programma conosce l'indirizzo e-mail in modo tale che invia al sito esatto una relazione simile alla seguente:

```
Package: w3m-ssl Version: 0.2.1-4 Severity: important After reloading a page containing complex tables sever
```

Questo messaggio genera un numero di bug che viene comunicato, inviato al manutentore e salvato nel database. Quando il bug è risolto, si riceverà una notifica. Ogni errore ha un indirizzo e-mail ad esso assegnato, che può essere usato, per esempio, per fornire ulteriori informazioni. Si può consultare il database dei bug, <http://bugs.debian.org>, in qualsiasi momento.

A volte i sistemi di monitoraggio dei bug hanno meccanismi per l'assegnazione di persone responsabili alla risoluzione di bug e per fissare una scadenza. Ci sono anche altri problemi, quali i lavori in sospeso, i miglioramenti richiesti, traduzioni, ecc, che richiedono simili meccanismi di gestione. Con il software libero non si possono usare meccanismi troppo rigidi per la gestione delle attività che ogni sviluppatore deve svolgere. Dopotutto, molti collaboratori sono volontari e non possono essere obbligati a fare qualcosa. Tuttavia, si possono definire i compiti e poi si può aspettare che qualcuno si sottoscriva al sistema e si assuma tali incarichi entro un periodo stabilito. Sebbene ci sia controllo su quello che certe persone possono fare o no, è sempre consigliabile controllare tutte le attività che devono essere fatte, da chi e da cosa dipendono, il loro livello d'importanza e chi sta lavorando su di essi. Molti progetti importanti gestiscono questi aspetti usando Bugzilla (*The Bugzilla guide*) [89] o i suoi derivati.

Spesso chi lavora su un progetto può scoprire un bug in un progetto che non ha niente a che vedere col suo e che ha un sistema di gestione dei bug diverso da quello a cui è abituato. Ciò è particolarmente vero per gli utenti delle distribuzioni che vogliono utilizzare un unico strumento di segnalazione e controllo per la risoluzione dei bug. Per facilitare la segnalazione e il controllo di questi bug, può essere utile *confederare* i diversi sistemi, come è stato fatto da *Malone* (The Bug Tracker Malone) [47].

## 8.8. Supporto per altre architetture

Il supporto minimo richiesto per lavorare con un programma portatile è l'accesso alle *compilation farms*, che permettono di compilare il programma su diverse architetture e su diversi sistemi operativi. Per esempio, SourceForge (vedi paragrafo 8.9.1) ha permesso per un certo tempo di utilizzare ambienti di sviluppo di tipo Debian GNU/Linux per piattaforme Intel x86, DEC Alpha, PowerPC e SPARC, oltre agli ambienti Solaris e Mac OS/X.

È utile anche saper testare (non solo compilare) il programma in tali ambienti di sviluppo. Ma questo servizio richiede più risorse e più tempo all'amministratore. La compilazione potrebbe già essere problematica di suo perché di solito è necessario fornire ambienti di compilazione per diversi linguaggi, con molte librerie. Se ciò che si vuole fare è testare qualunque programma, le complessità aumentano esponenzialmente, non solo perché è difficile avere le risorse necessarie disponibili, ma è anche per ragioni di sicurez-

za, che possono ostacolare l'amministrazione questi sistemi. Nonostante ciò, esiste un numero ristretto di servizi di *compilation farm*, con installazioni standard di molteplici architetture che consentono di testare alcune cose.

I servizi di compilazione citati richiedono un uso manuale. Lo sviluppatore copia i suoi file su una di quelle macchine, li compila e testa i risultati. Egli dovrà farlo periodicamente prima di rilasciare una versione significativa del programma. Potrebbe essere molto più interessante per le compilazioni e per l'esecuzione dei test di regressione se venissero eseguiti sistematicamente, in modo automatico, come ad esempio ogni sera, se vi sono stati cambiamenti nei codici sorgente. Questo è il modo in cui funzionano alcuni progetti importanti, che mettono a disposizione la loro stessa infrastruttura per sviluppatori esterni e che tende a essere chiamata *tinderbox*. Questo è il caso di Mozilla, finanziato da Netscape, il cui *tinderbox* (<http://www.mozilla.org/tinderbox.html>) [50] ha un'interfaccia web ai risultati della compilazione e sui test dei componenti del browser rispetto a tutte le architetture sul quale opera. Questa interfaccia è strettamente correlata a un server CVS e mostra quei risultati per differenti stati (tra molteplici rilasci di codice), identificando la radice che ha causato il problema e facilitando il progresso attraverso il superamento del problema finché non viene risolto. I Tinderbox vengono anche usati dai progetti OpenOffice e Free BSD.

## 8.9. Siti di supporto allo sviluppo

I siti di supporto allo sviluppo mettono a disposizione in modo più o meno integrato tutti i servizi sopra descritti ed altri servizi supplementari che permettono di ricercare i progetti per categorie e di classificarli secondo semplici parametri di attività. Ciò evita allo sviluppatore di installare e mantenere un'intera infrastruttura per la collaborazione, consentendogli di concentrarsi sul progetto.

### 8.9.1. SourceForge

A questo proposito, uno dei primi servizi che venne istituito e che diventò il più famoso di tutti, fu SourceForge (<http://sourceforge.net>) [61] gestito da OSDN (Open Software Development Network), una filiale di VA Software, che nel marzo 2007 ospitò più di 144,000 progetti. È strutturato attorno ad un insieme di programmi con lo stesso nome, che erano liberi fino alla versione 2.

SourceForge, come prototipo per questo tipo di siti, mette a disposizione un'interfaccia web o un portale d'accesso globale (<http://sourceforge.net/>) e un sotto portale a seconda del progetto (<http://proyecto.sourceforge.net>). L'interfaccia globale mette a disposizione notizie, pubblicità, collegamenti e un invito a diventare membri o ad accedervi se si è già membri. Per collaborare al sito, si consiglia di diventare membri ed è obbligatorio farlo se si vuole creare un nuovo progetto o partecipare ad un progetto già esistente. Se si vuole solo assistere, non è necessario diventare membri in quanto si possono vedere

i progetti che hanno maggior sviluppo (oppure quelli scaricati più frequentemente e ricercati per categoria o per descrizione) ed essi appariranno in ordine di livello di attività. Per ogni progetto si può vedere la sua descrizione, il suo stato (alfa, beta, produzione), i suoi sottoscrittori (linguaggio di programmazione, sistema operativo, tema, tipo di utilizzatori, lingua, licenza...), i suoi bug e i suoi aspetti in sospeso o restaurati, i livelli di attività nel tempo, o infine scaricare il progetto. Si può anche partecipare ai forum o segnalare bug, anche anonimamente, sebbene non è molto consigliato (perché ad esempio non si può avere risposta).

Ogni utilizzatore autenticato può richiedere di registrare un progetto, che gli amministratori ammetteranno a condizione che soddisfi le policy del sito che nel caso di Source Forge sono molto permissive. Dopo l'autorizzazione, il creatore può registrare altri utenti come amministratori supplementari o come sviluppatori con accesso a modificare le sorgenti. Dopo l'autenticazione, non ci sono più molti controlli sul progetto. Per questo ci sono molti progetti abbandonati. Questo, tuttavia, non confonde troppo gli utenti perché le ricerche sul progetto ordinano i progetti per il livello di attività e pertanto quelli che hanno attività bassa o nulla sono scarsamente visibili. Questi ultimi corrono perciò il rischio di essere eliminati dai proprietari del sito. I servizi che SourceForge offre e che ci si potrebbe aspettare da qualunque servizio simile sono i seguenti:

- Spazio web libero per le pagine del portale del progetto all'indirizzo *project.sourceforge.net*, aperto al pubblico. Queste pagine possono essere statiche o dinamiche (con CGI o PHP) e in quest'ultimo caso possono accedere ad un database (MySQL). Esse sono inserite direttamente attraverso ordini di copia da remoto ed essere gestite attraverso sessioni interattive da terminale remoto (SSH).
- Facoltativamente, un server virtuale che risponde agli indirizzi da un dominio separato, come *www.project.org* o *cvs.project.org*.
- Tutti i forum web e/o mailing list che si rendessero necessari a giudizio dell'amministratore.
- Un servizio di notifica degli aggiornamenti dove gli amministratori comunicano i progressi del progetto.
- *Trackers* per la segnalazione dei bug ed il controllo, le richieste di assistenza, le richieste di miglioramento o di integrazione delle correzioni. Gli amministratori assegnano un livello di priorità ai problemi e nominano uno sviluppatore per trovare una soluzione.
- I task managers, simili ai trackers, che permettono di definire i sottoprogetti con una serie di operazioni. A queste attività, viene assegnata una sca-

denza, oltre al livello di priorità. Periodicamente gli sviluppatori responsabili possono comunicare le percentuali di realizzazione dei compiti.

- Un CVS o una sottoversione con diritti di accesso iniziali per tutti gli sviluppatori.
- Un servizio di uploading e downloading per i pacchetti software. Il servizio, se usato, registra versioni entranti e le parti interessate possono ricevere una notifica, quando questo accade. Inoltre, l'upload iniziale comprende la creazione di numerose repliche a livello mondiale, che facilita la distribuzione.
- Un servizio per pubblicare i documenti in formato HTML. Chiunque può registrarli ma saranno visibili solo dopo l'approvazione di un amministratore.
- Una copia di back-up in caso di recupero di dati dopo un disastro, come ad esempio per un supporto di memoria danneggiato, bug di non utenti, o come l'eliminazione accidentale di un file.
- Un meccanismo integrato per donazioni agli utenti, ai progetti e a SourceForge.

Un utilizzatore autenticato avrà una pagina personale contenente tutte le informazioni rilevanti, come ad esempio i progetti associati all'utente, i temi o le attività in sospeso, oppure forum e file che ha affermato di voler supervisionare. Inoltre l'utente riceverà notifiche nella sua e-mail sulle ciò che desidera controllare, in modo che non debba gestire la sua pagina personale.

### 8.9.2. Gli eredi di SourceForge

Nel 2001, VA Software stava per andare in fallimento, nel pieno della crisi delle dotcom. Poi annunciò una nuova versione del suo software SourceForge, con una licenza a pagamento, nel tentativo di garantirsi un'entrata dalla vendita alle aziende per i loro sviluppi interni. Allo stesso tempo, VA Software eliminò i meccanismi che permettevano di abbandonare un progetto per il passaggio a un altro sito. Entrambi gli eventi vennero visti come una minaccia alle migliaia di progetti ospitati da SourceForge, in quanto sarebbero rimasti intrappolati nelle mani di una singola società che usava la piattaforma per mostrare il software non libero. A fronte di questo e della possibilità di chiusura del sito, vennero sviluppati i discendenti della versione libera e furono aperti portali basati sul software libero, in particolare Savannah ( <http://savannah.gnu.org>) [57], dedicato al progetto GNU e ad altri programmi con licenze di tipo copyleft, o BerliOS (BerliOS: the Open Source Mediator) [13], concepito come un punto d'incontro per gli sviluppatori e le aziende di free software. Tuttavia, questo è



solo un passo nella direzione di uno sviluppo di una piattaforma distribuita e replicata, dove nessuno ha il controllo assoluto sui progetti (Savannah The Next Generation, 2001) [98].

Un altro esempio di un sistema di gestione di un progetto di software libero è Launchpad (<https://launchpad.net>) [43], utilizzato da Ubuntu per lo sviluppo di ogni versione della distribuzione. Launchpad non è un repository per codice sorgente ma piuttosto è stato progettato per offrire supporto alla supervisione del codice, degli incidenti e delle traduzioni. Per raggiungere tale obiettivo, Launchpad utilizza lo strumento Malone, già menzionato, che consente di reindirizzare gli incidenti ad ogni repository di codice dei moduli interessati.

### **8.9.3. Altri siti e programmi**

Naturalmente, i sistemi di collaborazione sono stati e continuano a essere sviluppati; alcune aziende fondano il loro business sulla manutenzione e sul servizio a quei siti. Ad esempio, il progetto Tigris (Tigris.org: Open Source Software Engineering Tools) [64], che non solo sostiene progetti d'ingegneria free software, si avvale anche di un portale di collaborazione (SourceCast) gestito da una società di servizi (CollabNet), che si occupa dei siti dei singoli progetti, come OpenOffice.org. I nuovi siti emergenti adottano nuovi software liberi, come ad esempio G-Force (<http://gforge.org>) [30], utilizzato dal progetto Debian (<http://alioth.debian.org>) [5]. Possiamo trovare un confronto dettagliato di molti siti in: "Comparison of free/open source hosting (FOSPhost) sites available for hosting projects externally from project owners"[202].

## 9. Casi di studio

"Il termine GNU, che sta per "Gnu's Not Unix", ovvero GNU non è Unix, è il sistema di software completo e compatibile con Unix che sto scrivendo in modo che possa distribuirlo liberamente a tutti coloro che sanno usarlo. Molti altri volontari mi stanno aiutando. Qualsiasi contributo in termini di tempo, denaro, programmi e attrezzature sarà molto apprezzato".

Richard Stallman, "The GNU Manifesto" (1985)

Questo capitolo fornisce un'analisi più approfondita su alcuni dei progetti più interessanti di software libero in termini di impatto sul mondo del free software, dei risultati ottenuti, dei modelli di gestione, di sviluppo storico, etc. Il numero di progetti che qui possiamo discutere è sicuramente molto più piccolo rispetto al numero totale di progetti software libero esistenti (decine di migliaia); il che significa che questo capitolo non dovrebbe essere considerato come esaustivo, e mai lo potrà essere. Tuttavia, ci auguriamo che i lettori, dopo aver letto questo capitolo, avranno una conoscenza almeno basilare su come vengano messe in pratica le teorie che abbiamo discusso nel corso questo libro.

I progetti che abbiamo scelto vanno da un livello elementare di applicazioni, che interagiscono più con il sistema fisico del computer che con l'utente, fino ad ambienti di lavoro progettati per l'utilizzatore finale. Abbiamo incluso anche progetti di software libero che non sono, in linea di principio, strettamente progetti di sviluppo. Ciò è vero soprattutto per le distribuzioni, che tendono ad essere usate come sistemi di integrazione, in quanto usano un ampio ma limitato set di applicazioni indipendenti e li usano per creare un sistema in cui tutto interagisce in modo efficace, comprese le opzioni per l'installazione, l'aggiornamento e la cancellazione di nuove applicazioni, in base alle esigenze dell'utente.

I progetti di livello più basilare che vedremo sono Linux, il kernel del sistema operativo libero più famoso al giorno d'oggi, e FreeBSD, che unisce il kernel della famiglia BSD con una serie di applicazioni e utility fatta da progetti terzi, che è il metodo più puro per le distribuzioni. Gli ambienti di lavoro per utenti finali che saranno analizzati sono KDE e GNOME, che sono senza dubbio i più diffusi e conosciuti. Per quanto riguarda i server, uno degli aspetti principali dei sistemi liberi, esamineremo Apache, leader nel settore dei server WWW. Allo stesso modo sarà presentato Mozilla, uno dei client WWW (che in realtà è molto di più di questo) al quale possiamo fare affidamento nel mondo del free software. L'ultimo progetto che sarà preso in considerazione in questo capitolo è OpenOffice.org, un pacchetto Office IT libero (*suite*).

Per finire, abbiamo pensato che sarebbe stato opportuno studiare i dettagli di due delle distribuzioni più note, Red Hat Linux e Debian GNU/Linux, e di confrontare le loro dimensioni con quelle di altri sistemi ampiamente utilizzati come Microsoft Windows o Solaris.

Dopo aver trattato ciascun caso di studio, forniremo una tabella contenente le caratteristiche più importanti di ogni applicazione o progetto. Uno dei fattori che probabilmente i lettori troveranno più sorprendente, saranno i risultati delle stime di costo e di durata e il numero di sviluppatori richiesti. Questi risultati sono stati ottenuti con metodi tipicamente impiegati nell'ingegneria del software, in particolare con il modello di stima del costo dei software COCOMO. Il modello COCOMO (*Software Engineering Economics*, 1981) [93], prende come misurazione di partenza il numero di linee di codice sorgente e genera le stime del costo totale, il tempo di sviluppo e lo sforzo necessario per creare il software. COCOMO è un modello progettato per i processi "classici" di generazione dei software (il modello cascata o le evoluzioni del modello V) e per i progetti di medie o grandi dimensioni; pertanto, le cifre che tale modello fornisce per alcuni dei casi che analizzeremo, dovrebbero essere prese con alcune riserve. In ogni caso, i risultati possono contribuire a dare un'idea dell'ampia scala sulla quale stiamo lavorando e della quantità di sforzo che sarebbe necessario per raggiungere gli stessi risultati con un modello di sviluppo di software proprietario.

In generale, la cifra più sorprendente fra tutte quelle risultanti dal modello COCOMO è la stima di costo. In questa stima, vengono presi in considerazione due fattori: lo stipendio medio di uno sviluppatore e le *overheads*. Per il calcolo dei costi stimati, il salario medio di un programmatore di sistemi, a tempo pieno, viene preso a decorrere dall'anno 2000 "Salary survey 2000" [235]. Le spese generali (*overheads*) sono i costi aggiuntivi che tutte le società devono sostenere affinché il prodotto possa essere rilasciato, a prescindere dal salario versato per i programmatori. Questo va dagli stipendi delle segretarie e del team di marketing ai costi per le fotocopie, la luce, le apparecchiature hardware, etc. In sintesi, il costo calcolato dal COCOMO è il costo totale che una società dovrebbe sostenere per creare software delle dimensioni specificate e va ricordato che i programmatori ricevono solo una parte di questa somma per la progettazione del software. Una volta che questo viene calcolato, i costi non sembrano più così eccessivi.

## 9.1. Linux

Il kernel di Linux è senza dubbio l'applicazione star nel mondo dei software liberi, al punto che, nonostante costituisca solo una minuscola parte del sistema, il suo nome viene usato per definire il tutto. Inoltre, si potrebbe persino dire che lo stesso software libero viene spesso confuso con Linux, il che è un errore abbastanza grave dato che esistono software liberi che girano su sistemi non basati su Linux (in realtà, uno degli obiettivi più ambiti di questo movimento e di molti progetti di free software è quello di creare applicazioni che

possano girare su numerosi ambienti). D'altra parte, ci sono anche applicazioni che lavorano in Linux ma che in realtà non sono software liberi (Acrobat Reader, il lettore di documenti PDF, per il quale esiste anche una versione di Linux).

#### **Nota**

A dir la verità, esistono vari progetti che integrano e distribuiscono applicazioni libere che girano sui sistemi Windows, per evitare che il software libero venga associato esclusivamente ai sistemi Linux. Uno dei pionieri in questo settore (e che probabilmente è diventato il più noto e completo) è stato GNUWin, che fu distribuito su CD auto-avviabili con più di un centinaio di applicazioni libere per i sistemi Win32. La maggior parte di queste applicazioni sono disponibili anche nelle comuni distribuzioni di GNU/Linux, che hanno reso GNUWin un valido strumento per preparare una transizione semplice e graduale da un sistema Windows a uno GNU/Linux. Come all'inizio del 2007, ci sono altri sistemi simili che sono a disposizione, come ad esempio WinLibre.

### **9.1.1. Storia di Linux**

La storia di Linux è una delle storie più conosciute nel campo del software libero, probabilmente perché ha i tratti più di una leggenda che di una storia di un programma per computer. Nel 1991, uno studente finlandese chiamato Linux Torvalds decise che voleva imparare a utilizzare la modalità protetta 386 su una macchina che il suo modico reddito gli aveva permesso di acquistare. A quel tempo, c'era un kernel nel sistema operativo chiamato Minix, progettato per scopi accademici e per l'uso nei corsi universitari sui sistemi operativi; Minix è in uso ancora oggi. Andrew Tanenbaum, uno dei professori più prestigiosi dell'università, era il leader di un team che lavorava sullo sviluppo di Minix, basato sui sistemi Unix tradizionali. Minix era un sistema limitato ma piuttosto competente e ben progettato, ed era al centro di una grande comunità accademica e d'ingegneria.

Il sistema Minix aveva una licenza di distribuzione libera e poteva essere utilizzato facilmente per scopi accademici, ma aveva il grande svantaggio che non poteva essere migliorato dalle persone che non lavoravano o non studiavano all'Università di Amsterdam; al contrario, questi miglioramenti dovevano essere fatti in modo indipendente, di solito usando correzioni patches. Ciò significa che, in pratica, c'era una versione ufficiale di Minix che tutti utilizzavano e poi una lunga serie di correzioni che dovevano essere applicate in un secondo momento per ottenere funzioni supplementari.

A metà del 1991, Linus, l'ancora anonimo studente finlandese, inviò un messaggio al newsgroup Minix annunciando che stava per iniziare a lavorare su un kernel del sistema operativo basato su Minix, da zero, ma senza includere lo stesso codice. A quel tempo, anche se Linus non aveva esplicitamente dichiarato che stava per pubblicarlo con licenza libera, aveva notato che il sistema che stava per creare non avrebbe avuto le *barriere* che aveva Minix; questo rivelava che, a sua insaputa, e probabilmente senza volerlo in realtà, stava facendo il primo passo nel rendere sua la comunità che a quel tempo si riuniva attorno a Minix.

La versione 0.02, che risale all'ottobre 1991, pur essendo molto limitata, poteva già eseguire terminali *bash* e il compilatore GCC. Nel corso dei mesi successivi, il numero di contributi provenienti dall'esterno crebbe a tal punto che nel marzo 1992, Linus riuscì a pubblicare la versione 0.95, che fu ampiamente riconosciuta come quasi stabile. C'era ancora molta strada da fare comunque, prima di arrivare alla versione 1.0, che è comunemente considerata la prima versione stabile: nel dicembre 1993, per esempio, fu pubblicata la versione 0.99p114 (che la renderebbe la quattordicesima versione corretta della versione 0.99); nel marzo 1994, nacque finalmente Linux 1.0. Da quel momento in poi, Linux veniva pubblicato sotto i termini della licenza GPL: secondo lo stesso Torvalds, questa fu una delle migliori decisioni che abbia mai preso, in quanto è stata molto utile nella distribuzione e nella divulgazione del suo kernel. In "Evolution in open source software: a case study", [128] vi è un'analisi approfondita sull'evoluzione delle diverse versioni del kernel di Linux, dove viene messa in risalto la sua dimensione e modularità.

#### Note

Un altro evento significativo negli annali del software libero fu lo scontro tra Andrew Tanenbaum e Linus Torvalds nel newsgroup Minix, che avvenne verso la fine del gennaio 1992. Tanenbaum, che probabilmente era un pò infastidito dal successo di Torvalds con il suo "giocattolo", attaccò Linux e Linus in maniera un pò esagerata: la sua convinzione era che Linux è un sistema monolitico (il kernel integra tutti i gestori e il resto) e non un sistema *microkernel* (il kernel ha un design modulare, il che significa che può essere molto più piccolo e che i moduli possono essere caricati su richiesta). La discussione autentica, così come avvenne, si può trovare nel newsgroup "The Tanenbaum-Torvalds debate" [214].

#### 9.1.2. La modalità di lavoro di Linux

Il modo con cui Torvalds lavorava non era molto comune per quel periodo. Lo sviluppo era basato principalmente su una mailing list<sup>9</sup>. La mailing list non era solo un luogo dove la gente discuteva ma anche dove avvenivano dei progressi. Questo perché Torvalds era estremamente interessato a che la vita intera del progetto si riflettesse sulla mailing list, che è il motivo per cui chiedeva alle persone di inviare le loro correzioni a questa mailing list. Contrariamente a quanto ci si poteva aspettare (le correzioni inviate come allegati), Linus preferiva che il codice fosse inviato nel testo del messaggio così che lui e gli altri potevano commentare il codice. In ogni caso, sebbene molte persone fornivano le loro opinioni e spedivano correzioni o nuove funzioni, l'ultima parola spettava sempre a Linus Torvalds, che decideva quale codice Linux sarebbe stato incorporato. Questo è il modo in cui, in buona parte, funziona ancora nel 2007.

<sup>(9)</sup> L'elenco e-mail è [linux-kernel@vger.kernel.org](mailto:linux-kernel@vger.kernel.org). I messaggi storici possono essere consultati a <http://www.uwsg.indiana.edu/hypermail/linux/kernel/>.

**Note**

Il consolidamento di Linus Torvalds come "dittatore benevolo" ha dato origine a numerosi aneddoti legati a tale progetto. Per esempio, si dice che se un'idea piace, deve essere attuata. Se non piace, deve allo stesso modo essere attuata. Il corollario, quindi, è che le buone idee non sono di alcuna utilità, senza codice ovviamente. D'altra parte, se l'implementazione non è ben voluta, è fondamentale insistere. Un caso ben noto è quello di Gooch, per il quale Giacobbe era un principiante. Gooch fece fino a centoquarantasei correzioni in parallelo fino a che finalmente Linus decise di integrarle nel ramo ufficiale del kernel.

Un'altra delle idee innovative Torvalds era di sviluppare due rami del kernel in parallelo: uno stabile (il secondo numero della versione è solitamente pari, come in 2.4.18) e uno instabile (il secondo numero della versione è dispari, come ad esempio 2.5.12). Come sempre, Torvalds è la persona che decide cosa va in quale ramo (molte delle decisioni più controverse sono legate proprio a questo punto). In ogni caso, Linux non ha consegne prefissate, in tempi già stabiliti: è pronto quando sarà pronto e, nel frattempo, dovremo solo aspettare. Sicuramente a questo punto, molti lettori avranno intuito che la decisione sul fatto se il sistema è pronto o meno, dipenderà esclusivamente da Linus.

Il metodo di sviluppo usato in Linux si è rivelato molto efficace in termini di risultati: Linux è molto stabile e qualsiasi bug è corretto in modo estremamente rapido (a volte in pochi minuti), grazie al fatto che ha migliaia di sviluppatori. In questo senso, quando c'è un bug, la probabilità che qualcuno lo trovi è molto alta, e se la persona che lo scopre non è in grado di correggerlo, si presenterà qualcuno disposto a trovare una soluzione in poco tempo. In sintesi, questo mostra come Linux disponga di migliaia di persone che ogni mese lavorano sul suo sviluppo, che è il motivo per cui il suo successo non è del tutto sorprendente.

Va osservato, tuttavia, che questo modo di lavorare è molto costoso quando si parla di risorse. Non è raro che vi siano molte proposte per una nuova funzione che non si escludono una con l'altra, o che, per lo stesso bug, vengano ricevute decine di correzioni. Nella maggior parte dei casi, solo una delle correzioni sarà definitivamente inserita nel kernel, il che significa che il tempo e lo sforzo impiegati dagli altri sviluppatori nel formulare le correzioni sono stati vani. Il modello di sviluppo di Linux è, quindi, un modello che funziona molto bene in Linux, ma che non tutti i progetti si possono permettere.

**9.1.3. Lo stato attuale di Linux**

Agli inizi del 2007, Linux era alla versione 2.6, che comprendeva, in termini di miglioramenti apportati alla versione 2.4: NUMA (Non-Uniform Memory Access, molto utilizzato nei multiprocessori), nuovi sistemi di file, miglioramenti alla comunicazione nelle reti wireless e architetture sonore (ALSA) e molti altri progressi (se si è interessati ai dettagli delle modifiche rispetto alle versioni precedenti, si può consultare "The wonderful world of Linux 2.6" [186]).

Nel corso degli ultimi anni, il modello di sviluppo di Linux ha subito alcune modifiche. Sebbene la mailing list di sviluppo sia ancora il *cuore* del progetto, il codice non deve più necessariamente passare attraverso di essa. Una delle cose che ha maggiormente contribuito in questa direzione è BitKeeper, un sistema proprietario che esegue il controllo di revisione, sviluppato dalla società BitMover, che segue rigorosamente le raccomandazioni di Linus Torvalds. L'utilizzo di questo strumento proprietario ha sollevato numerose polemiche, nelle quali Linus, fedele alla forma, dimostrò ancora una volta il suo pragmatismo, in quanto secondo lui e secondo molti altri, il sistema di controllo della versione CVS era molto antiquato. Le divergenze furono messe a tacere con lo sviluppo di *git*, un sistema di controllo di revisione con caratteristiche simili a BitKeeper che è attualmente usato per lo sviluppo di Linux. Più specificamente, il processo di sviluppo di Linux segue una gerarchia piramidale, nella quale partendo dalla base, gli sviluppatori suggeriscono alcune correzioni condivise via e-mail tra gli stessi livelli; le correzioni devono poi essere accettate dal livello successivo, costituito dal supervisore e dai responsabili della manutenzione di file. Questi ultimi si trovano ad un livello superiore, mentre Linus Torvalds e Andrew Morton sono al livello più alto della piramide e hanno l'ultima parola per quanto riguarda l'accettazione delle correzioni.

Per riassumere, la seguente tabella fornisce una "radiografia" del progetto Linux, nella quale si vede che ora Linux ha più di cinque milioni di linee di codice e che può quindi essere considerato come uno fra i più importanti progetti di software libero (insieme a Mozilla e OpenOffice.org). Per quanto riguarda la stima del tempo necessario per elaborare tale progetto e la stima del numero di sviluppatori mediamente necessario, bisogna notare che la prima stima è sicuramente inferiore rispetto al tempo di vita di Linux. D'altra parte, questo è più che bilanciato dalla seconda stima, dato che il numero medio di sviluppatori full-time che sarebbe necessario per un progetto simile è superiore al numero che sia mai stato disponibile per Linux.

#### Nota

La stima del costo che COCOMO offre, è attorno ai 215 milioni di dollari USA, una somma che, nell'ambito dei calcoli giornalieri che si possono pensare, rappresenterebbe il doppio di quanto i migliori club di football sarebbero disposti a pagare per un importante giocatore.

**Tabella 4. Analisi di Linux**

Website	<a href="http://www.kernel.org">http://www.kernel.org</a>
Inizio del progetto	Primo messaggio su <a href="http://news.comp.os.minix">news.comp.os.minix</a> : Agosto 1991
Licenza	GNU GPL
Versione analizzata	2.6.20 (version stabile dal 20/02/2007)
Linee di codice sorgente	5,195,239
Stima del costo (secondo COCOMO di base)	\$ 215,291,772

Stima di tempo di progettazione (secondo COCOMO di base)	8,83 anni (105,91 mesi)
Stima del numero medio di sviluppatori (secondo COCOMO di base)	180,57
Numero approssimativo di sviluppatori	Questo numero è stimato in migliaia (anche se solo cento appaiono nei ringraziamenti [219])
Strumenti di assistenza allo sviluppo	Mailing list e <i>git</i>

La composizione di Linux in termini di linguaggi di programmazione mostra una chiara predominanza di C, che è considerato un linguaggio ideale per la progettazione di sistemi dove la velocità è un elemento chiave. Quando la velocità è un requisito inflessibile che nemmeno C può raggiungerlo, per la programmazione si usa direttamente un linguaggio assembly e ciò succede, come si può constatare, con una certa frequenza. Lo svantaggio del linguaggio assembly rispetto a C, è che non è molto portabile. Ogni architettura ha il proprio insieme di istruzioni particolari, il che significa che la maggior parte del codice scritto per un'architettura in linguaggio assembly deve essere portata su altre architetture. L'incidenza del resto dei linguaggi è marginale, come illustrato nella tabella qui sotto, e i linguaggi sono limitati alle funzioni delle installazioni e alle utility di sviluppo. La versione analizzata in questo libro è stata la Linux 2.6.20, poiché questo libro è stato pubblicato il 20 febbraio 2007 (senza includere tutte le successive correzioni).

**Tabella 5. Linguaggi di programmazione usati in Linux**

Linguaggio di programmazione	Linee di codice	Percentuale
C	4.972.172	95,71%
Assembler	210.693	4,06%
Perl	3.224	0,06%
Yacc	2.632	0,05%
Shell	2.203	0,04%

## 9.2. FreeBSD

Come abbiamo accennato nel capitolo sulla storia del free software, ci sono altri tipi di sistemi operativi liberi, oltre ai noti GNU/Linux. Una famiglia di questi sono gli "eredi" delle distribuzioni dell'Università di Berkeley, in California (USA): sistemi di tipo BSD. Il sistema BSD più antico e più conosciuto è FreeBSD, che fu creato all'inizio del 1993, quando Bill Jolitz smise di pubblicare gli aggiornamenti non ufficiali al 386BSD. Un gruppo di volontari decise di



portare avanti la creazione di questo sistema operativo libero con l'assistenza della società Walnut Creek CDROM, che successivamente cambiò il suo nome in BSDi.

L'obiettivo principale del progetto FreeBSD è la creazione di un sistema operativo che può essere usato senza alcun tipo di obblighi o vincoli, ma che ha tutti i vantaggi della disponibilità di codice e che viene processato accuratamente per garantire la qualità del prodotto. L'utente è libero di fare qualsiasi cosa con il software: modificarlo secondo i propri desideri, ridistribuirlo in forma aperta o persino in una forma chiusa, secondo i termini che desidera, con o senza modifiche. Come indica il nome stesso, il progetto FreeBSD è basato pertanto sulla filosofia delle licenze BSD.

### 9.2.1. Storia di FreeBSD

La versione 1.0 apparve verso la fine del 1993 ed era basata su 4.3BSD Lite (Net/2) e 386BDS. La versione 4.3BSD Net/2 aveva un codice che era stato creato negli anni settanta, quando AT&T stava sviluppando Unix. Tale versione, come poi si dimostrò, comportò una serie di problemi legali che non vennero risolti fino al 1995, quando la versione FreeBSD 2.0 fu pubblicata senza il codice originale sviluppato da AT&T ma basato su 4.4BSD-Lite, una versione *light* di 4.4BSD rilasciata dalla University of California (in quest'ultima versione molti di questi moduli erano stati eliminati per motivi legali, oltre al fatto che la *porta* per i sistemi Intel era ancora incompleta).

La storia di FreeBSD non sarebbe completa se trascurassimo le sue distribuzioni "sorelle": NetBSD e OpenBSD. NetBSD apparve come versione 0.8 a metà del 1993. Lo scopo principale era la massima portabilità (anche se all'inizio era solo un adattamento per i386); perciò il motto del prodotto fu: "Naturalmente può girarci NetBSD" (Of course it runs NetBSD). OpenBSD nacque a metà del 1996 dalla divisione di NetBSD causata da differenze filosofiche tra gli sviluppatori (così come dalle differenze personali). Il centro dell'attenzione è rivolto soprattutto alla sicurezza e alla crittografia e si dice sia il sistema operativo più sicuro che esista, anche se, siccome si basa su NetBSD, è altamente portabile.

### 9.2.2. Lo sviluppo in FreeBSD

Il modello di sviluppo utilizzato dal progetto FreeBSD è basato principalmente su due strumenti: Il sistema di controllo di versione CVS e GNATS, il software bug-tracking. L'intero progetto si basa su questi due strumenti, com'è confermato dal fatto che è stata creata una gerarchia in conformità a questi strumenti. In effetti, sono i *committers* (sviluppatori che hanno il permesso di scrittura nel repository CVS) che hanno la massima autorità sul progetto, direttamente o indirettamente attraverso la scelta del *core group*, come vedremo nella prossima sezione.

Nel software GNATS non è necessario essere *committer* per segnalare bug: ciò significa che chiunque desideri può fare un report di un bug. Tutti i contributi (*aperti*) in GNATS vengono valutati da un *committer* che può assegnare il compito (*analizzato*) ad un altro *committer* o richiedere maggiori informazioni da parte della persona che ha segnalato il bug all'inizio (*feedback*). Ci sono situazioni in cui il bug è stato risolto per alcuni rami recenti e perciò sarà indicato con lo stato sospeso. In ogni caso l'obiettivo è che il report venga *chiuso*, una volta che l'errore è stato corretto.

FreeBSD distribuisce il suo software in due forme: da un lato, le porte, un sistema che scarica codici sorgente, li compila e installa l'applicazione sul computer locale, e, dall'altro lato, i pacchetti, che sono semplicemente i codici sorgente delle porte precompilate e, pertanto, in (codice) binario. Il vantaggio più interessante delle porte rispetto ai pacchetti è che le porte permettono appunto all'utente di configurare e ottimizzare il software per i loro computer. D'altro canto però, nel sistema dei pacchetti, visto che sono già precompilati, ci vuole molto meno tempo per installare il software.

### 9.2.3. Il processo decisionale in FreeBSD

Il consiglio di amministrazione di FreeBSD, noto come il celebre *core team*, ha il compito di definire la direzione del progetto e assicurare che vengano raggiunti gli obiettivi, oltre alla mediazione nei casi in cui ci sono conflitti tra *committers*. Fino all'ottobre 2000 questo team è sempre stato un gruppo chiuso nel quale si poteva entrare a far parte solo tramite invito esplicito da parte del *core team* stesso. Dall'ottobre 2000, i membri vengono eletti periodicamente e democraticamente dai *committers*. Le regole più importanti per l'elezione del *core team* sono le seguenti:

- 1) Hanno diritto di voto i *committers* che hanno fatto almeno un *commit* nell'ultimo anno trascorso.
- 2) Il Consiglio di Amministrazione è rinnovato ogni due anni.
- 3) I membri del consiglio di amministrazione possono essere "espulsi" con un voto pari ai due terzi dei *committers*.
- 4) Se il numero dei membri del consiglio d'amministrazione è inferiore a sette, si terranno nuove elezioni .
- 5) Le nuove elezioni si tengono quando un terzo dei *committers* vota a questo proposito.
- 6) Eventuali modifiche alle regole richiedono un quorum pari ai due terzi dei *committers*.

#### 9.2.4. Aziende che lavorano intorno a FreeBSD

Ci sono numerose aziende che offrono prodotti e servizi basati su FreeBSD, che sono elencati sul sito web del progetto FreeBSD. In questa presentazione conosceremo gli aspetti più significativi di FreeBSD: BSDi e Walnut Creek CDROM.

La nascita di FreeBSD è dovuta in parte alla costituzione della società BSDi nel 1991 da parte dei membri del CSRG (Computer Systems Research Group) dell'Università di Berkeley, che dava sostegno commerciale al nuovo sistema operativo. Oltre alla versione commerciale del sistema operativo FreeBSD, BSDi sviluppò anche altri prodotti, come ad esempio un server Internet e un server gateway.

Walnut Creek CDROM è stato creato con l'obiettivo di commercializzare FreeBSD come prodotto finale, in modo che poteva essere considerato come una distribuzione secondo lo stile di quelli esistenti per GNU/Linux, ma con FreeBSD. Nel novembre 1998, Walnut Creek allargò i suoi orizzonti con la creazione del portale FreeBSD Mall, che commercializzava tutti i tipi di prodotti basati su FreeBSD (dalla distribuzione stessa a magliette, riviste, libri, ecc), annunciava prodotti di terzi sul suo sito web e forniva un sostegno professionale a FreeBSD.

Nel marzo 2000, avvenne la fusione fra BSDi e Walnut Creek sotto il nome di BSDi, allo scopo di unire le forze contro il fenomeno Linux che stava chiaramente lasciando in ombra i sistemi BSD in generale, e FreeBSD nello specifico. Un anno dopo, nel maggio 2001, Wind River acquisì la parte che era dedicata alla creazione del software BSDi, con il chiaro scopo di accelerare lo sviluppo di FreeBSD per il suo utilizzo in sistemi dedicati e dispositivi intelligenti connessi alla rete.

#### 9.2.5. Stato attuale di FreeBSD

Secondo gli ultimi dati del sondaggio che la Netcraft esegue periodicamente, il numero di web server che eseguono FreeBSD è di circa due milioni. Un nuovo utente che volesse installare FreeBSD potrebbe scegliere tra la versione 6.2 (che può essere considerata come la versione "stabile") o il ramo più avanzato o "sviluppato". Mentre il primo garantisce una maggiore stabilità, soprattutto in settori come il multiprocessing simmetrico che è stato completamente ridisegnato nelle versioni più recenti, il secondo permette agli utenti di godersi gli ultimi sviluppi. È altresì importante ricordare che le versioni sviluppate tendono a includere il codice test che influisce leggermente sulla velocità del sistema.

Una delle caratteristiche fondamentali di FreeBSD è quella che è conosciuta come *jails*. Le jails riducono al minimo i danni che potrebbero essere causati da un attacco sui servizi di rete di base, come ad esempio Sendmail per le e-mail o

BIND (Berkeley Internet Name Domain) per la gestione dei nomi. I servizi sono messi in una cella in modo che possano girare in un ambiente isolato. Le jails possono essere gestite utilizzando una serie di utility contenute in FreeBSD.

### 9.2.6. Radiografia di FreeBSD

Come abbiamo già accennato in nell'ultimo paragrafo, le funzioni di BSD non sono esclusivamente limitate allo sviluppo di un kernel del sistema operativo, ma includono anche l'integrazione di una gran numero di utility che sono distribuite secondo lo stile delle distribuzioni GNU/Linux. Il fatto che il processo di sviluppo di FreeBSD è strettamente legato al sistema di controllo delle versioni CVS significa che attraverso lo studio del sistema, ci si può fare un'idea di cosa consiste FreeBSD. I risultati riportati di seguito corrispondono alle analisi FreeBSD effettuate il 21 agosto 2003.

Uno degli aspetti più interessanti di FreeBSD è che i numeri sono analoghi a quelli che abbiamo già visto in KDE e GNOME: la dimensione del software supera senza problemi i cinque milioni di linee di codice, i file sono circa 250.000 e il numero totale di *commits* è approssimativamente due milioni. Tuttavia, è importante osservare che la principale differenza tra GNOME e KDE rispetto a FreeBSD è l'età del progetto. FreeBSD ha recentemente compiuto il decimo anno ed è stato presente per quasi il doppio del tempo in confronto agli ambienti desktop con i quali ci si confronta. Che la somiglianza sia simile è in parte dovuto al fatto che FreeBSD non ha attirato così tanti sviluppatori, sebbene il periodo di sviluppo è stato più lungo. Esiste un elenco di circa quattrocento sviluppatori con permesso di scrittura al CVS (*committers*), mentre il numero di sviluppatori elencati nel Manuale di FreeBSD è di circa un migliaio. È per questo motivo che l'attività registrata nel CVS di FreeBSD è inferiore alla media (cinquecento *commits* al giorno) rispetto a quella registrata in entrambi GNOME (novecento) e KDE (millesettecento, tra cui i *commits* automatici).

Come sistema base di FreeBSD si considera tutto ciò che è inserito nella directory *src/src* del modulo *root* del CVS. L'attività che è stata registrata nel sistema base negli ultimi dieci anni è costituita da più di mezzo milione di *commits*. Ci sono più di cinque milioni di linee di codice, anche se bisogna ricordare che questo numero non comprende solo il kernel ma molti servizi aggiuntivi, tra cui i giochi. Se prendiamo in considerazione solo il kernel (che si trova nella sottodirectory *sys*), la grandezza è di 1,5 milioni di linee di codice sorgente, prevalentemente in C.

È interessante osservare come la stima di tempo data da COCOMO corrisponde esattamente al tempo reale del progetto FreeBSD, sebbene la stima del numero medio di sviluppatori sia molto più elevata rispetto al numero effettivo. Bisognerebbe inoltre ricordare che durante l'anno appena trascorso, solo set-

tantacinque *committers* sono stati attivi, mentre COCOMO suppone che nel corso degli ultimi dieci anni di sviluppo, il numero di sviluppatori doveva essere 235.

Infine, dobbiamo ricordare, come abbiamo detto, che l'attività principale di FreeBSD è basata attorno al CVS, al sistema di controllo dei bug e alle attività di GNATS.

**Tabella 6. Analisi di FreeBSD**

Website	<a href="http://www.FreeBSD.org">http://www.FreeBSD.org</a>
Inizio del progetto	1993
Licenza	Di tipo BSD
Versione analizzata	4,8
Linee di codice sorgente.	7,750,000
Linee di codice sorgente (solo kernel)	1,500,000
Numero di file	250.000
Stima di costo	\$ 325.000.000
Stima di tempo di esecuzione	10,5 anni (126 mesi)
Stima del numero medio di sviluppatori	235
Numero approssimativo di sviluppatori	400 <i>committers</i> (1.000 collaboratori)
Numero di <i>committers</i> attivi nell'ultimo anno	75 (meno del 20% del totale)
Numero di <i>committers</i> attivi negli ultimi due anni	165 (approssimativamente il 40% del totale)
Numero di <i>commits</i> nel CVS	2.000.000
Numero medio di <i>commits</i> (totale) giornaliero	Circa 500
Strumenti di assistenza allo sviluppo	CVS, GNATS, mailing list e sito di notizie

C è il linguaggio predominante in FreeBSD e mantiene una maggiore distanza da C++ rispetto all'altro caso che abbiamo studiato in questo capitolo. È interessante notare che il numero di linee di codice in linguaggio assembly contenute in FreeBSD, corrisponde a quello di Linux in termini di dimensione, anche se quelli relativi al kernel sono solo venticinquemila in totale. In sintesi, potremmo dire che in FreeBSD predominano linguaggi più *classici* all'interno del software libero (C, Shell e Perl) e che gli altri linguaggi che sono stati esaminati in altre applicazioni e progetti (C++, Java , Python) non sono stati integrati.

**Tabella 7. Linguaggi di programmazione usati in FreeBSD**

Linguaggio di programmazione	Linee di Codice	Percentuale
C	7.080.000	92,0%
Shell	205.000	2,7%
C++	131.500	1,7%
Assembler	116.000	1,5%
Perl	90.900	1,20%
Yacc	5.800	0,75%

### 9.2.7. Studi Accademici su FreeBSD

Nonostante sia certamente un progetto molto interessante (si può guardare la sua storia attraverso l'analisi del sistema delle versioni, risalendo fino a dieci anni!), FreeBSD non ha suscitato molto interesse all'interno della comunità scientifica. Vi è, tuttavia, un gruppo di ricerca che ha concentrato la sua attenzione sul progetto FreeBSD da vari punti di vista ("Incremental and decentralized integration in FreeBSD") [149], che si è concentrato in particolare su come vengono risolti i problemi di integrazione di software in maniera incrementale e decentralizzata.

## 9.3. KDE

Anche se, con tutta probabilità, non era la prima soluzione in termini di ambienti desktop user-*friendly*, la diffusione del sistema operativo Windows 95 a metà del 1995 comportò un cambiamento radicale nel modo in cui gli utenti non specializzati interagivano con i computer. Dai sistemi mono-dimensionali delle linee di istruzioni (i terminali) nacque la metafora dell'ambiente desktop bidimensionale, dove il mouse ha cominciato ad essere più usato della tastiera. A Windows 95, più di un'innovazione tecnologica, va dato il credito per essere stato il sistema che è riuscito a raggiungere a tutti gli ambienti personali e di ufficio, fissando gli standard che sarebbero stati seguiti in futuro (regole tecniche e sociali di cui, in alcuni casi, si sta ancora soffrendo all'inizio del ventunesimo secolo).

Prima che i sistemi desktop fossero creati, ogni applicazione gestiva in modo autonomo il proprio tema grafico e il modo di interagire con l'utente. Sui desktop, tuttavia, le applicazioni devono avere caratteristiche comuni e un tema che sia condiviso dalle applicazioni, che aiuta ad alleggerire la pressione sull'utente; quest'ultimo infatti può *riutilizzare il modo di interagire* che ha appreso con un'applicazione, quando ne utilizza un'altra. Questo permetteva di alleviare la pressione anche sugli sviluppatori di applicazioni, in quanto non

avevano il problema di creare elementi interattivi a partire da zero (che è sempre un compito complesso), ma potevano iniziare da una struttura e da regole predefinite.

### 9.3.1. Storia di KDE

I seguaci di Unix notarono subito l'eccezionale successo di Windows 95 e, dato che gli ambienti Unix non avevano sistemi così intuitivi pur essendo liberi, decisero di mettersi al lavoro. Il progetto dell'ambiente desktop KDE K è nato da quest'opera nel 1996; è stato disegnato da Matthias Ettrich (creatore di LyX, un programma di editing nel comporre TeX) e da altri *hackers*. Il progetto KDE ha proposto i seguenti obiettivi:

- Fornire i sistemi Unix con un ambiente user-friendly che era allo stesso tempo aperto, stabile, affidabile e potente.
- Sviluppare un set di librerie per la scrittura di applicazioni standard su un sistema grafico per Unix X11.
- creare una serie di applicazioni che permettevano all'utente di raggiungere gli obiettivi in modo efficace ed efficiente.

#### Nota

Originariamente, il nome di KDE stava per Kool Desktop Environment, ma è stato poi modificato semplicemente in K Desktop Environment. La spiegazione ufficiale era che nell'alfabeto latino la lettera K è giusto prima della L, che sta per Linux.

Scoppiò una polemica quando i membri del progetto KDE appena creato, decisero di utilizzare una libreria orientata agli oggetti chiamata Qt, appartenente alla società norvegese Trolltech, che non aveva nessuna licenza per software libero. Si scoprì poi che, sebbene fossero sotto licenza GPL, le applicazioni KDE erano collegate a questa libreria, il che significava che era impossibile ridistribuirle. Di conseguenza, si violava una delle quattro libertà sancite da Richard Stallman nel Manifesto del Software Libero [117]. A partire dalla versione 2.0, Trolltech distribuisce Qt con doppia licenza che specifica che, se l'applicazione che usa la libreria opera sotto la licenza GPL, allora la licenza valida per Qt è la GPL. Grazie a questo, uno dei dibattiti più accesi nel mondo del software libero ha avuto, per fortuna, un lieto fine.

### 9.3.2. Sviluppo di KDE

KDE è uno dei pochi progetti di software libero, che di norma segue un piano per il lancio di una nuova versione (ricordiamo, per esempio, che ci sarà una nuova versione di Linux "quando sarà pronto", mentre, come vedremo più avanti, GNOME ha sempre subito notevoli ritardi quando si trattava di rilasciare nuove versioni). La numerazione delle nuove versioni segue una procedura ben definita. Le versioni di KDE hanno tre numeri di versione: uno superiore e due inferiori. Ad esempio, in KDE 3.1.2, il numero alto è il 3, mentre l'1 e 2 sono i numeri bassi. Versioni con lo stesso numero più alto hanno compatibilità binaria, il che significa che non è necessario ricompilare le applicazioni. Fino ad oggi, le modifiche nel numero più alto si sono verificati

in parallelo con i cambiamenti nella libreria Qt, che dimostra come gli sviluppatori volevano approfittare delle nuove funzionalità nella libreria Qt nella successiva versione di KDE.

Per quanto riguarda i numeri più bassi, le versioni con un unico numero basso sono versioni in cui sono state incluse nuove funzionalità e nelle quali i bug sono stati corretti. Le versioni con un secondo numero più basso non includono nuove funzionalità rispetto alle versioni con il primo numero inferiore, e contengono solo correzioni di bug. Il seguente esempio spiegherà meglio questa logica: KDE 3.1 è una versione di terza generazione KDE (numero alto 3) a cui sono state aggiunte nuove funzionalità, mentre KDE 3.1.1 è la versione precedente con le stesse funzionalità, ma dove sono stati corretti tutti i bug che sono stati trovati.

KDE è stato realizzato, poco dopo che iniziò il progetto, in un'associazione registrata in Germania (KDE e. V.) e, come tale, gli articoli dell'associazione han fatto sì che ci fosse un comitato di gestione. L'influenza di questo comitato di gestione sullo sviluppo è pari a zero, in quanto il suo compito principale è l'amministrazione dell'associazione, in particolare per quel che riguarda le donazioni ricevute dal progetto. Al fine di promuovere e di diffondere KDE, venne fondata la Lega KDE che comprende tutte le aziende interessate, come vedremo di seguito.

### 9.3.3. La lega KDE

La Lega KDE è un gruppo di aziende e di privati di KDE che hanno l'obiettivo di facilitare la promozione, la distribuzione e lo sviluppo di KDE. Le società e gli individui che partecipano alla lega KDE non devono essere direttamente coinvolti nello sviluppo di KDE (anche se i membri sono incoraggiati a partecipare), ma semplicemente rappresentano una struttura industriale e sociale che è sostenitore di KDE. Gli obiettivi della Lega KDE sono i seguenti:

- Promuovere, assicurare e facilitare l'educazione formale e informale delle funzionalità, capacità e altre qualità di KDE.
- Incoraggiare imprese, governi, aziende e privati ad utilizzare KDE.
- Incoraggiare imprese, governi, aziende e privati a partecipare allo sviluppo di KDE.
- Fornire conoscenza, informazione, gestione e posizionamento su KDE in termini del suo impiego e sviluppo.
- Favorire la comunicazione e la cooperazione tra gli sviluppatori di KDE.
- Favorire la comunicazione e la cooperazione tra gli sviluppatori di KDE e il grande pubblico attraverso pubblicazioni, articoli, siti web, convegni,



partecipazione a conferenze e mostre, articoli di stampa, interviste, materiali promozionali e comitati.

Le aziende che partecipano alla lega KDE sono principalmente designer di distribuzione (SuSE, ora parte di Novell, Mandriva, TurboLinux, Lindows e Hancom, una distribuzione software libera coreana), società di sviluppo (Trolltech e Klarälvdalens Datakonsult AB), il gigante IBM e una società creata allo scopo di promuovere KDE (KDE.com). Di tutti questi, in particolare va citato Trolltech, Novell e Mandriva Software, il cui coinvolgimento è stato essenziale e i cui modelli commerciali sono strettamente legati al progetto KDE:

- Trolltech è una società norvegese con sede a Oslo, che sviluppa Qt, la libreria che può essere usata come interfaccia grafica per l'utente e un API per gli sviluppatori, anche se può funzionare come un elemento integrato nel PDA (ad esempio come la Sharp Zaurus). L'importanza del progetto KDE per Trolltech è evidenziato da due elementi fondamentali nella sua strategia commerciale: da un lato, riconosce KDE come il suo metodo di promozione principale, incoraggiando lo sviluppo del desktop e accettando e implementando i miglioramenti o le modifiche proposte; dall'altro lato, alcuni dei più importanti sviluppatori di KDE lavorano in modo professionale per Trolltech. L'esempio più noto è quello di Matthias Ettrich stesso, colui che fondò il progetto, che senza dubbio trae vantaggio sia dal progetto KDE sia dalla società stessa. Il coinvolgimento di Trolltech nel progetto KDE non è limitato esclusivamente alle librerie Qt, come prova il fatto che uno dei principali sviluppatori di KOffice, pacchetto software office di KDE, attualmente ha un contratto part-time.
- SuSE (ora parte di Novell), ha sempre mostrato una particolare predilezione per il sistema desktop KDE, dovuto in parte al fatto che la maggior parte dei suoi sviluppatori sono di origine tedesca o dell'Europa centrale, come l'azienda stessa. SuSE, consapevole del fatto che maggiore è la facilità dell'ambiente desktop che la sua distribuzione offre, maggiore è la sua attuazione e quindi aumentano le vendite e le richieste di supporto, ha sempre avuto una procedura molto attiva in termini di budget destinato alla professionalizzazione delle posizioni chiave all'interno del progetto KDE. Per fare un esempio, l'attuale amministratore del sistema di controllo di versione e due dei principali sviluppatori, sono tutti sul libro paga di SuSE. Allo stesso modo, all'interno della forza lavoro di SuSE sono presenti una decina di sviluppatori che possono trascorrere parte del loro tempo lavorando allo sviluppo di KDE.
- La distribuzione Mandriva è un altro dei più grandi sostenitori di KDE e alcuni dei suoi principali sviluppatori lavorano per KDE. La sua situazione finanziaria, compreso il suo fallimento a partire dal 2003, ha causato la sua perdita di potere degli ultimi anni.

### 9.3.4. Stato attuale di KDE

Dopo la pubblicazione di KDE il 3 maggio 2002, l'opinione generale è che i desktop liberi sono alla pari con i loro concorrenti proprietari. Alcuni dei suoi più grandi successi rappresentano, in primo luogo, l'integrazione di un sistema di componenti (KParts) che permette di incorporare alcune applicazioni in altre (una parte di un foglio di calcolo KSpread nel processore word KWord) e in secondo luogo, lo sviluppo di DCOP, un sistema semplice che fa sì che processi comunichino l'uno con l'altro dopo la procedura di autenticazione. DCOP è stato l'incarico del progetto che ha agito a danno alle tecnologie CORBA, un argomento largamente discusso all'interno del mondo dei desktop liberi, soprattutto per GNOME, dove è stato deciso che sarebbero stati usati CORBA e le tecnologie KDE. La storia sembra aver messo ogni tecnologia al suo posto, come si evince dalla proposta DBUS (un tipo DCOP avanzato) sul sito FreeDesktop.org; quest'ultimo è un progetto volto a promuovere l'interoperabilità e l'utilizzo di tecnologie congiunte nei desktop liberi, che è guidato casualmente da uno dei più noti *hackers* di GNOME.

La seguente tabella riassume le caratteristiche più importanti del progetto KDE. Il progetto accetta licenze a seconda che siano per un'applicazione o per una biblioteca. Le licenze di biblioteca offrono una maggiore "flessibilità" nei confronti dei terzi: permettono di creare applicazioni proprietarie legate alle librerie.

Al momento attuale, l'ultima versione di KDE è la 3.5.6 mentre la quarta generazione KDE 4, che sarà basata su Qt4, si prevede che arrivi a metà 2007. Il ricambio generazionale comporta un grande impegno per adeguare la versione, che è un compito noioso e porta via molto tempo. Tuttavia, questo non significa che le "vecchie" applicazioni non funzioneranno più. In generale, sono state incluse anche le vecchie versioni delle librerie sulle quali erano basate in modo che le applicazioni continuino a lavorare; ciò significa, tuttavia, che le varie versioni delle librerie devono essere caricate nella memoria in modo simultaneo, con il conseguente spreco di risorse di sistema. Gli sviluppatori di KDE vedono questo effetto come parte integrante dello sviluppo del progetto e, pertanto, come un male minore.

### 9.3.5. Radiografia di KDE

Per quanto riguarda la dimensione di KDE, i dati che ora discuteremo corrispondono allo stato di CVS nell'agosto del 2003, il che significa che devono essere presi con le solite riserve che abbiamo già accennato, più una: alcuni dei moduli che sono stati usati in questa analisi sono ancora in fase di sviluppo e non soddisfano i criteri di un prodotto finito. Questo non dovrebbe avere alcun effetto sulle finalità qui proposte dal momento che il nostro interesse riguarda più la grandezza dei risultati che l'esattezza dei numeri.

Il codice sorgente incluso nel CVS di KDE è la somma totale di sei milioni di linee di codice in diversi linguaggi di programmazione, come vedremo di seguito. Il tempo necessario per creare KDE sarebbe circa nove anni e mezzo, che è più dei sette anni del progetto, e la stima del numero medio di sviluppatori che lavorerebbero a tempo pieno è di duecento. Se si tiene conto del fatto che nel 2003 KDE aveva circa ottocento persone con permesso di scrittura a CVS (di cui la metà sono stati inattivi negli ultimi due anni) e il fatto che gli sviluppatori KDE, con contratti full-time, non sono mai stati più di venti, si può constatare come il livello di produttività di KDE è molto, molto più elevato della stima fornita da COCOMO.

### Nota

Una società che intendeva sviluppare un prodotto di tale portata a partire da zero doveva investire più di 250 milioni di dollari; a fini comparativi, questa somma è come se fosse pari all'investimento che un produttore d'auto desidera effettuare per creare un nuovo impianto di produzione nell'Europa dell'Est o alla cifra che un'affermata società di petrolio intende spendere per aprire duecento distributori di benzina in Spagna.

E' anche interessante vedere che gran parte dello sforzo, quasi la metà di quello speso per lo sviluppo del progetto KDE, corrispondeva alla traduzione dell'interfaccia utente e della documentazione. Anche se pochissime linee di programmazione (circa mille) si occupano di questo compito, il numero dei file dedicato a questo scopo sale a settantacinquemila per le traduzioni (una somma che arriva a centomila se si calcola la documentazione nei diversi formati); tale numero comprende almeno un quarto (terzo) dei 310.000 file che ci sono in CVS. L'attività combinata di CVS è di milleduecento *commits* al giorno, il che significa che il tempo medio tra *commits* è di circa un minuto<sup>10</sup>.

<sup>(10)</sup> Su questo punto, è doveroso fare due osservazioni: la prima è che, quando viene eseguito un *commit* che comprende vari file, è come se ci fosse un *commit* separato per ogni file; la seconda è che, il numero di *commits* è un importo calcolato, in quanto il progetto ha una serie di *scripts* che eseguono *commits* automaticamente.

Per quanto riguarda gli strumenti, i luoghi d'informazione e i casi di assistenza allo sviluppo, vedremo che la gamma delle possibilità offerta da KDE è molto più ampia di quella usata in Linux. A prescindere dal sistema di controllo di versione e dalle mailing list, KDE ha una serie di siti web che forniscono informazioni e documentazioni, tecniche o meno, sul progetto. Fra questi, c'è anche un sito di notizie all'interno del quale vengono presentate nuove soluzioni e si discutono le proposte. Questo sito, tuttavia, non può essere considerato come una sostituzione della mailing list, che, come succede con Linux, è il luogo dove si svolgono i veri dibattiti, dove vengono prese le decisioni ed elaborate le future strategie. Il sito di notifiche assomiglia più ad un luogo d'incontro per utenti. Infine, KDE ha organizzato incontri periodici per tre anni, in cui gli sviluppatori e i collaboratori s'incontrano per circa una settimana per presentare le innovazioni più recenti, per elaborare, discutere, conoscersi e divertirsi (non necessariamente in questo ordine).

**Tabella 8. Analisi KDE**

Website	<a href="http://www.kde.org">http://www.kde.org</a>
Inizio del progetto	1996

Licenza (per applicazioni)	GPL, QPL, MIT, Artistic
Licenza (per librerie)	LGPL, BSD, X11
Versione analizzata	3.1.3
Linee di codice sorgente.	6,100,000
Numero di file (codice, documentazione, etc.)	310.000 file
Stima del costo	\$ 255.000.000
Stima del tempo di esecuzione	9,41 anni (112,98 mesi)
Stima del numero medio di sviluppatori	200,64
Numero approssimativo di sviluppatori	Circa 900 <i>committers</i>
Numero di <i>committers</i> attivi nell'ultimo anno	Circa 450 (approssimativamente il 50% del totale)
Numero di <i>committers</i> attivi negli ultimi due anni	Circa 600 (approssimativamente il 65% del totale)
Numero approssimativo di traduttori (attivi)	Circa 300 traduttori per più di 50 linguaggi (compreso Esperanto).
Numero di <i>commits</i> (da parte degli sviluppatori) nel CVS	Circa 2.000.000 (cifra stimata che non comprende i <i>commits</i> automatici)
Numero di <i>commits</i> (da parte dei traduttori) nel CVS	Circa 1.000.000 (cifra stimata che non comprende i <i>commits</i> automatici)
Numero medio di <i>commits</i> (totale) giornaliero	1.700
Strumenti, documentazione and casi di assistenza allo sviluppo	CVS, mailing lists, website, sito di notizie, meeting annuali

Quando si parla invece di linguaggi di programmazione utilizzati in KDE, predomina C++. Ciò è dovuto principalmente al fatto che questo è il linguaggio madre di Qt, anche se viene spesa molta energia per creare collegamenti che consentano sviluppi in altri linguaggi di programmazione. Certamente, il numero di linee di codice nei linguaggi minori corrisponde quasi totalmente all'attuale progetto per la creazione del link. Questo però non significa che tali linguaggi non vengono usati affatto, come tra l'altro, dimostrano i numerosi progetti esterni a KDE che li utilizzano.

**Tabella 9 Linguaggi di programmazione utilizzati in KDE**

Linguaggio di Programmazione	Linee di codice	Percentuale
C++	5.011.288	82,05%
C	575.237	9,42%
Objective C	144.415	2,36%

Linguaggio di Programmazione	Linee di codice	Percentuale
Shell	103.132	1,69%
Java	87.974	1,44%
Perl	85.869	1,41%

## 9.4. GNOME

Lo scopo principale del progetto GNOME è quello di creare un sistema desktop per l'utente finale che sia completo, libero e facile da usare. Allo stesso modo, l'idea è che GNOME sia una piattaforma molto potente per gli sviluppatori. La sigla GNOME sta per *GNU Network Object Model Environment*. Da quando GNOME cominciò, sono stati proposti vari modi per tradurlo lingua spagnola, ma nessuna di queste proposte è mai riuscita a soddisfare tutti i soggetti coinvolti. Tuttavia, come si evince dal nome, GNOME fa parte del progetto GNU. Attualmente, tutto il codice contenuto in GNOME deve essere posto sotto licenza GNU GPL o GNU LGPL. Si può notare che sono altresì importanti le reti e il modello orientato all'oggetto.

### 9.4.1. Storia di GNOME

Mentre nell'estate del 1997 l'assenza di licenze per KDE era ancora al centro delle discussioni, il destino volle che Miguel de Icaza incontrò Nat Friedman durante alcuni seminari organizzati da Microsoft a Redmond. È probabile che questo incontro rappresentasse una svolta radicale per entrambi, che ebbe come risultato la creazione di GNOME da parte di Miguel de Icaza dopo il suo ritorno in Messico (insieme a Federico Mena Quintero) e la sua ammirazione per le tecnologie di oggetti distribuiti. De Icaza e Mena decisero di creare un ambiente che sarebbe stato un'alternativa a KDE, poiché capirono che una nuova implementazione di una libreria proprietaria sarebbe stato un compito destinato a fallire. E così nacque GNOME.

Dall'antico 1997, GNOME è cresciuto in modo graduale e continua a svilupparsi con le sue ripetute pubblicazioni. La versione 0.99 fu lanciata nel novembre del 1998, ma la prima versione riconosciuta a tutti gli effetti, distribuita in pratica con qualsiasi distribuzione GNU/Linux, era GNOME 1.0, pubblicata nel marzo 1999. Va notato che l'esperienza di questa prima versione *stabile* di GNOME non è stata molto soddisfacente perché molti ritenevano fosse piena di bug non trascurabili. Per questo motivo, GNOME October (GNOME 1.0.55) viene considerata come la prima versione dell'ambiente desktop GNOME che fu realmente stabile. Come si può osservare, con GNOME October, gli sviluppatori non utilizzarono una versione di pubblicazione numerata evitando in questo modo di entrare in una "gara alla versione" contro KDE. Il primo GUADEC, conferenza europea per utenti e sviluppatori GNOME, ebbe luogo a Parigi nel 2000 e poco mancò che coincidesse con la pubblicazione della

nuova versione di GNOME, chiamato GNOME April. È stata l'ultima versione a prendere il nome di un mese, in quanto si scoprì che questo sistema creava più confusione che altro (per esempio, GNOME April è stato lanciato dopo GNOME October, anche se si poteva perdonare chi riteneva il contrario). Nel mese di ottobre dello stesso anno, dopo numerosi dibattiti durati mesi nelle diverse mailing list, venne fondata la fondazione GNOME, della quale parleremo nei prossimi paragrafi.

GNOME 1.2 rappresentò un passo in avanti in termini di architettura usata da GNOME, che comunque continuava ad essere adoperata per GNOME 1.4. Questo periodo fu caratterizzato dalla seconda conferenza GUADEC, che ebbe luogo a Copenhagen. Ciò che nacque come un piccolo incontro di pochi *hackers*, diventò un grande evento che attirava l'attenzione di tutta l'industria del software.

Nel frattempo, la discussione riguardo la libertà di KDE venne risolta quando si finì col dare a Qt una doppia licenza, grazie al cambio di opinione di Trolltech (la doppia licenza era per le applicazioni che funzionano con il software). Oggi, non c'è dubbio che sia GNOME sia KDE siano ambienti desktop liberi; ciò significa che lo sviluppo di GNOME ha incoraggiato la creazione non di un solo ambiente di desktop libero, ma di due.

#### **9.4.2. La Fondazione GNOME**

Il problema più difficile da valutare quando si sente parlare per la prima volta di GNOME è l'organizzazione di oltre mille collaboratori al progetto. È paradossale che un progetto con una struttura tendente verso l'anarchia abbia così successo e raggiunga obiettivi complessi che solo poche multinazionali nel settore IT sarebbero in grado di raggiungere.

Sebbene GNOME venne creato con il chiaro obiettivo di fornire un ambiente user-friendly e potente, al quale sarebbero stati gradualmente aggiunti nuovi programmi, divenne presto evidente che era necessario fondare un organismo che avesse certe responsabilità, che permettesse di promuovere e incrementare l'utilizzo, lo sviluppo e la diffusione di GNOME: di conseguenza, venne fondata la Fondazione GNOME nel 2000, con sede a Boston, USA.

La Fondazione GNOME è un'organizzazione no-profit e non un consorzio industriale, con le seguenti funzioni:

- Coordinare le pubblicazioni.
- Decidere quali progetti fanno parte di GNOME.
- È il portavoce ufficiale (per la stampa e per organizzazioni commerciali e non) del progetto GNOME.

- Promuovere conferenze attinenti a GNOME (come il GUADEC).
- Rappresentare GNOME in altre conferenze.
- Creare standard tecnici.
- Promuovere l'uso e lo sviluppo di GNOME.

Inoltre, la Fondazione GNOME riceve fondi per promuovere e rafforzare le funzioni di cui sopra, dal momento che era impossibile farlo in modo trasparente prima che venisse creata la fondazione.

La Fondazione GNOME ha attualmente un impiegato full-time che si occupa di risolvere tutte le questioni burocratiche e organizzative che vengono normalmente svolte da un'organizzazione no-profit che tiene periodicamente riunioni e conferenze.

Generalmente parlando la Fondazione GNOME si divide in due grandi comitati: un comitato di gestione e un comitato di consulenza.

Il comitato di gestione (il *Consiglio d'Amministrazione*) è formato al massimo da quattordici membri eletti democraticamente dagli aderenti della Fondazione GNOME. Si segue un modello meritocratico e perciò, per diventare membri della GNOME Foundation, è indispensabile aver cooperato in un modo o nell'altro al progetto GNOME. Il contributo non deve necessariamente implicare il codice sorgente; ci sono anche compiti da svolgere che richiedono traduzione, organizzazione, diffusione, ecc, e successivamente, chi vuole può chiedere l'adesione alla Fondazione GNOME in modo da avere il diritto di voto. Pertanto, sono i membri della Fondazione che possono proporsi per il consiglio di amministrazione e sono sempre i membri che, democraticamente, scelgono i loro rappresentanti nel consiglio fra le persone che si sono fatte avanti. Allo stato attuale il voto si svolge tramite posta elettronica. La durata del mandato di membro del consiglio di amministrazione è di un anno, dopo di che si tengono nuove elezioni.

Ci sono alcune regole base per garantire la trasparenza del consiglio di amministrazione. La più importante è la limitazione del numero dei membri affiliati alla stessa società che non può superare i quattro dipendenti. È importante sottolineare che in questo modo i membri del consiglio di amministrazione sono sempre a titolo personale, e mai in rappresentanza di una società. Tuttavia, dopo una lunga discussione, si è convenuto che fosse inserita tale clausola per evitare diffidenze.

L'altra commissione all'interno della Fondazione GNOME è la commissione di consulenza, che non ha potere decisionale, ma serve come mezzo di comunicazione con il comitato di gestione. È formata da società commerciali che operano nel settore del software ma anche da organizzazioni non commercia-

li. I suoi membri al momento includono Red Hat, Novell, Hewlett-Packard, Mandrake, Sun Microsystems, Red Flag Linux, Wipro, Debian e la Fondazione Free Software. Tutte le aziende con più di dieci dipendenti sono tenute a pagare una tassa per entrare a far parte del consiglio dei consulenti.

#### 9.4.3. Il settore che lavora attorno a GNOME

Sostanzialmente GNOME è riuscito a entrare nel settore in modo tale per cui diverse società hanno partecipato molto attivamente al suo sviluppo. Di tutte queste, le più rilevanti sono: Novell Inc., Eazel, i laboratori RHAD da Red Hat e, più recentemente, Sun Microsystems. Ora descriveremo per ogni azienda citata, le motivazioni e i contributi più importanti che hanno portato all'ambiente desktop GNOME:

- Ximian Inc. (originariamente chiamata Helix Inc.) è il nome della società che fu stata fondata nel 1999 da Miguel de Icaza, il co-fondatore di GNOME, e Nat Friedman, uno degli *hackers* GNOME. L'obiettivo principale era quello di riunire i più importanti sviluppatori GNOME sotto lo stesso ombrello per massimizzare lo sviluppo, che è il motivo per cui non ci sorprende che fra i suoi dipendenti presenti e passati si trovino una ventina degli sviluppatori più attivi di GNOME. L'applicazione in cui Ximian ha investito maggiormente fin dall'inizio è stata Evolution, un sistema di gestione completo delle informazioni personali secondo lo stile di Microsoft Outlook, che comprendeva un client di posta elettronica, agenda e una rubrica di contatti. I prodotti che Ximian vendette furono: Desktop Ximian (una versione di GNOME per scopi più aziendali), Red Carpet (un sistema di distribuzione principalmente del software di GNOME ma non solo) e infine, MONO (una nuova implementazione della piattaforma di sviluppo .NET). Quest'ultimo progetto in realtà non è, per ora, legato in alcun modo a GNOME. Ulteriormente, Ximian sviluppò un'applicazione che permette a Evolution di interagire con un server Exchange 2000. Quest'applicazione è stata molto discussa, pur essendo piuttosto piccola, perché è stata pubblicata con licenza non libera (in seguito, nel 2004, gli è stata concessa la licenza come software libero). Nel mese di agosto 2003, Novell comprò Ximian come parte della sua strategia per entrare nel desktop GNU/Linux.
- Eazel è stata fondata nel 1999 da un gruppo di persone che lavoravano per Apple, con l'obiettivo di rendere l'ambiente GNU/Linux facile da usare come quello Macintosh. L'applicazione sulla quale concentrarono i loro sforzi si chiamava Nautilus e doveva essere il file manager che avrebbe definitivamente mandato in pensione il leggendario Midnight Commander, sviluppato da Miguel de Icaza. La mancanza di un modello commerciale e la crisi dei "dotcom", che portò gli investitori di rischio a rimuovere tutto il capitale che era necessario alla società per continuare a lavorare, costrinse Eazel a dichiarare il fallimento il 15 maggio 2001 e alla chiusura definitiva. C'era comunque tempo per rilasciare la versione Nautilus 1.0 prima del



tracollo, anche se la numerazione era piuttosto artificiale, dato che la stabilità che ci si aspettava in una versione 1.0 era incomparabile. Due anni dopo il fallimento di Eazel, si è visto come Nautilus si era sviluppato ed era diventato un completo e maneggevole gestore di file integrato in GNOME. La storia di Eazel e Nautilus può perciò essere considerata come un caso paradigmatico di un programma che sopravvive nonostante la scomparsa della società che l'ha creato: qualcosa che è possibile quasi esclusivamente nel mondo del software libero.

- Red Hat creò la Red Hat Advanced Development Labs, RHAD, con lo scopo di garantire che il desktop GNOME acquisisca facilità d'uso e potenza. A tal fine, Red Hat utilizzò metà dei dodici *hackers* più importanti di GNOME e diede loro la libertà di sviluppare ciò che decidevano fosse più appropriato. Dal Labs RHAD si ha ORBit, l'implementazione di CORBA utilizzata dal progetto GNOME conosciuta come "la più veloce del west". Un altro aspetto importante è il lavoro che è stato svolto sulla nuova versione di GTK+ e sul sistema di configurazione di GNOME, GConf.
- I microsystemi SUN vennero coinvolti nello sviluppo di GNOME in uno stadio avanzato, visto che GNOME era un prodotto relativamente maturo a partire dal settembre 2000. L'intenzione di SUN era di utilizzare GNOME come sistema desktop per il sistema operativo Solaris e perciò costituì un team che lavorasse con GNOME, i cui meriti più conosciuti sono l'usabilità e l'accessibilità di GNOME. Nel giugno 2003, Sun annunciò che avrebbe distribuito GNOME 2.2 con la versione 9 di Solaris.

#### 9.4.4. Lo stato attuale di GNOME

Ad oggi 2007, GNOME è alla versione 2.18. La maggior parte delle tecnologie chiave sulle quali è basato sono maturate, com'è evidente dal numero della versione. Ad esempio, il *broker* CORBA utilizzato oggi è ORBit2, mentre l'ambiente grafico e API, GTK+, subirono modifiche frutto dell'esperienza accumulata durante le precedenti versioni di GNOME. Una novità importante è l'inserimento di una libreria accessibile, proposto da Sun, che permette a persone con problemi di accessibilità di utilizzare l'ambiente GNOME. Un richiamo particolare va fatto per Bonobo, il sistema di componenti GNOME. Bonobo lasciò un'impronta a GNOME per un certo periodo mentre veniva sviluppato Evolution, il programma di gestione delle informazioni personali. Tuttavia, il tempo ha dimostrato che le aspettative suscitate da Bonobo erano troppo alte e che, inoltre, il riutilizzo dello sforzo speso attraverso l'impiego dei suoi componenti non è stato così intenso come ci si aspettava inizialmente.

**Nota**

La biblioteca ATK è una libreria di classi astratte, che rende accessibili le applicazioni. Ciò significa che le persone con qualche forma di disabilità (non vedenti, daltonici, persone con problemi agli occhi che impediscono di usare il mouse, la tastiera, etc) possono ancora utilizzare GNOME. L'interesse di SUN volto a garantire la massima accessibilità è dovuto al fatto che, se vuole offrire i suoi servizi al governo degli Stati Uniti, deve rispondere ad una serie di standard di accessibilità. Il lavoro è stato a tal punto preso sul serio che esiste persino un programmatore non vedente nel team di sviluppo GNOME che lavora a SUN. Nel settembre 2002, l'architettura dell'accessibilità di GNOME ha vinto il premio Helen Keller Achievement Award.

**9.4.5. Radiografia di GNOME**

I dati e i numeri indicate nella tabella 10 ci portano alla fine della nostra presentazione su GNOME. Le cifre corrispondono allo stato di CVS di GNOME al 14 agosto 2003. In quella data, c'erano più di nove milioni di linee di codice ospitate nel repository CVS di proprietà del progetto GNOME. Anche se la cosa più naturale sarebbe quella di confrontare GNOME a KDE, dobbiamo avvertire i lettori che le differenze dal punto di vista dell'organizzazione di questi progetti, rendono sconsigliabile questo paragone, se ciò vuole essere fatto in condizioni di parità. Ciò è dovuto, per esempio, dal fatto che il CVS di GNOME comprende GIMP (un programma per la creazione e la gestione della grafica) che di per sé rappresenta più di 660.000 linee di codice, o la libreria GTK, su cui si concentra lo sviluppo di GNOME, e che da sola ha 330.000 linee. Se aggiungiamo a questo il fatto che il repository CVS di GNOME è più incline ad aprire nuovi moduli per i programmi (che ne ha settecento in totale) rispetto a KDE (che ne ha meno di cento), possiamo capire perché GNOME ha più linee di KDE pur essendo di un anno e mezzo più giovane. Il repository GNOME ospita più di 225.000 file, che sono stati aggiunti e modificati quasi due milioni di volte ( *vedi* il numero di *commits* alcune righe sotto, nella tabella).

**Nota**

Secondo il modello COCOMO usato nel corso di questo capitolo, un'azienda che volesse creare un software della grandezza di quello di GNOME avrebbe dovuto stipulare un contratto di più di undici anni con una media di circa duecentocinquanta sviluppatori per ottenere un prodotto con un'estensione simile. Il costo relativo sarebbe approssimativamente di 400 milioni di dollari, una cifra simile a quella che investì una compagnia di telefonia mobile consolidata nel 2003 per rafforzare la capacità della sua rete o simile alla cifra che un'azienda di produzione di macchine pagherebbe per aprire un impianto di produzione a Barcellona.

Le risorse umane di GNOME contano almeno mille sviluppatori con permesso di scrittura al sistema di revisione e controllo di CVS, dei quali, almeno venti lavorano professionalmente per GNOME (part-time o full-time). Di questi, solo il 25% è stato attivo nell'ultimo anno mentre il 40% è stato attivo negli ultimi due anni. Il numero in media di *commits* giornalieri registrati da quando iniziò il progetto, è quasi un migliaio. Gli strumenti per l'assistenza allo sviluppo usati dal progetto GNOME sono fondamentalmente gli stessi usati da KDE e perciò non saranno presi in considerazione in questo paragrafo.

**Tabella 10. Analisi di GNOME**

Website	<a href="http://www.gnome.org">http://www.gnome.org</a>
Inizio del progetto	Settembre 1997
Licenza	GNU GPL e GNU LGPL

Versione analizzata	2,2
Linee di codice sorgente.	9,200,000
Numero di file (codice, documentazione, etc.)	228.000
Stima di costo	\$ 400.000.000
Stima del tempo di esecuzione	11,08 anni (133,02 mesi)
Stima del numero medio di sviluppatori	Circa 250
Numero di sotto-progetti	Più di 700 moduli nel CVS.
Numero approssimativo di sviluppatori	Almeno 1.000 con accesso di scrittura a CVS.
Numero di <i>committers</i> attivi nell'ultimo anno	Circa 500 (approssimativamente il 55% del totale)
Numero di <i>committers</i> attivi negli ultimi due anni	Circa 700 (75% del totale)
Numero di <i>commits</i> nel CVS	1.900.000
Numero medio di <i>commits</i> (totale) giornaliero	Circa 900
Strumenti di assistenza allo sviluppo	CVS, mailing lists, website, sito di notizie, meeting annuali

Mentre in KDE il linguaggio più usato è senza dubbio C++, in GNOME è C. Questo è dovuto al fatto che, in GNOME come in KDE, la libreria viene scritta in C, il che significa che il linguaggio *nativo* è C, mentre i programmatori che desiderano usare altri linguaggi devono aspettare che appaiano i collegamenti. Il collegamento più avanzato in GNOME è quello incluso in *gnome--*, che non è altro che C++: questo è il motivo per cui non è sorprendente che questo sia il secondo linguaggio in classifica. Perl è sempre stato ampiamente accettato all'interno della comunità di GNOME e un esempio di questo fatto è che in GNOME è possibile usare molti linguaggi per la programmazione. La sua implementazione, tuttavia, non è stata così ampia come ci si sarebbe potuto aspettare ed è leggermente più estensiva di Shell. D'altro canto, Python e Lisp sono stati accettati in modo abbastanza diffuso in GNOME, com'è dimostrato dall'importanza relativa di questa classificazione, mentre Java non è mai veramente decollato, probabilmente a causa di un collegamento incompleto.

**Tabella 11. Linguaggi di programmazione usati in GNOME**

Linguaggio di programmazione	Linee di codice	Percentuale
C	7.918.586	86,10%
C++	576.869	6,27%
Perl	199.448	2,17%
Shell	159.263	1,73%

Linguaggio di programmazione	Linee di codice	Percentuale
Python	137.380	1,49%
Lisp	88.546	0,96%

#### 9.4.6. Studi Accademici su GNOME

Gli studi più importanti di GNOME nella sfera accademica sono due: "Results from software engineering research into open source development projects using public data" [158] e "The evolution of GNOME"[132].

- [158] è uno dei primi studi su larga scala nel settore del software libero. Gli autori della ricerca hanno approfittato del fatto che i dettagli dello sviluppo sono di norma accessibili al pubblico al fine di misurare gli sforzi e confrontarli con i modelli di stima dei costi, il tempo tradizionale e le misurazioni di sforzo. Uno dei modelli classici con i quali li hanno confrontati è stato quello usato in questo capitolo, il modello COCOMO.
- [132] va brevemente oltre gli obiettivi di GNOME e la sua breve storia, così come l'uso del progetto GNOME della tecnologia.

### 9.5. Apache

L'Apache HTTP server è una delle applicazioni star del mondo del software libero, in quanto secondo un'indagine real-time è il web server più implementato ( [http://news.netcraft.com/archives/2003/08/01/august\\_2003\\_web\\_server\\_survey.html](http://news.netcraft.com/archives/2003/08/01/august_2003_web_server_survey.html) ) [167]. Per esempio, nel maggio 1999, il 57% dei web server lavorava con Apache, mentre nel maggio 2003, la percentuale salì al 68%. Apache è disponibile per tutti i tipi di Unix (BSD, GNU/Linux, Solaris ...), Microsoft Windows e altre piattaforme minori.

#### 9.5.1. Storia di Apache

Nel marzo 1989, Tim Berners Lee, uno scienziato inglese che lavorava al CERN (Svizzera) propose un nuovo metodo per gestire l'enorme quantità di informazioni provenienti dai progetti CERN. Il metodo era una rete di documenti con collegamento ipertestuale (ipertesto, come Ted Nelson l'aveva già chiamato nel 1965). Nacque il WWW. Tuttavia, il primo software WWW non venne rivelato prima del novembre 1990: un pacchetto chiamato il World Wide Web comprendeva un browser web con un'interfaccia grafica e un editor WYSIWYG ( "what you see is what you get" ). Due anni più tardi, l'elenco dei server WWW aveva circa trenta voci, tra cui NCSA HTTPd.

La vera storia di Apache è iniziata quando Rob Mc Cool lasciò la NCSA nel marzo 1995. Apache 0.2 sarebbe nato il 18 marzo 1995, basato sul NCSA HTTPd 1.3 server, costruito dallo stesso Rob McCool mentre si trovava a NCSA. Durante quei primi mesi, Apache era una collezione di correzioni applicate al server NCSA, fino a che Robert Thau lanciò Shambhala 0.1: una re-implementazione quasi completa, che comprendeva già l'API per i moduli che in seguito si rivelò avere molto successo.

#### **Nota**

Il nome del progetto Apache è basato sulla sua filosofia di sviluppo e organizzazione. Com'è avvenuto con la tribù Apache, gli sviluppatori di Apache decisero che il loro metodo organizzativo doveva basarsi sui meriti personali degli sviluppatori in confronto al resto della comunità Apache. Tuttavia, vi è una leggenda che si è diffusa che dice che in realtà il nome Apache derivava dal fatto che nelle fasi iniziali era semplicemente un server NCSA corretto o *unpatchy server*.

La prima versione stabile di Apache non apparve fino al gennaio 1996, quando venne rilasciato Apache 1.0, che comprendeva il caricamento di moduli durante l'esecuzione tramite modalità test-mode, così come altre funzioni interessanti. I primi mesi di quell'anno si rivelarono particolarmente produttivi per il progetto, mentre la versione 1.1, che aveva moduli di autenticazione che venivano controllati rispetto alle basi di dati (come ad esempio MySQL), venne pubblicata solo due mesi più tardi. Da allora in poi, gli eventi più importanti per il progetto sono stati: l'introduzione della totale conformità con lo standard HTTP 1.1 (aggiunto in Apache 1.2 nel mese di aprile 1997), l'inserimento della piattaforma Windows NT (che avvenne nel luglio 1997 con le versioni di prova di Apache 1.3), l'unificazione della configurazione dei file in un unico file (che non accadeva dall'ottobre 1998, in Apache 1.3.3) e il lancio, ancora in fase di test, della successiva generazione di Apache, Apache 2.

Nel frattempo, nel giugno 1998, IBM decise che, invece di sviluppare il suo HTTP, avrebbe usato Apache come motore del suo prodotto WebSphere. Questo fu interpretato come un riconoscimento enorme per il progetto Apache da parte del Big Blue e per il software libero in generale, anche se era necessario modificare leggermente la licenza Apache originale per farlo funzionare.

### **9.5.2. Lo sviluppo di Apache**

L'Apache HTTP server è il progetto principale tra i tanti che la Fondazione Apache Software gestisce. Il design modulare di Apache ha reso possibile l'esistenza di una serie di progetti satellite, basati su Apache, alcuni dei quali sono stati persino più grandi di Apache stesso. Per esempio, l'Apache HTTP server contiene il kernel del sistema con le funzionalità base, mentre quelle aggiuntive sono fornite da diversi moduli. I moduli più conosciuti sono *mod\_perl* (un interprete del linguaggio di scrittura Perl, incorporato nel web server) e Jakarta (un potente server di applicazioni). Nei paragrafi seguenti descriveremo solamente il processo di sviluppo che si è seguito per il server http, con moduli simili o no, senza prendere in considerazione gli altri moduli.

Lo sviluppo dell'Apache HTTP server si basa sul lavoro di un piccolo gruppo di sviluppatori chiamato il Gruppo Apache. Il Gruppo Apache è costituito dagli sviluppatori che hanno lavorato insieme al progetto per un lungo periodo, generalmente più di sei mesi. Lo sviluppatore, essendo stato invitato a partecipare da un membro del Gruppo Apache, viene eletto da tutti gli altri membri. All'inizio il Gruppo Apache era composto da otto sviluppatori; questo numero poi salì a dodici e attualmente ci sono venticinque membri.

Il Gruppo Apache è responsabile per lo sviluppo del web server e, quindi, per specifiche decisioni che riguardano lo sviluppo in un certo momento. È importante distinguere il Gruppo Apache dagli sviluppatori nel *core group*, che è sempre attivo. Il carattere volontario del lavoro svolto dalla maggior parte degli sviluppatori rende improbabile che tutte le persone appartenenti al Gruppo Apache siano attive in qualsiasi momento. Ciò significa che il *core group* è definito dalle persone che possono prendersi cura delle operazioni in Apache in un certo periodo di tempo. In generale, le decisioni che devono essere prese dagli sviluppatori appartenenti a questo gruppo centrale sono limitate al voto per l'inclusione o meno di codice, anche se in realtà è riservato solo per le modifiche su larga scala e le domande sul design. D'altro canto, tali sviluppatori hanno di solito accesso in scrittura al repository CVS, il che significa che essi fungono da guardiani per il codice di entrata, assicurando che esso sia corretto e di buona qualità.

### 9.5.3. Radiografia di Apache

I dati riportati di seguito corrispondono alla versione dell'Apache HTTP server, che era disponibile per il download dal server CVS del progetto Apache come dal 18 aprile 2003. Nessuno dei numerosi moduli del progetto Apache è stato preso in considerazione in questa sede. Come vedremo, il progetto Apache è relativamente piccolo rispetto agli altri casi studiati in questo capitolo. Nonostante sia stato già detto, è importante sottolineare la modularità di Apache che ha questo specifico vantaggio: kernel è piccolo e maneggevole. Il repository CVS del progetto Apache, che contiene il kernel del web server e molti moduli aggiuntivi, ospita più di quattro milioni di linee di codice sorgente, una cifra che è leggermente inferiore a quella dei progetti come KDE e GNOME.

La versione 1.3 di Apache aveva appena più di 85.000 linee di codice sorgente. Secondo il modello COCOMO, questo richiedeva il lavoro di una media di venti sviluppatori che lavorano a tempo pieno per un anno e mezzo. Il costo totale del progetto, a quel tempo, era intorno ai quattro milioni di dollari. Al fine di preparare il web server Apache, erano necessari fino a sessanta *commit-ters* diversi, mentre il numero di sviluppatori che fornivano input, secondo i calcoli, sarebbe stato di circa quattrocento.

**Tabella 12. Analisi di Apache**

Website	<a href="http://www.apache.org">http://www.apache.org</a>
---------	---

Inizio el progetto	1995
Licenza	Apache Free Software License
Versione Analizzata	2.2.4
Linee di codice sorgente.	225,065
Numero di file	2.807
Stima di costo	\$ 7.971.958
Stima del tempo di esecuzione	2,52 anni (30,27 mesi)
Stima del numero medio di sviluppatori	23,4
Numero approssimativo di sviluppatori	60 <i>committers</i> (400 sviluppatori)
Strumenti di assistenza allo sviluppo	CVS, mailing lists, sistema di bug report

Apache 1.3 è scritto quasi interamente in linguaggio C e non ci sono molti altri linguaggi di programmazione, soprattutto se considera che la maggior parte di linee scritte nel secondo linguaggio, Shell, corrispondono ai file di configurazione all'assistenza alla compilazione.

**Tabella 13. Linguaggi di programmazione usati in Apache**

Linguaggio di programmazione	Linee di codice	Percentuale
C	208.866	92,8%
Shell	12.796	5,69%
Perl	1.649	0,73%
Awk	874	0,39%

## 9.6. Mozilla

Il progetto Mozilla lavora su una serie di applicazioni integrate per Internet, che sono libere e multiplatforma, e la maggior parte dei prodotti degni di nota sono il browser web Mozilla Firefox e Mozilla Thunderbird e-mail e le news client. Questo insieme è anche concepito come una piattaforma per lo sviluppo di altre applicazioni, il che significa che ci sono molti browser che usano Gecko, il motore HTML di Mozilla (come Galeon).

Il progetto è gestito dalla Fondazione Mozilla, un'organizzazione no-profit che crea il software libero ed è "dedicata a preservare la scelta e a promuovere l'innovazione su Internet". Per questo motivo, i prodotti di Mozilla si basano su tre principi fondamentali: devono essere di software libero, rispettare gli standard ed essere portabili su altre piattaforme.

### 9.6.1. Storia di Mozilla

La storia di Mozilla è lunga e complicata, ma anche molto interessante perché ci permette di apprendere la storia del WWW. La ragione di questo è che se rintracciamo tutte le persone e le istituzioni che sono state coinvolte nello sviluppo di Mozilla, allora si arriva al punto di partenza di Internet, con il lancio del primo browser internet completo.

Come nel caso del predecessore di Apache, fu in NCSA che nacque il primo browser internet completo nel 1993: Mosaico. Molti dei membri del team di sviluppo, con Marc Andreessen e Jim Clark al timone, fondarono una piccola azienda per scrivere partendo da zero (questo perché c'erano problemi con il copyright sul codice di Mosaic e il progetto tecnico del programma aveva i suoi limiti *vedere Speeding the Net: The Inside Story di Netscape and how it challenged Microsoft* [189]), quello che sarebbe poi diventato il browser Netscape Communicator, che era, indiscutibilmente, il leader del mercato dei browser internet fino all'arrivo di Microsoft Internet Explorer. Oltre all'innovazione puramente tecnologica che il browser Netscape rappresentava, Netscape Inc. era innovativa anche nel modo in cui riuscì a monopolizzare il mercato. La sua applicazione principale, il browser WWW, era libero e ciò andava totalmente contro al buon senso di quel tempo (e poteva anche essere distribuito con alcune limitazioni). Questo approccio che era completamente sconosciuto al mondo delle imprese, provocò una certa sorpresa ma si rivelò essere giusto per la strategia di Netscape Inc.; e fu solamente il gigante Microsoft che fu in grado di superarlo con tattiche più aggressive (e probabilmente dannose per la concorrenza del libero mercato).

Intorno al 1997, la quota di mercato di Netscape scese drasticamente a causa della diffusione di Microsoft Explorer. Di conseguenza, Netscape Inc. studiò nuove modalità per recuperare la sua posizione dominante che aveva in precedenza. Una relazione tecnica pubblicata dall'ingegner Frank Hecker ("Setting up shop: the business of open source software", 1998) [142] proponeva che la soluzione migliore al problema era quella di rilasciare il codice sorgente del browser e di trarre beneficio dagli effetti della comunità del free software, come descritto da Eric Raymond in "La Cattedrale e il Bazaar". Nel gennaio 1998, Netscape Inc. annunciò ufficialmente che avrebbe rilasciato il codice sorgente del suo browser, segnando una svolta estremamente importante all'interno della breve storia del software libero: una società stava per pubblicare l'intero codice sorgente sotto una licenza di software libero di un'applicazione che, fino ad allora, era stata un prodotto commerciale. La data di lancio era stata fissata per il 31 marzo 1998.

Nei mesi tra gennaio e marzo, il personale di Netscape era freneticamente attivo cercando di preparare tutto. L'elenco dei compiti era enorme e complicato ("Freeing the source: the story of Mozilla", 1999) [134]. Sul piano tecnico, è stato necessario contattare le aziende che facevano i moduli per chiedere loro il consenso al cambiamento di licenza. Se la risposta era negativa, il modulo



doveva essere eliminato. Inoltre, tutte le parti scritte in Java dovevano essere re-implementate visto che si riteneva che Java non fosse libero. Decisero quindi di chiamare il progetto libero Mozilla, così come gli sviluppatori di Netscape avevano chiamato Netscape il loro elemento principale, e il dominio di Mozilla.org venne acquisito per costruire una comunità di sviluppatori e assistenti basata su questo sito. Alla fine del processo, vennero rilasciate più di un milione e mezzo di linee di codice sorgente.

#### **Nota**

Il nome Mozilla è un gioco di parole caratterizzato da una piccola dose di umorismo da parte del team di sviluppo Netscape Inc. Il nome Mozilla venne creato adattando il nome Godzilla, il mostro che scatenava disordini nei film horror giapponesi negli anni Cinquanta, per dare l'idea di *Mosaic Killer*, visto che il nuovo browser dotato di tecnologie avanzate aveva lo scopo di rendere obsoleto Mosaic.

D'altra parte, c'è la questione legale. Le licenze libere esistenti a quel tempo non convincevano i dirigenti di Netscape che non riuscivano a capire come questo poteva essere "compatibile" con la natura commerciale di un'impresa. Netscape voleva una licenza più *flessibile* che fosse in grado di raggiungere accordi con i terzi così da includere il loro codice a prescindere dalla licenza o dal fatto che altri sviluppatori commerciali dovevano contribuirvi, in modo tale che potevano difendere i loro interessi finanziari in qualunque modo scegliessero. E pur non avendo previsto di creare una nuova licenza inizialmente, alla fine giunsero alla conclusione che questo era l'unico modo per ottenere ciò che volevano. Questo è come venne creata la Licenza Netscape Public (NPL): una licenza basata sui principi fondamentali delle licenze di software libero, ma che, dal punto di vista della Fondazione Free Software, dava anche alcuni diritti supplementari a Netscape Inc. che lo rese una licenza *non libera*. Quando la bozza di NPL venne pubblicata per una discussione pubblica, la clausola che garantiva ulteriori diritti a Netscape, venne fortemente criticata. Netscape Inc. reagì rapidamente in risposta a queste critiche e creò una licenza aggiuntiva, la Mozilla Public License (MPL), che era identica alla NPL tranne per il fatto che Netscape, in quella licenza, non aveva nessun diritto addizionale.

La decisione finale fu di rilasciare il codice di Netscape sotto la licenza NPL, che assicurava diritti aggiuntivi a Netscape, e che ogni nuovo codice che veniva incluso doveva essere rilasciato sotto la licenza MPL (o una licenza compatibile). Le correzioni al codice originale (licenza con la NPL), erano incluse in questa licenza.

#### **Nota**

Al giorno d'oggi Mozilla accetta contributi ai sensi delle proprie licenze, la MPL, GPL e la LGPL. Cambiare licenza non fu affatto facile perché si dovevano trovare tutte le persone che avevano contribuito al codice in qualsiasi punto in modo che davano il loro consenso al passaggio dal NPL/MPL a MPL/GPL/LGPL. Al fine di dare una nuova licenza a tutto il codice, venne creato un sito web che conteneva una lista di trecento "hackers persi", ("Have you seen these hackers?") [38]. Ad oggi maggio 2007, stanno ancora ricercando due di questi sviluppatori.

Sviluppare il codice originale di Netscape Communicator era senza dubbio più complicato di quanto ci si aspettava all'inizio. Le condizioni iniziali erano già pessime perché ciò che venne rilasciato fu a volte incompleto e difficilmente funzionava (tutti i moduli di terzi per i quali non era stato dato il consenso per il rilascio vennero rimossi). Come se non bastasse, oltre al problema tecnico di far funzionare Mozilla su un grande numero di sistemi operativi e piattaforme, c'erano i difetti presi da Netscape Inc., con cicli di rilascio che erano troppo lunghi e inefficienti per Internet e che non si distinguevano tra i propri interessi e la comunità nate intorno Mozilla. Tutto questo s'inasprì esattamente l'anno successivo, quando uno dei programmatori più attivi da prima e dopo il rilascio, Jamie Zawinsky, decise di gettare la spugna con una lettera amara ("Resignation and post-mortem", 1999) [ 237] in cui egli descriveva la sua disperazione e desolazione.

Il 15 luglio 2003, Netscape Inc. (ora di proprietà di America On Line) annunciò che non aveva più intenzione di sviluppare il browser Netscape e, pertanto, non si sarebbe più occupato attivamente del progetto Mozilla. Come una sorta di "accordo di licenziamento" Netscape approvò la creazione della Fondazione Mozilla che sostenne con un contributo di due milioni di dollari. Allo stesso modo, tutto il codice che era sotto la NPL (licenza pubblica Netscape) venne donato alla Fondazione e ridistribuito con le licenze precedentemente pubblicate dal progetto Mozilla: MPL, GPL e LGPL.

Il 10 marzo 2005, la Fondazione Mozilla annunciò che non avrebbe pubblicato le versioni ufficiali dell'applicazione Mozilla Application Suite, che venne sostituita da Mozilla SeaMonkey che comprendeva un web browser, un client di posta elettronica, una rubrica, un editor HTML ed un client IRC. D'altra parte, il progetto Mozilla ospita varie applicazioni indipendenti, fra cui le più importanti sono: Mozilla Firefox (web browser), che è senza dubbio la più conosciuta, Mozilla Thunderbird (e-mail e news client), Mozilla Sunbird (calendario), Mozilla Nvu (editor HTML), Camino (web browser progettato per Mac OS X) e Bugzilla (strumento bug-tracker basato sul web).

Col passare del tempo, nonostante i dubbi e i lunghi periodi in cui sembrava che fosse destinato a fallire, il progetto ora sembra andare meglio. Grazie alla versatilità e alla portabilità delle sue applicazioni, nonostante richiedano molte risorse per il tempo di esecuzione, queste vengono utilizzate in molti casi (in genere, ma in particolare Firefox), come la coppia di OpenOffice.org sul desktop dell'utente finale.

### **9.6.2. Radiografia di Mozilla**

I numeri che discuteremo in questo paragrafo appartengono ad uno studio di Firefox, l'applicazione più conosciuta del progetto. Secondo le stime del modello COCOMO, un'azienda che desidera creare un software di questa gran-

dezza dovrebbe investire all'incirca 111 milioni di dollari. Il tempo che richiederebbe sarebbe pari a sette anni e il numero medio di programmatori full-time che l'azienda utilizzerebbe, sarebbe circa 120.

**Tavola 14. Stato attuale di Mozilla Firefox**

Website	<a href="http://www.mozilla-europe.org/es/products/firefox/">www.mozilla-europe.org/es/products/firefox/</a>
Inizio del progetto	2002
Licenza	MPL/LGPL/GPL
Versione	2.0
Linee di codice sorgente.	2.768.223
Stima di costo	\$ 111.161.078
Stima di tempo d'esecuzione	6,87 anni (82,39 mesi)
Stima del numero medio di sviluppatori	120
Numero approssimativo di sviluppatori	50 <i>committers</i>
Strumenti di assistenza allo sviluppo	CVS, mailing lists, IRC, Bugzilla.

C++ e C sono i linguaggi più usati in ordine di priorità. L'uso di Perl ha dovuto principalmente al fatto che gli strumenti di assistenza allo sviluppo creati dal progetto Mozilla, come per esempio BugZilla e Tinderbox, sono progettati per questo linguaggio. La cosa più sorprendente è l'ammontare di linee di codice in un linguaggio assembly nell'applicazione dell'utente finale. Un'ispezione del codice nel repository mostra che in effetti esiste una quantità piuttosto grande di file codificati nel linguaggio assembly.

**Tavola 15. Linguaggi di programmazione usati in Mozilla Firefox**

Linguaggio di programmazione	Linee di codice	Percentuale
C++	1.777.764	64,22%
C	896.551	32,39%
Assembler	34.831	1,26%
Perl	26.768	0,97%
Shell	16.278	0,59%
C#	6.232	0,23%
Java	5.352	0,19%
Python	3.077	0,11%
Pascal	459	0,02%

## 9.7. OpenOffice.org

Openoffice.org è una delle applicazioni star nell'attuale scenario di software libero. Si tratta di un'applicazione multiplatforma per software di produttività personale che comprende le applicazioni chiave in un ambiente di office desktop, come ad esempio: un processore word (Writer), un foglio di calcolo (Calc), un programma di presentazioni (Impress), un editor di grafica (Draw), un strumento per la creazione e la modifica di formule matematiche (Math) e, infine, un editor del linguaggio HTML (allegato in Writer). L'interfaccia fornita da OpenOffice.org è omogenea e intuitiva, con un aspetto e delle funzionalità simili a quelle di altre applicazioni office, in particolare quella che è più diffusa oggi: Microsoft Office.

Scritto in C++, OpenOffice.org include l'API di Java e ha i propri componenti per sistemi dedicati, che permette di unire ad esempio le tabelle provenienti da un foglio di calcolo nel processore word in modo molto semplice e intuitivo. Uno dei suoi vantaggi è che si possono gestire una grande quantità di formati di file, compresi quelli di Microsoft Office. I suoi formati di file nativi, diversamente da quelli della suite da ufficio di Microsoft, sono basati su XML; il che dimostra che sono chiaramente orientati alla versatilità, alla facilità di trasformazione e alla trasparenza. OpenOffice.org è stato attualmente tradotto in più di venticinque lingue e gira su Solaris (il suo sistema operativo nativo), GNU/Linux e Windows. Le versioni per FreeBSD, IRIX e Mac OS X sono attese in un futuro non troppo lontano.

OpenOffice.org ha preso il suo nome definitivo (OpenOffice, come tutti sanno che, più la tag *.org*), dopo un caso giudiziario in cui è stato accusato di violazione di un marchio registrato da un'altra società.

### 9.7.1. Storia di OpenOffice.org

A metà del 1980, venne fondata l'azienda StarDivision nella Repubblica federale tedesca con l'obiettivo principale di creare un'applicazione di produttività personale: StarOffice. Nell'estate del 1999, SUN Microsystems decise di acquistare la società StarDivision e dare un contributo significativo a StarOffice, con il chiaro intento di strappare una parte della quota di mercato che Microsoft deteneva in quel momento. Nel giugno 2000, l'azienda lanciò la versione 5.2 di StarOffice, che poteva essere scaricata liberamente da Internet.

Tuttavia, il successo StarOffice era limitato perché il mercato era già fortemente dominato dal pacchetto di Microsoft Office. SUN decise di cambiare strategia e, com'è avvenuto con Netscape e il progetto Mozilla, volle approfittare del software libero per acquisire importanza e implementare i suoi sistemi. Le future versioni di StarOffice (un prodotto proprietario di SUN) venivano di con-

sequenza create con OpenOffice.org (un prodotto libero) come una sorgente, rispettando le interfacce di programmazione applicativa (API) e dei formati dei file e servendo come l'implementazione standard.

### 9.7.2. Organizzazione di OpenOffice.org

OpenOffice.org mira ad avere una struttura decisionale in cui tutti i membri della comunità si sentano partecipi. Venne perciò elaborato un sistema di modo che il processo decisionale aveva il massimo consenso possibile. Il progetto OpenOffice.org è suddiviso in una serie di sottoprogetti che vengono presi dai membri del progetto, gli assistenti e da un leader unico. Naturalmente, i membri di un progetto possono lavorare su più progetti, così come il leader. Tuttavia, nessuno può condurre più di un progetto alla volta. I progetti sono suddivisi in tre categorie:

- Progetti accettati. Questi possono essere tecnici o non tecnici. I leader di ogni progetto hanno diritto a un voto quando si tratta di prendere decisioni a livello globale.
- Progettative-*lang*. Questi sono tutti i progetti di internazionalizzazione e localizzazione di OpenOffice.org. Come abbiamo accennato, attualmente ci sono più di venticinque team che stanno lavorando per tradurre le applicazioni OpenOffice.org in diverse lingue e convenzioni. Come un insieme, i progetti *native-lang* hanno un singolo voto sulle decisioni a livello globale.
- Progetti in incubazione. Questi sono i progetti promossi dalla comunità (in genere sono sperimentali o piccoli). Possono diventare progetti accettati dopo un periodo di sei mesi. In, la comunità OpenOffice.org può garantire che i progetti accettati sono basati su un reale interesse, visto che il *tasso di mortalità* di nuovi progetti nel mondo del free software è molto alto. In totale, i progetti in incubazione hanno un voto sulle decisioni prese.

### 9.7.3. Radiografia di OpenOffice.org

L'applicazione di produttività personale di OpenOffice.org comprende circa quattro milioni di linee di codice sorgente distribuite in quarantacinquemila di file.

Il modello COCOMO stima che il lavoro richiesto per costruire un "clone" di OpenOffice.org dovrebbe essere eseguito da centottanta programmatori a tempo pieno per quasi otto anni. Secondo le stime COCOMO, il costo di sviluppo sarebbe di circa 215 milioni di dollari.

I risultati discussi in questa sezione sono stati ottenuti da uno studio del codice sorgente della versione stabile 2.1 di OpenOffice.org.

**Tabella 16. Stato attuale di OpenOffice.org**

Website	<a href="http://www.openoffice.org">http://www.openoffice.org</a>
Inizio del progetto	giugno 2000 (prime versioni libere)
Licenza	LGPL e SISSL
Versione	2.1
linee di codice sorgente.	5.197.090
Stima di costo	\$ 215.372.314
stima del tempo di esecuzione:	8,83 anni (105,93 mesi)
Stima del numero medio di sviluppatori	180
Numero approssimativo di sviluppatori	200 <i>committers</i>
Strumenti di assistenza per lo sviluppo	CVS, mailing lists

Per quanto riguarda i linguaggi di programmazione utilizzati in OpenOffice.org, il più diffuso è C++. È interessante notare come l'acquisto di SUN portò all'integrazione di gran parte del codice Java nell'applicazione di produttività personale, che addirittura superava l'ammontare del linguaggio in C.

**Tabella 17. Linguaggi di programmazione utilizzati in OpenOffice.org**

Linguaggio di programmazione	linee di codice	Percentuale
C + +	4.615.623	88,81%
Java	385.075	7,41%
C	105.691	2,03%
Perl	54.063	1,04%
Shell	12.732	0,24%
Yacc	6.828	0,13%
C #	6.594	0,13%

## 9.8. Red Hat Linux

Red Hat Linux è stata una delle prime distribuzioni commerciali di GNU/Linux. Oggi è probabilmente una delle più note ed è quella che può essere indubbiamente considerata la più "canonica" di tutte le distribuzioni commerciali. Il lavoro dei distributori è fondamentalmente legato alle attività d'integrazione

e non tanto allo sviluppo del software. Naturalmente Red Hat e altre distribuzioni possono avere sviluppatori che lavorano per loro, ma il loro lavoro è secondario alle finalità di una distribuzione. In generale, si suppone che il compito delle distribuzioni sia semplicemente di prendere i pacchetti sorgente (solitamente i file pubblicati dagli sviluppatori stessi) e di unirli in modo che rispondano a determinati criteri (sia tecnici che organizzativi). Il prodotto di questo processo è una distribuzione: una serie di pacchetti opportunamente organizzati che fan sì che l'utente possa installarli, disinstallarli e aggiornarli.

Le distribuzioni sono anche responsabili della qualità del prodotto finale, che è un aspetto molto importante se si considera che molte delle applicazioni incluse sono state sviluppate da volontari nel loro tempo libero. Gli aspetti della sicurezza e della stabilità sono perciò essenziali per una distribuzione.

### 9.8.1. Storia di Red Hat

Red Hat Software Inc. è stata fondata da Bob Young e Marc Ewing nel 1994. L'obiettivo principale era quello di compilare e commercializzare una distribuzione GNU/Linux che si chiamava (e si chiama ancora), Red Hat Linux [236]. Si trattava praticamente di una versione che raggruppava ciò che esisteva in Internet in quel periodo, compresa la documentazione e il supporto. La versione 1.0 di questa distribuzione nacque nell'estate del 1995. Pochi mesi dopo, in autunno, venne pubblicata la versione 2.0, che usava la tecnologia RPM (*gestore di pacchetti RPM*). Il gestore di pacchetti RPM divenne uno standard *di fatto* per il pacchetto nei sistemi GNU/Linux. Nel 1998, la versione 5.2 di Red Hat venne rilasciata al grande pubblico. Per una storia completa dei nomi delle varie versioni di Red Hat, si prega di leggere "The truth behind Red Hat names" [201].

#### Nota

A partire dalla versione 1.1 di Linux Standard Base (una specifica progettata per ottenere la compatibilità binaria tra le distribuzioni GNU/Linux, di cui si prende cura il Gruppo Free Standards), venne scelto RPM come il gestore dei pacchetti standard. Il progetto Debian continua con il proprio formato di pacchetti, così come fanno molte distribuzioni che dipendono dal sistema di gestione dei pacchetti Debian, e vengono adeguati al formato standardizzato utilizzando uno strumento di conversione chiamato *alien*.

Prima che esistesse il sistema di gestione RPM, quasi tutte le distribuzioni GNU/Linux offrivano la possibilità di installare il software tramite una procedura basata su un menu. Tuttavia non era facile apportare modifiche a un'installazione esistente, in particolare aggiungendo nuovi pacchetti software dopo l'installazione. RPM fece un passo al di là di ciò che era effettivamente possibile dando agli utenti la possibilità di gestire i propri pacchetti ("Maximum RPM. Taking the Red Hat package manager to the limit", 1998) [83]: ciò rese possibile più semplice cancellare, installare o aggiornare qualsiasi pacchetto software esistente nella distribuzione. Il sistema dei pacchetti RPM continua ad essere il più diffuso sistema di gestione dei pacchetti nelle varie distribuzioni GNU/Linux. Le statistiche delle distribuzioni Linux ("Facts and figures" 2003 [92] un sito web contenente informazioni qualitative e quantitativi

ve sulla maggioranza delle distribuzioni), mostrano che nel maggio 2003 una larga maggioranza (sessantacinque) delle centodiciotto distribuzioni utilizzate per i calcoli, utilizzava RPM (circa il 55% del totale). In confronto, il formato dei pacchetti Debian (noto come *deb*) veniva utilizzato solo in sedici distribuzioni (circa il 14% del totale).

Tuttavia, Red Hat Inc. non era solamente conosciuta per la sua distribuzione software basata su Linux. Nell'agosto 1999, Red Hat divenne pubblico e le sue azioni raggiunsero l'ottavo guadagno più alto di tutta la storia di Wall Street solamente il primo giorno. Quattro anni più tardi, il valore delle azioni Red Hat si era ridotto ad un centesimo del valore massimo che ha raggiunto prima della crisi delle dotcom. Tuttavia, i suoi inizi di successo sul mercato azionario portarono Red Hat sulle prime pagine di giornali e di riviste che non erano specializzate direttamente in IT. In ogni caso, sembra che Red Hat sia riuscito a superare i problemi che altre società nel mondo del lavoro hanno avuto con il software libero e i numeri pubblicati nel corso dell'ultimo trimestre del 2002 erano in attivo per la prima volta in tutta la sua storia.

Un altro degli eventi storici più importanti che riguardano Red Hat è stata l'acquisizione di Cygnus Solutions nel novembre 1999, una società fondata dieci anni prima, che aveva già dimostrato com'era possibile guadagnare con una strategia integrata basata su software libero ("Future of Cygnus Solutions. An entrepreneur's account") [216]. Cygnus scelse il difficile mercato dei compilatori per lasciare il segno. La sua strategia commerciale era basata sullo sviluppo e sull'adattamento degli strumenti di sviluppo software GNU (sostanzialmente GCC e GDB) sulle esigenze del cliente.

Nel settembre 2003, Red Hat decise di concentrare il suo lavoro di sviluppo sulla versione aziendale della distribuzione e delegò la versione comune a Fedora Core, un progetto indipendente a codice aperto di Red Hat.

Nel giugno 2006, Red Hat acquisì la società di JBoss, Inc., diventando l'azienda incaricata di sviluppare il server più importante delle applicazioni open source: J2EE.

### **9.8.2. Stato attuale di Red Hat.**

I prodotti più importanti di Red Hat Inc. al giorno d'oggi sono Fedora Core e Red Hat Network, un servizio di aggiornamento del software Internet. Questi tipi di servizi sono pensati per l'utente finale e non tanto per l'ambiente aziendale, ma sono utili a Red Hat per farsi pubblicità e rafforzare la sua strategia di marca.

La "reale" strategia commerciale di Red Hat è basata sui prodotti che realizza per il mondo aziendale. Questi tipi di prodotti sono molto meno noti ma costituiscono una parte rilevante del fatturato di Red Hat, di gran lunga maggiore di quello dei suoi prodotti star più famosi in senso letterale.



Red Hat ha una distribuzione che è orientata al mondo aziendale, integrata intorno a un server di applicazioni chiamato Red Hat Enterprise Linux AS. I clienti che acquistano questo software ricevono anche un supporto. L'equivalente di Red Hat Network per gli utenti commerciali è Red Hat Enterprise Network, che comprende la gestione del sistema e la possibilità di ottenere aggiornamenti. D'altro canto, Red Hat offre anche servizi di consulenza IT e un programma di certificazione analogo a quello offerto da Microsoft nel mondo di Windows.

### 9.8.3. Radiografia di Red Hat

Red Hat ha recentemente superato il traguardo dei cinquanta milioni di linee di codice, il che lo rende una delle più grandi distribuzioni software che sia mai esistita, superando, come vedremo in questo capitolo, la dimensione dei sistemi operativi proprietari. La versione 8.1 Red Hat era composta da 792 pacchetti e perciò si può ipotizzare che la versione più recente avrà più di ottocento pacchetti, se si considera che il numero tende ad aumentare leggermente da versione a versione.

Come negli casi precedenti, è stato utilizzato il modello COCOMO per stimare l'investimento e lo sforzo che sarebbe stato necessario al fine di creare una generazione di software della stessa grandezza. Tuttavia, nel caso di Red Hat, abbiamo preso in considerazione il fatto che si tratta di un prodotto preparato con una serie di applicazioni indipendenti. Di conseguenza, è stata usata una stima indipendente COCOMO per ognuno dei pacchetti di Red Hat e poi abbiamo aggiunto la stima dei costi e del personale che sarebbero stati necessari. Al fine di analizzare il tempo ottimale di progettazione per Red Hat, abbiamo scelto il pacchetto più grande in quanto, in teoria, tutti i pacchetti sono indipendenti e perciò potrebbero essere progettati contemporaneamente. Per questo motivo, il tempo ottimale di progettazione per Red Hat è simile a quello che degli altri progetti presentati nei precedenti paragrafi di questo capitolo.

Secondo COCOMO, al fine di progettare la versione Red Hat Linux 8.1 partendo da zero, sarebbero necessari circa sette anni e mezzo e un team di sviluppatori costituito da una media di milleottocento sviluppatori. Il costo dello sviluppo finale sarebbe di circa 1.800 milioni di dollari.

#### Nota

Milleottocentomilioni di dollari è la somma che il ministero della Difesa spagnolo ha stanziato per rinnovare la sua flotta di elicotteri nell'ultimo bilancio. Di tale somma, metà saranno investiti per l'acquisto di ventiquattro elicotteri. Quindi si può dire che il prezzo di Red Hat sarebbe equivalente a quello di quarantotto elicotteri da combattimento. Allo stesso livello, milleottocento milioni di dollari è anche il guadagno totale a livello mondiale del film *Titanic*.

#### Tabella 18. Stato di Red Hat Linux.

Website	<a href="http://www.redhat.com">http://www.redhat.com</a>
---------	---

Inizio del progetto	1993
Licenza	
Versione	9.0
Linee di codice sorgente.	Più di 50,000,000
Numero di pacchetti	792
Stima di costo	\$ 1.800.000.000
Stima del tempo di esecuzione	7,35 anni (88,25 mesi)
Stima del numero medio di sviluppatori	1.800
Numero approssimativo di sviluppatori	Impiegati Red Hat (generalmente solo integrazione)
Strumenti per l'assistenza allo sviluppo	CVS, mailing lists

A causa dei molti pacchetti, i linguaggi di Red Hat sono più diversificati rispetto a quelli che abbiamo visto nelle applicazioni più importanti del software libero. In generale, C è il più rilevante con oltre il sessanta per cento delle linee di codice. Al secondo posto, con più di dieci milioni di linee di codice, abbiamo C++, seguito con un grande distacco da Shell. È interessante notare che dopo Perl, appare Lisp (principalmente a causa del suo utilizzo in Emacs), il linguaggio assembly (di cui un quarto corrisponde al linguaggio in dotazione con Linux) e un linguaggio il cui uso è decisamente in declino: Fortran.

**Tabella 19. Linguaggi di programmazione usati in Red Hat.**

Linguaggio di programmazione	Linee di codice	Percentuale
C	30,993,778	62,13%
C++	10.216.270	20,48%
Shell	3.251.493	6,52%
Perl	1.106.082	2,22%
Lisp	958.037	1,92%
Assembler	641.350	1,29%
Fortran	532.629	1,07%

## 9.9. Debian GNU/Linux

Debian è un sistema operativo libero che attualmente utilizza il kernel di Linux per la sua distribuzione (anche se si prevede che in futuro ci saranno distribuzioni Debian basate su altri kernel, come è il caso di "the HURD"). È attualmente disponibile per diverse architetture differenti, tra cui Intel x86, ARM, Motorola, 680x0, PowerPC, Alpha e SPARC.

Debian non è solo la più grande distribuzione GNU/Linux esistente, ma anche una delle più stabili e ha ricevuto diversi premi talmente è preferita dagli utenti. Anche se la sua base di utenti è difficile da stimare, in quanto il progetto Debian non vende cd o altro supporto con il suo software e il software può ancora essere ridistribuito da chiunque voglia, possiamo supporre con un ragionevole grado di certezza che si tratta di una rilevante distribuzione all'interno del mercato GNU/Linux.

C'è una categorizzazione in Debian che dipende dalle condizioni di licenza e distribuzione dei pacchetti. Il kernel della distribuzione Debian (la sezione chiamata *main* che comprende una gran varietà di pacchetti) consiste solo in software libero in conformità con le DFSG (Debian Free Software Guidelines) [104]. Può essere scaricato da Internet e molti ridistributori vendono su CD o su altri supporti.

Le distribuzioni Debian sono create da quasi mille volontari (in genere professionisti IT ed esperti). Il lavoro di questi volontari consiste nel prendere i programmi sorgente, nella maggior parte dei casi dagli autori originali, configurandoli, compilandoli e raggruppandoli in modo che l'utente medio di distribuzione Debian debba solo selezionare il pacchetto e infine il sistema lo installerà senza alcun problema. Quello che all'inizio può sembrare semplice può diventare complesso non appena vengono presi in considerazione altri fattori, come ad esempio le dipendenze tra i diversi pacchetti (il pacchetto A ha bisogno del pacchetto B per lavorare) e le diverse versioni di tutti questi pacchetti.

Il lavoro svolto dai membri del progetto Debian è lo stesso di quello eseguito per qualsiasi altra distribuzione, ovvero, l'integrazione del software in modo che funzioni correttamente. A prescindere dal lavoro di adattamento e raggruppamento, gli sviluppatori Debian sono incaricati di mantenere un'infrastruttura di servizi basata su Internet (sito web, file online, sistema di gestione dei bug, le mailing list di assistenza, il supporto e lo sviluppo, ecc), vari progetti di traduzione e internazionalizzazione, lo sviluppo di vari strumenti specifici per Debian e, in generale, sono responsabili di tutto ciò che è richiesto per far funzionare la distribuzione Debian.

Oltre alla sua natura volontaria, il progetto Debian ha una caratteristica unica: il contratto sociale ([http://www.debian.org/social\\_contract.html](http://www.debian.org/social_contract.html)) [106]. Questo documento non solo descrive gli scopi principali del progetto Debian ma anche i mezzi impiegati per raggiungerli.

Debian anche è noto per avere una procedura molto rigida dei pacchetti e delle versioni, elaborata per ottenere la miglior qualità del prodotto ("Debian policy manual") [105]. In questo modo, esistono contemporaneamente tre diversi *tipi* di Debian: una versione stabile, una instabile e una di prova. Come indica il nome stesso, la versione stabile è quella consigliata per sistemi e utenti che necessitano una stabilità completa. Il software deve essere sottoposto ad un periodo di "congelamento", dove vengono corretti tutti i bug. La regola è che la versione stabile di Debian non deve avere nessun bug cruciale conosciuto. D'altra parte, questa versione stabile non ha solitamente le ultime versioni del software (le aggiunte più recenti).

Ci sono altre due versioni di Debian che si affiancano a quella stabile per coloro che desiderano avere il software più recente. La versione instabile include pacchetti che sono in corso di stabilizzazione, mentre la versione di prova, come indica il nome stesso, ha una maggiore tendenza a fallire ma che contiene le novità più recenti in termini di ultimi software.

Quando è stato fatto il primo studio, la versione stabile di Debian era Debian 3.0 (noto anche come Woody), quella instabile era soprannominata Sid e la versione di prova Sarge. Tuttavia, anche Woody attraversò una fase instabile e, prima ancora, una di prova. Questo è importante, perché ciò che analizzeremo in questo paragrafo, comprenderà le diverse versioni stabili di Debian da quando la versione 2.0 venne pubblicata nel 1998. Per esempio abbiamo Debian 2.0 (alias Hamm), Debian 2.1 (Slink), Debian 2.2 (Potato) e, infine, Debian 3.0 (Woody).

#### **Nota**

I soprannomi delle versioni Debian corrispondono ai personaggi principali nel film d'animazione *Toy Story*; una tradizione che iniziò, un po' seriamente e un po' per scherzo, quando la versione 2.0 venne pubblicata e Bruce Perens, l'allora leader del progetto e fondatore successivamente dell'iniziativa Open Source e della frase *open source*, stava lavorando per l'azienda che creò il film. Per maggiori dettagli riguardante la storia di Debian e la sua distribuzione in generale, si raccomanda "A brief history of Debian"[122].

### **9.9.1. Radiografia di Debian**

Debian GNU/Linux è probabilmente la più grande raccolta di software libero che funziona in modo coordinato e, senza dubbio, uno dei più grandi prodotti software mai costruiti. La versione 4.0, rilasciata nel mese di aprile 2007 (chiamata Etch), è costituita da più di diecimila pacchetti sorgente, con oltre 288 milioni di linee di codice.

Il numero di linee di codice nella versione 3.0 è di 105 milioni. Secondo il modello COCOMO, dovrebbe essere pagata una somma di circa 3.600 milioni di dollari per ottenere un software simile a quello impacchettato con questa distribuzione. Come con Red Hat, venne calcolato lo sforzo necessario alla creazione separata di ogni pacchetto e le cifre risultanti sono state poi aggiunte le une alle altre. Per la stessa ragione, il tempo che ci sarebbe voluto per sviluppare Debian è di solo sette anni, poiché i pacchetti potevano essere creati tutti allo stesso tempo come gli altri. Tuttavia, sarebbero dovuti essere mobilitati una media di circa quattromila sviluppatori in questi sette anni.

**Note**

Tremilaseicentomilioni di dollari è il budget stanziato dal sesto EC Framework Programme per la ricerca e lo sviluppo sulla società informatica. È uguale anche alla somma che Telefonica vuole investire in Germania per attivare l'UMTS.

**Table 20. Stato di Debian**

Website	<a href="http://www.debian.org">http://www.debian.org</a>
Inizio del progetto	16/08/1993
Licenza	Quelle che rispettano il DFSG
Versione usata	Debian 4.0 (alias Etch)
Linee di codice sorgente.	288.500.000
Numero di pacchetti	10.106
Stima di costo	\$ 10.140 million
Stima del tempo di esecuzione	8,84 anni
Numero approssimativo di maintainers	Circa 1.500
Strumenti di assistenza allo sviluppo	Mailing lists, sistema di bug report

Il linguaggio già comunemente usato in Debian 4.0 è C, con oltre il 51% delle linee di codice. Tuttavia, come vedremo più avanti, l'importanza di C è in declino col passare del tempo, visto che l'80% del codice nella prime versioni di Debian, era in C. Il secondo linguaggio maggiormente usato, C++, si prende una buona parte della "colpa" per il declino di C. Tuttavia C è stato in particolare colpito da *linguaggi di scripting*, come Perl, Python e PHP. Tra questi, riescono a volte ad entrare linguaggi come Lisp e Java (che è scarsamente rappresentato in Debian a causa della sua politica di non accettare codice che dipende da una macchina virtuale privata di Sun).

**Table 21. Linguaggi di programmazione usati in Debian GNU/Linux 4.0**

Linguaggio di programmazione	Linee di codice (in milioni)	Percentuale
C	155	51%
C++	55	19%
Shell	30	10%
Perl	8.1	2,9%

Linguaggio di programmazione	Linee di codice (in milioni)	Percentuale
Lisp	7.7	2,7%
Python	7.2	2,5%
Java	6.9	2,4%
PHP	3.5	1,24%

La tabella 22 mostra come vengono sviluppati i linguaggi più importanti in Debian.

**Tabella 22. Linguaggi più usati in Debian**

Linguaggio	Debian 2.0		Debian 2.1		Debian 2.2		Debian 3.0	
C	19.400.000	26,67%	27.800.000	24,89%	40.900.000	29,12%	66.500.000	63,08%
C++	1.600.000	6,16%	2.800.000	7,57%	5.980.000	10,11%	13.000.000	12,39%
Shell	645.000	2,55%	1.150.000	3,10%	2.710.000	4,59%	8.635.000	8,19%
Lisp	1.425.000	5,64%	1.890.000	5,10%	3.200.000	5,41%	4.090.000	3,87%
Perl	425.000	1,68%	774.000	2,09%	1.395.000	2,36%	3.199.000	3,03%
Fortran	494.000	1,96%	735.000	1,98%	1.182.000	1,99%	1.939.000	1,84%
Python	122.000	0,48%	211.000	0,57%	349.000	0,59%	1.459.000	1,38%
Tcl	311.000	1,23%	458.000	1,24%	557.000	0,94%	1.081.000	1,02%

Ci sono linguaggi che si potrebbero considerare in minoranza ma che raggiungono posizioni piuttosto alte nella classifica. Questo è dovuto al fatto che tali pacchetti sono piuttosto grandi, sebbene siano presenti in un basso numero di pacchetti. Questo è il caso di Ada, che nonostante sia in solo tre pacchetti (GNAT, un compilatore ADA; libgkda, un collegamento alla libreria GTK; ASIS, un sistema di gestione delle risorse di Ada), comprende 430.000 linee di codice sorgente su un totale di 576.000 che sono state contate in Debian 3.0 per Ada. Un altro caso simile è Lisp, che appare solamente in GNU Emacs e Xemacs, ma che ha più di 1.200.000 linee su un totale di circa quattro milioni nell'intera distribuzione.

### 9.9.2. Confronto con altri sistemi operativi

C'è un proverbio che dice che tutti i confronti sono odiosi; questo è particolarmente vero quando si confronta il software libero con quello proprietario. Le radiografie dettagliate di Red Hat Linux e Debian furono possibili perché c'erano esempi di software libero. Avere accesso al codice (e ad altre informazioni che sono state fornite in questo capitolo) è essenziale per studiare i numeri di linee, i pacchetti, i linguaggi di programmazione etc. delle diverse ver-

sioni. Ma i vantaggi dei software liberi vanno oltre perché rendono semplice la loro analisi ai terzi, che siano team di ricerca o semplicemente persone interessate.

Uno studio come questo nei sistemi proprietari in generale, sarebbe completamente impossibile. Infatti, i dati forniti qui sotto furono ottenuti dalle stesse aziende che stanno dietro allo sviluppo del software proprietario, il che significa che non siamo in posizione di garantire la loro veridicità. Per finire, in molti casi non conosciamo se si sta parlando di linee di codice sorgente fisiche, come abbiamo visto nel corso di questo capitolo o se comprendono anche le linee vuote e i commenti. Inoltre, non si sa per certo nemmeno cosa queste aziende comprendono nel loro software e perciò non si sa nemmeno se certe versioni di Microsoft Windows inseriscono Microsoft Office suite o meno.

In ogni caso, considerando tutto ciò che si è detto su questo argomento nei paragrafi precedenti, crediamo che sia interessante includere questo confronto dato che aiuta a vedere in un contesto più ampio la posizione in cui le diverse distribuzioni Red Hat e Debian si trovano. È fuori discussione che sia Debian e Red Hat, ma soprattutto il primo, siano le più grandi collezioni di software che l'umanità abbia mai visto fino ad oggi.

I numeri riportati qui sotto provengono da Mark Lucovsky [168] per Windows 2000, dai Microsistemi SUN [171] per Star Office 5.2, da Gary McGraw [169] per Windows XP e da Bruce Schneier [200] per tutti gli altri sistemi. La tabella 23 fornisce un confronto dal più piccolo al più grande.

**Tabella 23. Confronto con i sistemi proprietari**

Sistema	Data di pubblicazione	Linee di codice (approssimative)
Microsoft Windows 3.1	Aprile 1992	3.000.000
SUN Solaris 7	Ottobre 1998	7.500.000
SUN StarOffice 5.2	Giugno 2000	7.600.000
Microsoft Windows 95	Agosto 1995	15.000.000
Red Hat Linux 6.2	Marzo 2000	18.000.000
Debian 2.0	Luglio 1998	25.000.000
Microsoft Windows 2000	Febbraio 2000	29.000.000
Red Hat Linux 7.1	Aprile 2001	32.000.000
Debian 2.1	Marzo 1999	37.000.000
Windows NT 4.0	Luglio 1996	40.000.000
Red Hat Linux 8.0	Settembre 2002	50.000.000

Sistema	Data di pubblicazione	Linee di codice (approssimative)
Debian 2.2	Agosto 2000	55.000.000
Debian 3.0	Luglio 2002	105.000.000

## 9.10. Eclipse

La piattaforma Eclipse consiste in un IDE aperto ed estendibile (*Integrated Development Environment*). Un IDE è un programma che consiste in un insieme di strumenti che sono utili per uno sviluppatore software. Gli elementi fondamentali di un IDE sono un codice editor, un compilatore/interprete e un debugger. Eclipse è un IDE in Java e fornisce numerosi strumenti per lo sviluppo del software. Supporta inoltre altri linguaggi di programmazione come C/C++, Cobol, Fortran, PHP e Python. *Plug-ins* può essere aggiunto alla piattaforma base di Eclipse per aumentarne la funzionalità.

Il termine Eclipse si riferisce anche alla comunità di software libero che sviluppa la piattaforma Eclipse. L'attività si divide in progetti che hanno lo scopo di creare una piattaforma robusta, ridimensionabile e di qualità per lo sviluppo del software con l'IDE di Eclipse. Questo lavoro viene coordinato dalla Fondazione Eclipse, che come organizzazione no-profit, fu creata per la promozione e lo sviluppo della piattaforma Eclipse e che sostiene sia la comunità sia l'ecosistema Eclipse.

### 9.10.1. Storia di Eclipse

Molta della programmazione Eclipse è stata effettuata da IBM, prima che il progetto Eclipse venne creato in quanto tale. Il predecessore di Eclipse era VisualAge ed è stato realizzato utilizzando Smalltalk in un ambiente di sviluppo chiamato Envy. Dopo che Java apparve negli anni novanta, IBM sviluppò una macchina virtuale che lavorava sia con Smalltalk e sia con Java. La rapida crescita di Java e dei suoi vantaggi concentrati su un Internet che si stava notevolmente espandendo, costrinsero IBM a considerare di abbandonare questa doppia macchina virtuale e di costruire da zero una nuova piattaforma basata su Java. Il prodotto finale fu Eclipse, che nel 2001 era già costato a IBM circa 40 milioni di dollari.

Verso la fine del 2001, IBM insieme a Borland, diede vita alla fondazione no-profit Eclipse aprendosi così al mondo open source. Questo consorzio è stato gradualmente raggiunto da importanti società di sviluppo software a livello globale: Oracle, Rational Software, Red Hat, SuSE, HP, Serena, Ericsson e Novell, tra gli altri. Ci sono due assenze importanti: Microsoft e Sun Microsystems. Microsoft è stata esclusa a causa del monopolio sul mercato e Sun Microsystems aveva il suo proprio IDE, costituendo il concorrente principale di



Eclipse: NetBeans. In realtà, il nome di Eclipse è stato scelto perché l'obiettivo era quello di creare un IDE in grado di "Eclipse Visual Studio" (Microsoft) e di "eclissi di sole" (Sun Microsystems).

L'ultima versione stabile di Eclipse è disponibile per Windows, Linux, Solaris, AIX, HP-UX e Mac OS X sistemi operativi. Tutte le versioni di Eclipse devono avere una Java Virtual Machine (JVM) installata nel sistema, preferibilmente JRE (Java Runtime Environment) o JDK (Java Developer Kit) di Sun, che ad oggi 2007, non sono ancora liberi (anche se Sun ha annunciato che il loro sarà JVM).

### 9.10.2. Stato attuale di Eclipse

Tutto il lavoro preparato per il consorzio Eclipse è organizzato in progetti diversi. Questi progetti sono divisi in sotto-progetti e a loro volta i sotto-progetti in componenti. I progetti di alto livello sono gestiti dai comitati della fondazione Eclipse (PMC, *project management committees*). La seguente lista mostra i progetti di alto livello:

- Eclipse. Piattaforma base per il resto dei componenti. Questa piattaforma sarà libera, robusta, completa e di buona qualità per lo sviluppo delle *rich client platforms* (RCP) e strumenti integrati (*plug-ins*). Il kernel del tempo di esecuzione della piattaforma Eclipse è chiamato Equinox ed è un'implementazione della specifica OSGi (Open Services Gateway Initiative), che descrive un'architettura orientata al servizio (SOA) per applicazioni.
- Strumenti (ETP, *Eclipse tools project*). Vari strumenti e componenti comuni per la piattaforma Eclipse.
- Web (WTP, *web tools project*). Strumenti per lo sviluppo di applicazioni web e JEE (Java Enterprise Edition).
- *Test and Performance Tools Project* (TPTP). Gli strumenti di prova e misure del livello di performance così che gli sviluppatori possono tenere sotto controllo le loro applicazioni e renderle più produttive.
- Web reports (BIRT, *business intelligence and reporting tools*). Sistema di generazione di report sul Web.
- Modelling (EMP, *Eclipse modelling project*). Strumenti di sviluppo basati sui modelli.
- Data (DTP, *data tools platform*). Supporto per tecnologie che gestiscono dati.

- Dispositivi dedicati (DSDP, *device software development platform*). Strumenti per lo sviluppo di applicazioni che devono essere eseguiti su dispositivi con hardware limitato, in altre parole, servizi dedicati.
- *Architettura orientata al servizio* (SOA). Strumenti per sviluppare progetti orientati al servizio.
- Tecnologia Eclipse. Ricerca, diffusione e sviluppo della piattaforma Eclipse.

I principi che guidano lo sviluppo della comunità Eclipse sono i seguenti:

- Qualità. Il software sviluppato in Eclipse deve rispettare gli standard di qualità dell'ingegneria del software.
- Sviluppo. La piattaforma Eclipse, e tutti gli strumenti su cui si basa, si devono sviluppare in modo dinamico secondo le richieste dell'utente.
- Meritocrazia. Più uno contribuisce, più ha responsabilità.
- Ecosistema Eclipse. Ci saranno risorse donate dalla comunità open source al consorzio Eclipse. Queste risorse saranno impiegate in modo che la comunità possa trarne beneficio.

Il processo di sviluppo per Eclipse segue alcune fasi predefinite. In primo luogo, c'è una fase chiamata pre-proposta, in cui un individuo o una società dichiara l'interesse nello stabilire un progetto. Se la proposta viene accettata, si decide se sarà un progetto di alto livello o un sottoprogetto. Il passo successivo è quello approvare il progetto in termini di applicabilità e qualità. Dopo una fase in cui il progetto è in incubazione, ci sarà una revisione finale. Se il progetto passa questa revisione, avrà dimostrato la sua validità di fronte alla comunità Eclipse e si passerà alla fase di attuazione.

### 9.10.3. X-ray of Eclipse

Eclipse è distribuito sotto una licenza EPL (Eclipse Public License). Questa licenza è considerata libera da FSF e OSI. Con la licenza EPL, è possibile utilizzare, modificare, copiare e distribuire nuove versioni del prodotto sotto licenza. Il predecessore EPL è il CPL (Common Public License). Il CPL è stato scritto con IBM, mentre l'EPL è il lavoro del consorzio Eclipse.

Dare una stima dell'investimento e dello sforzo speso in Eclipse non è un compito facile. Ciò è dovuto al fatto che il codice sorgente che comprende l'ecosistema Eclipse è distribuito in numerosi progetti e depositi software.

Di seguito sono riportati i risultati ottenuti impiegando il modello COCOMO per la piattaforma Eclipse, che viene utilizzato come base per il resto dei *plugins*.

**Tabella 24. Analisi di Eclipse**

Website	<a href="http://www.eclipse.org">http://www.eclipse.org</a>
Inizio del progetto	2001
Licenza	Eclipse Public License
Versione analizzata	3.2.2
Linee di codice sorgente.	2.163.932
Numero di file	15.426
Stima del costo	\$ 85.831.641
Stima del tempo di esecuzione	6,22 anni (74,68 mesi)
Stima del numero medio di sviluppatori	102,10
Numero approssimativo di sviluppatori	133 <i>committers</i>
Strumenti di assistenza allo sviluppo	CVS, mailing lists, sistema di bug-tracking (Bugzilla)

La seguente tabella mostra i linguaggi di programmazione usati in Eclipse 3.2.2:

**Tabella 25. Linguaggi di programmazione usati in Eclipse**

Linguaggio di programmazione	Linee di codice	Percentuale
Java	2.066.631	95,50%
C	85.829	3,97%
Perl	3.224	0,06%
C++	5.442	0,25%
JSP	3.786	0,17%
Perl	1.325	0,06%
Lex	1.510	0,03%
Shell	849	0,04%
Python	46	0,00%
PHP	24	0,00%

## 10. Altre risorse libere

"If you want to make an apple pie from scratch, you must first create the universe".

"Se vuoi fare una torta di mele partendo da zero, devi prima creare l'universo".

Carl Sagan

Si possono estendere anche ad altre risorse le idee che si nascondono dietro ai programmi liberi? Alcune risorse d'informazione, che possono essere facilmente copiate elettronicamente, sono simili ai programmi e si possono applicare ad esse le stesse libertà, regole e i modelli di sviluppo e business. Tuttavia ci sono alcune differenze, e le implicazioni di tali differenze hanno fatto sì che i programmi non si sviluppassero con la stessa forza. La principale differenza è che, per farli funzionare, è necessario copiare i programmi, mentre quando vengono copiati altri tipi d'informazione, devono passare attraverso un processo più o meno costoso prima che possano iniziare ad essere in qualche modo utili: il che può andare dall'imparare un documento, alla fase di produzione dell'hardware scritta in un linguaggio appropriato.

### 10.1. Le più importanti risorse libere

Abbiamo già discusso la documentazione di programmi e altri documenti tecnici nel paragrafo 3.2.5. Ora vedremo altri tipi di creazioni che possono essere anche testuali e che non sono legati al software ma piuttosto all'ambito scientifico, tecnico e artistico.

#### 10.1.1. Articoli scientifici

Il modo in cui la scienza evolve è dovuto in larga misura al fatto che i ricercatori che la rendono un progresso per il bene dell'umanità, pubblicano i risultati dei loro lavori su riviste che raggiungono un vasto pubblico. Grazie a questa diffusione, i ricercatori sviluppano una carriera che consente loro di progredire verso posizioni di maggiore reputazione e responsabilità, nonostante ricevano profitti da contratti di ricerca ottenuti grazie al loro crescente prestigio.

Questo modo di diffusione degli articoli rappresenta un *modello commerciale* che si è dimostrato produttivo. Affinché questo modello funzioni, la qualità del lavoro deve essere garantita e gli articoli devono essere ampiamente distribuiti. L'ostacolo che impedisce tale diffusione è la grande quantità di riviste esistenti, di costo notevole, che possono essere acquistate solo grazie a generosi budget. La qualità è garantita dal fatto che gli articoli vengono revisionati da specialisti o persone di livello simile.

Legato a questo, sono nate molte riviste online fra cui si ricorda il veterano *First Monday* ("*First Monday: peer reviewed journal on the Internet*") [26] o il progetto *Public Library Of Science* (PLOS <http://www.publiclibraryofscience.org> [55]). La "Directory of Open Access Journals" [22] ne cita altri ancora. Devono persone diverse dagli autori avere il permesso di pubblicare modifiche a questo tipo di articoli? Ci sono obiezioni a tale riguardo che vanno dalla possibilità di articoli di qualità inferiori o equivoci di opinioni o dei risultati, fino al pericolo di persone che possono facilmente plagiare gli articoli e raggiungere un grado elevato senza fatica, negando i meriti ottenuti con impegno dei veri autori. Tuttavia, il fatto che tutti gli scrittori abbiano l'obbligo di citare l'autore originale e di sottoporre gli articoli ad una revisione paritetica per la pubblicazione in una rivista prestigiosa, può controbilanciare questi problemi (vedere paragrafo 10.2.2).

Può essere stabilita un'analogia fra il software libero e la scienza dato che il modello di sviluppo del primo richiede una grande quantità di diffusione, di revisioni paritetiche (presumibilmente esperti) e di riutilizzo dei risultati ("Free software/free science", 2001) [154].

### 10.1.2. Leggi e standard.

Ci sono documenti di natura regolatoria che definiscono il modo di fare le cose così da migliorare la co-esistenza tra le persone o in modo tale che programmi o macchine possano lavorare insieme. Questi documenti devono essere diffusi in quanto ogni ostacolo potrà essere controproducente. Per questo motivo è incomprensibile che ricevano un trattamento speciale come dimostrato nell'atto di proprietà intellettuale spagnolo:

"I provvedimenti legali o regolatori e le relative bozze, le sentenze degli organi giurisdizionali e gli atti, gli accordi, le deliberazioni e le decisioni degli enti pubblici, e le traduzioni ufficiali di tutti questi testi, non possono essere oggetto di proprietà intellettuale".

L'equivalente tecnologico di queste leggi sarebbero le norme o gli standard. Nella programmazione, i protocolli di comunicazione sono specialmente importanti, sia fra le macchine remote sia tra i moduli nella stessa macchina. È ovvio che non dobbiamo limitare la loro diffusione soprattutto se si desidera che fioriscano i programmi che operano con altri ma nonostante questo, gli enti che tradizionalmente regolano questi problemi, sono ad esempio ISO<sup>11</sup> e ITU<sup>12</sup>, vendono i loro regolamenti e standard perfino in formati elettronici e proibiscono la loro redistribuzione. Nonostante questo possa essere giustificato fino ad un certo punto, rivendicando il bisogno di coprire parte dei costi, la distribuzione libera degli standard è stata molto più produttiva; questo è il caso di W3C<sup>13</sup> linee guida e soprattutto per quel che riguarda gli standard Internet, i documenti chiamati RFCs (*richiesta di commenti*) che sono esistiti fin dall'inizio in formati elettronici che possono essere letti usando qualsiasi forma di text editor.

<sup>(11)</sup>Organizzazione Internazionale per la standardizzazione

<sup>(12)</sup>Unione internazionale delle telecomunicazioni

<sup>(13)</sup>Consorzio World Wide Web

Tuttavia, il successo dei protocolli Internet non è dovuto esclusivamente alla loro disponibilità. Altri fattori includono il *modello di sviluppo*, che è molto simile al software libero per la sua apertura a far partecipare chiunque sia interessato e l'uso di mailing list e di elementi simili. Questo processo è descritto in "The Internet standards process - revision 3" [94] e "GNU make" [36].

È consentito modificare i testi delle leggi e dei regolamenti? Ovviamente non se porta a confusione. Ad esempio, un RFC deve essere modificato al fine di semplificarlo o aggiungere commenti esemplificativi, considerando che nemmeno questo è consentito senza autorizzazione esplicita per le raccomandazioni del W3C (<http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231>) [65]. Le licenze stesse sono documenti legali che non possono essere modificate. È possibile creare nuovi regolamenti derivanti da altri già esistenti utilizzando i documenti originali? Ciò provocherebbe probabilmente la naturale diffusione di regolamenti simili e incompatibili che potrebbero creare confusione e potrebbero aiutare le società che dominano il mercato a promuovere le proprie varianti incompatibili, come in effetti accade in particolare nel settore Internet. Tuttavia, per quanto riguarda la legislazione dello stato, molto spesso le leggi sono state letteralmente copiate da quelle di altri paesi e adattate con piccole modifiche alle particolarità locali.

Esiste un modello commerciale per le leggi e regolamenti? Ci sono numerosi professionisti che lavorano sulle leggi che si occupano di idearle, interpretarle e farle rispettare (legislatori, giuristi, avvocati, giudici, ecc.) Ci sono laboratori che procurano i certificati di conformità per i regolamenti. Gli organismi di regolamentazione sussistono, o dovrebbero sussistere, sui contributi dei loro membri che desiderano, per esempio, promuovere norme perché la loro attività è basata su prodotti in grado di interagire.

Allo stesso modo in cui è conveniente avere una definizione di *free software* o di *open software*, è anche necessario disporre di una definizione che sia adeguata per *open standards*. Bruce Perens (<http://perens.org/OpenStandards>) [15] propone la seguente definizione basata su questi principi:

- 1) Disponibilità: se possibile, gli standard aperti devono essere disponibili affinché tutti possano leggerli e implementarli.
- 2) Massimizzare la scelta dell'utente finale.
- 3) Gli standard aperti devono essere liberi per tutti da implementare senza nessun diritto d'autore o tassa (le certificazioni di conformità possono comportare una quota, anche se Bruce Perens consiglia che dovrebbero essere disponibili strumenti per la *self-certification*).
- 4) Nessuna discriminazione per favorire un responsabile dell'implementazione rispetto ad un altro.

- 5) L'estensione o i permessi del sottoinsieme (non certificabili).
- 6) Astensione di pratiche avide da parte dei produttori dominanti. Tutti i sottoinsiemi proprietari devono avere un'implementazione open standard.

### 10.1.3. Enciclopedie

Nel 1999, Richard Stallman lanciò l'idea di un'enciclopedia libera ("The free universal encyclopaedia and learning resource", 2001) [210] come meccanismo per evitare l'appropriazione della conoscenza e per garantire l'accesso universale all'apprendimento e la relativa documentazione. Si tratterebbe di articoli forniti dalla comunità, senza alcun controllo centralizzato, in cui diversi attori svolgerebbero compiti differenti, come ad esempio rivedere o controllare gli articoli non come obbligo ma come raccomandazione. Questa enciclopedia non contiene solo testo, ma anche multimedia e software didattici liberi.

Sono nate varie iniziative per far sì che ciò diventasse realtà. Ad esempio, Nupedia (<http://www.nupedia.com>) [178] cercò di creare un'enciclopedia di qualità ma il tentativo fallì probabilmente perché richiedeva un formato che era relativamente difficile da imparare (TEI), sebbene sembra che avvenne più a causa del fatto di avere tutti gli articoli modificati, rivisti da scienziati e esaminati dal punto di vista dello stile, etc.

Il successore di Nupedia fu Wikipedia, che ebbe molto più successo (<http://www.wikipedia.org>) [69]. Wikipedia è un'enciclopedia multilingue libera, basata sulla tecnologia *wiki*. Wikipedia è scritta in modo cooperativo da volontari e la stragrande maggioranza degli articoli può essere modificata da chiunque abbia un web browser. Il suo successo è basato sulla sua struttura che è più flessibile in termini di editing, che elimina gli ostacoli che erano presenti in Nupedia e che la rende più vicina all'idea che Stallman aveva in mente. La parola *wiki* deriva dall'Hawaiano *wiki wiki* (veloce). La tecnologia *Wiki* permette a chiunque di modificare qualsiasi documento usando una sistema di testo strutturato, che è straordinariamente semplice come si è visto nel paragrafo 8.6.2. Nel febbraio 2007, il numero di articoli in inglese di Wikipedia superava 1.500.000 ed esistono più di 200.000 articoli in spagnolo.

## Nota

Wikipedia è un progetto dell'organizzazione no-profit Wikimedia, che porta avanti anche i seguenti progetti basati sullo stesso modello di Wikipedia:

- Wiktionary (<http://www.wiktionary.org>) [66]. Questo è un progetto cooperativo volto a creare un dizionario multilingue libero con definizioni, etimologie e pronunce nei linguaggi richiesti.
- Wikibooks (<http://www.wikibooks.org/>) [67]. Questo è un progetto che mira a fornire libri di testo, manuali, lezioni o altri testi didattici a chiunque desideri questi elementi, senza pagare.
- Wikiquote (<http://www.wikiquote.org>) [70]. È una raccolta di frasi famose in tutte le lingue, che include le fonti quando queste sono sconosciute.
- Wikisource. È una libreria di testi originali che sono di pubblico dominio o che sono stati pubblicati con un GFDL (GNU Free Documentation License).
- Wikispecies (<http://species.wikimedia.org/>) [71]. È un repertorio aperto di specie animali, specie vegetali, funghi, batteri e tutte le forme di vita conosciute.
- Wikinoticias (<http://wikinews.org/>) [68]. È una fonte di contenuto di notizie libere nel quale gli utenti sono gli editors.
- Commons (<http://commons.wikimedia.org/>) [19]. È un deposito libero di immagini e di contenuto multimediale.
- Wikiversidad (<http://wikiversity.org/>) [72]. È una piattaforma aperta e libera basata su progetti di insegnamento a tutti i livelli didattici.
- Meta-Wiki (<http://meta.wikimedia.org/>) [48]. È il sito web che supporta tutti i progetti della fondazione Wikimedia.

Va inoltre menzionata la *Concise Encyclopedia of Mathematics*, che ha un concetto più limitato di cosa significhi libero (può essere consultata solo su internet) e un modello di sviluppo nel quale è necessario sottomettere tutti i contributi ad una redazione prima della pubblicazione.

### 10.1.4. Corsi

Allo stesso scopo delle enciclopedie, è possibile produrre materiale di apprendimento libero, come appunti, lucidi, esercizi, libri, agende o software didattico. Esiste una tendenza a considerare le università come aziende che producono e vendono conoscenza, il che contraddice i principi base. Le ragioni per cui un'università può rendere disponibili questi materiali a tutti sono le seguenti:

- Il raggiungimento della sua missione, come un agente che diffonde conoscenza.
- Il basso prezzo della messa a disposizione del materiale esistente in tutto il mondo.
- Il fatto che questi materiali non possono sostituire l'insegnamento di persona.



- L'idea di questi materiali come una pubblicità che può attrarre studenti e contribuire al prestigio dell'universo.
- La possibilità di creare una comunità di insegnanti che rivede e migliora i materiali didattici.

L'iniziativa più prominente in questo campo è quella del MIT (<http://ocw.mit.edu>) [174], che ha lo scopo di rendere accessibile più di duemila risorse ben catalogate in modo uniforme e coerente.

#### **10.1.5. Collezioni e databases**

La mera raccolta di informazioni che segue determinati criteri, che organizza l'informazione e la mette a disposizione è di per sé un prodotto di informazioni importante, indipendentemente dal contenuto stesso, che è quindi il prodotto dei suoi autori e, di conseguenza, soggetto alle restrizioni sulla libertà di accesso, di modifica o ridistribuzione del contenuto. Pertanto, se si vuole informazione libera, si possono anche richiedere collezioni libere.

Ad esempio, possiamo desiderare di classificare le informazioni importanti su Internet, organizzare e commentare i link. Questo è ciò che fa la ODP (Open Directory Project <http://dmoz.org> [109]); opera Netscape e mantiene i redattori volontari organizzati secondo una struttura gerarchica. La directory completa può essere copiata liberamente in formato RDF e pubblicata con alcune modifiche, come fa Google e molti altri motori di ricerca che ne traggono vantaggio. Netscape, che possiede la directory, garantisce un "Open Directory Project social contract"[53] ispirato a quello della distribuzione Debian ([http://www.debian.org/social\\_contract.html](http://www.debian.org/social_contract.html)) [106], che facilita i contributi esterni assicurando che l'Open Directory Project sarà sempre libero, con le policy pubbliche e auto-regolamentate dalla comunità e che gli utenti sono la prima priorità.

Altri esempi di collezioni che potrebbero interessare, sono le distribuzioni di software libero, con i programmi modificati in modo che si adattano perfettamente e che sono precompilati così che possono girare con facilità.

#### **10.1.6. Hardware**

Per quanto concerne l'hardware, ci sono due aspetti principali che riguardano l'essere libero. Il primo è la necessità di interfacce e gruppi di istruzioni per renderli liberi, in modo tale che chiunque possa creare un responsabile per dispositivi o un compilatore per un'architettura. Il secondo punto è che ci dovrebbero essere informazioni ed energia sufficienti per riprodurre un design hardware, modificarlo e combinarlo con altri. I progetti di design possono essere considerati software in un linguaggio appropriato (VHDL, Verilog, ecc.) Tuttavia, non è facile farli funzionare, in quanto devono essere fabbricati, il

che è costoso e lento. D'altra parte ci sono iniziative in questo senso, tra le quali possiamo citare OpenCores ( <http://www.opencores.org> ) [52], per i circuiti integrati.

### 10.1.7. Letteratura e arte

Per terminare la nostra analisi sulle risorse libere, non si può dimenticare arte e letteratura, il cui ultimo scopo non è tanto utilitario quanto estetico. Quali motivi può avere un artista per dare alle persone la libertà di copiare, modificare o ridistribuire il proprio lavoro? Da un lato, ciò può aiutare a renderli famosi e facilitare la diffusione del loro lavoro, che gli permette di ricevere entrate da altre attività come ad esempio concerti o commissioni, e dall'altro lato può promuovere la sperimentazione e la creatività. Nell'arte, si hanno le stesse situazioni di quelle tecniche. L'innovazione è progressiva ed è difficile a volte distinguere fra un plagio e un lavoro che è rappresentativo o che segue un movimento o trend artistico.

Creazione e interpretazione non sono ovviamente la stessa cosa e nemmeno musica e letteratura. La musica, la pittura, la fotografia e il cinema sono molto simili ai programmi nel senso che si possono far subito funzionare su un computer, mentre ciò non è vero ad esempio per la scultura. Non esistono molte iniziative open source nell'arte e nella letteratura e quelle che sono presenti, sono molto diverse. Si possono citare i romanzi di Wu Ming (<http://www.wumingfoundation.com>) [29] collettive.

## 10.2. Licenze per altre risorse libere

Le licenze per il software libero sono state una fonte di ispirazione per altre risorse intellettuali a tal punto che molte di queste sono state adottate direttamente, soprattutto per quanto riguarda la documentazione e, in altre occasioni, sono state leggermente adattate, come succede con la licenza pionieristica Open Audio ([http://www.eff.org/IP/Open\\_licenses/eff\\_oal.html](http://www.eff.org/IP/Open_licenses/eff_oal.html)) [114]. La maggior parte di queste licenze sono *copyleft*, se permettono lavori derivati.

La licenza di documentazione libera di GNU (*vedi* paragrafo 10.2.1) è stata usata ed è spesso usata per tutti i tipi di testo, anche se le licenze Creative Commons (*vedi* paragrafo 10.2.2) vengono gradualmente accettate.

Le licenze di programma (GPL e LGPL) sono state infatti utilizzate anche per l'hardware, anche se questo argomento è complesso e difficile da conciliare con l'attuale legge. In effetti, i disegni e diagrammi possono essere usati, senza che vengano fisicamente copiati, per estrarre le idee che vengono usate per nuovi design chiusi. Ad esempio, la licenza pubblica generale dell'OpenIPCore Hardware ("OpenIPCore Hardware General Public License") [155] stabilisce che questa appropriazione non è consentita ma la validità giuridica del documento è discutibile [209]. L'unico modo possibile per proteggere queste idee

è di utilizzare una qualche forma di brevetto libero, che è qualcosa che non è stata ancora sviluppato ed è fuori della portata di quelli che non intendono o non sono in grado di creare un business costruito sulle idee.

### 10.2.1. Licenza di documentazione libera GNU

Una delle più note licenze *copyleft* per la documentazione tecnica, sia che corrisponda a programmi o a qualsiasi altro argomento, è quella della Free Software Foundation. Dopo aver scoperto che un documento non è la stessa cosa di un programma, Richard Stallman promosse una licenza per i documenti che andavano con i programmi e per gli altri documenti di natura tecnica o didattica.

Al fine di agevolare lo sviluppo di versioni derivate, deve essere resa disponibile a chiunque ne abbia bisogno una copia *trasparente* del documento, come spiegato nel paragrafo 3.2.5, e allo stesso modo le copie *opache*, in un'analogia tra i codici sorgente e gli oggetti dei programmi.

Uno dei motivi per cui avere una licenza è stabilire la paternità e garantire che le idee o le opinioni espresse dall'autore non vengano definite false. È per questo che le attività derivate devono avere un titolo sulla copertina diverso da quello delle versioni precedenti (salvo che non sia stata data espressa autorizzazione) e deve indicare esplicitamente il luogo in cui si può ottenere l'originale. Devono essere elencati i nomi dei principali autori dei documenti originali, così come i nomi delle persone che hanno eseguito qualunque modifica, e devono essere preservati tutti gli appunti sulla proprietà intellettuale. Allo stesso modo, devono essere conservati eventuali riconoscimenti e dediche e deve essere rispettata la sezione cronologia, se esiste, quando vengono aggiunte nuove modifiche. È anche possibile, e questo è l'aspetto della licenza che è stato più criticato, nominare sezioni invarianti e testi di copertina, che nessuno può modificare o eliminare; anche se, la licenza consente solo che i testi *non tecnici* siano considerati come sezioni invarianti, a cui la licenza si riferisce come sezioni *secondarie*.

Questa licenza ha creato molte polemiche nel campo del software libero, al punto che il progetto di distribuzione Debian sta attualmente (al momento della pubblicazione di questo libro) discutendo se eliminare la licenza o segnalare come *non liberi* tutti i documenti che hanno la licenza e considerarli non ufficiali. È importante ricordare che le sezioni invariate potrebbero essere aggiunte in seguito, anche se non ci sono, in quanto i lavori derivati possono essere soggetti ai termini della licenza stessa. Si è sostenuto per esempio, che ci possono essere sezioni invarianti errate o obsolete, che tuttavia, devono essere preservate. In ogni caso, la licenza è incompatibile con le linee guida del software libero di Debian ( <http://www.debian.org/>

#### Advice

Le prime versioni di questo testo erano sotto la licenza GFDL, ma gli autori decisero successivamente di usare una licenza Creative Commons (*vedi* paragrafo 10.2.2), che è più appropriata per le caratteristiche di un libro.

social\_contract.html#guidelines) [104], ma il problema è se la documentazione deve seguire queste linee guida (ad esempio, i testi delle licenze non possono nemmeno essere modificati).

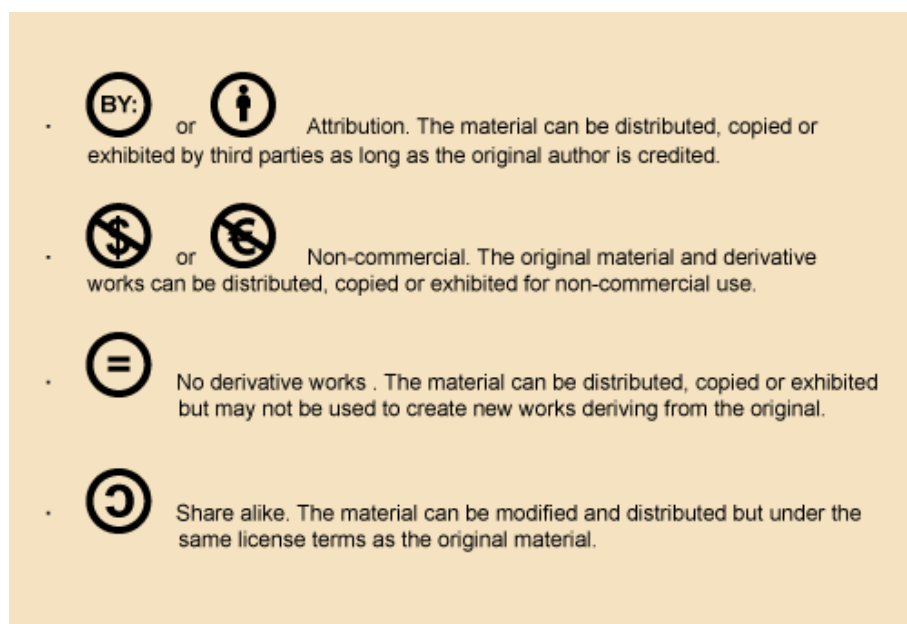
### 10.2.2. Licenze Creative Commons







La Creative Commons (<http://creativecommons.org>) [21] è un'organizzazione no-profit che venne fondata nel 2001 da esperti in proprietà intellettuali e in legge nella società informatica, con lo scopo di favorire la creazione, la conservazione e l'accessibilità delle risorse intellettuali cedute alla comunità in svariati modi. La Creative Commons è basata sull'idea che alcune persone possano non voler usare tutti i diritti della proprietà che la legge permette, in quanto ciò potrebbe impedire la loro ampia distribuzione.

Le prime licenze Creative Commons per lavori creativi, delle quali ci furono varie versioni, si svilupparono nel tardo 2002. Queste licenze erano progettate per essere:

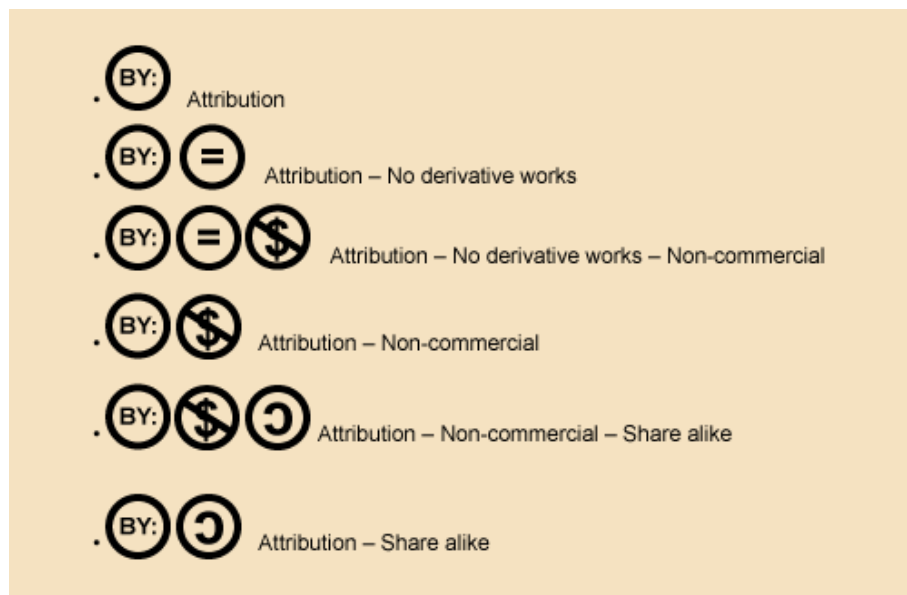
- abbastanza forti da resistere un esame della corte, nei diversi paesi;
- abbastanza semplici affinché possano essere usati anche da non avvocati;
- abbastanza sofisticati da essere identificati da varie applicazioni web.

Le diverse licenze permettono al creatore di selezionare quali tipi di libertà sono permesse, oltre alla copia, in conformità a quattro punti:




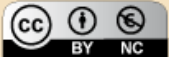
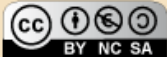



-  or  Attribution. The material can be distributed, copied or exhibited by third parties as long as the original author is credited.
-  or  Non-commercial. The original material and derivative works can be distributed, copied or exhibited for non-commercial use.
-  No derivative works. The material can be distributed, copied or exhibited but may not be used to create new works deriving from the original.
-  Share alike. The material can be modified and distributed but under the same license terms as the original material.

Nella versione 1.x delle licenze Creative Commons, c'erano undici tipi di licenza che associavano le quattro caratteristiche fondamentali appena citate. Il 98% degli autori scelse l'opzione "attribuzione"; di conseguenza, a partire dalla versione 2.x delle licenze Creative Commons, l'attribuzione è un requisito. Questo riduce gli undici tipi di licenze a sei, che sono le seguenti:



La tabella di sotto mostra uno schema delle licenze con le corrispondenti icone. Questa icona è di solito un link ad un riassunto della licenza, ospitato da [21].

	Allows modifications.	Allows modifications if shared alike.	Does not allow modifications.
Allows commercial use.			
Does not allow commercial use.			

È possibile usare l'icona generica<sup>14</sup> al posto dell'icona che rappresenta la licenza ma deve essere collegata alla licenza scelta dall'autore. Si può ottenere il codice HTML del link per la licenza dalla Creative Commons [21]. Una volta che è stata scelta la licenza e aggiunta l'icona corrispondente, il lavoro avrà ottenuto la licenza e voi riceverete il:

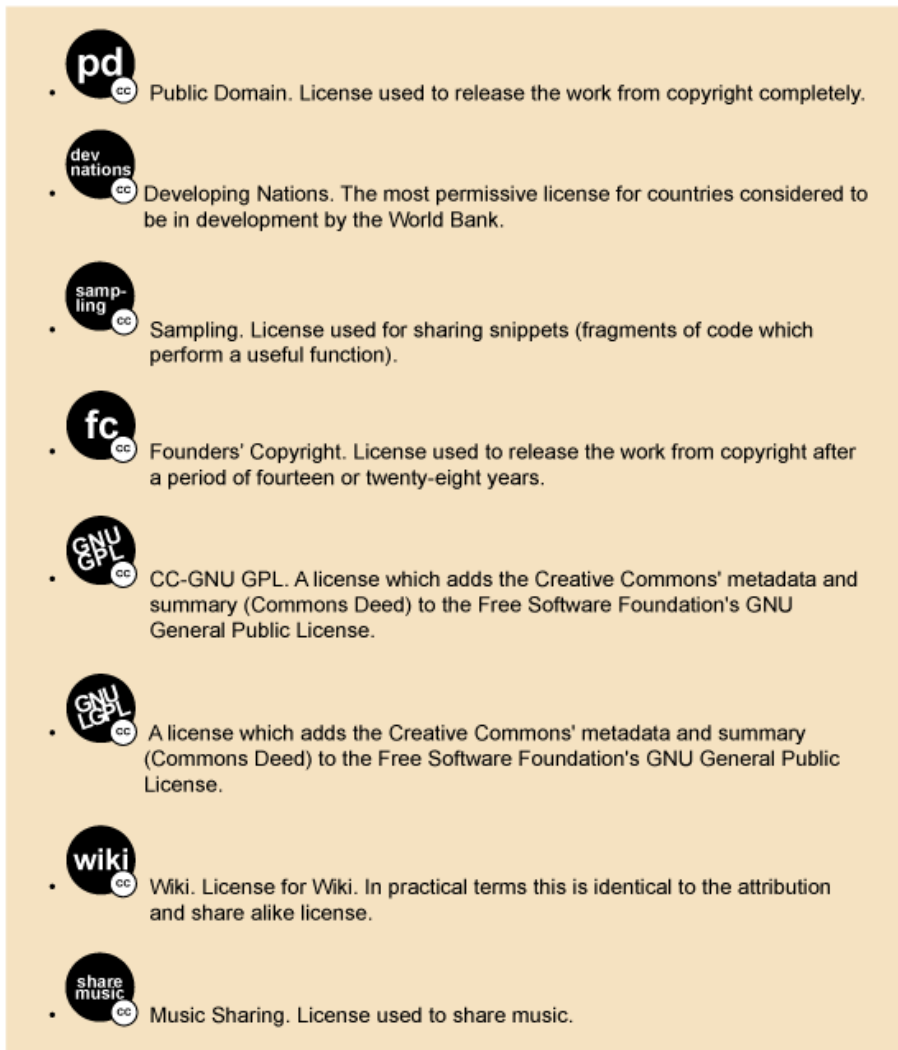


- *Commons deed.* Un riassunto della licenza con le icone specifiche per la licenza. Questo riassunto verrà mostrato quando si clicca il link ottenuto dalla Creative Commons [21].

- *Legal Code*. Questo è il testo legale completo e le clausole scritte in piccolo sul quale si basa la licenza. A questo testo si può accedere dal riassunto qui sopra descritto.
- *Digital code*. Questo è la descrizione RDF (*resource description framework*), che i motori di ricerca e altre applicazioni usano per identificare la licenza per i lavori e le condizioni d'uso.

Nel febbraio 2007, venne pubblicata la versione 3.0 delle licenze Creative Commons. Questo è un aggiornamento che corregge molte delle colpe che la gente trovava. La prima grande modifica è che la licenza generica non è più basata sul modello statunitense ma sulla terminologia della convention di Berna. In secondo luogo, i diritti morali e la società di gestione vengono specificatamente citate, in quanto sono state prese diverse decisioni in ogni giurisdizione. Come terzo e ultimo, entrambi i testi del *commons deed* e del *legal code* che andavano con ciascuna licenza, vennero modificati per mettere in chiaro che la clausola relativa al riconoscimento della paternità non permette al licenziatario di implicare o dare l'impressione di avere una relazione o essere associato in alcun modo al licenziante.

Inoltre, la Creative Commons fornisce altri tipi di licenze per specifiche applicazioni. Come ad esempio:



Non tutte le licenze Creative Commons sono considerate libere dai settori legati al free software in quanto le quattro libertà fondamentali devono applicarsi prima che le licenze vengano definite tali (*vedi* paragrafo 1.1.1). Benjamin "Mako" Hill (sviluppatore Debian e Ubuntu) creò il sito [Freedomdefined.org](http://freedomdefined.org/) (<http://freedomdefined.org/>) [28], con l'obiettivo di fornire una migliore definizione di cosa sia la cultura libera e cosa non lo è. Su queste premesse, delle sei licenze base della Creative Commons, solo due sono strettamente libere: attribution alone (BY) e attribution-share-alike (BY-SA), l'ultima delle quali ha anche *copyleft*.





## Bibliografia

- [1] Aap Project: <http://www.a-a-p.org>
- [2] Ada Core Technologies: <http://www.gnat.com/>
- [3] Alcôve: <http://www.alcove.com>
- [4] Alcôve-Labs: <http://www.alcove-labs.org>
- [5] Alioth: <http://alioth.debian.org>
- [6] Anjuta: <http://www.anjuta.org>
- [7] The Apache Ant Project: <http://ant.apache.org>
- [8] Arch Revision Control System: <http://www.gnu.org/software/gnu-arch/>
- [9] artofcode LLC: <http://artofcode.com/>
- [10] Autoconf: <http://www.gnu.org/software/autoconf>
- [11] Barrapunto: <http://barrapunto.com>
- [12] Bazaar GPL Distributed Version Control Software: <http://bazaar-vcs.org/>
- [13] Berlios. The Open Source Mediator: <http://berlios.de>
- [14] Bitkeeper Source Management: <http://www.bitkeeper.com>
- [15] Bruce Perens: <http://perens.com/OpenStandards/Definition.html>
- [16] Caldera: <http://www.sco.com>
- [17] Cisco Enterprise Print System: <http://ceps.sourceforge.net/>
- [18] Code::blocks: <http://www.codeblocks.org>
- [19] Commons: <http://commons.wikimedia.org/>
- [20] Concurrent Version System: <http://ximbiot.com/cvs/>
- [21] Creative Commons. <http://creativecommons.org>
- [22] Directory of Open Access Journals: <http://www.doaj.org>
- [23] Eclipse - An Open Development Platform: <http://www.eclipse.org>
- [24] eCos: <http://sources.redhat.com/ecos/>
- [25] eCos license 2.0: <http://www.gnu.org/licenses/ecos-license.html>
- [26] *First Monday. Peer Reviewed Journal on the Internet* : <http://firstmonday.org>
- [27] Free Software Foundation: <http://www.fsf.org>
- [28] Freedom Defined (Free Cultural Works): <http://freedomdefined.org/>
- [29] Fundación Wu Ming: <http://www.wumingfoundation.com>
- [30] GForge: <http://gforge.org>
- [31] Gettext: <http://www.gnu.org/software/gettext>
- [32] GNU Automake: <http://www.gnu.org/software/automake>
- [33] GNU Emacs: <http://www.gnu.org/software/emacs/>
- [34] GNU Libc: <http://www.gnu.org/software/libc>
- [35] GNU Libtool: <http://www.gnu.org/software/libtool>

- [36] GNU Make: <http://www.gnu.org/software/make/make.html>
- [37] GNU Troff: <http://www.gnu.org/software/groff/groff.html>
- [38] "Have you seen these hackers?": <http://www.mozilla.org/MPL/missing.html>
- [39] "History of TeX": <http://www.math.utah.edu/software/plot79/tex/history.html>
- [40] IBM Public License Version 1.0: <http://opensource.org/licenses/ibmpl.php>
- [41] Jam Product Information: <http://www.perforce.com/jam/jam.html>
- [42] KDevelop: <http://www.kdevelop.org>
- [43] Launchpad: <https://launchpad.net>
- [44] The Linux Documentation Project: <http://www.tldp.org>
- [45] LinuxCare: <http://www.levanta.com>
- [46] Mailman, the GNU Mailing List Manager: <http://www.list.org>
- [47] The Malone Bug Tracker: <https://launchpad.net/products/malone>
- [48] Metawiki: <http://meta.wikimedia.org/>
- [49] Mozilla Public License 1.1: <http://www.mozilla.org/MPL/MPL-1.1.html>
- [50] Mozilla Tinderbox: <http://www.mozilla.org/tinderbox.html>
- [51] NetBeans: <http://www.netbeans.org>
- [52] Open Cores: <http://www.opencores.org>
- [53] Open Directory Project Social Contract:
- [54] Open Source Initiative: <http://www.opensource.org>
- [55] Public Library of Science: <http://www.publiclibraryofscience.org>
- [56] Red Hat: <http://www.redhat.com>
- [57] Savannah: <http://savannah.gnu.org> and <http://savannah.nongnu.org>
- [58] Slashdot: News for Nerds. <http://slashdot.org>
- [59] Sleepycat License: <http://www.sleepycat.com/download/oslicense.html>
- [60] Sleepycat Software: <http://www.sleepycat.com/>
- [61] SourceForge: Open Source Software Development Website: <http://sourceforge.net>
- [62] Subversion: <http://subversion.tigris.org>
- [63] Texinfo - The GNU Documentation System:
- [64] Tigris.org: Open Source Software Engineering: <http://tigris.org>
- [65] W3c Document License:  
<http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231>
- [66] Wiktionary: <http://www.wiktionary.org>
- [67] Wikibooks: <http://www.wikibooks.org/>
- [68] Wikinews: <http://wikinews.org/>
- [69] Wikipedia: <http://www.wikipedia.org>
- [70] Wikiquote: <http://www.wikiquote.org>

- [71] Wikispecies: <http://species.wikimedia.org/>
- [72] Wikiversity: <http://wikiversity.org/>
- [73] X Window System Release 11 License: [http://www.x.org/Downloads\\_terms.html](http://www.x.org/Downloads_terms.html)
- [74] Ximian: <http://www.novell.com/linux/ximian.html>
- [75] Zope Corporation: <http://www.zope.com/>
- [76] Zope Public License 2.0: <http://www.zope.org/Resources/ZPL>
- [77] Law on Intellectual Property. Royal Legislative Decree 1/1996, of 12th April (April 1996):
- [78] Affero General Public License, 2002: <http://www.affero.org/oagpl.html>
- [79] Law on Intellectual Property. Law 23/2006, of 7th July (July 2006):
- [80] Flossimpact Study. Technical Report, European Comission, 2007: <http://flossimpact.eu>
- [81] ISO JTC 1/SC 34. Standard Generalized Markup Language (SGML, ISO 8879), 1986:
- [82] **Antoniades, I.; Samoladas, I.; Stamelos, I.; Bleris, G. L.** "Dynamical simulation models of the open source development process" En: Koch [157].
- <http://www.wi.wu-wien.ac.at/~koch/oss-book/>
- [83] **Bailey, E. C.** (1998). *Maximum RPM. Taking the Red Hat package manager to the limit* . <http://rikers.org/rpmbook/>
- [84] **González Barahona, J. M.** (2000). "Software libre, monopolios y otras yerbas". *Todo Linux* (3). <http://sinetgy.org/~jgb/articulos/soft-libre-monopolios/>
- [85] **González Barahona, J. M.** (2002). "¿Qué se hace con mi dinero?". *Todo Linux* (17).
- <http://sinetgy.org/~jgb/articulos/sobre-administracion/>
- [86] **González Barahona, J. M.; Robles, G.** Libre Software Engineering Web Site.
- <http://libresoft.dat.escet.urjc.es/>
- [87] **González Barahona, J. M.; Robles, G.** (2003, mayo). "Unmounting the *code god* assumption". En: *Proceedings of the Fourth International Conference on eXtreme Programming and Agile Processes in Software Engineering* . Génova, Italia.
- [88] **González Barahona, J. M.; Robles, G.; Ortuño Pérez, M. A.; Roderio Merino, L.; Centeno González, J.; Matellán Olivera, V.; Castro Barbero, E. M.; De las Heras Quirós, P.** "Anatomy of two GNU/Linux distributions". En: Koch [157].
- <http://www.wi.wu-wien.ac.at/~koch/oss-book/>
- [89] **Barnson, M. P.** *The Bugzilla guide* .
- <http://www.bugzilla.org/docs214/html/index.html>
- [90] **Baudis, P.** "Cogito manual page".
- <http://www.kernel.org/pub/software/scm/cogito/docs/>
- [91] **Bezroukov, N.** (1998, diciembre). "A second look at the cathedral and the bazaar". *First Monday* , 4(12).
- [http://www.firstmonday.org/issues/issue4\\_12/bezroukov/index.html](http://www.firstmonday.org/issues/issue4_12/bezroukov/index.html)
- [92] **Bodnar, L.** (2003). "Linux distributions. Facts and figures".
- <http://www.distrowatch.com/stats.php?section=packagemanagement>
- [93] **Boehm, B. W.** (1981). *Software Engineering Economics* . Prentice Hall.

[94] **Bradner, S.** (1996, octubre). "The Internet standards process. Revision 3 (rfc 2026, bcp 9)".

<http://www.ietf.org/rfc/rfc2026.txt>

[95] **Cederqvist, P.; GNU** (1993). "CVS - concurrent versions system". <http://www.gnu.org/manual/cvs/index.html>

[96] **Collins-Sussman, B.; Fitzpatrick, B. W.; Pilato, C. M.** (2004). *Version control with Subversion*. O'Reilly & Associates ( <http://www.ora.com> ).

<http://svnbook.red-bean.com/>

[97] **Cunningham, W.** "Wiki design principles".

[98] **Dachary, L.** (2001). "Savannah, the next generation".

<http://savannah.gnu.org/docs/savannah-plan.html>

[99] **Autonomous Government of Andalucía** (2003, March). Decree 72/2003, of 18th March, on Measures to Promote the Knowledge Society in Andalucía.

<http://www.andaluciajunta.es/SP/AJ/CDA/Ficheros/ArchivosPdf/DecretoConocimiento.pdf>

[100] **De Boor, A.** *Pmake. A tutorial* . <http://docs.freebsd.org/44doc/psd/12.make/paper.html>

[101] **De Icaza, M.** "The story of the GNOME Project".

<http://primates.ximian.com/~miguel/gnome-history.html>

[102] **Senate of the Republic of France** . Forum sur la proposition de loi tendant à généraliser dans

l'administration l'usage d'Internet et de logiciels libres.

<http://www.senat.fr/consult/loglibre/index.htm>

[103] **De las Heras Quirós, P.; González Barahona, J. M.** (2000). "Iniciativas de las administraciones públicas en relación al software libre". *Bole. TIC, ASTIC magazine* (14).

[104] **Debian.** "Debian free software guidelines".

[http://www.debian.org/social\\_contract.html#guidelines](http://www.debian.org/social_contract.html#guidelines)

[105] **Debian.** *Debian policy manual* .

<http://www.debian.org/doc/debian-policy/>

[106] **Debian.** "Debian social contract".

[http://www.debian.org/social\\_contract.html](http://www.debian.org/social_contract.html)

[107] **Schriftenreihe der KBSt** (2003, July). Leitfaden für die migration von basissoftware-komponenten auf serverund arbeitsplatzsystemen. Technical report, Koordinierungs-und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung (KBSt).

[http://www.kbst.bund.de/download/mlf\\_v1\\_de.pdf](http://www.kbst.bund.de/download/mlf_v1_de.pdf)

[108] **DiBona, C.; Ockman, S.; Stone, M.** (ed.) (1999). *Open sources. Voices from the open source revolution* . O'Reilly & Associates.

<http://www.oreilly.com/catalog/opensource/>

[109] **Open Directory Project** . <http://dmoz.org>

[110] **Ehrenkrantz, J. R.** (2003, May). "Release management within open source projects". In: *Proceedings of the 3rd Workshop on Open Source Software Engineering at the 25th International Conference on Software Engineering* . Portland, USA

[111] **European Council** (1991). Council Directive 91/250/CEE of 14th May 1991, on the legal protection of computer programmes.

<http://europa.eu.int/scadplus/leg/es/lvb/l26027.htm>

[112] **Feller, J.; Fitzgerald, B; Hissam, S.; Lakhani, K.** (ed.) (2003). *Making sense of the bazaar* . O'Reilly.

[113] **Fogel, K.; Bar, M.** (2001). *Open source code development with CVS* (2nd edition). Paraglyph Press.

<http://cvsbook.red-bean.com>

[114] **Electronic Frontier Foundation.** Open Audio.

[http://www.eff.org/IP/Open\\_licenses/eff\\_oal.html](http://www.eff.org/IP/Open_licenses/eff_oal.html)

[115] **Free Software Foundation.** GPLv3.

<http://gplv3.fsf.org>

[116] **Free Software Foundation.** LGPLv3. First discussion draft.

<http://gplv3.fsf.org/pipermail/info-gplv3/2006-July/000008.html>

[117] **Free Software Foundation** (1985): "The GNU Manifesto".

<http://www.gnu.org/philosophy/>

[118] **Free Software Foundation** (1991, junio). GNU General Public License, version 2. <http://www.fsf.org/licenses/gpl.html>

[119] **Free Software Foundation** (1999, February). GNU Lesser General Public License, version 2.1.

<http://www.fsf.org/licenses/lgpl.html>

[120] **Free Software Foundation.** "Free software definition".

<http://www.gnu.org/philosophy/free-sw.html>

[121] **Free Software Foundation.** "Free licenses".

<http://www.gnu.org/licenses/license-list.html>

[122] **Garbee, B.; Koptein, H.; Lohner, N.; Lowe, W.; Mitchell, B.; Murdock, I.; Schulze, M.; Small, C.** "A brief history of Debian". In the package: *Debian-history* .

[123] **Germán, D.** (2002, May). "The evolution of GNOME. In: *Proceedings of the 2nd Workshop on Open Source Software Engineering at the 24th International Conference on Software Engineering* . Florida, USA

[124] **Germán, D.; Mockus, A.** (2003, May): "Automating the measurement of open source projects". In: *Proceedings of the 3rd Workshop on Open Source Software Engineering at the 25th International Conference on Software Engineering* . Portland, USA

[125] **Ghosh, R. A.** (1998, March). "Cooking pot markets: an economic model for the trade in free goods and services on the Internet. *First Monday* , 3(3).

[http://www.firstmonday.dk/issues/issue3\\_3/ghosh/index.html](http://www.firstmonday.dk/issues/issue3_3/ghosh/index.html)

[126] **Ghosh, R. A.; Glott, R.; Krieger, B.; Robles, G.** (2002). *Free/libre and open source software: Survey and study* . Part iv: "Survey of developers".

[http://www.infonomics.nl/FLOSS/report/FLOSS\\_Final4.pdf](http://www.infonomics.nl/FLOSS/report/FLOSS_Final4.pdf)

[127] **Ghosh, R. A.; Prakash, V. V.** (2000, July). "The orbited free software survey". *First Monday* , 5(7).

[http://www.firstmonday.dk/issues/issue5\\_7/ghosh/index.html](http://www.firstmonday.dk/issues/issue5_7/ghosh/index.html)

[128] **Godfrey, M. W.; Tu, Q.** (2000, August). "Evolution in open source software. A case study". In: *Proceedings of the 2000 International Conference on Software Maintainance* .

- [129] **González, J. A.** (2002, March). "Carta al congresista Villanueva".  
<http://www.gnu.org.pe/mscarta.html>
- [130] **Goosens, M.; Rahtz, S.** (1999). *The LaTeX Web Companion*. Addison Wesley.
- [131] **Grad, B.** (2002, January-March). "A personal recollection: IBM's unbundling of software and services". In: *IEEE Annals of the History of Computing*, 24(1):64-71.
- [132] **Working Group on Libre Software** (1999). "Free software / open source. Information society opportunities for Europe?".  
<http://eu.conecta.it/paper.pdf>
- [133] **GrULIC**. "Legislation on the use of free software by the State".  
<http://proposicion.org.ar/doc/referencias/index.html.es>
- [134] **Hamerly, J; Paquin, T.; Walton, S.** (1999). "Freeing the source. The story of Mozilla". <http://www.oreilly.com/catalog/opensources/book/netrev.html>
- [135] **Hammel, M. J.** (1991, December). "The history of xfree86". *Linux Magazine*.  
[http://www.linux-mag.com/2001-12/xfree86\\_01.html](http://www.linux-mag.com/2001-12/xfree86_01.html)
- [136] **Harris, S.** (2001, August). *The Tao of IETF. A novice's guide to the Internet engineering task force* (RFC 3160, FYI 17).  
<http://www.ietf.org/rfc/rfc3160.txt>
- [137] **Harrison, P.** (2002). "The rational street performer protocol".  
<http://www.logarithmic.net/pfh/RSPP>
- [138] **Hasan, R.** "History of Linux".  
<http://ragib.hypermart.net/linux/>
- [139] **Hauben, M.; Hauben, R.** (1997). *Netizens. On the history and impact of Usenet and the Internet*. IEEE Computer Society Press.
- [140] **Healy, K.; Schussman, A.** (2003, January). "The ecology of open source software development". <http://opensource.mit.edu/papers/healyschussman.pdf>
- [141] **Hecker, F.** (1998, May). "Setting up shop. The business of open-source software".  
<http://www.hecker.org/writings/setting-up-shop.html>
- [142] **Hecker, F.** (1998). "Setting up shop. The business of open-source software".  
<http://www.hecker.org/writings/setting-up-shop.html>
- [143] **Hertel, G.; Niedner, S.; Herrmann, S.** (2003). "Motivation of software developers in open source projects. An Internet-based survey of contributors to the Linux kernel".  
<http://opensource.mit.edu/papers/rp-hertelniednerherrmann.pdf>
- [144] **Himanen, P.** (2001). *The hacker ethic and the spirit of the information age*. Random House.  
<http://www.hackerethic.org>
- [145] **Hunt, F.; Johnson, P.** (2002). "On the Pareto distribution of SourceForge projects. Technical report". Centre for Technology Management, Cambridge University Engineering Department, Mill Lane, Cambridge CB2 1RX.  
<http://www-mmd.eng.cam.ac.uk/people/fhh10/Sourceforge/Sourceforge%20paper.pdf>
- [146] **Open Source Initiative**. "History of the OSI".

<http://www.opensource.org/docs/history.php>

[147] **Hamilton, J. R.** (US ambassador to Peru) (2002, June). "Carta al presidente del Congreso de la República".

<http://www.gnu.org.pe/lobbyusa-congreso.html>

[148] **Jones, P.** (2000, May). "Brook's law and open source. The more the merrier?".

<http://www-106.ibm.com/developerworks/opensource/library/os-merrier.html?dwzone=opensource>

[149] **Jorgensen, N.** "Incremental and decentralized integration in FreeBSD". In: Feller *et al.* [112]. <http://www.dat.ruc.dk/~nielsj/research/papers/bazaar-freebsd.pdf>

[150] **Brooks, F. P.** (1975). *The mythical man-month. Essays on software engineering*. Addison-Wesley.

[151] **Kalt, C.** (2000, April). "Internet relay chat: architecture (RFC 2810)".

<http://www.ietf.org/rfc/rfc2810.txt>

[152] **Kelsey, J.; Schneier, B.** (1998, November). "The street performer protocol". In: *Third USENIX Workshop on Electronic Commerce Proceedings*. USENIX Press.

[http://www.counterpane.com/street\\_performer.html](http://www.counterpane.com/street_performer.html)

[153] **Kelsey, J.; Schneier, B.** (1999, June). "The street performer protocol and digital copyrights". *First Monday*, 4(6).

[http://www.firstmonday.dk/issues/issue4\\_6/kelsey/](http://www.firstmonday.dk/issues/issue4_6/kelsey/)

[154] **Kelty, C. M.** (2001, December). "Free software/free science". *First Monday*, 6(12).

[http://firstmonday.org/issues/issue6\\_12/kelty/index.html](http://firstmonday.org/issues/issue6_12/kelty/index.html)

[155] **Khatib, J.** "OpenIPCore Hardware General Public License".

[http://www.opencores.org/OIPC/OHGPL\\_17.shtml](http://www.opencores.org/OIPC/OHGPL_17.shtml)

[156] **Knuth, D.** (1989). *The TeXbook*. Addison Welsley.

[157] **Koch, S.** (ed.) (2003). *Free/open source software development*. Idea Group Inc.

<http://www.wai.wu-wien.ac.at/~koch/oss-book/>

[158] **Koch, S.; Schneider, G.** (2000). "Results from software engineering research into open source development projects using public data". In: *Diskussionspapiere zum Tätigkeitsfeld Informationsverarbeitung und Informationswirtschaft, H.R. Hansen und W.H. Janko (Hrsg.), Nr. 22*, Wirtschaftsuniversität Wien.

[159] **Kovács, G. L.; Drozdik, S.; Succi, G.; Zuliani, P.** (2004). "Open source software for the public administration". In: *Proceedings of the 6th International Workshop on Computer Science and Information Technologies (CIST 2004)*. Budapest, Hungary.

[160] **Krishnamurthy, S.** (2002, May). "Cave or community? An empirical examination of 100 mature open source projects". *First Monday*, 7(6).

[http://www.firstmonday.dk/issues/issue7\\_6/krishnamurthy/index.html](http://www.firstmonday.dk/issues/issue7_6/krishnamurthy/index.html)

[161] **Laffitte; Trégouet; Cabanel** (1999). Proposition de loi numéro 495. Senate of the Republic of France.

<http://www.senat.fr/consult/loglibre/texteloi.html>

[162] **Laffitte; Trégouet; Cabanel** (2000). Proposition de loi numéro 117. Senate of the Republic of France.

<http://www.senat.fr/consult/loglibre/texteloi.html>

[163] **Lamport, L.** (1994). *LaTeX user's guide and reference manual* (2nd edition). Addison Welsley, Reading, Mass.

[164] **Lancashire, D.** (2001, December). "Code, culture and cash. The fading altruism of open source development". *First Monday* , 6(12).

[http://www.firstmonday.dk/issues/issue6\\_12/lancashire/index.html](http://www.firstmonday.dk/issues/issue6_12/lancashire/index.html)

[165] **Lehman, M. M.; Ramil, J. F; Wernick, P. D.** (1997, November). "Metrics and laws of software evolution. The nineties view". In: *Proceedings of the 4th International Symposium on Software Metrics* .

<http://www.ece.utexas.edu/~perry/work/papers/feast1.pdf>

[166] **Leiner, B. M.; Cerf, V. G.; Kahn, R. E.; Clark, D. D.; Kleinrock, L.; Lynch, D. C.; Postel, J.; Roberts, L. G.; Wolff, S.** (1997). "A brief history of the Internet". In: *Communications of the ACM* .

<http://www.isoc.org/internet/history/brief.shtml>

[167] **Netcraft Ltd. August 2003 Web Server Survey, 2003.**

[http://news.netcraft.com/archives/2003/08/01/august\\_2003\\_web\\_server\\_survey.html](http://news.netcraft.com/archives/2003/08/01/august_2003_web_server_survey.html)

[168] **Lucovsky, M.** (2000). "From NT OS/2 to Windows 2000 and beyond. A software-engineering odyssey".

[http://www.usenix.org/events/usenix-win2000/invitedtalks/lucovsky\\_html/](http://www.usenix.org/events/usenix-win2000/invitedtalks/lucovsky_html/) >

[169] **McGraw, G.** "Building secure software: how to avoid security problems the right way". Cited by: David A. Wheeler in <http://www.dwheeler.com/sloc/>

[170] **McKusick, M. K.** (1999). "Twenty years of Berkeley Unix. From ATT owned to freely redistributable". In: DiBona *et al.* [108].

<http://www.oreilly.com/catalog/opensources/>

[171] **SUN Microsystems** (2000). "Sun microsystems announces availability of StarOffice source code on OpenOffice.org".

[http://www.collab.net/news/press/2000/openoffice\\_live.html](http://www.collab.net/news/press/2000/openoffice_live.html)

[172] **Mockus, A.; Fielding, R. T.; Herbsleb, J. D.** (2000, June). "A case study of open source software development: the Apache server". In: *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)* , pages 263272. Limerick, Ireland ACM Press.

[173] **Molenaar, B.** "What is the context of charityware?".

<http://www.moolenaar.net/Charityware.html>

[174] **MIT Open Course Ware.**

<http://ocw.mit.edu>

[175] **Nagel, L. W.** (1996, september). "The life of SPICE". In: *1996 Bipolar Circuits and Technology Meeting* . Minneapolis, MN, US

<http://www.icsl.ucla.edu/aagroup/Life%20of%20SPICE.html>

[176] **Narduzzo, A.; Rossi, A.** (2003, May). "Modularity in action: GNU/Linux and free/open

source software development model unleashed".

<http://opensource.mit.edu/papers/narduzzorossi.pdf>

[177] **Newman, N.** (1999). "The origins and future of open source software".

<http://www.netaction.org/opensrc/future/>

[178] **Nupedia.**



<http://www.nupedia.com>

[179] **Villanueva Núñez, E.** (2002, April). "Letter to Microsoft Peru".

<http://www.gnu.org.pe/rescon.html>

[180] **Danish Board of Technology** (2002, October). "Open-source software in e-Government, analysis and recommendations drawn up by a working group under the danish board of technology. Technical report".

[181] **Open Source Initiative.** "Open source licenses".

<http://www.opensource.org/licenses/index.html>

[182] **Pareto, W.** (1896). "Course of Political Economy". Lausanne.

[183] **Perens, P.; The Open Source Initiative** (1998). "The open source definition".  
[http://www.opensource.org/docs/definition\\_plain.html](http://www.opensource.org/docs/definition_plain.html)

[184] **GNU Peru.** "Proyectos ley de software libre en la Administración pública del Gobierno peruano, Congreso de la República".

<http://www.gnu.org.pe/proleyap.html>

[185] **Pinheiro, P.** (1999, December). Proposição pl-2269/1999: Dispõe sobre a utilização de programas abertos pelos entes de direito público e de direito privado sob controle acionário da administração pública. Câmara dos Deputados do Brasil.

[http://www.camara.gov.br/Internet/sileg/Prop\\_Detalhe.asp?id=17879](http://www.camara.gov.br/Internet/sileg/Prop_Detalhe.asp?id=17879)

<http://www.fenadados.org.br/software.htm>

[186] **Pranevich, J.** (2003). "The wonderful world of Linux 2.6".

<http://www.kniggit.net/wwol26.html>

[187] **The Debian Project.** "Debian developer map".

<http://www.debian.org/devel/developers.loc>

[188] **Puigcercós Boixassa, J.** (2002). Draft Bill on Measures for Implementing Free Software in Public Administration.

[http://www.congreso.es/public\\_oficiales/L7/CONG/BOCG/B/B\\_244-01.PDF](http://www.congreso.es/public_oficiales/L7/CONG/BOCG/B/B_244-01.PDF)

[189] **Quittner, J.; Slatalla, M.** (1998). *Speeding the net: the inside story of Netscape and how it challenged Microsoft*. Atlantic Monthly Pr.

[190] **Rasch, C.** "A brief history of free/open source software movement".

<http://www.openknowledge.org/writing/open-source/scb/brief-open-source-history.html>

[191] **Rasch, C.** (2001, May). "The Wall Street performer protocol. Using software completion bonds to fund open source software development". *First Monday*, 6(6).

[192] **Raymond, E. R.** (2001, January). *The cathedral and the bazaar. Musings on Linux and open source by an accidental revolutionary*. O'Reilly & Associates (<http://www.ora.com>).

<http://catb.org/~esr/writings/cathedral-bazaar/>

[193] **Reis, C R.; De Mattos Fortes, R. P.** (2002, February). "An overview of the software engineering process and tools in the Mozilla Project".

<http://opensource.mit.edu/papers/reismozilla.pdf>

[194] **Rideau, F. R.** (2000). "Patents are an economic absurdity".

<http://fare.tunes.org/articles/patents.html>

[195] **Roberts, L.** (1978, November). "The evolution of packet switching". *Proceedings of the IEEE*, (66).

[196] **Robles, G.; González Barahona, J. M.; Centeno González, J.; Matellán Oliveira, V.; Rodero Merino, L.** (2003, May). "Studying the evolution of libre software projects using publicly available data". In: *Proceedings of the 3rd Workshop on Open Source Software Engineering at the 25th International Conference on Software Engineering* . Portland, US.

[197] **Robles, G.; Scheider, H.; Tretkowski, I.; Weber, N.** (2001): "Who is doing it? Knowing more about libre software developers".

<http://widi.berlios.de/paper/study.pdf>

[198] **Rochkind, M.** (1986, May). "Interview with Dick Haight". *Unix Review* .

[199] **Scacchi, W.** (2003). "Understanding open source software evolution. Applying, breaking and rethinking the laws of software evolution".

<http://www.ics.uci.edu/~wscacchi/Papers/New/Understanding-OSS-Evolution.pdf>

[200] **Schneier, B.** (2000). "Software complexity and security".

<http://www.counterpane.com/crypto-gram-0003.html>

[201] **Smoogen, S. J.** "The truth behind Red Hat names".

[http://www.smoogespace.com/documents/behind\\_the\\_names.html](http://www.smoogespace.com/documents/behind_the_names.html)

[202] **Haggen So.** "Comparison of free/open source hosting (FOSPhost) sites available for hosting projects externally from project owners".

<http://www.ibiblio.org/fosphost/exhost.htm>

[203] **Stallman, R.** "GNU coding standards".

<http://www.gnu.org/prep/standards.html>

[204] **Stallman, R.** "Why *free software* is better than *open source* ".

<http://www.fsf.org/philosophy/free-software-for-freedom.html>

[205] **Stallman, R.** (1998). "Copyleft: pragmatic idealism".

<http://www.gnu.org/philosophy/pragmatic.html>

[206] **Stallman, R.** (1998). "Why *free software* is better than *open source* ".

<http://www.gnu.org/philosophy/free-software-for-freedom.html>

[207] **Stallman, R.** (1998). "Why software should not have owners".

<http://www.gnu.org/philosophy/why-free.html>

[208] **Stallman, R.** "The GNU Project". In: DiBona *et al.* [108].

<http://www.fsf.org/gnu/thegnuproject.html>

[209] **Stallman, R.** (1999, June). "On free hardware". *Linux Today* .

[http://features.linuxtoday.com/news\\_story.php3?ltsn=1999-06-22-005-05-NW-LF](http://features.linuxtoday.com/news_story.php3?ltsn=1999-06-22-005-05-NW-LF)

[210] **Stallman, R.** (2001). "The free universal encyclopedia and learning resource".

<http://www.gnu.org/encyclopedia/free-encyclopedia.html>

[211] **Stallman, R.** (2002). *Free software, free society. Selected essays of Richard M. Stallman* . Joshua Gay.

[212] **Stallman, R.** (2003). "Some confusing or loaded words and phrases that are worth avoiding".

<http://www.gnu.org/philosophy/words-to-avoid.html>

[213] **Stoltz, M.** (1999). "The case for government promotion of open source software".

<http://www.netaction.org/opensrc/oss-report.html>

[214] **Tanenbaum, A.; Torvalds, L.** (1999). "The Tanenbaum-Torvalds debate".

<http://www.oreilly.com/catalog/opensources/book/appa.html>

[215] **The Open Source Initiative.** "The open source definition".

[http://www.opensource.org/docs/definition\\_plain.html](http://www.opensource.org/docs/definition_plain.html)

[216] **Tiemann, M.** "Future of Cygnus Solutions. An entrepreneur's account". In: DiBona *et al.* [108].

<http://www.oreilly.com/catalog/opensources/book/tiemans.html>

[217] **Torvalds, L; Diamond, D.** (2001). *Just for fun: the story of an accidental revolutionary*. Texere.

[218] **Linus Torvalds, Hamano, J. C.; Ericsson, A.** "Git manual page".

<http://www.kernel.org/pub/software/scm/git/docs/>

[219] **Tuomi, I.** (2002). "Evolution of the Linux credits file: methodological challenges and reference data for open source research".

<http://www.jrc.es/~tuomiil/articles/EvolutionOfTheLinuxCreditsFile.pdf>

[220] **Various authors.** "Open letter to WIPO".

<http://www.cptech.org/ip/wipo/kamil-idris-7july2003.pdf>

[221] **Vigo i Sallent, P.; Benach i Pascual, E.; Huguet i Biosca, J.** (2002, May). Proposició de llei de programari lliure en el marc de l'Administració pública de Catalunya.

<http://www.parlament-cat.es/pdf/06b296.pdf>

[http://www.hispalinux.es/modules.php?  
op=modload&name=Sections&file=index&req=viewarticle&artid=49](http://www.hispalinux.es/modules.php?op=modload&name=Sections&file=index&req=viewarticle&artid=49)

[222] **Villanueva Núñez, E.** (2001, December). Free software project bill, number 1609.

<http://www.gnu.org.pe/proley1.html>

[223] **Villanueva Núñez, E.; Rodrich Ackerman, J.** (2002, April). Bill on the use of free software by the Public Administration, number 2485.

<http://www.gnu.org.pe/proley4.html>

[224] **W3C** (2000). *Extensible markup language (xml) 1.0* (2nd edition).

[225] **Walsh, N.; Muellner, L.; Stayton, B.** (2002). *DocBook: the definitive guide*. O'Reilly.  
<http://docbook.org/tdg/en/html/docbook.html>

[226] **Welke, L; Johnson, L.** (1998). How the ICP Directory began.

<http://www.softwarehistory.org/history/Welke1.html>

[227] **Wheeler, D. A.** (2000, July). "Estimating Linux's size".

<http://www.dwheeler.com/sloc>

[228] **Wheeler, D. A.** (2001, June). "More than a gigabuck: estimating GNU/Linux's".

<http://www.dwheeler.com/sloc>

[229] **Wiesstein, E.** "Concise encyclopedia of mathematics".

<http://mathworld.wolfram.com/>

[230] **Wikipedia.** "Gini coefficient".

[http://www.wikipedia.org/wiki/Gini\\_coefficient](http://www.wikipedia.org/wiki/Gini_coefficient)

[231] **Wikipedia**. "Lorenz curve".

[http://www.wikipedia.org/wiki/Lorenz\\_curve](http://www.wikipedia.org/wiki/Lorenz_curve)

[232] **Wikipedia**. "Pareto".

<http://www.wikipedia.org/wiki/Pareto>

[233] **Wikipedia**. "TeX".

<http://www.wikipedia.org/wiki/TeX>

[234] **Wilson, B.** "Netscape Navigator".

<http://www.blooberry.com/indexdot/history/netscape.htm>

[235] **Computer World** (2000). "Salary survey 2000".

<http://www.computerworld.com/cwi/careers/surveysandreports>

[236] **Young, R.** (1999). "Giving it away. how Red Hat software stumbled across a new economic model and helped improve an industry".

<http://www.oreilly.com/catalog/opensources/book/young.html>

[237] **Zawinsky, J. W.** (1999). "Resignation and postmortem".

<http://www.jwz.org/gruntle/nomo.html>