Lezione 12: il software Open Source

- Copiare il software: un caso in di network externality (da Shy, 2000)

- Possiamo pensare che nel caso dell'industria del software si abbia un caso in cui fare copie del software aumenta i profitti del produttore di software? Da un lato se potesse discriminare il prezzo è possibile che permettere di fare copie aumenti i profitti: ad esempio certe licenze per imprese o istituzioni hanno un prezzo più alto perché si considera il fatto che lo stesso software venga copiato per i dipendenti. D'altro canto le imprese stesse saranno disponibili a pagare un prezzo più alto perché ad esempio potranno dotare i dipendenti di software che può essere usato nel lavoro da casa, nei fine settimana, ecc.

- Altro motivo per spiegare come i profitti di un'impresa che produce software possano aumentare quando non viene messa la protezione contro la possibilità di copiare il software. Il ragionamento si basa sul concetto di *network externality*: gli utilizzatori del software traggono un beneficio dal fatto che esista un elevato numero di utilizzatori dello stesso software. Questo perché pongono un valore alla compatibilità tra i software e alla possibilità di scambiarsi files.

- L'argomento è il seguente: se viene posta una protezione e il software non si può copiare, allora alcuni lo compreranno ed altri no. In particolare il software non sarà acquistato da coloro che sono disposti esclusivamente a copiarlo. Per ciascun utilizzatore il valore di possedere un software aumenta con il numero degli utilizzatori quindi, se la protezione non ci fosse ciascun acquirente avrebbe l'incentivo a copiare e diffondere il software. Se si rimuovesse la protezione, è possibile che gli acquirenti legali siano disposti a pagare un prezzo superiore, perché possono beneficiare del fatto che il software si diffonde.

- Quando si rimuove la protezione, ovviamente, il numero degli utilizzatori aumenta. Questo produce due effetti: da un lato aumenta il valore che gli utilizzatori pongono sul software e quindi il prezzo che potenzialmente può essere praticato, dall'altro si riduce il numero di acquirenti potenziali dato che il software può essere copiato gratuitamente. L'effetto netto dipenderà dal numero di utilizzatori legali, disposti cioè a pagare un certo prezzo (ma non qualsiasi prezzo). In questo caso si aggiungono servizi accessibili solo agli utilizzatori legali (ad esempio l'assistenza), per favorire le vendite. Oppure società e istituzioni della pubblica amministrazione, per motivi di reputazione o più semplicemente perché sono più facilmente controllabili, continueranno ad acquistare legalmente il software.

- Consideriamo un esempio numerico.

Numero di utilizzatori ($n=n^b+n^p$)	1	2	3	4
Disponibilità a pagare	€200	€300	€450	€525

- Supponiamo che ci siano 4 utilizzatori potenziali del software. Due sono "*support-oriented*", cioè interessati ad acquistare legalmente il software perché hanno una forte preferenza per ricevere l'assistenza, due invece sono disposti solo a copiarlo. Nella tabella si riporta la disponibilità a pagare da parte dei primi, che aumenta all'aumentare del numero di utilizzatori. Si definiscono con n^b e n^p rispettivamente il numero di "support-

oriented" e di "pirati". Qui si considera il caso di un monopolista che ha due possibilità: o protegge il software (e quindi ne vende due copie), oppure permette che venga copiato. Nel primo caso $n=n^b=2$ e $n^p=0$. L'impresa può fare un profitto pari a €300*2=€600. Nel secondo caso $n^b=2$, $n^p=2$ e $n=4$. Il profitto dell'impresa è €525*2=€1050 > €600.

- Se il prezzo che si determina è sufficientemente basso da permettere la vendita, allora l'impresa può aumentare i profitti permettendo di copiare il software. In questo caso si è data una spiegazione su una tendenza alla riduzione di protezione dei software osservata empiricamente nella seconda metà degli anni '80.

- In questo caso è l'impresa stessa a non tentare di proteggere la propria proprietà intellettuale. L'implicazione economica è che in alcuni casi può essere più conveniente, anche per le imprese, permettere di copiare i software invece di destinare risorse, ad esempio, per rintracciare e "punire" i pirati o per criptare il software.

- Il software open source (OS) (Lerner e Tirole, 2002, JIE, Garzarelli, 2002)

- I software sono comunemente scritti in linguaggi di programmazione come C, Java e Basic. Il software proprietario solitamente è venduto come "oggetto", cioè come sequenza di 0 e 1 che può essere direttamente interpretata dalla macchina.

- Prima era di cooperazione per lo sviluppo di un sistema operativo (Unix) negli anni 70, grazie anche alla presenza di Usenet, un network creato nel 1979 che collegava i membri della comunità di programmatori di Unix. Questo periodo dura fino ai primi anni '80, quando cominciano a essere messi dei vincoli da AT&T, presso i quali laboratori erano iniziati i lavori di sviluppo di Unix, rispetto ai suoi diritti di proprietà intellettuale su Unix.

- Stallman (MIT) crea nel 1983 la Free Software Foundation per disseminare diverse varietà di software senza costi. L'idea è anche quella di definire una licenza d'uso per il software prodotto collettivamente, in modo da precludere la pretesa di diritti di proprietà intellettuale su di esso. In particolare per sviluppare GNU, sistema operativo alternativo a Unix (le varianti che utilizzano il kernel di Linux sono chiamate GNU/Linux). Viene codificata una licenza formale (*General Public License*, GPL) che accompagna il software open source. Alcuni punti sono:

1) in cambio della possibilità di modificare e distribuire il software, l'utilizzatore deve accettare di renderlo a sua volta liberamente accessibile ad altri;
2) gli sviluppatori non devono porre restrizioni ad altri;
3) tutti i miglioramenti al codice devono essere concessi in licenza agli stessi termini.

- Si noti che c'è una differenza rispetto al cosiddetto *shareware*, che è un software è reso liberamente disponibile, spesso per un certo periodo ma solo sotto forma di codice binario; e rispetto al software di pubblico dominio, su cui non viene posta nessuna restrizione.

- Dal punto di vista organizzativo questo sistema aveva le seguenti caratteristiche: i) venivano accettati contributi da diversi sviluppatori; ii) la versione ufficiale del programma era gestita da un numero ristretto di individui strettamente coinvolti nel progetto; iii) il fondatore del progetto (o un suo “successore”) veniva riconosciuto come

leader.

- Negli anni '90 importanti sviluppi: la comunità di *open sourcers* si espande (anche grazie alla diffusione di Internet), numerosi nuovi progetti vengono lanciati (il più noto è forse il sistema operativo Linux, lanciato da Linus Torvalds nel 1991), inizia l'interazione tra imprese commerciali e la comunità OS.

- Modifiche anche nei contratti di licenza: ad esempio quella dell'organizzazione Debian. Si tratta di una licenza meno restrittiva, nel senso che permette di "abbinare" in qualche modo il software OS con quello proprietario.

- Tra le caratteristiche dei prodotti OS c'è stata spesso, almeno inizialmente, una predisposizione ad essere utilizzati da utenti esperti. Questo ha costituito motivo di dibattito nella comunità ad esempio rispetto all'uso di interfacce grafiche (come Gnome e KDE per Linux), insieme ad altre questioni come la disponibilità di manuali, assistenza, ecc. Tutto ciò ad esempio non accade per software proprietari come Windows, concepiti per essere estremamente *user friendly*.

- La comunità di open sourcers partecipa però in modo asimmetrico ed è piuttosto elitista, cioé un numero ristretto di individui è responsabile per la maggior parte dei progetti ed un numero elevato di individui dà contributi sporadici (al limite segnala solamente dei *bugs*). Si viene a creare un *core group*, a cui viene riconosciuto un status molto elevato dagli altri partecipanti.

- Esempi di successo, misurato dalle quote di mercato sono: Apache (web server), Linux (sistema operativo), Perl (linguaggio di programmazione), Sendmail (programma di gestione di server per posta elettronica).

- Apache: sviluppato nel 1994 da B. Behelendorf, che manteneva il server del sito web della rivista *Wired*. Il programma che usava era stato realizzato dal *National Center for Supercomputer Applications* (NCSA). Il codice sorgente del programma era liberamente disponibile e si era creato un gruppo di utilizzatori che interagiva con il NCSA per apportare modifiche, miglioramenti, ecc. Il fatto di ottenere scarse risposte dal NCSA in relazione a vari problemi incontrati porta Behelendorf e il suo gruppo a realizzare Apache, che presenta caratteristiche particolari ad esempio la modularità che permette a gruppi separati di lavorare su parti specifiche del programma senza interferire con il resto. In quel momento sul mercato non c'erano valide alternative, e Apache conquista una posizione dominante. Malgrado il successivo ingresso sul mercato di società come Netscape e Microsoft, Apache ha mantenuto una posizione dominante: nel novembre del 2000 aveva il 59,7% del mercato. Nel 1999 viene creata la *Apache Software Foundation* allo scopo di gestire e supervisionare lo sviluppo del prodotto.

- Linux: creato nel 1991 da Linus Torvalds essenzialmente per curiosità intellettuale, piuttosto che per risolvere un problema pratico. Linux unisce “Linus” e “Unix”. Torvalds voleva sviluppare il “kernel”, cioé il cuore di un sistema operativo basato su una versione di pubblico dominio di Unix. Inizialmente il codice venne messo a disposizione sul server di un'università. Intorno al 1994 Linux aveva circa mezzo milione di utilizzatori, oggi si parla di alcuni milioni in tutto il mondo. Linux costituisce un concorrente di Microsoft Windows. Fin dall'inizio Torvalds ha mantenuto la posizione di leader, pur essendosi

sviluppata un'ampia comunità di sviluppatori. Si sono successivamente avuti investimenti in Linux da parte di società (come Red Hat), che vendono "pacchetti" Linux, generalmente facili da installare, che contengono manuali, e per i quali è garantita l'assistenza, e vendono prodotti complementari di natura proprietaria. Altre imprese di software e hardware hanno realizzato massicci investimenti nello sviluppo di Linux.

- Sendmail: inizialmente sviluppato alla fine degli anni '70 da uno studente di Berkeley (E Allman) per risolvere un problema di compatibilità tra le diverse reti che esistevano nel campus rispetto all'invio di emails. Questo si afferma successivamente come leader nei programmi di gestione dei serve di posta elettronica (si stima che intorno al 2000 circa il 75% del traffico di email è gestito da Sendmail). Allman ha creato una società nel 1997 che ha ottenuto finanziamenti di *venture capital*, cioé destinati a imprese nuove e generalmente rischiosi. La società vende prodotti legati a Sendmail (come interfacce), fornisce assistenza, ma continua a sostenere lo sviluppo del software nella forma OS.

La teoria economica e l'open source.

- La teoria economica può contribuire a rispondere ad alcune domande: i) cosa spinge a partecipare a progetti OS? ii) Perché esistono progetti OS? Iii) Come reagiscono le imprese tradizionali allo sviluppo del OS?

- Apparentemente non si riescono a comprendere gli incentivi di un l'inventore di un software (che ha dunque sostenuto dei costi di ricerca) lo mette a disposizione, rivela cioè il codice sorgente, così come quelli dei programmatori che formano la comunità di sviluppatori.

- i) Dal punto di vista economico un programmatore partecipa ad un progetto OS se ne può derivare un beneficio netto, dato dal *payoff* immediato (beneficio corrente – costo corrente) più il *payoff* differito (beneficio differito – costo differito).

- I costi sono essenzialmente: i costi opportunità, il tempo che dedicano lavorando all'OS potrebbe essere dedicato al lavoro retribuito in un'impresa. Se già sono assunti da un'impresa il tempo passato per l'OS li distoglie dalle loro mansioni principali. Il tempo di realizzazione di un progetto aumenta, il tempo dedicato alla ricerca (caso di un programmatore universitario) diminuisce, il tempo di ottenimento di un titolo di studio (caso di un programmatore studente) aumenta. Questo può implicare dei costi differiti.

- I benefici sono di due ordini: i primi immediati: può essere il caso che il tempo passato sull'OS aumenti la produttività del programmatore (ad esempio se impara qualcosa di utile, oppure il fatto di dedicarsi sul suo posto di lavoro a questa attività costituisce una riduzione della routine). I benefici futuri sono a loro volta di due tipi: il primo è più prettamente monetario: distinguendosi all'interno della comunità dei programmatori, cioé segnalandosi come programmatori validi, è possibile ottenere guadagni futuri in termini di maggiori stipendi (*career concern incentive*); il secondo ha una forte componente non pecuniaria e riguarda la reputazione all'interno della comunità (*ego gratifcation incentive*). Questi due tipi di incentivi sono definibili come incentivi alla segnalazione (*signaling incentives*).

- Dal punto di vista della leadership, risulta in parte più difficile comprendere cosa spinge

l'iniziatore di un progetto a distribuire liberamente il codice. Ci sono certamente incentivi dati dallo status che si può ottenere o altre prospettive di carriera, ma questo può apparentemente non essere comparabile con i forti guadagni che possono derivare dal mantenere segreto il codice.

- Confronto tra gli incentivi dei programmatori in ambiente OS o *closed source* (CS). Consideriamo i benefici immediati. In un'impresa c'è naturalmente un vantaggio in termini di salario. L'impresa può pagare salari perché ottiene dei profitti dallo sfruttamento del software proprietario. In ambiente OS il programmatore ha però benefici di altra natura: "alumni effect", essendo il codice libero, il programmatore può avere avuto la possibilità di conoscerlo a scuola o all'università, e questo può ridurre i suoi costi di lavorare al progetto OS; il fatto di contribuire a un programma OS può dare poi dei benefici al programmatore o alla sua impresa se egli è in grado ad esempio di risolvere un problema del software che gli si era presentato.

- Consideriamo i benefici differiti. I *signaling incentives* sono maggiori maggiore è la visibilità della performance e maggiore è la possibilità di attribuire la performance a un dato individuo. In un ambiente OS questi incentivi possono essere superiori che in un ambiente CS perché: in un ambiente CS non è del tutto agevole per osservatori esterni osservare il contributo di un singolo perché il codice non è visibile; viceversa in ambiente OS è perfettamente osservabile il contributo del singolo, se il suo contributo ha funzionato, se il problema da risolvere era complesso, ecc... La performance del programmatore è maggiormente misurata e attribuita ad un individuo in ambiente OS perché è stabilita direttamente dal singolo e non ad esempio da un suo superiore. Inoltre, il capitale umano accumulabile in ambiente OS dovrebbe essere maggiormente trasferibile, nel senso che molti elementi sviluppati in un progetto OS, essendo liberamente trasferibili, vengono poi incorporati in altri progetti. Dal punto di vista di un lavoratore, è preferibile accumulare capitale umano "generico", cioé trasferibile in diverse occupazioni, perché questo permette di ottenere guadagni in diverse occupazioni.

- Quindi, individui per i quali i *signaling incentives* sono forti, saranno portati a partecipare a progetti OS. Può essere importante "segnalarsi" per ottenere buone occupazioni, fare carriera in un'organizzazione, ottenere finanziamenti di *venture capital*...

- Si tenga presente che partecipare significa a volte risolvere un problema individuale (questo ha motivato la nascita di alcuni progetti OS come Sendmail). Ma uno degli aspetti più rilevanti è la possibilità di vedere riconosciuti in modo pubblico i propri meriti rispetto allo sviluppo di un progetto OS. Nei siti web dei vari progetti solitamente si trovano le indicazioni dei nomi dei *contributors*. Questo ha spesso permesso ad alcuni di loro di avere delle credenziali per accedere a posizioni ben remunerate presso imprese commerciali, o di ottenere *venture capital*.

- Altre caratteristiche della produzione OS: organizzazione della produzione e *governance*. Da un lato molti dei prodotti sono modulari: cioé consentono a diversi gruppi di lavorare contemporaneamente a diverse parti del programma senza creare interferenze. All'avvio dei progetti è importante il ruolo del leader nell'attrarre diversi programmatori: ad esempio deve fare apparire i problemi da risolvere come *challenging.*

- Si noti che, invece di essere un'organizzazione quasi anarchica, nella comunità OS sono

ben definite le *leaderships* (non vere e proprie gerarchie). In alcuni casi si tratta di singoli individui (come Torvalds per Linux), in altri di circoli ristretti. Questo è importante perché l'autorità, che non è formale (nel senso che non può ordinare a qualcuno di fare qualcosa) ma "sostanziale", svolge alcuni ruoli essenziali: definisce l'indirizzo della ricerca, la "visione" iniziale e gli altri obiettivi durante il progresso del progetto, può evitare il *forking*, cioè la frammentazione della comunità in piccoli gruppi che perseguono obiettivi senza avere sufficiente massa critica e quindi dissipando risorse, mediante l'accettazione delle modifiche apportate dagli altri programmatori certifica la qualità del prodotto (importante all'esterno per ottenere la fiducia di eventuali utenti).

- La comunità deve avere fiducia nei confronti del leader. Questa di solito è conquistata tramite la reputazione, l'aver dimostrato che le eventuali indicazioni sono nell'interesse della comunità, l'essere stati trasparenti rispetto a decisioni cruciali per lo sviluppo del progetto, ad esempio rendendo pubbliche le procedure di controllo di nuove proposte di software.

- Nelle compagnie private invece: il *signaling incentive* ci può essere solo parzialmente. Ad esempio in alcuni casi viene dato il credito agli sviluppatori (come per Eudora), ma spesso ciò non accade perché le imprese temono così facendo di segnalare all'esterno i loro migliori lavoratori che potrebbero essere sottratti dalla concorrenza, anche se, allo stesso tempo riconoscono che questa pratica potrebbe essere un sistema per reclutare i migliori (trade-off). In questo senso i programmatori leaders nelle imprese commerciali guadagnano visibilità tanto quanto negli ambienti OS. Invece a quelli operanti a livelli più bassi vengono concesse meno opportunità di rendersi visibili. Inoltre, sebbene teoricamente si potrebbero adottare delle pratiche OS all'interno delle imprese, questo non viene quasi mai fatto.

- Le imprese commerciali possono capitalizzare il successo di OS offrendo prodotti complementari (vedi Red Hat, Mandrake, etc.). Possono a loro volta rendere pubblici i codici sorgenti, ad esempio con l'obiettivo di coprire poi segmenti complementari, cioè offrire prodotti complementari di tipo proprietario. Questo può valere se l'impresa si attende di ottenere dei profitti nei segmenti complementari maggiori di quelli perduti per avere reso disponibile il codice del software del segmento "primario". Questo può accadere se l'impresa è troppo piccola per potere competere nel segmento primario.

- In generale, la comunità di persone impegnate nei progetti OS rappresenta una forma particolare di organizzazione del lavoro. Queste persone sono appunto su un medesimo livello, si possono monitorare a vicenda, hanno incentivi allineati e non contrastanti, condividono un problema di reputazione (nel senso che cercano di massimizzare la loro reputazione nella comunità). Inoltre questo tipo di organizzazione riduce di molto l'incertezza del lavoro di scoperta: numerosi individui provano simultaneamente diversi approcci per risolvere lo stesso problema (basti pensare alla frequenza elevata di nuove versioni di software OS rispetto a quello proprietario), e si caratterizza per la fiducia reciproca dei membri. Non tutti sono identici, ma condividono un nucleo di competenze (Garzarelli, 2002).

- La velocità con la quale vengono risolti i *bugs* contribuisce alla qualità del prodotto OS, che può in parte spiegare le quote di mercato guadagnate da tali prodotti.

- Considerazioni:

- Il software è un tipico frutto dell'attività inventiva che costituisce input per altro software.

- Questo già di per sé è un argomento a favore della cautela rispetto all'attribuzione di forti diritti di proprietà intellettuale ai creatori di software.

- Il caso OS dimostra che è possibile che si determini un sistema di incentivi che favorisce l'innovazione continua *in assenza* di tutela della proprietà intellettuale (almeno nelle forme tradizionali del brevetto e del copyright).

- In questo caso la ragione è da ascrivere alla natura del prodotto (ad esempio la modularità, al fatto che ci può lavorare una comunità tramite internet) e al tipo di organizzazione del lavoro che si è creata nella comunità di *open sourcers*.