Introduzione al software *free/open source* (FOSS)

Ottobre 2005

Alberto Cammozzo (mmzz @ stat.unipd.it)

revisione 1

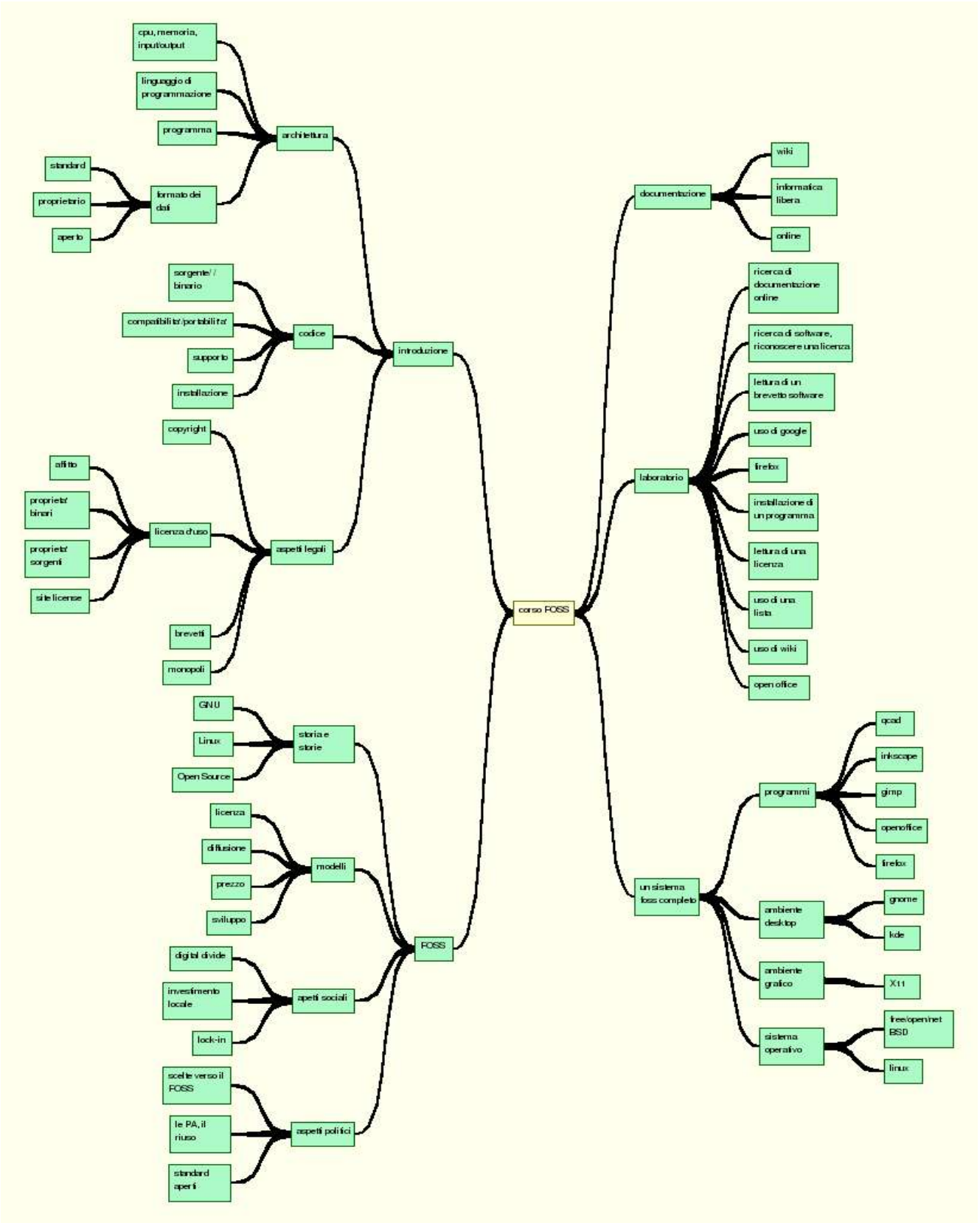
1 Note sul corso

Questa e' la documentazione del corso su *free software e open source.* Non sostituisce il corso, non e' uno strumento indispensabile alla comprensione dello stesso, ma una traccia per il suo svolgimento, un poco come fosse una sua radiografia: puo' essere consultato per comprendere a che punto si e', per rivedere gli aspetti essenziali, per ripassare punti poco chiari, o come base per porre delle domande.

Come e' possibile vedere dalla mappa (pag 2), il corso si articola in 5 moduli:

1. una introduzione teorica al **calcolatore** e in particolare al **software**, o meglio agli aspetti piu' importanti per la comprensione del software libero. Per chi gia' conosce il computer un mezzo per ripassare o riorganizzare le proprie conoscenze, per chi non lo conosce per districarsi nella terminologia. Verranno spiegati tutti i termini tecnici necessari.
2. Una introduzione al ***free software*, e all'*open source***, e sui vari fenomeni associati ad essi e varie considerazioni sulle conseguenze della loro diffusione. E' il cuore del corso.
3. Un **viaggio in un sistema *libero*** attraverso il quale, possibilmente mettendo le mani sulle tastiere, vedere concretamente in cosa consistono i vari elementi che compongono un sistema libero. Non solo i programmi usati dagli utenti, ma anche tutti gli altri elementi, piu' o meno nascosti.
4. Una introduzione alla **documentazione**: senza la documentazione e' difficile usare i programmi. Il mondo del *free software* si avvale anche della diffusione libera della conoscenza, per cui la documentazione dei programmi liberi ha modalita' e caratteristiche particolari.
5. Alcune **esercitazioni**, da svolgere al calcolatore, non solo su alcuni piu' diffusi programmi liberi, ma su alcuni aspetti specifici, quali la ricerca di programmi, le licenze, eccetera.

Il percorso effettivo del corso non seguira' l'ordine dei moduli, ma li percorrera' trasversalmente. Tutto il materiale del corso, se non gia' distribuito, sara' disponibile online o richiedendolo al docente. Potra' essere liberamente riprodotto e distribuito nella sua completezza, purche' non vi venga apportata alcuna modifica che non sia stata concordata con l'autore.



2 Richiami sui calcolatori elettronici

2.1 Questa prima parte affronta schematicamente alcuni elementi fondamentali dei calcolatori elettronici con particolare riguardo alla terminologia, mettendo in evidenza gli aspetti rilevanti

per il FOSS.

2.2 Architettura, hardware

2.2.1 Cosa si intede per architettura riferendosi a un calcolatore elettronico? Come nel caso di un edificio, si tratta degli elementi caratterizzanti, costitutivi, quelli che ne definiscono le differenze con altri edifici. Tutti gli edifici hanno degli elementi comuni (pareti, tetto, vani, aperture). La destinazione di un edificio ne caratterizza alcuni aspetti: l'uso civile (case), industriale (capannoni), eccetera. Altri aspetti saranno invece determinati dalle scelte di chi li ha costruiti.

2.2.2 Anche i calcolatori elettronici hanno degli elementi comuni, senza i quali non si possono definire tali (cpu, memoria, input e output), altri determinati dal loro uso: se sono dei server, dei PC domestici, se devono essere usati dentro a un telefonino, un'astronave o una lavatrice. Altre scelte sono determinate da chi li costruisce, in base alle funzionalita' che ha in mente e alle tecnologie che ha a disposizione. L'insieme di queste scelte deteminano l'*architettura*.

2.2.3 Parliamo ad esempio di *architettura Intel* per definire i calcolatori che fanno uso di componenti chiave (in particolare la CPU) prodotti dalla ditta Intel. Tutti i calcolatori con un'architettura comune condivideranno tra loro determinate caratteristiche, e si diranno tra loro **compatibili** se l'insieme di caratteristiche comuni e' sufficientemente ampio.

2.2.4 Il termine architettura si applica prevalentemente alle scelte di componenti fisici, detti *hardware*. Si contrappone al termine *ambiente* che si riferisce prevalentemente agli aspetti *software*, relativi ai componenti non fisici.

2.2.5 Esempi di architetture sono: Intel x86, Apple/PowerPC. Esempi di ambienti: Windows, Apple, ambiente Linux, ambiente Symbian.

2.3 Cpu, memoria, programmi, input/output

2.3.1 La *CPU* (*central processing unit* – unita' centrale di elaborazione) e' l'elemento che **esegue** le istruzioni che sono contenute nella **memoria centrale.** I *programmi* sono collezioni di istruzioni ordinate secondo una rigorosa logica, come le ricette di cucina. Ogni particolare CPU esegue un determinato insieme di istruzioni elementari. Versioni diverse delle stesse CPU possono condividere lo stesso insieme (o sottoinsieme) di istruzioni, percio' si dicono tra loro **compatibili**.

2.3.2 Esempi di CPU sono: Intel Pentium, Motorola PowerPC.

2.3.3 La ***memoria*** *centrale* e' il luogo ove risiedono i programmi che possono essere eseguiti dalla CPU e dove risiedono i dati usati dai programmi. Non va confusa con la *memoria di massa*, che e' il luogo ove dati e i programmi risiedono in modo permenente in attesa di esser *caricati in memoria* (centrale) per essere eseguiti. E' importante ricordare che la memoria centrale e' volatile (cioe' svanisce se viene spento il computer) mentre quella di massa e' permanente, cioe' rimane finche' i dati non vengono cancellati, o comunque per un lungo periodo di tempo.

2.3.4 Esempi di memoria di massa: floppy, chiave USB, CD-ROM, DVD, hard disk. Di norma la memoria centrale risiede stabilmente all'interno del calcolatore, le memorie di

massa possono essere rimosse piu' o meno facilmente assieme ai dati che contengono.

2.3.5 Semplificando possimo dire che i *dispositivi di* ***input/output*** identificano le unita' periferiche (per contrapporle all'unita' centrale, che ospita memoria e CPU): per interagire con le periferiche e' necessario che il calcolatore ospiti dei circuiti appositi, detti appunto dispositivi di input/output, brevemente *I/O*.

2.3.6 Esempi di periferiche sono: la tastiera, il mouse, il video, lo scanner.

2.3.7 Esempi di dispositivi di I/O sono: la scheda video, la scheda di rete. In generale dietro a un qualsiasi connettore presente sull'unita' centrale e' presente un dispositivo di I/O.

2.3.8 I programmi specializzati che servono a guidare i dispositivi di I/O si chiamano *driver* ("guidatori, conduttori"). Fanno parte del sistema operativo.

2.4 Istruzioni e programmi

2.4.1 Ogni CPU esegue delle istruzioni *espresse come numeri*: ad esempio un numero significa l'istruzione: “esegui la somma dei prossimi due numeri”, un'altra istruzione, espressa da un diverso numero, significhera' “esegui l'istruzione presente all'indirizzo di memoria X”.

2.4.2 I numeri che codificano le istruzioni sono espressi in *base due*, cioe' possono essere rappresentati solo dai numeri 0 e 1, detti *bit*, che sta per *binary digit*, o numero binario. Questo e' dovuto al fatto che la CPU e' un dispositivo elettronico che, descritto in modo estrememente semplificato, e' composto da interruttori che fanno passare o non fanno passare corrente elettrica in un circuito. La presenza di corrente e' rappresentata da un 1, l'assenza da uno 0.

2.4.3 Le istruzioni che possono essere eseguite direttamente dalla CPU sono dette *elementari*, cioe' si riferiscono alle operazioni piu' semplici che possano essere eseguite da un calcolatore. Sono ad esempio somme, sottrazioni, confronti tra numeri.

2.4.4 Collezioni piu' o meno complesse di istruzioni elementari, che ricevono dati in input e che forniscono altri dati in output, che operano scelte in base ai dati in ingresso o allo stato in cui si trovano, si chiamano *programmi*. I programmi vengono scritti dai programmatori.

2.5 Sistemi operativi, ambiente, software

2.5.1 Viene denominato *sistema operativo* una collezione di programmi che rendono utilizzabile un calcolatore per una serie di scopi generali quali: leggere e scrivere un file, caricare in memoria centrale i programmi e i dati presenti sulle memorie di massa, organizzare i dati nelle memorie di massa, effettuare stampe, collegarsi alla rete, eccetera.

2.5.2 Esempi di sistema operativo sono: GNU/Linux, Microsoft Windows, Digital VMS, Symbian OS, FreeBSD.

2.5.3 Il sistema operativo si divide in una *interfaccia*, che e' cio' con cui gli utenti entrano in relazione, composta dai principali comandi e l'aspetto esteriore, e il nucleo (*kernel*) che ha il vero controllo dell'hardware e dei processi.

2.5.4 Il sistema operativo (s.o.) ha grande importanza nel definire le possibilita' di *compatibilita':* le caratterische dei s.o. sono talmente diverse tra loro da rendere i programmi che possono essere caricati ed eseguiti in un dato s.o. incompatibili, cioe' non

eseguibili negli altri, a meno di eccezioni. Questo accade anche se l'architettura della CPU e' la stessa.

2.5.5 Come si parla di *architettura* per le CPU, cosi' nel caso dei sistemi operativi si usa il termine *ambiente,* proprio per indicare qualcosa all'interno del quale gli altri programmi andranno ospitati per essere eseguiti.

2.5.6 Ad esempio un programma pensato per essere usato in ambiente GNU/Linux su *architettura* Intel non potra' essere usato in *ambiente* Microsoft sulla stessa architettura.

2.6 il codice: linguaggio macchina e linguaggio di alto livello.

2.6.1 Un determinato insieme di istruzioni per calcolatore si chiama *linguaggio*, ed e' costituito da parole chiave o numeri che hanno un determinato, preciso, significato.

2.6.2 L'insieme delle istruzioni che una CPU puo' eseguire direttamente si chiama *linguaggio macchina*. Fornire al computer direttamente le istruzioni elementari a lui comprensibili si dice *programmare in linguaggio macchina.*

2.6.3 Per il fatto di dover scrivere le istruzioni sotto forma di numeri e per il fatto che quelle istruzioni sono estrememante elementari la programmazione in linguaggio macchina e' pratica rara.

2.6.4 Ovunque possibile i programmatori usano invece linguaggi di alto livello, cioe' linguaggi che fanno uso di parole e non numeri per dare istruzioni, e in cui ogni parola raggruppa istruzioni piu' complesse di quelle elementari.

2.6.5 Esempi di parole chiave di un linguaggio di alto livello sono *fai* questo *finche non succede* quest'altro, oppure *se* questo *allora* ...; solitamente le parole chiave sono espresse in lingua inglese (`do...while`, `if...then`).

2.6.6 Esempi di linguaggi di alto livello sono: BASIC, C, Perl, PHP.

2.6.7 Quando si dice *linguaggio di programmazione* solitamente si intende linguaggio di alto livello. Un sinonimo molto usato di *programma* e' *codice*. Ad esempio si dice: codice C, programma in C, programma in BASIC, codice PHP, codice macchina...

2.6.8 I programmi scritti in un linguaggio di alto livello devono essere *tradotti* in linguaggio macchina per essere eseguiti dalla CPU. A questo scopo vengono usati altri programmi chiamati *compilatori* oppure *interpreti*. Ogni linguaggio di alto livello ha il proprio compilatore.

2.6.9 Esempi di traduttori sono: compilatore C, interprete BASIC, interprete PHP.

2.6.10 Il testo dei programmi scritti direttamente da un programmatore in un linguaggio di alto livello o piu' raramente in linguaggio macchina si chiamano *codice sorgente*, e per opposto si chiama *codice binario* o *codice eseguibile* quello tradotto dal compilatore. Riassumendo: a partire dal *codice sorgente* in un *linguaggio di alto livello*, scritto da un programmatore, verra' generato dal *compilatore* del *codice binario* scritto in *linguaggio macchina*.

2.7 formato dei dati

2.7.1 Anche i dati che devono essere elaborati o i risultati forniti dai programmi,

analogamente alle istruzioni, sono rappresentati in modo codificato. Sia perche' sono, come le istruzioni, espressi in forma binaria, ma soprattutto perche' il calcolatore elabora solo dati elementari espressi come numeri, percio' dati di natura diversa dai numeri dovranno comunque essere tradotti in forma numerica.

2.7.2 Esempi di dati complessi sono: un file di testo, un'immagine, una registrazione audio, un filmato video, un disegno di un progetto, un diagramma.

2.7.3 Il modo, le regole, in cui dati complessi vengono codificati per essere archiviati si chiama *formato*.

2.7.4 Esempi di formato: “formato word”: il formato in cui il programma Microsoft Word codifica i propri file di testo, “formato jpeg”: un formato grafico standard.

2.7.5 formati standard

2.7.5.1 I formati possono essere standard, cioe' rispondere a norme, spesso internazionali, definite in modo rigoroso da enti preposti allo scopo.

2.7.5.2 Esempi di formati standard: il formato XML e' un noto standard per codificare diverse forme di dati complessi; il formato ASCII e' un antico standard per la codifica di semplici file di testo; il formato HTML e' un formato standard per la descrizione delle pagine Web.

2.7.5.3 E' bene notare che gli enti che emettono standard sono composti spesso da molti membri, percio' sono di solito piuttosto lenti nell'emettere o aggiornare gli standard.

2.7.6 formati proprietari

2.7.6.1 I formati possono essere, all'opposto di quelli standard, proprietari, cioe' definiti in modo riservato, o segreto, e in certi casi coperto da brevetto, da parte del produttore del programma che elabora i dati in quel formato.

2.7.6.2 Anche se il formato non e' divulgato, e' talvolta possibile ricostruire il formato proprietario analizzando un gran numero di file di quel formato. Questo procedimento si chiama *reverse engineering* ovvero “ingegnerizzazione alla rovescia”. In certi casi questo procedimento e' illegale.

2.7.6.3 Esempi di formato proprietario: Microsoft Word 95, Microsoft Excel, Apple “mov”.

2.7.7 formati aperti e interoperabilita'

2.7.7.1 Alcuni produttori di programmi, pur non aderendo a standard definiti da enti, rendono pubblici i formati dei file generati dai propri programmi, cioe' “aprono” al pubblico i loro formati.

2.7.7.2 Esempi di formati aperti: il formato "PDF", Portable Document Format

2.7.7.3 La possibilita' per due programmi diversi di scambiarsi dati con un formato comune, di qualsiasi tipo viene denominata *interoperabilita'*.

2.8 codice sorgente e binario, portabilita'

2.8.1 Alcuni aspetti di quanto visto nel capitolo precedente sono della massima importanza per comprendere il fenomeno del FOSS, e sono relativi al codice, in particolare se i programmi sono diffusi come codice binario (o eseguibile) o come codice sorgente.

2.8.2 Il *codice sorgente* viene scritto dal programmatore, tradotto in *linguaggio macchina* o *binario* (cioe' in istruzioni elementari) da un compilatore, per essere eseguito dalla CPU.

2.8.3 Una volta tradotto in codice binario, le istruzioni del linguaggio di alto livello vengono sostituite da gruppi di istruzioni elementari, ed e' difficile, talvolta impossibile, effettuare l'operazione opposta, cioe' tradurre da linguaggio macchina in linguaggio di alto livello.

2.8.4 La piu' importante conseguenza di cio' e' che per modificare un programma scritto in un linguaggio di alto livello, e' necessario intervenire sul codice sorgente.

2.8.5 Una seconda conseguenza e' che uno stesso programma, scritto in codice sorgente, potra' essere compilato (cioe' tradotto) per diverse *architetture* (insiemi di istruzioni di CPU). Si dice in questo caso che il programma, sviluppato originariamente per una data architettura, viene *portato* su un'altra architettura.

2.8.6 Ad esempio, Microsoft Office, originariamente scritto per l'architettura Intel, e' stato *portato* da Microsoft sull'architettura Apple. La maggior parte dei programmi scritti per il sistema Unix sono stati portati su Linux.

2.9 Compatibilita'

2.9.1 Diverse CPU hanno diversi *insiemi di istruzioni* elementari. I numeri che rappresentano un'istruzione elementare (ad es. la somma) sulla CPU *Intel* sono diversi dai numeri che rappresentano la stessa istruzione elementare nelle CPU *PowerPC*. Questo significa che le due CPU sono tra loro *incompatibili.*

2.9.2 Se invece diciamo che l'insieme di istruzioni della CPU Intel 8086 e' *compatibile* con quella della CPU Intel Pentium IV, significa che l'insieme delle istruzioni elementari eseguibili dalla prima e' presente anche nella seconda. Benche' il Pentium sia un'evoluzione della stessa *architettura*, non e' vero il contrario: molte delle istruzioni del Pentium non sono presenti sull'8086. Significa che i programmi scritti in codice macchina 8086 possono essere eseguiti su un pentium, ma non viceversa. I programmi 8086 sono *compatibili* con il Pentium, ma viceversa quelli Pentium sono *incompatibili* con quelli 8086.

2.9.3 L'incompatibilita', come vedremo, non riguarda solo gli insiemi di istruzioni delle CPU, ma anche gli *ambienti,* cioe' i *sistemi operativi*. I programmi scritti per un PC con Windows sono incompatibili con Linux: significa che non verranno eseguiti sullo stesso PC con Linux.

2.10 Pacchetti/distribuzione/supporto

2.10.1 I programmi, come i dati, esistono su un qualche *supporto*: anche cartaceo se sono scritti “a mano” dal programmatore, o piu' sovente in archivi elettronici che contengono il codice sorgente, o l'eseguibile (codice binario).

2.10.2 Esempi di supporto: il CD audio e' il supporto della musica, il CD dati e' il supporto

dei file che contiene, il disco fisso e' il supporto dei programmi in esso contenuti, Internet, pur non essendo un supporto in senso proprio, sta divenedo il supporto virtuale per molti programmi.

2.10.3 I programmi pronti per essere utilizzati dall'utente finale, senza necessita' di essere modificati o personalizzati con interventi di un programmatore, si chiamano *pacchetti*. L'idea e' che l'utente apre il pacchetto e ha gia' tutto quello che gli serve, senza dover intervenire nemmeno per integrare il programma con il sistema operativo. Altri programmi, che invece richiedono l'intervento dei programmatori per essere messi in opera, si chiamano *custom*.

2.10.4 Esempi: i programmi dell'INPS per la gestione dei dati delle pensioni, scritti solo per l'INPS, sono programmi *custom*. Microsoft Office e' un *pacchetto* che raccoglie diversi programmi per uso di ufficio. I giochi per le console di videogiochi sono *pacchetti*.

2.11 Installazione

2.11.1 Installare un programma significa trasferirlo da un supporto utile al suo trasporto a un'altro pronto per l'uso. Di solito comporta: 1) il trasferimento sul disco fisso direttamente collegato al calcolatore, e 2) il perfezionamento di operazioni che renderanno il programma utilizzabile in modo integrato con gli altri programmi presenti.

2.11.2 Alcuni programmi, come ad esempio quelli di *webmail* (per la lettura della posta elettronica) presenti sui portali web non richiedono installazione, perche' non risiedono su un supporto dell'utente ma sui server del fornitore del servizio di mail.

2.12 Aspetti legali

2.12.1 Per comprendere il fenomeno del FOSS, vi sono anche alcuni aspetti legali da comprendere, legati alla cosiddetta tutela della cosiddetta *proprieta' intellettuale*.

2.12.2 L'attivita' di programmazione e' considerata un'attivita' di autore, analogamente allo scrivere libri, poesie o canzoni. I programmatori godono quindi dei diritti d'autore nei confronti del codice sorgente di cui sono autori, e cedono alle ditte che li impiegano i *diritti di sfruttamento* del codice da loro scritto, analogamente a quanto accade con un autore di un romanzo nei confronti del suo editore.

2.12.3 Inoltre le *software house* (le ditte che producono software) hanno dei *segreti industriali* che tutelano con vari mezzi. Le violazioni dei *copyright* e di altri diritti scatenano feroci e costosissime battaglie legali.
Ad esempio una battaglia tutt'ora in corso ruota attorno a porzioni del codice sorgente di Linux, e vede opposte le ditte SCO e IBM. Una analoga battaglia sulla proprieta' del codice sorgente del sistema operativo BSD si e' svolta anni fa e ne ha tenuto fermo lo sviluppo per anni.

2.12.4 Copyright

2.12.4.1 Il programmatore e' l'autore del codice che scrive. Come tale e' tutelato dalle leggi sul diritto d'autore, dalla convenzione di Berna, eccetera.

2.12.4.2 Nel diritto statunitense l'autore puo' cedere il © *in toto* alla ditta che lo paga. Per

il diritto europeo mantiene dei diritti, detti morali, che sono inalienabili, che cioe' non potrebbe cedere nemmeno se volesse farlo.

2.12.4.3 I diritti che solitamente un programmatore cede alla software house per la quale lavora sono quelli di *sfruttamento economico* e di poter trarre dei *lavori derivati,* cioe' di prodotti che sono un evoluzione di quello originale.

2.12.5 Licenza d'uso

2.12.5.1 Il produttore di software cede al suo cliente, cioe' all'*utente finale*, un altro sottoinsieme dei diritti: quello di eseguire il programma, di farne un certo numero di copie, di impiegare l'output del programma come gli piace, incluso per ricavare del denaro dal suo impiego, eccetera.

2.12.5.2 Esempio: se un programmatore scrive un programma per la ricerca di pozzi petroliferi e lo vende a una ditta che fa prospezioni, quest'ultima non dovra' una percentuale di quanto ricavato grazie al programma al programmatore.

2.12.5.3 Le licenze solitamente vietano esplicitamente attivita' di *reverse engineering* e di modifica del codice. L'unico modo per ottenere la modifica del funzionamento di un programma e' richiedere al produttore che sia lui ad applicare le modifiche richieste.

2.12.6 Affitto

2.12.6.1 In certi casi il software viene affittato, cioe' puo' essere impiegato dall'utente finche' paga per farlo. Scaduto l'affitto, deve essere cancellato dai supporti dell'utente, o comunque non puo' piu' essere usato.

2.12.6.2 Esempio: il programma statistico SAS si rifiuta di eseguire le istruzioni impartitegli se rileva, in base alla data di sistema e alle date dei file, che la licenza e' scaduta.

2.12.7 Proprieta' del codice eseguibile

2.12.7.1 La licenza consente all'utente che acquista dei programmi nella forma binaria (codice eseguibile) di installarne una o piu' copie di sua proprieta'. Con un cambiamento di *architettura* o di sistema operativo sara' tuttavia necessario riacquistare i programmi per poterli impiegare sui nuovi sistemi.

2.12.7.2 Esempio: La maggior parte dei pacchetti hanno questo tipo di licenza. In certi casi la licenza obbligano l'utente a usare il programma solo sul calcolatore sul quale e' stato installato la prima volta. Il programma non puo' essere *reinstallato* su altre macchine.

2.12.8 Proprieta' sorgenti

2.12.8.1 In certi casi il committente desidera possedere completamente il codice sorgente scritto per lui. Questo avviene quando vuole poter essere del tutto indipendente dal produttore, o se desidera poter intervenire autonomamente sui programmi per modificarli.

2.12.8.2 Esempio: tutti i programmi *custom* scritti *ad-hoc* per la Pubblica

Amministrazione Italiana devono essere ceduti completi di sorgente, di cui la PA diviene proprietaria. Questo implica che la software house non puo' vendere gli stessi programmi ad altri.

2.12.9 Licenze libere

2.12.9.1 All'utente viene concesso l'uso del programma senza limitazioni di scopo e la possibilita' di accedere ai sorgenti, anche per modificarli. Vi sono molte diverse possibilita' su quali diritti e quali obblighi l'utente si assume accettando una di queste licenze.

2.12.10 Site license

2.12.10.1 A volte la facolta' da parte del cliente di effettuare un certo numero di copie non e' limitata dal loro numero, ma dal fatto che vengano effettuate all'interno di una struttura delimitata, ad es. un'azienda.

2.12.10.2 Esempio: una licenza campus e' una site license estesa a tutta un'universita'.

2.12.11 Brevetti

2.12.11.1 Un modo molto controverso delle software house di tutelare i propri investimenti e' quella di brevettare dei metodi associati ai programmi che scrivono. Non si tratta di tutelare, come per il diritto d'autore (copyright), solo quelle particolari forma di codice, ma tutta la procedura eseguita dal codice. Vigono criteri diversi per la brevettabilita' negli USA e nell'UE. La brevettabilita' del software e' oggetto di forti critiche.

2.12.11.2 Ad esempio: l'acquisto *on-line* con un singolo click.

2.12.12 Monopoli

2.12.12.1 Problemi legali nascono anche quando vi sono condizioni di monopolio nel mercato.

2.12.12.2 Ad esempio Microsoft e' stata accusata di approfittare della propria posizione preminente nel mercato dei Sistemi Operativi per conquistare altri mercati, come quello dei browser o dei riproduttori multimediali.

3 FOSS: *Free/Open Source Software*

Riferendosi al *free software* si parla di *fenomeno* o anche di *rivoluzione.* La sua esplosione repentina dopo una lunga maturazione ha le caratteristiche di una rivoluzione e ha sorpreso tutti, inclusi quelli che l'anno promossa. La caratteristica di fenomeno nasce dal fatto che non sia descrivibile solo in termini semplici di una sola disciplina, quale quella dell'ingegneria del software: sul free software e sull'open source proliferano studi articolati in tutti i campi: informatica, economia, sociologia, psicologia, politica. Parte da premesse molto semplici ma ha conseguenze e ripercussioni estremente complesse

3.1 Premesse

3.1.1 Sorgente/ eseguibile

3.1.1.1 Tutto il fenomeno FOSS ruota attorno al semplice concetto che il codice sorgente dei programmi deve essere liberamente disponibile a chi lo usa.

3.1.1.2 E' bene premettere che sui motivi per cui questo deve avvenire, e sulle precise modalita' vi sono diversi pareri. Qualcuno sottolinea piu' gli aspetti ideali (liberta', equita'), altri quelli economici (stimolo all'industria del software), altri ancora quelli tecnici (il software libero e' tecnicamente migliore).

3.1.1.3 Avere la disponibilita' del codice sorgente significa essere in grado di *modificare* il programma oltre che di eseguirlo, mentre quella del codice *eseguibile* consente solo di eseguirlo senza modificarlo.

3.1.2 Libero/gratuito

3.1.2.1 E' bene subito chiarire anche che il *software libero*, anche se spesso e' gratuito, non necessariamente lo e'. E gran parte del software gratuito circolante non e' software libero.

3.1.2.2 Il termine *software libero* si contrappone a *software proprietario*.

3.1.2.3 Nella lingua inglese la parola gratuito e libero coincidono con il termine *free,* e questo ha creato non pochi equivoci, tanto che il termine *open source* e' stato coniato proprio per evitare confusioni.

3.1.2.4 Non e' nemmeno detto che i programmi free/open siano disponibili in Internet e liberamente scaricabili, anche se cio' accade molto spesso.

3.2 Storia e storie

3.2.1 Nascita del mercato del software

3.2.1.1 I calcolatori elettronici si diffondono commercialmente negli anni '60 del secolo scorso. Si trattava di grossi calcolatori (*mainframe*) in cui il software veniva venduto assieme all'hardware.

3.2.1.2 I calcolatori erano impiegati presso industrie, esercito, universita'. I programmi, specie in ambito accademico, venivano scambiati tra i (pochi) possessori dello stesso

tipo di calcolatori *compatibili* tra loro. Gli utenti dei calcolatori erano programmatori esperti in grado di modificare i programmi, adattarli alle proprie esigenze e correggerli in caso ospitassero errori.

3.2.1.3 Questi utenti/programmatori, specie in ambito accademico, costituivano delle *comunita'* di persone in contatto tra loro con la consuetudine di scambiarsi informazioni, programmi (in codice sorgente) e documentazione.

3.2.1.4 Negli anni '70 vennero prodotti i minicalcolatori e poi i personal computer (anni '80): invece che pochi calcolatori per molte persone, un calcolatore per ogni lavoratore. Il mercato dell'hardware si amplio' abbastanza da giustificare un *mercato del software indipendente*. Chi costruiva il calcolatore non forniva il software, e viceversa. Nacquero ditte che fornivano esclusivamente software, ad esempio sistemi operativi o programmi "applicativi" come word processor, fogli elettronici.

3.2.1.5 Gli utenti per mini e soprattutto personal computer non costituivano comunita' di utenti/programmatori: si trattava spesso di persone senza conoscenze di programmazione che usavano il calcolatore per lavoro. Pochi sentivano l'esigenza di modificare i programmi e di adattarli alle proprie esigenze. Gli utenti piu' esigenti potevano richiedere le modifiche al produttore dei programmi.

3.2.1.6 Le ditte che producevano software non diffondevano il codice sorgente, ma solo quello eseguibile. Sia perche' gli utenti non sentivano il bisogno di averlo, ma soprattutto per proteggersi dalla concorrenza che avrebbe potuto copiarlo. Il codice sorgente era un loro patrimonio industriale da proteggere al riparo dalla concorrenza, ma anche lontano dagli occhi degli utenti.

3.2.2 Prima storia: GNU

3.2.2.1 Chi si sentiva piu' penalizzato dalla segretezza del codice sorgente erano quegli utenti/programmatori che prima potevano modificare, adattare, correggere, migliorare i programmi scritti anche da altri e che facevano parte di comunita' di appassionati. Tra questi Richard Stallman, ricercatore presso l'MIT di Boston, vede questa limitazione come un fatto grave, una pesante limitazione alla liberta' personale, un modo per "*impedire di fare cose utili*", e vi si ribella.

3.2.2.2 Nel 1983 avvia un progetto, che chiama GNU (Gnu Non e' Unix), con lo scopo di rifare da zero un sistema Unix (sistema operativo allora molto diffuso in ambito accademico) rendendone disponibili i sorgenti.

3.2.2.3 I fondamenti ideologici del progetto GNU sono riassunti dalle cosiddette *4 liberta'*:
0: Libertà di eseguire il programma, per qualsiasi scopo.
1: Libertà di studiare come funziona il programma, e adattarlo alle proprie necessità.
2: Libertà di ridistribuire le copie in modo da aiutare il prossimo.
3: Libertà di migliorare il programma, e distribuirne pubblicamente i miglioramenti, in modo tale che tutta la comunità ne tragga beneficio.

3.2.2.4 Per proteggere la liberta' del codice sorgente del progetto GNU viene elaborata una ingegnosa licenza, che basandosi sulle leggi che proteggono i diritti dell'autore (copyright) impedisce che qualcuno lo "chiuda" o ne sviluppi programmi non liberi.

Questo meccanismo viene chiamato *copyleft*.

3.2.2.5 Come tutti i sistemi operativi, anche Unix (e di conseguenza il progetto GNU) e' composto da un nucleo e da un parte piu' esterna con i comandi e l'interfaccia utente. Il progetto sviluppa fino agli anni '90 una versione libera di buona parte dei programmi esterni, in particolare un ottimo compilatore C (GCC), ma incontra delle difficolta' nella costruzione del nucleo (*kernel*) del sistema operativo.

3.2.2.6 Per le necessita' di finanziare e sostenere il progetto nasce la Free Software Foundation (FSF), che assumera' nel tempo un ruolo importante.

3.2.2.7 L'aspetto ideale del progetto GNU/FSF e' efficacemente riassunto da queste citazioni di Richard Stallman dal film Revolution OS:
- "il software proprietario divide e soggioga gli utenti";
- “un sistema operativo libero dara' agli utenti la liberta' mentre usano i loro computer”;
- “La liberta' di cooperare con altri utenti, di avere una comunita', e' importante per la qualita' della vita, [...] per avere una buona societa' nella quale vivere, ed e' [...] ancora piu' importante che avere software potente e affidabile”.

3.2.3 Seconda storia: Linux

3.2.3.1 Indipendentemente dal progetto GNU, nel 1991 uno studente finlandese Linus Torvalds, spinto dalla mancanza del denaro necessario ad acquistare un sistema Unix per i suoi studi, decide di farsene uno. Avendo a disposizione gia' tutti gli strumenti del progetto GNU, gli resta da fare "solo" il nucleo (*kernel*).

3.2.3.2 Il suo progetto di *kernel* e' piu' tradizionale, meno ambizioso ed innovativo di quello di Stallman, e in breve tempo e' in grado di costruirne una prima versione, rudimentale ma funzionante.

3.2.3.3 Annuncia il suo progetto (battezzato *Linux*) in Internet, e riceve un entusiastica accoglienza. Molte persone si dichiarano disposte a collaborare allo sviluppo del nuovo nucleo: dalla collaborazione via Internet di programmatori sparsi per il globo nasce un nuovo sistema operativoe e un inedito modo di sviluppare codice di qualita'.

3.2.3.4 Il kernel Linux unito ai programmi del progetto GNU formano un sistema operativo simile a Unix cosi' efficace e stabile da soppiantare Unix stesso in quasi tutti i campi. Il *kernel* di Linus Torvalds si amplia e diviene sempre piu' affidabile, completando il progetto GNU di Stallman. E' nato il sistema GNU/Linux.

3.2.4 Terza storia: Open Source

3.2.4.1 La diffusione del free software negli anni '80 incuriosisce sempre di piu' l'industria del software, forse piu' interessata dalla gratuita' di prodotti, piu' che dagli aspetti di liberta'.

3.2.4.2 Il successo tecnico e la continua diffusione di GNU/Linux diventa motivo di preoccupazione per alcuni, crescente interesse per altri. L'opportunita' di affari collegati allo sviluppo, assistenza e diffusione di software libero si concretizza nell'attivita' di alcune ditte che si specializzano in vendita e assistenza a sistemi GNU/Linux. Altre decidono di rilasciare il proprio software con licenze libere.

3.2.4.3 La parola *free* non piace a chi produce software perche' troppo carica di significati ideali. Inoltre in inglese *free* significa sia libero che gratuito: percio' viene coniato il termine *open source*, dietro al quale si riconoscono meglio coloro che sono interessati specialmente agli aspetti commerciali del software libero.

3.2.4.4 Alcuni programmi liberi, tra cui il server web *Apache* riscuotono un grande successo per le ottime caratteristiche tecniche, e superano in diffusione i programmi proprietari concorrenti.

3.2.4.5 Dagli anni 2000 anche le piu' grandi imprese, come IBM, riconoscono e sfruttano il potenziale economico dell'*open source.*

3.3 Modelli e mercato

Anche se prodotto in modo quasi completamente artigianale, il software e' al centro di una ricchissima industria in un mercato talvolta molto competitivo. Le modalita' di produzione, diffusione, attribuzione dei prezzi e delle licenze sono di capitale importanza e hanno conseguenze economiche rilevanti. I punti seguenti descrivono un modello fortemente semplificato del mercato del software, per dare un'idea dei ruoli dei vari soggetti coinvolti.

3.4.1 Mercato "tradizionale"

3.4.1.1 I programmatori, stipendiati dalle software house, producono codice sorgente che rimane segreto aziendale. Solo il programma eseguibile viene distribuito ai clienti tramite dei supporti e accompagnato da una licenza che vieta di farne delle altre copie, di modificarlo o manometterlo.

3.4.1.2 La software house ottiene i suoi guadagni soprattutto dalla vendita del programma.

3.4.1.3 Il cliente, in caso abbia bisogno di modifiche al programma o abbia trovato dei difetti o malfunzionamenti (detti *bug*), puo' segnalarli al produttore (con un *bug report*) che potra' correggerli e inserire le modifiche nel successivo rilascio del programma.

3.4.1.4 Un consulente puo' fare da tramite tra software house e utente, o in parte adattarlo (*configurarlo*) per le esigenze del cliente.

3.4.2 Mercato *foss*

3.4.2.1 I programmatori, stipendiati dalle software house o indipendenti, producono codice sorgente che diviene pubblico. Sono disponibili agli utenti sia il programma eseguibile che quello in forma sorgente. Il programma viene accompagnato da una licenza che obbliga il fornitore a rendere disponibili i sorgenti e impedisce al cliente di ridistribuire solo il codice eseguibile.

3.4.2.2 La software house ottiene i suoi guadagni soprattutto da attivita' di servizio, quali personalizzazioni su richiesta, consulenze, corsi, certificazioni.

3.4.2.3 L'utente, in caso di necessita', puo' modificare e correggere il programma direttamente o pagare un programmatore o un consulente per farlo. Se lo desidera puo' ridistribuire le modifiche apportate.

3.4.2.4 Un consulente o un programmatore indipendente puo' adattare un programma alle

esigenze del cliente e contribuire allo sviluppo del programma.

3.4.3 Modelli di licenza

3.4.3.1 Come si e' visto il software viene ceduto all'utente assieme ad una licenza che stabilisce i suoi diritti e obblighi. Eccone alcune tra le principali.

3.4.3.2 EULA. La licenza che accompagna il software proprietario e' quella di solito contrassegnata dall'acronimo EULA (*end user license agreement*, accordo di licenza con l'utente finale). Solitamente descrive i termini in cui l'utente puo' installare, eseguire e copiare il programma nella sua forma binaria. E' accompagnata da una serie di divieti, quali quelli di modificare o cercare di "smontare" (attivita' di *reverse engineering*) il programma.

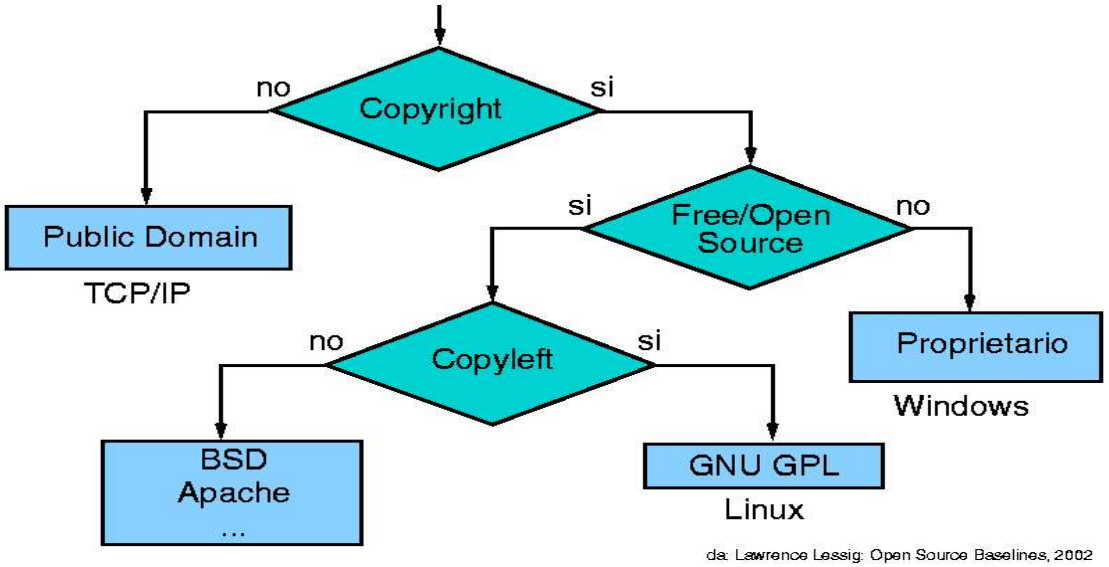
3.4.3.3 GPL. E' la General Public License del progetto GNU/FSF. E' studiata in modo da rendere *irreversibile* il processo di liberazione del codice sorgente: chi lo riceve e' obbligato a usare la GPL anche per i *prodotti derivati* dal codice originario. Esiste una versione meno restrittiva, la LGPL (Lesser GPL, o GPL minormente generale) che prevede certe deroghe che consentono il collegamento stretto di altri programmi non liberi, vietato dalla GPL.

3.4.3.4 BSD. E' una famiglia di licenze derivate da quella originale BSD (Berkeley Standard Distribution) che accompagnava i programmi dell'Universita' di Berkeley, California. Si basa sul principio del dono liberale del codice, ma richiede che venga sempre citata nel codice sorgente stesso la paternita' originale. Il codice rilasciato sotto licenze di tipo BSD puo' essere reso proprietario.

3.4.3.5 Pubblico Dominio. Il programmatore cede ogni diritto di sfruttamento economico (negli USA anche i diritti morali) sul codice. Il programma puo' essere modificato, distribuito, perfino attribuito ad altri autori. Naturalmente puo' anche essere reso proprietario.

3.4.3.6 Dual Licensing. Il produttore di un programma puo' decidere di applicare piu' di una licenza, a scelta del cliente, per un dato programma. Ad esempio una licenza GPL e un'altra senza limitazioni sui prodotti derivati, in modo da lasciare all'utente la facolta' di scegliere la licenza che desidera per i prodotti derivati. Alcuni applicano licenze diverse per lo stesso prodotto a stadi diversi di maturazione (ad es. l'ultima versione del programma e' rilasciato con licenza proprietaria, le versioni precedenti con licenza libera). Possono bentinteso applicarsi tariffe diverse a seconda della licenza applicata.

3.4.3.7 Lo schema 2 a pagina 16 illustra i vari gruppi di licenze in base alle scelta di applicare il copyright o di rendere il programma free/open e infine di avvalersi o meno del copyright per impedire che il programma o prodotti derivati possano divenire proprietari in futuro (*copyleft*).



schema 2 tipi di licenze

3.4.4 Modelli di diffusione/distribuzione del software

3.4.4.1 Solo il produttore di software che applica prezzi molto bassi o distribuisce il proprio prodotto gratuitamente puo' usare Internet per diffonderlo presso gli utenti. Gli altri solitamente offrono in rete solo prodotti di tipo dimostrativo, (*demo),* o dalla funzionalita' limitata, o protetti da codici di attivazione.

3.4.4.2 La distribuzione diretta del software via rete ha dei vantaggi:
- il mancato ritardo tra acquisto e uso del programma e' motivo di gratificazione immediata dell'utente[1];
- il produttore puo' distribuire piu' frequentemente gli aggiornamenti del prodotto, offrendo piu' rapidamente innovazioni tecnologiche e migliorie;
- riduzione dei costi di distribuzione basati su supporti tradizionali, con relativi imballaggi, intermediari, ecc.

3.4.5 Modelli di prezzo

3.4.5.1 A pagamento: il programma viene offerto in cambio della proprieta' del codice eseguibile, o del codice sorgente, o in affitto a tempo, o secondo qualsiasi altra modalita'. Contrariamente a quanto molti ritengono, il codice libero puo' essere venduto; anche la licenza GPL prevede che possa essere pagato il supporto sul quale viene scambiato il programma.

3.4.5.2 Personalizzazione. Un programmatore puo' essere pagato per apportare delle modifiche a del codice sorgente.

3.4.5.3 Freeware. Sono programmi distribuibili, copiabili ed eseguibili gratuitamente,in forma binaria (eseguibile). Sono accompagnati da licenze che possono vietare di sfruttarli per scopi commerciali: in tal caso il programma va comprato. Non si tratta si software libero, mancando la disponibilita' del codice sorgente.

3.4.5.4 Shareware. Analogo al Freeware, ma non gratis. Il programma puo' essere copiato,

1 Messerchmitt, Szipersky: *Software Ecosystem*, p 94

ma si chiede all'utente di pagarlo, solitamente cifre modeste.

3.4.5.5 La Tabella 1 a pagina 17 illustra, in funzione dei vari modelli di licenza, quali sono le facolta' attribuite all'utente.

	Esecuz.	©	Lettura/Copia	Modifica	Distribuzion
Public Domain	Si	Si	Si	Si	Si/$i
BSD	Si	No	Si	Si	Si/$i
IBM Public license	Si	No	Si	Si	Si/$i
GNU GPL	Si	No	Si	Si/No	Si/$i
Shared Source	$i	No	$i/No	$i/No	No
Freeware	Si	No	No	No	Si
Shareware	$i	No	No	No	$i
Proprietaria tradizionale	$i	No	No	No	No

Tabella 1 Licenze e prezzi. Nota: "$i": Si, a pagamento

3.4.6 Modelli di sviluppo

3.4.6.1 Il free software viene spesso considerato un nuovo modo di sviluppare il software. Questa opinione e' stata largamente diffusa grazie ad alcuni popolari articoli (ad es. *La cattedrale e il bazaar*, di Eric Raymond). Sicuramente alcuni programmi, come Linux, sono stati sviluppati grazie ad Internet in modo cooperativo, collaborativo, da una comunita' di persone auto-organizzatasi, che ha riconosciuto od eletto un leader o un coordinatore. Tuttavia non si puo' identificare il free software con questo modello di sviluppo. Vi sono programmi sviluppati in modo analogo che non sono liberi e programmi liberi che sono stati sviluppati in modo tradizionale.

3.4.6.2 Tra le varie definizioni di modelli di sviluppo vi e' quella tradizionale dall'alto al basso, secondo un modello gerarchico, che Raymond identifica con la *cattedrale*, cui viene contrapposto quello del *bazaar*, in cui i vari elementi del programma possono essere reperiti tra il codice libero disponibile e analogamente i programmatori si riconoscono e aderiscono al progetto invece che essere arruolati.

3.4.6.3 Tra le altre possibili suddivisioni in base al coordinamento del progetto vi sono quelle dette *single guru* (guru singolo), in cui un programmatore-guru sviluppa il programma da solo in completa autonomia; quella del *dittatore benevolo*, in cui un singolo coordinatore detta le linee direttive; o quello del *gruppo di progetto (project team)* in cui un gruppo democraticamente decide sul destino del progetto.

3.4.7 Aspetti innovativi e ruolo di Internet

3.4.7.1 Da quanto esposto risulta che non tutti gli elementi che hanno contribuito al successo del free software sono originali o innovativi. Spesso si tratta di una combinazione di effetti: il fenomeno della condivisione del codice sorgente esiste continuativamente dagli anni '70, ma grazie ad Internet ha avuto un impulso straordinario, rendendo possibile uno *sviluppo coordinato* a distanza che prima era impensabile: con la rete e' possibile costituire gruppi di persone che collaborano strettamente anche a distanza.

3.4.7.2 Altro elemento che dipende da Internet e' *la distribuzione*, completamente svincolata dal supporto. E' possibile acquisire un programma senza dipendere dalla catena di distribuzione: non serve contattare un rivenditore, ottenere una copia di prova, valutarla, e ripetere l'operazione con tutti i prodotti concorrenti. Basta scaricare e provare quello che mi serve in breve tempo. Anche questo non e' un meccanismo esclusivo del software libero, dato che e' caratteristico anche di shareware e freeware, ma e' stato determinante per il suo successo.

3.4.7.3 La *gratuita'*, caratteristica di quasi tutti i progetti free, non e' un requisito del foss ed e' comune a molti altri programmi proprietari, ma di sicuro e' stata uno dei motivi della straordinaria diffusione del software libero.

3.4.7.4 Elementi che contraddistinguono in modo assolutamente caratteristico il software libero sono invece *le licenze*, che tutelano l'utente, il programmatore e la liberta' del codice sorgente, anziche' solo il produttore del codice.

3.4.7.5 Nella Tabella 2 sono riassunte le caratteristiche innovative o esclusive dei vari modelli di sviluppo, licenza, distribuzione, tra foss e proprietario.

	esclusivo foss	nuovo	comune con non-foss	non esclusivo foss
sviluppo	nessun modello esclusivo	no	tradizionale, single-guru, master-disciple, core-team	
licensing	GPL, BSD, PD	no	no	EULA
prezzo	no	no	affitto servizio, affitto uso	shareware, freeware, gratis per uso non commerciale
business	no	no	servizi, ASP	
distribuzione	via Internet, con codice sorgente	SI	alcuni mezzi in comune	limitato dal mezzo di distribuzione
uso di tecnologie innovative	no	no	tutte	non esclusivo foss
caratteristiche del rilascio	dinamico	no	tradizionale	limitato dal modello di distribuzione
arruolamento	spontaneo		tradizionale	AAA cercasi

Tabella 2 Elementi di novita'

3.5 La sicurezza

3.5.1 La sicurezza informatica di un programma indica non solo l'assenza di errori nella progettazione e nella codifica del programma (che cioe' non sia soggetto a guasti, interruzioni), ma anche la lealta' (che non faccia cose non previste, ad esempio consenta l'accesso ai dati aggirando i sistemi di sicurezza) e l'affidabilita' (che non danneggi o cancelli i dati o agevoli intrusioni nei sistemi)

3.5.2 La disponibilita' del codice sorgente ha una immediata conseguenza: la sua *ispezionabilita'*. E' possibile verificare che il programma non contenga del codice malevolo o tipici errori di programmazione che ne pregiudichino la sicurezza. A qyesto scopo anche ditte che producono software proprietario hanno proposto a certi selezionati clienti (governi) di visionare il codice sotto stretti accordi di non diffusione (*non disclosure*

agreement)

3.5.3 Inoltre, se il codice e' liberamente disponibile e visionabile da molte persone, ci saranno molti occhi a controllare l'affidabilita' del programma, aumentando la probabilita' che eventuali errori (in gergo informatico: *bachi*, o "*bug"*) vengano identificati. Non e' possibile per il programmatore scrivere codice sciatto confidando nel fatto che nessuno lo guardera' mai: scrivere codice che verra' distribuito e visionato da molti altri programmatori spinge ad essere piu' scrupolosi. La prospettiva non e' solo quella di consegnare del codice che funzioni, ma qualcosa che porta il nome di chi lo ha scritto difronte a una comunita' di pari.

3.5.4 La consuetudine a distribuire il codice online, cosa frequente nel mondo foss, facilita l'aggiornamento dei programmi, riducendo quelli difettosi (in gergo: *bacati*) che non vengono aggiornati.

3.6 Aspetti sociali

Oltre agli aspetti ideali promossi dalla Free Software Foundation e da altri, vi sono altri aspetti importanti da valutare sul piano sociale. Il primo per importanza e' quello della disponibilita' ed accessibilta' dei dati.

3.7.1 I Dati

3.7.1.1 Abbiamo gia' visto che i dati sono sempre codificati per essere manipolati dal calcolatore. Il formato in cui questi dati sono codificati e' della massima importanza perche' questi possano continuare ad essere accessibili nel tempo.

3.7.1.2 La situazione ideale e' quella che il formato dei dati sia aperto e definito in base a degli standard concordati tra i produttori, gi utenti e in generale tutti gli interessati. Gli organismi che formulano gli standard pero' hanno tempi lunghi per approvarli, spesso vi sono problemi nel giungere ad accordi tra ditte concorrenti, o a superare divergenze di natura tecnica.

3.7.1.3 Anche se un formato non e' formulato come uno standard, puo' essere cosi' ben documentato da chi lo crea da rendere chiunque in grado di manipolare file in quel dato formato. Chiunque cioe' puo' scrivere un programma che legga e scriva file in quel formato senza perdita o alterazione di informazioni.

3.7.1.4 Molte ditte ritengono che il formato dei file creati dai loro programmi sia una loro proprieta' e che tenerlo segreto sia un loro diritto. Indubbiamente questo monopolio sulla gestione di un dato formato rappresenta un vantaggio: i dati dei loro clienti puo' essere letto e alterato esclusivamente con i loro programmi. Per superare la diffidenza degli utenti nei confronti di questi formati vengono spesso forniti gratuitamente dei programmi per *leggere* i file in quel formato. Anche ammettendo che l'utente sia sempre soddisfatto del prodotto e desideroso di rimanere legato a quel formato, rimane aperto il problema di come accedere ai dati da altre architetture o ambienti, o nel caso la ditta fallisca o non rilasci nuove versioni del programma.

3.7.1.5 Il software libero, fornendo il codice sorgente attraverso il quale manipolare i file di un dato formato, fornisce anche una descrizione completa del formato stesso. Questo

per la natura stessa del codice sorgente, che e' una rappresentazione in un linguaggio ad alto livello delle operazioni necessarie a manipolare il file stesso. Questo vale anche per formati nuovi, caratteristici di un dato programma. Va precisato che in questo caso e' importante anche la *qualita'* del codice sorgente, che cioe' deve essere facilmente leggibile.

3.7.2 Pericolo di chiusura dei dati o *lock-in*

3.7.2.1 Un utente che ha dei dati in un formato proprietario non disponibile, cioe' noto solo a chi produce i programmi che manipolano dati in quel formato, dipende da quel produttore per leggere i propri dati. Questo fenomeno, che si chiama *lock-in*, garantisce la fedelta' dei clienti al prodotto. E' vantaggioso per il produttore, che in questo modo acquista un vantaggio sulla concorrenza: maggiore e' la mole di dati in un formato chiuso, minore sara' la propensione a cambiare il programma usato per manipolarli. Questa strategia e' ben riassunta dalla frase "prendi i loro dati, cuore e mente seguiranno"[2]

3.7.2.2 Esempio: un ufficio gestisce tutti i fax con un programma fornito gratuitamente che li salva in un formato proprietario. Quando e' ora di aggiornare il sistema operativo alla versione successiva, si accorge che quel dato programma non e' piu' disponibile gratuitamente per la nuova versione, ma solo a pagamento. Le alternative sono rinunciare a leggere i fax ricevuti fino a quel momento, rinunciare ad aggiornare il sistema oppure acquistare il programma.

3.7.3 Il divario digitale (*digital divide*)

3.7.3.1 analogamente a quanto accade con la possibilita' di accedere all'istruzione, l'accesso alle tecnologie digitali amplia le possibilita dell'individuo di comunicare, accedere alle informazioni, di apprendere, amplia le possibilita' di lavoro e di guadagno. La differenza di possibilita' tra chi puo' usare i calcolatori e chi non puo' farlo si chiama divario digitale, o *digital divide*.

3.7.3.2 A esserne colpiti sono individui con malattie o disabilita', poveri, anziani, analfabeti; possono essere intere comunita' di persone che vivono in luoghi non raggiunti dall'elettricita' o dalle linee telefoniche; possono anche essere comunita' soggette a dittature che censurano il loro accesso libero alle informazioni, anche in rete. Anche la semplice ignoranza della lingua inglese costituisce un ostacolo nel mondo telematico (e non solo in quello).

3.7.3.3 Il free software puo' avere un ruolo nell'aiutare a colmare il divario digitale. In parte per il basso costo di acquisto, che lo rende facilmente accessibile a scuole e persone senza mezzi, ma anche per i minori requisiti nell'hardware (richiede spesso computer meno potenti) che ne riducono ulteriormente il costo. L'adattabilita' del software puo' aiutare ulteriormente a colmare altri aspetti del divario digitale, per esempio la lingua.

2 Cusumano, *The business of Software*, p. 64

3.7.4 Free software e paesi in via di sviluppo

3.7.4.1 Il free software puo' costituire un aiuto nel colmare il divario digitale per paesi in via di sviluppo, e non solo per l'immediato vantaggio del basso costo delle tecnologie.

3.7.4.2 Il libero accesso al codice sorgente consente l'adattamento alla lingua (e all'alfabeto) locali, operazione che potrebbe non essere redditizia per un produttore di software proprietario.

3.7.4.3 Il codice sorgente, descrivendo con istruzioni di alto livello le operazioni che compie il programma, costituisce di per se una possibile forma di *trasferimento tecnologico*: vale l'analogia di chi, invece di scavare un pozzo, insegna come farlo. Non solo il programma eseguibile compie le operazioni richieste, ma il codice sorgente mostra come quelle operazioni possono essere compiute. Perche' questo sia possibile serve naturalmente che vi siano le conoscenze informatiche necessarie (saper programmare).

3.7.4.4 La possibilita' di adattare i programmi alle esigenze locali facilita l'emergere di competenze professionali locali che con il tempo possono trasformarsi in una vera e propria industria, e non solo di attingere a competenze e software prodotto all'estero.

3.8 Aspetti politici

3.8.1 Il ruolo delle Pubbliche Amministrazioni

3.8.1.1 Le PA hanno un ruolo particolare nei confronti dei cittadini, che *devono* entrare in relazione con loro. Le scelte dei programmi, e in particolare dei formati, usati dalle PA sono strategiche: tutti i cittadini dovranno uniformarvisi.

3.8.1.2 Ad esempio se l'Universita' di Padova pubblica i bandi di concorso *esclusivamente* in un formato dati proprietario caratteristico di un dato word-processor, i candidato *dovranno* munirsi di quel programma per leggere il bando e compilare la domanda. Se invece vengono usati formati aperti o standard, il candidato potra' scegliere tra i programmi che possono leggere quel formato. Se l'Ufficio delle Entrate accetta le dichiarazioni dei redditi telematiche solo se effettuate con un *browser* disponibile solo su un dato sistema operativo, il cittadino dovra' munirsi di quel sistema operativo per presentare la sua dichiarazione in via telematica, o farlo con il modulo cartaceo.

3.8.1.3 Le PA hanno una responsabilita' nei confronti della collettivita' sulla conservazione e accessibilita' di dati che sono un patrimonio pubblico: devono garantire che questi dati siano accessibili senza interruzioni indipendentemente dalla disponibilita" dei propri fornitori.

3.8.1.4 Se ad esempio un fornitore dei programmi per il calcolo delle pensioni dell'INPS fallisse, dovrebbe esservi garanzia che i programmi possano continuare a funzionare e ad essere manutenuti senza disagi o interruzioni di servizio.

3.8.1.5 Anche se le scelte ben fatte non possono dipendere esclusivamente da valutazioni economiche, le PA devono giustificare (almeno moralmente) alla collettivita' l'impiego del pubblico denaro: se vi sono alternative di costo diverso, la scelta deve essere ben motivata.

3.8.1.6 Un altro ruolo chiave e' quello della pubblica istruzione: i programmi o i sistemi operativi usati ed insegnati nelle scuole e nelle universita' saranno quelli maggiormente usati dalla popolazione. E' opportuno che, nel caso vi siano piu' alternative, queste vengano, se non insegnate, perlomeno illustrate.

3.8.2 Normativa italiana e riuso del codice

3.8.2.1 Il principio cui una PA si deve ispirare per l'acquisto di programmi e' quella di massimizzare il valore in rapporto al costo.

3.8.2.2 I principi che ispirano la normativa italiana sono (dal sito del Ministro per l'Innovazione e le Tecnologie):
- la trasferibilità ad altre Amministrazioni delle soluzioni acquisite
- l'interoperabilità e la cooperazione applicativa tra le amministrazioni
- la non dipendenza da un unico fornitore o da un'unica tecnologia proprietaria
- la disponibilità del codice sorgente per ispezione e tracciabilità
- la esportabilità di dati e documenti in più formati, di cui almeno uno di tipo aperto

3.8.2.3 Una direttiva del Ministro per l'Innovazione e le Tecnologie del 2003[3] prescrive che ogni programma per le PA venga scelto in base a criteri che privilegiano l'apertura dei formati, delle interfacce e in generale interoperabilita' e riuso. I criteri della direttiva sono stati raccolti e rinforzati nel recente *codice dell'amministrazione digitale*[4], in vigore da gennaio 2006.

3.8.2.4 L'art. 69 del *codice dell'amministrazione digitale* prescrive che ogni programma sviluppato appositamente per una PA *dovra'* essere concesso gratuitamente in riuso a un'altra PA che lo richiedesse; che i programmi scritti per le PA siano facilmente *portabili* su altre piattaforme (architetture o ambienti). Consente inoltre che nei contratti con le PA vi siano clausole riguardanti il riuso e che obblighino i fornitori di programmi a prestare *servizi che consentano il riuso delle applicazioni.*

3.8.2.5 Il coordinamento delle attivita' di riuso e' svolto dal CNIPA (Centro nazionale informatica per le PA).

3.8.3 Standard aperti

3.8.3.1 Le scelte politiche devono imporre l'adozione di almeno un formato dati standard o comunque aperti per i programmi che riguardino le pubbliche amministrazioni.

3.8.3.2 Il Codice dell'amministrazione digitale prevede appunto che le PA adottino "soluzioni informatiche che assicurino l'interoperabilità e la cooperazione applicativa, [...] e che consentano la rappresentazione dei dati e documenti in più formati, di cui almeno uno di tipo aperto, salvo che ricorrano peculiari ed eccezionali esigenze."

3.8.3.3 "Il CNIPA istruisce ed aggiorna, con periodicità almeno annuale, un repertorio dei formati aperti utilizzabili nelle pubbliche amministrazioni".

3 direttiva MIT 19 dic 2003, cd. *direttiva Stanca*

4 decreto legislativo 5 mar 2005, n. 82 pubblicato GU n. 112 del 16-5-2005 -Suppl. Ordinario n. 93.

3.8.4 Scelte nei confronti del FOSS e licenze

3.8.4.1 Certe scelte politiche possono avere ripercussioni piu' o meno dirette nei riguardi del software libero o open source.

3.8.4.2 Il foss puo' essere ad esempio sostenuto direttamente con programmi di finanziamento a chi sviluppa codice libero, o indirettamente, tramite sostegno alla ricerca o agevolazioni di vario genere.

3.8.4.3 Solitamente le amministrazioni decidono che le scelte debbano avvenire in base al miglior rapporto prezzo/prestazioni.

3.8.4.4 Altre amministrazioni hanno effettuato scelte piu' esplicite verso il codice sorgente aperto prescrivendo che tutto il software sviluppato da e per la pubblica amministrazione sia accompagnato da licenze libere o addirittura rilasciato nel pubblico dominio. In questi casi la scelta delle licenze comportera' delle conseguenze determinanti: l'impiego di licenze con *copyleft* come la GPL non consente di rendere proprietario il codice sviluppato con denaro pubblico, mentre l'impiego di di licenze piu' liberali lo consente.

4 Viaggio in un sistema foss completo

4.1 Sistema operativo

4.1.1 kernel GNU/Linux

4.1.1.1 Sulla storia del sistema GNU e del *kernel* inux si e' gia' detto. L'Attualmente esistono diverse *distribuzioni*, cioe' modi di collegare insieme i vari elementi del sistema. Alcune distribuzioni sono piu' orientate ad un'utenza esperta, altre a imprese, altre ancora a dispositivi di controllo di apparecchiature (come lavatrici, fotocopiatori) o a utenti inesperti.

4.1.1.2 Il kernel Linux viene sviluppato ancora da un gruppo distribuito di persone, ma alcuni tra i principali soggetti del mercato informatico hanno fondato un consorzio chiamato OSDL (*Open System Development Laboratories* - laboratori di sviluppo sistemi aperti) per i quali attualmente Linus Torvalds lavora, che si occupano di sviluppo, collaudo, promozione e accelerazione della diffusione di Linux.

4.1.2 Kernel free/open/net BSD

4.1.2.1 Quello Linux non e' l'unico kernel libero disponibile. La famiglia dei kernel BSD, hanno tutti origine lontana dal kernel che diede origine a Unix e sono vitali ed attivi, anche se meno in vista. FreeBSD, NetBSD e OpenBSD sono tre sistemi, con caratteristiche diverse, piuttosto diffusi anche se rivolti a utenti esperti.

4.1.2.2 I kernel BSD subirono nel 1991 una lunga interuzione nel loro sviluppo a causa di una controversia legale su una violazione di copyright.

4.2 Ambiente grafico

4.2.1 Interfaccia grafica X11

4.2.1.1 Quasi tutti i sistemi liberi condividono uno stesso sottosistema grafico chiamato X11. Si tratta dell'l'insieme di programmi che gestiscono l'interfaccia utente basata su schede grafiche e dispositivi ad alta risoluzione anziche' terminali a carattere.

4.2.1.2 L'origine del progetto X11 e' lontana, ed e' nata al MIT (*Massachussets Institute of Technology*) di Boston per dare un'interfaccia ad alta risoluzione al sistema Unix, molto diffuso in ambito accademico. Attualmente e' gestita da X.org.

4.3 Ambiente desktop

4.3.1 In una progressiva stratificazione, al sottosistema grafico X11 si aggiungono altri elementi, quali per esempio il sistema di gestione delle finestre *(window manager),* che si occupa di tutti gli aspetti dell'intefaccia legati alle finestre e alcuni menu a tendina.

4.3.2 Diversamente da quento accade nei sistemi operativi proprietari, quelli liberi offrono un gran numero di diverse possibilita' di scelta, sia per i *window manager* che per i programmi per gestire graficamente i file (per copiarli, spostarli, rinominarli, cancellarli). Questo offre molta liberta' all'utente ma nello stesso tempo pone dei problemi tecnici di interoperabilita'

e di mancanza di standard.

4.3.3 Due progetti liberi concorrenti si sono posti l'obiettivo di creare un ambiente integrato e standard nel quale sviluppare complessi programmi grafici.

4.3.4 KDE

4.3.4.1 KDE nasce nel 1996 per iniziativa di un tedesco, Mattias Ettrich. Si propone di sviluppare un ambiente grafico coerente. La vita del progetto e' legata a quella di uno dei componenti usati di programmi KDE e sviluppati dalla ditta Trolltech: le *librerie grafiche QT*, e alla licenza di queste librerie. Per *libreria* si intende una ampia collezione di programmi che svolgono funzioni elementari che vengono ripetute spesso. In questo caso si tratta di funzioni grafiche.

4.3.4.2 Le librerie QT inizialmente erano coperte da una licenza non libera, e successivamente da una doppia licenza: GPL e "commerciale".

4.3.4.3 KDE e' oggi molto diffuso e vanta una apia collezione di programmi che lo rendono un ambiente gradevole e funzionale.

4.3.5 Gnome

4.3.5.1 Nasce nel 1997 per iniziativa di due programmatori messicani, Miguel de Icaza e Federico Mena, come alternativa a KDE, proprio per evitare la licenza, inizialmente non libera, delle librerie QT.

4.3.5.2 Gnome usa la licenza GNU LGPL, che a differenza della GPL non obbliga a coprire con la stessa licenza i programmi collegati strettamente. E' possibile quindi sviluppare programmi proprietari per l'ambiente Gnome, cosa che e' non possibile fare nell'ambiente KDE se non acquistando una licenza commerciale della libreria QT.

4.3.5.3 Per questo motivo Gnome e' preferito dalle *distribuzioni* di orientamento piu' commerciale.

4.4 programmi

4.4.1 Mozilla Firefox

4.4.1.1 E' un *browser* (programma per la navigazione in Internet) di grande successo, disponibile su molte architetture ed ambienti, con alcune caratteristiche innovative come la facile personalizzazione con moduli, i *bookmark attivi,*il *tabbed browsing.*

4.4.1.2 La Mozilla Foundation che lo sviluppa deriva da quella Mosaic communication corp. (poi Netscape), fondata da Tim Berners Lee, che sviluppo' il primo browser. La Netscape rilascio' il codice sotto licenza *open* nel 1998, sotto la pressione della concorrenza di Microsoft Explorer, che allora veniva distribuito insieme ai sistemi operativi Windows. Fu il primo caso di codice proprietario rilasciato da una software house di successo.

4.4.2 Openoffice

4.4.2.1 Programma di automazione di ufficio, composto da vari moduli, tra i piu' importanti quello per scrivere testi, il foglio elettronico, quello per le presentazioni elettroniche.

4.4.2.2 Deriva dal programma StarOffice, originariamente sviluppato in Gemania dalla ditta Stardivision, poi acquistato nel 1999 della Sun Microsystem. Sun distribuisce StarOffice gratuitamente ma con licenza proprietaria, mentre sostiene il progetto OpenOffice.org rilasciato sotto licenza LGPL.

4.4.2.3 Garantisce una elevata interoperabilita' con gli altri programmi di automazione di ufficio e dispone alcune caratteristiche originali molto apprezzate, per esempio la produzione diretta di documenti in formato pdf.

4.4.3 Altri programmi

4.4.3.1 Grafica: The Gimp e' un complesso programma di elaborazione e manipolazione di immagini

4.4.3.2 Grafica: Inkscape un pogramma completo per la grafica vettoriale.

4.4.3.3 CAD: Qcad: Programma per il disegno tecnico.

4.4.3.4 Email: Evolution e Mozilla Firebird sono programmi di gestione della posta elettronica molto sofisticato.

4.4.3.5 Server: Apache e' il programma per server web piu' diffuso, probabilmente uno dei principali fattori di diffusione di GNU/Linux sui server.

5 Documentazione

Altrettanto importante rispetto al software e' la sua documentazione e il modo di condividere le conoscenze relative a come scriverlo, usarlo, installarlo, configurarlo, adattarlo, eccetera.
Caratteristica del software libero e' di avere della documentazione altrettanto libera.
Uno degli aspetti piu' interessanti dell'informatica libera e' che non solo il codice puo' essere modificato dagli utenti, ma anche la documentazione puo' essere scritta da loro.

5.1 Pagine di manuale

5.1.1 Nella tradizione del sistema operativo Unix, ogni programma viene accompagnato da una o piu' pagine di manuale che viene installata nel sistema insieme al programma stesso. Un comando apposito *(man)* serve a accedere alle pagine di manuale, a mostrarle all'utente, ed eventualmente effettuare delle ricerche.
Nei sistemi non Unix vi sono analoghi sistemi di aiuto. Ad esempio nei sistemi DOS/Windows il sistema di aiuto viene attivato solitamente con il tasto F1.

5.2 Online

5.2.1 I programmi liberi di solito hanno un sito web a loro dedicato, nel quale e' reperibile la documentazione, i forum di discussione, gli aggiornamenti dei programmi. Di solito vi e' sempre una sezione destinata alla documentazione, anche se spesso e' in inglese.

5.2.2 Esempio: http://www.openoffice.org e' il sito del programma openoffice.

5.2.3 Il modo piu' semplice di risolvere un problema che si incontra e che non riusciamo a risolvere con la documentazione allegata al programma o presente nel sito e' quella di inserire direttamente in un motore di ricerca il messaggio di errore che il programma ci presenta.

5.2.4 Un altro mezzo efficace e' rappresentato dalle *mailing list* di aiuto, che possono essere generiche o specifiche per un dato programma. Il vantaggio delle mailing list di aiuto e' che chi pone un quesito per un problema che ha incontrato potra' spiegare a qualcun'altro come farlo una volta superato l'ostacolo. In questo modo e' possibile distribuire il carico dell'assistenza agli utenti tra gli utenti stessi.

5.3 PLUTO ILDP

5.3.1 il PLUTO Project (http://www.pluto.it) e' un gruppo informale che da diversi anni ha un progetto di traduzione dall'inglese di documentazione, effettuato da volontari.

5.3.2 Chiunque abbia conoscenza dell'inglese tecnico puo' parteciparvi.

5.4 HOWTO e FAQ

5.4.1 il termine HOWTO, che vuol dire alla lettera “come fare a”, e' una forma di documentazione su come far funzionare i programmi con un approccio molto pragmatico, anche se orientato prevalentemente alla gestione di sistemi.

5.4.2 il termine FAQ sta per *frequently asked questions*, ovvero domande poste di frequente. Sono collezioni di domande su un determinato argomento, e le relative risposte. Utile per

trovare rapidamente soluzioni a problemi semplici.

5.5 Appunti di informatica libera

5.5.1 Un corposo progetto di documentazione su GNU/Linux, distribuzioni, vari programmi liberi, con ampie spiegazioni sull'hardware, installazioni, eccetera. Non sempre aggiornato in tutte le parti, ma in italiano. Reperibile facilmente online, ad es. http://a2.pluto.it oppure http://a2.swlibero.org

5.6 Wiki

5.6.1 Un sito wiki e' un sito di cui chiunque puo' aggiornare, creare e modificare le pagine. E' l'ideale per creare siti di documentazione.

5.6.2 Esempi: http://www.wikipedia.org e' una ricca enciclopedia in piu' lingue, creata esclusivamente dai contributi di volontari, http://foss.stat.unipd.it e' un sito destinato all'uso del Free/Open source software nell'Ateneo di Padova.

5.7 Comunita' locali di utenti

5.7.1 Gli utenti appassionati di software libero spesso si ritrovano in gruppi informali o associazioni, e sono solitamente desiderosi di aiutare chi si trova in difficolta' o i neofiti.

5.7.2 Esempio: i vari FSUG (*free software users group*) e LUG (*linux users group*). A Padova agisce il FSUG Pluto Padova.

6 Laboratorio

Esercitazioni con software libero su aspetti del FOSS trattati durante il corso.

6.1 Programmi liberi: openoffice

Cenni di uso del programma openoffice, affinita' e differenze con Microsoft Office.

6.2 Programmi liberi: firefox

Uso del browser libero: plugin, tabbed browsing.

6.3 Scrivere documentazione online: uso di un wiki

Registrarsi, sviluppare una pagina personale, scrivere della documentazione, le regole di formattazione del testo. Interazione con una comunita', pagina di discussione.

Esempi da http://www.wikipedia.org e esercitazione da http://www.foss.stat.unipd.it

6.4 Porre bene le domande: uso di una lista di discussione

Trovare la lista giusta, iscriversi, disiscriversi, liste pubbliche e private, gruppi di liste, gli archivi, le FAQ, presentarsi. Porre bene una domanda: documentare il proprio sistema, esporre con chiarezza il problema. versione del programma. Come citare, netiquette.

6.5 Le motivazioni dei programmatori: un credit file

La lettura di un file di riconoscimenti (*credit file*) di un programma libero, l'importanza del riconoscimento tra pari e dell'organizzazione del lavoro a distanza con Internet.
Il proprio nome in Internet. Il *ranking* dei motori di ricerca, il prestigio personale.

6.6 Licenze: lettura di una licenza

Lettura di una licenza EULA e della licenza GNU GPL. Confronti, differenze, opinioni... Dibattito...

6.7 Procurarsi un programma

Valutazione delle proprie esigenze, scegliere le parole giuste per la ricerca in Internet, valutazione del programma: aggiornamento, sicurezza, licenza. Scaricamento, installazione, uso.

I repository: navigare in http://www.sourceforge.net.

6.8 Lettura di un brevetto software

Ricerca, valutazione e analisi di un brevetto software.

6.9 Ricerca di documentazione online

Impiego di un motore di ricerca per il reperimento di documentazione. Formulare le domande. Domande generiche, ricerche specifiche. Valutazione delle varie fonti:

affidabilita' e aggiornamento.

Indice Alfabetico

Indice