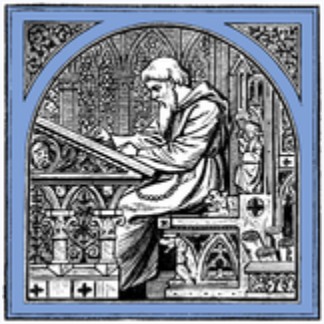
Il calderone magico

Eric Steven Raymond



1999

Questo testo è completo.

Questo saggio analizza il substrato economico in evoluzione del fenomeno open source. Innanzitutto, si sfatano alcuni miti predominanti circa il finanziamento dello sviluppo dei programmi e la struttura del prezzo del software. Quindi si presenta un'analisi della stabilità dell'open source, basata sulla teoria dei giochi, per poi passare a nove modelli di finanziamento sostenibile per la gestione open source; due no profit e sette a scopo di lucro. Si continua esponendo una teoria qualitativa delle circostanze in cui la scelta del software commerciale è più razionale, in termini economici. In seguito, si esaminano alcuni nuovi meccanismi aggiuntivi che il mercato sta inventando per finanziare lo sviluppo open source con finalità di lucro, ivi compresa la reinvenzione del sistema del patrocinio e la costituzione di mercati specifici. Si conclude con alcune previsioni ipotetiche degli sviluppi futuri.

Indice

Nella mitologia gallese, la dea Ceridwen possiede un enorme pentolone che produce cibo prelibato, ogni qualvolta gliene venga dato ordine con una formula magica nota soltanto alla dea. Nella scienza moderna, Buckminster Fuller ci ha fornito il concetto di "effimerizzazione", secondo cui la tecnologia diventa sempre più efficace e sempre meno costosa, via via che le risorse materiali investite nei progetti precedenti vengono sostituite da una quantità crescente di informazioni. Arthur C. Clarke ha messo in collegamento le due cose affermando che "ogni tecnologia sufficientemente avanzata è indistinguibile dalla magia".

A molte persone, i successi della comunità open source sembrano un esempio inverosimile di magia. Software di alta qualità si materializzano "gratis", il che è bello finché dura, ma sembra a malapena sostenibile nel mondo reale, in cui dominano competizione e scarsità di risorse. Ma dov'è il trucco? La pentola magica di Ceridwen è soltanto un gioco di prestigio? E se non lo è, come funziona l'effimerizzazione in questo contesto? Qual è la formula magica della dea?

Sicuramente, l’esperienza della cultura open source ha fatto vacillare molte certezze in coloro che hanno studiato la gestione del software in un ambiente ad essa esterno. "La cattedrale e il bazaar" [CatB] descriveva i modi in cui la gestione decentrata e cooperativa del software contravveniva efficacemente alla legge di Brooks, portando a livelli di affidabilità e qualità dei singoli progetti senza precedenti. "Colonizzare la noosfera" [HtN] esaminava le dinamiche sociali in cui si colloca questo modello di gestione a "bazaar", sostenendo che la prospettiva migliore per comprenderlo si trovava non già nelle convenzioni dell’economia di scambio, bensì in quella che gli antropologi chiamano la "cultura del dono", i cui membri competono per il proprio status regalando vari oggetti. In questo saggio, cominceremo sfatando alcuni miti diffusi sull’economia della produzione di software, per poi proseguire l’analisi di [CatB] e [HtN] nelle sfere dell’economia, della teoria dei giochi e delle strategie aziendali, sviluppando nuovi strumenti concettuali necessari per comprendere il modo in cui la cultura del dono, tipica degli sviluppatori open source, possa sostenersi in un’economia di scambio.

Per perseguire questa linea di analisi senza lasciarci fuorviare, dovremo abbandonare (o almeno decidere di ignorare temporaneamente) il livello di spiegazione relativo alla "cultura del dono". [HtN] sosteneva che comportamenti riconducibili alla cultura del dono si manifestano in situazioni in cui i beni di prima necessità siano sufficientemente abbondanti da far diminuire l’interesse nel gioco dello scambio; ma se questa sembra abbastanza convincente come spiegazione psicologica e comportamentale, non è sufficiente a chiarire il variegato contesto economico in cui opera la maggioranza degli sviluppatori open source. Per la maggior parte di essi, il gioco dello scambio ha perso le sue attrattive, ma non la capacità di opporre limiti. Il loro comportamento deve risultare abbastanza sensato in un’economia a scarsità di materiale per mantenersi in una zona di surplus che possa sostenere la cultura del dono.

Pertanto, considereremo ora (interamente dall’interno di un’economia di scarsità) le modalità di cooperazione e scambio che sostengono lo sviluppo open source. Nel far ciò, risponderemo al quesito pragmatico: "Come fare soldi con questa cosa?", dettagliatamente e fornendo esempi. Prima, però, mostreremo che gran parte della tensione sottostante questa domanda deriva da modelli divulgativi assai diffusi sull’economia della produzione del software, che si rivelano falsi alla prova dei fatti.

(Una nota conclusiva prima di venire all’esposizione: la discussione e il sostegno accordato allo sviluppo open source in questo saggio non devono essere scambiati per una tesi secondo cui lo sviluppo commerciale sarebbe intrinsecamente errato, né per un pamphlet contro i diritti di proprietà intellettuale nel software e nemmeno per un appello altruistico alla "condivisione". Benché questi argomenti siano ancora assai cari a una minoranza rumorosa nella comunità open source, le esperienze fatte dai tempi di [CatB] ne hanno svelato il carattere superfluo. Un argomento perfettamente sufficiente in favore dello sviluppo open source può basarsi sui suoi risultati a livello ingegneristico ed economico: migliore qualità, maggiore affidabilità, costi inferiori e varietà di scelta).

Per iniziare, occorre far notare che i programmi informatici, come tutte le altre classi di strumenti o beni capitali, hanno due tipi distinti di valore economico: hanno, cioè, un *valore d'uso* e un *valore di vendita*.

Il *valore d'uso* di un programma è il suo valore economico in quanto strumento. Il *valore di vendita* di un programma è il suo valore in quanto articolo commerciabile (nel gergo professionale degli economisti, il valore di vendita è il valore come bene finito, mentre il valore d'uso è il valore come bene intermedio). Quando il profano cerca di ragionare sull'economia della produzione di software, tende a prendere come esempio la fabbrica, sfruttando un modello fondato sulle seguenti premesse.

1. La maggior parte dell'orario di lavoro degli sviluppatori è retribuito in base al valore di vendita.
2. Il valore di vendita di un software è proporzionale ai costi dello sviluppo (cioè al costo delle risorse necessarie per duplicarlo in modo funzionale) e al valore d'uso.

In altre parole, c'è una forte tendenza a presupporre che il software condivida le caratteristiche di valore di un tipico bene manifatturiero. Ma è possibile dimostrare che entrambe queste presupposizioni sono false.

Innanzitutto, il codice scritto per la vendita non è che la punta dell'iceberg rappresentato dalla programmazione. Nell'era precedente il microcomputer, era opinione comune che il 90% di tutto il codice diffuso nel mondo fosse scritto all'interno di aziende, come banche o compagnie di assicurazione. Probabilmente, questo non è più vero: oggi, altre industrie sono assai più coinvolte nel software, per cui la frazione del totale realizzata dall'industria finanziaria è diminuita. Ma vedremo, tra breve, che esistono prove empiriche che circa il 95% del codice viene ancora scritto all'interno di aziende.

Tale codice comprende la maggior parte dei sistemi di gestione informatica, compresi gli adattamenti del software a scopo finanziario e la gestione di banche dati necessarie a tutti i media e alle grandi imprese. Comprende codice tecnico-specialistico come i device driver (ma quasi nessuno fa soldi vendendo device driver e su questo punto torneremo in seguito), nonché tutti i tipi di software incorporato per le nostre macchine, sempre più basate sui microchip, dagli

strumenti meccanici agli aerei di linea a reazione, dalle macchine ai forni a microonde, fino ai tostapane.

Gran parte di questo codice, prodotto a livello aziendale, è integrato nel suo ambiente attraverso modalità che rendono molto difficili il riutilizzo e la copia (ciò è vero sia che per "ambiente" si intenda una serie di procedure interne agli uffici di un’impresa, sia che si intenda il dispositivo di iniezione del carburante in una mietitrebbia). Perciò, con il variare dell’ambiente, occorre sempre molto lavoro per l’adattamento del software.

Si tratta della "manutenzione" che, come sa qualsiasi ingegnere o perito informatico, rappresenta la stragrande maggioranza (più del 75%) del lavoro per cui sono pagati i programmatori. Di conseguenza, il programmatore passa gran parte del suo tempo (e guadagna gran parte del suo salario) scrivendo e aggiornando codice aziendale che non ha alcun valore di vendita: il lettore potrà facilmente verificare questo fatto esaminando le offerte di lavoro per programmatori in qualsiasi giornale che contenga una sezione di annunci economici.

Scorrere gli annunci di lavoro su di un quotidiano locale è un esperimento illuminante a cui vorrei invitare il lettore o lettrice. Esaminate le offerte di lavoro per programmazione, elaborazione di dati, ingegneria informatica e cercate lavori che coinvolgano lo sviluppo di software. Suddividete le varie offerte a seconda del fatto che il software venga realizzato per l’uso o per la vendita.

Emergerà ben presto che, anche attenendosi alla più elastica definizione di "vendita", almeno diciannove su venti dei salari offerti proverranno esclusivamente dal valore d’uso (ossia dal valore del bene intermedio). Qui si trova la nostra ragione di ritenere che solo il 5% dell’industria sia sostenuta dal valore di vendita. Notate, però, che il resto dell’analisi svolta nel presente saggio è relativamente insensibile a questo dato: anche se fosse il 15 o addirittura il 20%, le conseguenze economiche resterebbero essenzialmente le stesse.

(Quando prendo la parola in convegni tecnici, solitamente inizio il mio intervento ponendo due quesiti: quante persone nel pubblico sono pagate per scrivere software, e per quante il salario dipende dal valore di vendita del software. Generalmente, ottengo una selva di mani alzate alla prima domanda, poche o addirittura nessuna alla seconda, e molta sorpresa da parte del pubblico davanti a una tale proporzione.)

In secondo luogo, la teoria per cui il valore di vendita di un software è collegato ai suoi costi di sviluppo o sostituzione si sfata ancora più facilmente, esaminando il comportamento stesso dei consumatori. Sono molti i beni per cui una proporzione di questo genere regge (prima della svalutazione): cibo, automobili, strumenti meccanici. Ci sono anche molti beni non tangibili il cui valore di vendita è strettamente collegato ai costi di gestione e sostituzione: i diritti di riproduzione di musica, cartine o banche dati, per esempio. Tali beni possono mantenere o addirittura aumentare il proprio valore di vendita, una volta uscito di scena il produttore originale.

Al contrario, quando un produttore software cessa l'attività (o se il prodotto sta andando fuori produzione), il prezzo massimo pagato dal consumatore andrà rapidamente verso lo zero, a prescindere dal valore d'uso teorico o dai costi di sviluppo di un equivalente funzionale. (Per verificare questa affermazione, basta osservare gli scatoloni delle rimanenze al negozio di software più vicino a casa vostra.)

Il comportamento dei rivenditori, quando un produttore chiude, è assai rivelatore. Ci fa capire che i rivenditori sanno qualcosa che i produttori non sanno. Ciò che sanno è che il prezzo pagato dal consumatore è determinato, in realtà, dalle *previsioni sul valore futuro dei servizi del produttore* (dove "servizi" comprende, in questo caso, personalizzazioni, aggiornamenti e progetti in progressione.)
In altre parole, il software è in gran parte un'industria di servizi che opera sotto la persistente, ma infondata illusione che si tratti di un'industria manifatturiera.

Vale la pena esaminare il motivo per cui, normalmente, tendiamo a credere il contrario. Potrebbe essere semplicemente perché la piccola porzione dell'industria del software che produce a livello industriale per la vendita è anche la sola a pubblicizzare i suoi prodotti. Inoltre, alcuni dei prodotti più visibili e fortemente pubblicizzati sono effimeri, come giochi che hanno ben pochi requisiti in materia di servizi di aggiornamento (che sono l'eccezione, piuttosto che la regola) SH.

Vale anche la pena notare che l'illusione manifatturiera incoraggia un assetto dei prezzi patologicamente scollato rispetto alla reale scomposizione dei costi di sviluppo. Se (com'è generalmente accettato) oltre il 75% dei costi del ciclo vitale di un normale progetto software è legato alla manutenzione, alla messa a punto e alle estensioni, la comune politica dei prezzi, consistente nell'imporre un elevato

prezzo fisso all'acquisto e contributi per l'assistenza relativamente bassi o addirittura nulli, è destinata a condurre a risultati deludenti per entrambe le parti.

I consumatori perdono perché, anche se il sofware è un'industria di servizi, gli incentivi nel modello della fabbrica vanno tutti contro l'offerta da parte del produttore di un servizio *competente*. Se il denaro del produttore proviene dalla vendita di articoli, la maggior parte dei suoi sforzi si concentrerà sul produrne il più possibile e mandarli fuori al più presto; l'ufficio assistenza, non attraendo profitti, diventerà una discarica per i meno efficienti e avrà soltanto le risorse strettamente necessarie a non alienarsi un numero eccessivo di clienti.

L'altra faccia della medaglia è che la maggior parte dei produttori che accettano questo modello della fabbrica finirà per fallire, nel lungo periodo. Finanziare spese di assistenza a scadenza indefinita, a partire da un prezzo fisso, è fattibile solo in un mercato in espansione abbastanza rapida da coprire i costi per l'assistenza e il ciclo vitale applicabili alle vendite di ieri, ma con i guadagni di domani. Una volta che il mercato raggiunge la maturazione e le vendite rallentano, la maggior parte dei produttori non avrà altra scelta se non tagliare le spese, abbandonando il prodotto.

Che ciò sia fatto in modo esplicito (mettendo il prodotto fuori catalogo) o implicito (rendendo più difficile l'accesso all'assistenza), l'effetto è comunque la fuga di clientela verso la concorrenza (poiché distrugge il valore futuro del prodotto, che è parte integrante del servizio). Nel breve periodo, si può sfuggire a questa trappola rilasciando file che correggono errori di programmazione ("bug-fix") facendoli passare per nuovi prodotti, con un prezzo nuovo, ma i consumatori si stancano facilmente. Nel lungo periodo, tuttavia, l'unico modo per cavarsela è non avere concorrenza, il che equivale a detenere un vero e proprio monopolio sul mercato. Alla fine, ne rimarrà soltanto uno.

E, in effetti, abbiamo visto diverse volte questo modello uccidere concorrenti anche forti al secondo posto in una nicchia di mercato. (Lo schema dovrebbe risultare particolarmente chiaro a chiunque abbia mai esaminato la storia dei sistemi operativi proprietari per il personal computer, dei word processor, dei programmi di contabilità e del software per l'impresa in generale). I perversi incentivi insiti nel modello della fabbrica conducono a una dinamica di mercato per cui il vincitore prende tutto e in cui anche i clienti del vincitore finiscono per perdere.

E se non il modello della fabbrica, quale? Per far fronte in modo efficiente alla reale struttura dei costi del ciclo vitale di un software (sia nel senso informale di "efficienza", sia nel suo significato in gergo economico), serve una struttura dei prezzi fondata su contratti per determinati servizi, abbonamenti e scambio di valore *continuativo* tra produttore e consumatore. Pertanto, date le condizioni di ricerca dell'efficienza imposte dal libero mercato, possiamo prevedere che, se questo tipo di struttura dei prezzi si imporrà, il risultato finale sarà un'industria del software matura.

Fin qui, ci siamo addentrati nel motivo per cui il software open source pone una sfida sempre più stringente e non puramente tecnologica, ma anche economica, nei confronti dell'ordine costituito. Pare che l'effetto del rendere un software "libero" sia quello di costringerci a entrare in un mondo dominato dallo schema servizio/costo e mostrarci come il valore di vendita degli articoli commerciali sia sempre stata un'impalcatura relativamente fragile.

Il termine "libero" è fuorviante anche in un altro senso. Abbassare il costo di un bene tende a far aumentare, piuttosto che diminuire, gli investimenti totali nell'infrastruttura che lo sostiene. Quando il prezzo delle automobili diminuisce, la richiesta di meccanici per auto aumenta: è per questo che, probabilmente, anche quel 5% di programmatori, attualmente retribuiti in base al valore di vendita, non ne risentirebbe in un mondo open source. Chi perderebbe, nella transizione, non sarebbero i programmatori, bensì gli investitori che hanno scommesso su strategie di non disponibilità del codice sorgente in casi in cui non sono opportune.

C’è un altro mito, uguale e contrario all’illusione del modello della fabbrica, che spesso inganna chi riflette sull’economia del software open source: è l’idea che "l’informazione voglia essere libera". Ciò conduce solitamente all’affermazione secondo cui un costo marginale uguale a zero per la riproduzione di informazioni digitali, implicherebbe che anche il costo alla vendita sia zero.
La forma più diffusa di questo mito si sfata facilmente, considerando il valore di un’informazione che dia accesso a un bene ambito, come la mappa di un tesoro, o il numero di un conto corrente in Svizzera, o ancora l’accesso a servizi quali una password di account per un computer. Benché l’informazione richiesta possa essere duplicata a costo zero, lo stesso non si può dire dell’oggetto richiesto. Perciò, la richiesta stessa dell’informazione fa sì che anche i costi marginali non equivalgano più a zero.

Riportiamo questo mito per dimostrare la sua non attinenza con gli argomenti economici e utilitaristici in favore dell’open source: come vedremo in seguito, questi argomenti, in generale, reggerebbero anche supponendo che i software presentino veramente una struttura del valore, diversa da zero, tipica di un bene industriale. Non c’è dunque bisogno di trattare il problema se il software "debba" essere libero o meno.

Dopo aver analizzato un modello assai diffuso con occhio critico, vediamo di costruirne un altro: una spiegazione economica solida di che cosa possa rendere sostenibile la cooperazione open source.

Si tratta di un quesito che richiede un'indagine a un paio di livelli diversi. Al primo livello, occorre spiegare il comportamento dei singoli individui che collaborano a progetti open source; al secondo, occorre comprendere le forze economiche che sostengono la cooperazione a progetti open source come Linux o Apache.

Ancora una volta, occorre, innanzitutto, confutare un modello divulgativo assai diffuso che ostacola la comprensione. Sopra ogni tentativo di spiegare il comportamento cooperativo aleggia l'ombra della Tragedia dell'area comune di Garret Hardin[1].

Hardin, com'è noto, ci chiede di immaginare un prato a uso comune in un villaggio di coltivatori, che vi conducono le proprie mandrie al pascolo. Ma gli effetti del pascolo degradano l'area comune, estirpando l'erba e lasciando macchie brulle in cui l'erba ricresce molto lentamente. Se manca una politica condivisa (e messa in atto!) per la distribuzione dei diritti di pascolo, al fine di prevenire gli eccessi, l'interesse di ciascuna delle parti starà nel far pascolare il maggior numero di capi possibile al più presto possibile, nel tentativo di trarre il massimo valore dal pascolo, prima che degeneri in un mare di fango.

La maggior parte di noi ha un modello intuitivo del comportamento cooperativo che funziona più o meno in questo modo. In realtà, non si tratta di una diagnosi valida per i problemi economici dell'open source, che sono legati al free riding [2] (e quindi alla scarsità di ricavi) piuttosto che alla congestione di un bene pubblico (per sovrautilizzo).

Ciononostante, questa è l'analogia che sento nella maggior parte delle obiezioni improvvisate.

La tragedia dell'area comune non fa che predire tre possibili risultati. Uno è il mare di fango. Un altro è che una figura con poteri coercitivi imponga una politica di ripartizione delle aree a nome del villaggio (una soluzione comunista). Il terzo è che l'area comune venga suddivisa via via che i vari membri del

villaggio recingono aree che sono in grado di difendere e di mantenere in modo sostenibile (la soluzione del diritto di proprietà).

Quando, con una riflessione, si applica questo modello alla cooperazione open source, ci si aspetta che avrà vita molto breve. Poiché non c'è alcun modo ovvio per mettere in atto una politica di ripartizione del tempo trascorso dai programmatori su Internet, questo modello conduce direttamente alla previsione che l'area comune si spezzetterà, che i vari software diverranno commerciali, e che ci sarà sempre meno lavoro disponibile nello spazio comune.

In effetti, la realtà empirica prova che la linea di tendenza è quella opposta. L'ampiezza e il volume dello sviluppo open source (misurati, per esempio, attraverso i contributi pervenuti ogni giorno a Metalab o gli annunci pubblicatiogni giorno su freshmeat.net) sono in crescita costante. È chiaro che il modello della tragedia dell'area comune non è in grado, per motivi sostanziali, di descrivere quello che sta succedendo in realtà.

Un aspetto della risposta si trova senz'altro nel fatto che l'utilizzo dei software non ne diminuisce il valore. Anzi, l'uso massiccio di software open source tende a farne *aumentare* il valore, con la possibilità da parte degli utenti di aggiungere correzioni e funzionalità a loro piacimento ("patches" del codice). In quest'area comune all'inverso, l'erba cresce più alta quando si è in tanti a pascolare.

Un altro aspetto della risposta si trova nel fatto che il presunto valore di mercato di una piccola correzione apportata al codice sorgente è difficile da stimare. Supponiamo che io scriva una correzione per un difetto seccante e supponiamo anche che molti si accorgano che la mia correzione ha un valore in denaro: ma come faccio a ricavarlo da tutte quelle persone? I sistemi di pagamento tradizionali hanno costi aggiuntivi abbastanza alti da costituire un problema per le tipologie di micropagamento che si rivelerebbero adatte in questi casi.

Forse, sarebbe più appropriato affermare che questo valore non è solo difficile da stimare, ma che, in generale, è anche difficile da *assegnare*. Supponiamo, a titolo di esempio, che Internet sia fornita di un sistema di micropagamento ideale, almeno a livello teorico: sicuro, accessibile a tutti, senza spese aggiuntive. Ora supponiamo che voi abbiate scritto una patch dal titolo "Correzioni varie per il kernel di Linux". Come stabilirete il prezzo da chiedere?

E come potrà l’eventuale acquirente, non avendo ancora visto la patch, valutare quanto sia ragionevole sborsare per averla?

Quella che abbiamo di fronte è una specie di immagine a galleria degli specchi di ciò che F. A. Hayek [3]chiamò "il problema del calcolo": servirebbe un superuomo, capace di stabilire il valore funzionale delle correzioni e abbastanza fidato da fissare i giusti prezzi, per oliare gli ingranaggi del commercio.

Sfortunatamente, c’è una seria carenza di superuomini, per cui l’autore di patches, signor John Smith Hacker, ha solo due possibilità: o tenersi la patch, oppure metterla gratuitamente nello spazio comune. La prima scelta non presenta alcun guadagno. La seconda potrebbe non presentare un guadagno, ma potrebbe anche incoraggiare la distribuzione a catena ad altri utenti che, in futuro, si rivolgerebbero al signor John Smith per eventuali problemi. La seconda scelta, apparentemente altruistica, è in realtà egoista al punto giusto, nella prospettiva della teoria dei giochi.

In un’analisi di questa tipologia di cooperazione, è importante notare che, se esiste un problema di free riding (il lavoro può essere poco remunerativo in assenza di denaro o di altri compensi equivalenti) esso non si aggrava in proporzione al numero degli utenti. La complessità e il costo delle comunicazioni che ruotano intorno a un progetto open source sono da considerarsi quasi del tutto una funzione del numero di sviluppatori coinvolti: la presenza di più utenti che non esaminano mai il codice sorgente, in realtà, non costa nulla. Può far aumentare il numero di domande banali che compaiono nella mailing list del progetto, ma si può facilmente farvi fronte gestendo una pagina di Frequently Asked Questions e ignorando candidamente i quesiti da cui risulti ovvio che la pagina non è stata letta (e, infatti, entrambi i comportamenti sono consueti).

I veri problemi creati dal free riding, nel campo del software open source, dipendono più che altro dai costi contingenti alla distribuzione stessa delle patches. Un collaboratore che abbia poco da perdere nel gioco culturale della reputazione (vedi [HtN]), in assenza di un compenso in denaro, potrebbe pensare: "Non vale la pena distribuire questa correzione, perché dovrei ripulire la patch, scrivere un ChangeLog e firmare le carte di licenza della Free Software Foundation...". È proprio per questo che il numero dei collaboratori (e, in seconda istanza, il successo dei progetti) è fortemente e inversamente proporzionale alle difficoltà cui l’utente deve far fronte per collaborare a un progetto. Tali costi contingenti possono essere di natura sia politica, sia meccanica, e insieme

possono spiegare perché una cultura sconnessa e amorfa come quella di Linux abbia attratto enormemente più energie cooperative rispetto a sforzi organizzati sistematicamente e più centralizzati, come BSD, e perché la FSF abbia iniziato a perdere importanza con la diffusione di Linux.

Tutto questo va benissimo, finché dura. Si tratta, però, di una spiegazione post factum di ciò che fa il signor John Smith Hacker della sua patch, una volta pronta. Ma ci manca ancora una spiegazione economica di come questo signore, tanto per cominciare, sia riuscito a scrivere la patch, invece di lavorare su un programma commerciale che avrebbe potuto apportargli un guadagno sul valore di vendita. Quali sono, quindi, i modelli commerciali che creano nicchie in cui possa fiorire lo sviluppo dell'open source?

Note

1. ↑ Garret Hardin: professore emerito di Biologia presso l'Università di Santa Barbara (CA), Hardin è l'autore, fra l'altro, di un saggio dal titolo "w:The Tragedy of Commons", considerato un contributo fondamentale all'ecologia, alla teoria del contenimento demografico e all'economia politica *(N.d.T)*
2. ↑ Free riding: nel linguaggio economico, indica l'accesso a un servizio che consente l'ottenimento di un beneficio senza oneri per sé. L'esistenza di free riders nel campo dell'open source è una diretta conseguenza della disponibilità stessa del codice sorgente. *(N.d.T)*
3. ↑ F. A. Hayek (1899-1992): vincitore del premio Nobel per l'Economia nel 1974, Hayek fu tra i pionieri della teoria monetaria e principale proponente della dottrina del libero arbitrio nel ventesimo secolo. Ha insegnato alle Università di Londra, Chicago e Friburgo. La sua dottrina sostiene che, per effettuare un calcolo economico, occorre un mercato dei mezzi di produzione, in mancanza del quale non c'è alcun modo per stabilire il valore dei suddetti mezzi e, di conseguenza, alcun modo per trovare le giuste modalità di produzione *(N.d.T)*

Prima di passare a una tassonomia dei modelli aziendali open source, ci occuperemo dei vantaggi dell'esclusione in generale. Che cosa proteggiamo esattamente, quando chiudiamo l'accesso al codice sorgente?

Supponiamo che incarichiate qualcuno di scrivere un pacchetto software di contabilità, specifico per la vostra impresa. Il problema non sarà più facile da risolvere, che il codice sorgente sia aperto o chiuso: l'unica motivazione razionale per tenerlo segreto è se si intende vendere il pacchetto ad altre persone o impedirne l'uso da parte della concorrenza.

La risposta più ovvia è che stiate proteggendo il valore di vendita, ma questo non si applica al 95% dei software scritti per uso interno. Quindi, quali altri benefici apporta la chiusura?

La seconda possibilità (proteggere il vantaggio competitivo) richiede un esame più attento. Supponiamo che optiate per la disponibilità del codice sorgente del pacchetto contabile. Il pacchetto si diffonderà e migliorerà, grazie ai perfezionamenti apportati dalla comunità. Ma anche la vostra concorrenza inizierà a usarlo, ne trarrà beneficio senza pagare costi di sviluppo e minaccerà la vostra impresa. Può considerarsi un argomento contro l'open source?

Può darsi, ma può darsi anche di no. Il problema reale è se il beneficio ottenuto dalla ripartizione degli oneri di sviluppo supera le perdite dovute all'aumento della competizione a causa del free riding. Molti tendono a fare ragionamenti semplicistici su questo argomento, (a) ignorando il vantaggio funzionale derivante dall'impiego di più addetti allo sviluppo; (b) non considerando i costi di sviluppo come costi sommersi. Tanto per formulare un'ipotesi, dal momento che i costi di sviluppo sarebbero da pagare ugualmente, sarebbe errato calcolarli come costi dell'open source (qualora si scelga questa strada).

Ci sono altre ragioni puramente irrazionali che motivano la non disponibilità del codice sorgente. Potreste, per esempio, star lavorando nell'illusione che il software commerciale tuteli l'impresa contro i cracker e gli intrusi. Se è così, vi consiglio immediatamente una conversazione a scopo terapeutico con un crittografo. I veri paranoici del lavoro sanno che è meglio non fidarsi della sicurezza dei programmi commerciali, perché la dura esperienza ha insegnato loro che è meglio non farlo. La sicurezza è un aspetto dell'affidabilità: soltanto gli

algoritmi e le applicazioni che siano stati accuratamente controllati e rivisti da più persone potranno essere considerati sicuri.

Un fattore chiave evidenziato dalla distinzione tra valore d'uso e di vendita è che solo il *valore di vendita* è minacciato dal passaggio dal software commerciale all'open source, contrariamente al valore d'uso.

Se il valore d'uso, anziché il valore di vendita, è veramente il principale propulsore dello sviluppo del software e se (come spiegato in [CatB]) lo sviluppo dell'open source è veramente più efficace ed efficiente di quello commerciale, allora dobbiamo aspettarci di trovare circostanze in cui il presunto valore d'uso sia sufficiente per finanziare lo sviluppo dell'open source in modo sostenibile.

E, in effetti, non è difficile individuare almeno due modelli degni di nota in cui il salario di uno sviluppatore open source a tempo pieno proviene interamente dal valore d'uso.

Supponiamo che lavoriate per una ditta che, per condurre i propri affari, abbia necessità di un webserver molto potente e altrettanto affidabile. Potrebbe trattarsi di commercio elettronico o di un'azienda pubblicitaria di grande rilievo o anche di un portale Web. Comunque sia, vi serve un collegamento sette giorni su sette e ventiquattro ore su ventiquattro, vi servono velocità e possibilità di personalizzare.

Ma come ottenere tutte queste cose? Sono tre le strategie di base che si possono perseguire:

Acquistare un webserver proprietario. In questo caso, si scommette sul fatto che le priorità del produttore coincidano con le proprie e che il produttore abbia la competenza tecnica sufficiente per metterle in pratica nel migliore dei modi. Ma anche supponendo che entrambi questi requisiti sussistano, è probabile che il prodotto non sia molto personalizzabile: si potrà modificare soltanto attraverso gli appigli che il produttore avrà voluto fornirvi. Infatti, la strada del webserver proprietario non è molto popolare.

In proprio. Costruire il proprio webserver non è un'opzione da scartare in partenza: i webserver non sono molto complessi, e certamente lo sono meno dei browser. Inoltre, un webserver specializzato può essere molto potente ed efficiente. Percorrendo questa strada, si possono ottenere esattamente le prestazioni e le caratteristiche di personalizzazione desiderate, anche se si dovranno poi pagare in termini di tempo di sviluppo. Inoltre, la ditta potrebbe trovarsi in difficoltà, una volta che l'autore si licenzi o vada in pensione.

Unirsi al gruppo Apache. Il server Apache è stato costruito da un gruppo di webmaster in collegamento su Internet, che si sono accorti che era preferibile unire i propri sforzi per migliorare una base comune di codice piuttosto che gestire una moltitudine di sforzi di sviluppo paralleli. Così facendo, sono riusciti a ottenere quasi tutti i vantaggi derivanti dall'autonomia, ma anche la maggiore potenzialità di debugging derivante da una revisione effettuata da più persone in maniera intensiva e in parallelo.

I vantaggi offerti dalla scelta di Apache sono fortissimi. Per verificare quanto, basta consultare il sondaggio mensile Netcraft, che mostra la costante crescita di Apache sul mercato fin dai suoi inizi, contro tutti i webserver proprietari. Al giugno 1999, Apache e i suoi derivati occupano il 61% del

mercato, benché non ci sia, dietro questo progetto, alcun proprietario legale, alcuna forma di promozione né una vera e propria organizzazione di servizi.

La storia di Apache generalizza un modello in cui gli utenti di software trovano vantaggioso sostenere lo sviluppo dell'open source perché, così facendo, ottengono un prodotto migliore di come potrebbe essere altrimenti, a un costo inferiore.

Alcuni anni fa, due programmatori della Cisco (produttrice di apparecchiature per networking) hanno ottenutol'incarico di scrivere un sistema distribuito per la stampa in spooling, ad uso della rete aziendale di Cisco. È stata una vera e propria sfida. Oltre a supportare la possibilità di un ipotetico utente A di stampare su un'ipotetica stampante B (che avrebbe potuto trovarsi nella stanza accanto, così come a migliaia di chilometri), il sistema doveva garantire che, in mancanza di carta o di toner, il comando di stampa sarebbe stato ridiretto a un'altra stampante, vicina a quella ricevente il comando. Inoltre, il sistema avrebbe dovuto notificare problemi di questo tipo a un operatore di stampa.

I due programmatori riuscirono a effettuare una valida serie di modifiche al software standard Unix per la stampa in spooling, oltre ad alcuni script di contorno, che permisero il raggiungimento degli obiettivi. Ma poi si accorsero di avere un problema (e Cisco con loro).

Il problema era che nessuno dei due programmatori sarebbe rimasto alla Cisco per sempre. Alla fine, entrambi se ne sarebbero andati, e il software, rimasto senza manutenzione, avrebbe iniziato a deteriorarsi (cioè a perdere gradualmente sincronia con le condizioni del mondo reale). A nessuno sviluppatore piace veder accadere cose di questo genere al frutto del proprio lavoro, e l'intrepida coppia aveva capito che Cisco, nell'adottare questa soluzione, si era basata sul presupposto non troppo assurdo che sarebbe sopravvissuta ai loro due posti di lavoro.

Per questo, andarono dal loro capo e lo indussero ad autorizzare il rilascio del software per la stampa in spooling come open source. Le loro ragioni ruotavano intorno al fatto che Cisco non aveva alcun valore di vendita da perdere, mentre aveva molte altre cose da guadagnare. Incoraggiando la formazione di una comunità di utenti e di co-sviluppatori diffusi tra molte grandi imprese, Cisco avrebbe potuto effettivamente difendersi contro i rischi legati alla perdita degli autori originali del software.

La storia della Cisco generalizza un modello in cui l'open source serve non tanto ad abbassare i costi, quanto a ripartire il rischio. Tutte le parti in gioco si accorgono che la disponibilità del codice sorgente, così come la presenza di una comunità che agisce in collaborazione, finanziata da più fonti di reddito

indipendenti, fornisce una garanzia di sopravvivenza che ha un valore economico intrinseco o, comunque, un valore sufficiente per finanziarla.

L’open source rende piuttosto difficile trarre un valore di vendita diretto dai software. Non si tratta di una difficoltà tecnica: il codice sorgente non è né più né meno copiabile del binario e l’istituzione del copyright e di diritti di licenza che permettano la riscossione del valore di vendita, non sarebbe necessariamente più difficile per i prodotti open source che per quelli commerciali.

La difficoltà sta piuttosto nella natura del contratto sociale che si trova alla base dello sviluppo open source. Per tre motivi che si rinforzano a vicenda, la principale licenza open source proibisce la maggior parte delle forme di restrizione d’uso, ridistribuzione e modifica che faciliterebbero la riscossione di profitti derivanti dal valore di vendita. Per comprendere questi motivi, occorre esaminare il contesto sociale nel quale si sono evolute le licenze: la cultura hacker di Internet <http://www.catb.org/~esr/faqs/hacker-howto.html>.

Nonostante i miti sulla cultura hacker ancora assai diffusi (nel 1999) al suo esterno, nessuna di queste ragioni ha a che fare con l’ostilità nei confronti del mercato. Se è vero che una minoranza di hacker resta ancora ostile alla logica del profitto, la volontà generalizzata da parte della comunità di cooperare con aziende i cui prodotti si basano su Linux, come Red Hat, SUSE e Caldera dimostra che la maggioranza degli hacker sarebbe felice di lavorare a fianco del mondo dell’impresa, quando è nei loro interessi. Le ragioni reali per cui gli hacker guardano con scetticismo alle licenze che consentono guadagni diretti, sono assai più sottili e interessanti.

Una delle ragioni riguarda la simmetria. Mentre la maggior parte degli sviluppatori open source non oppone obiezioni sostanziali al fatto che altri traggano profitto dai loro doni, i più chiedono che nessuno (ad eccezione, eventualmente, del creatore di una parte del codice) si trovi in una posizione privilegiata per fare profitti. Il signor John Smith Hacker è d’accordo che la Fubarco tragga profitti dalla vendita del suo software o delle sue patches, ma solo nel caso che anche lui stesso, almeno in teoria, possa farlo.

Un’altra ragione riguarda le conseguenze indesiderate. Gli hacker hanno osservato che le licenze che comportano la restrizione e il pagamento di contributi per uso "commerciale" o vendita (il tentativo più comune e, a prima vista, nemmeno tanto assurdo, di recuperare il valore di vendita diretto) hanno effetti davvero inquietanti. Un caso specifico è la possibilità di controllare segretamente

la legalità di attività come la ridistribuzione in antologie economiche su CD-ROM, che in linea di massima sarebbero da incoraggiare. Più in generale, le restrizioni sull'uso, la vendita, le modifiche, la distribuzione (e altre clausole presenti sulle licenze) comportano spese aggiuntive per la conformità della distribuzione e (con l'aumento dei pacchetti in circolazione e in utilizzo) una combinazione esplosiva di incertezza soggettiva e potenziali rischi legali. Poiché questi risultati si considerano dannosi, si registra una forte pressione sociale per mantenere le licenze semplici ed esenti da restrizioni.

L'ultima e più importante ragione riguarda la dinamica di revisione e di cultura del dono descritta in [HtN]. Le restrizioni sulle licenze, designate per proteggere la proprietà intellettuale o per percepire un valore di vendita diretto, hanno spesso l'effetto di rendere legalmente impossibile la biforcazione dei progetti (questo avviene, per esempio, per le licenze di Sun, le cosiddette "Community Source", per Jini e Java). Se la biforcazione incontra molto scetticismo ed è considerata come l'ultima risorsa (per le ragioni spiegate diffusamente in [HtN]), la presenza di quest'ultima risorsa si considera di capitale importanza in caso di incompetenza o defezione del gestore (a causa, per esempio, di una licenza troppo restrittiva).

La comunità hacker mostra una certa elasticità in materia di simmetria: per questo, tollera licenze, come NPL di Netscape, che concedono alcuni privilegi in materia di profitti agli autori originali del codice (nel caso specifico di NPL, il diritto esclusivo di utilizzare il codice open source di Mozilla in prodotti derivati, anche commerciali). Ma ha meno elasticità nei confronti delle conseguenze indesiderate e nessuna sul mantenimento della possibilità di biforcazione (ed è per questo che gli schemi elaborati dalla Sun per la "Community License" di Java e Jini sono stati in gran parte rifiutati dalla comunità).

Queste ragioni spiegano le clausole della Open Source Definition, scritta per esprimere il parere congiunto della comunità hacker sui requisiti essenziali delle licenze standard (GPL, licenza BSD, licenza MIT e Licenza Artistica). Queste clausole hanno l'effetto (ma non l'intenzione) di rendere molto difficile l'ottenimento di un valore di vendita.

Ciononostante, esistono modi per costruire mercati nei servizi legati al software che comportino un valore di vendita indiretto. Sono cinque i modelli conosciuti e due quelli teorici (ma se ne potranno sviluppare altri in futuro).

In questo modello, si usa il software open source per creare o mantenere una posizione sul mercato per un software proprietario che genera una fonte diretta di profitti. Nella variante più comune, un software client "libero" agevola le vendite di un software server, o profitti da abbonamento/pubblicità in relazione a un portale Web.

La Netscape Communications perseguiva questa strategia quando, all'inizio del 1998, rese disponibile il codice sorgente del browser Mozilla. Il giro d'affari legato al browser ammontava al 13% dei guadagni ed era in calo, in seguito al lancio di Internet Explorer da parte della Microsoft. Le intense attività di marketing concentrate su Internet Explorer (nonché l'equivoca vendita congiunta di sistema operativo e software, che sarebbe stata al centro di una causa legale antitrust) hanno ben presto intaccato la fetta di mercato del browser Netscape, creando preoccupazioni che la Microsoft intendesse monopolizzare il mercato dei browser, per poi sfruttare il proprio controllo de facto sull'HTML, per eliminare Netscape dal mercato server.

In effetti, rendendo disponibile il codice sorgente del browser Netscape, ancora assai diffuso, Netscape ha negato alla Microsoft la possibilità di istituire un monopolio del browser. Netscape prevedeva che lo spirito di collaborazione dell'open source avrebbe accelerato lo sviluppo e il debugging del browser. Sperava altresì che Microsoft Internet Explorer si sarebbe trovato costretto a tenere il passo e dunque impossibilitato a definire l'HTML in modo esclusivo.

Questa strategia ha funzionato. Nel novembre 1998, infatti, Netscape ha cominciato a riguadagnare la sua quota di mercato rispetto a Internet Explorer. Al momento dell'acquisizione di Netscape da parte di America On-Line (AOL), all'inizio del 1999, il vantaggio competitivo di aver mantenuto in gioco Mozilla era abbastanza chiaro perché uno dei primi impegni dichiarati da AOL fosse proprio quello di continuare a sostenere il progetto Mozilla, benché si trovasse ancora allo stadio iniziale.

Questo modello riguarda i produttori di hardware (hardware, in questo contesto, comprende tutto a cominciare da schede Ethernet e simili fino a interi sistemi). Le pressioni del mercato hanno costretto le società che trattano hardware a scrivere e gestire software (dai device driver agli strumenti di configurazione, fino a interi sistemi operativi), ma il software stesso non è fonte di profitti, bensì una spesa aggiuntiva, spesso consistente.

In questa situazione, la disponibilità del codice sorgente non dà molto da pensare. Non ci sono guadagni da perdere, né aspetti negativi. Ciò che il produttore acquista è una risorsa di sviluppo di gran lunga maggiore, una risposta più rapida e flessibile alle esigenze dei clienti e maggiore affidabilità, grazie alla possibilità di revisione reciproca. Inoltre, apre la strada ad altri ambienti gratuitamente e, probabilmente, accresce anche la fedeltà dei clienti, in quanto il personale tecnico dedica più tempo al codice per esaudire le richieste di personalizzazione della clientela.

Ma i produttori, solitamente, sollevano alcune obiezioni specifiche nei confronti della disponibilità del codice sorgente per i driver hardware. Invece di mescolarle a discussioni di carattere più generale in questa sede, ho dedicato un’intera appendice a questo argomento in particolare.

Il margine di sicurezza dell’open source è particolarmente evidente nel campo del debugging di elementi di interfaccia ("widget frosting"). I prodotti hardware hanno produzione e tempi di assistenza limitati, scaduti i quali, i clienti si ritrovano da soli. Ma se hanno accesso al codice di altri driver, con la possibilità di utilizzarli per produrre patches secondo il bisogno, è più probabile che rimangano clienti della stessa società.

Un esempio assai illuminante dell’adozione del "widget frosting" ci è fornito dalla decisione da parte di Apple Computer, a metà marzo del 1999, di rendere disponibile il codice sorgente di "Darwin", il fulcro del sistema operativo MacOSX server.

In questo modello, si rende disponibile il codice sorgente di un software per creare una nicchia di mercato non nel settore del software commerciale (come nel caso Articolo civetta/posizionatore sul mercato), bensì nei servizi.

(Tempo fa, chiamavo questa strategia "Regalare il rasoio, vendere le lame", ma il legame non è tanto stretto quanto implica l'analogia tra rasoio e lama.)

Ecco come agiscono Red Hat e altri distributori di Linux. Ciò che vendono, in realtà, non sono i software, ossia i bit, ma il valore aggiunto nell'assemblaggio e nelle prove di un sistema operativo funzionante e con garanzie (anche solo implicite) di commerciabilità e compatibilità con altri sistemi operativi della stessa marca. Altri elementi del valore da essi offerto comprendono l'assistenza gratuita per l'installazione e l'offerta di opzioni per contratti di assistenza continuativi.

Le possibilità di costruire mercati nell'open source sono estremamente significative, specie per imprese che si trovino, per propria natura, nel settore dei servizi. Un caso recente e assai istruttivo è rappresentato da Digital Creations, un'impresa di Web design aperta nel 1998 e specializzata in banche dati complesse e siti per la gestione di transazioni. Il suo strumento principale, la punta di diamante della proprietà intellettuale dell'impresa, è un object publisher che oggi, dopo essere passato attraverso diversi nomi e forme, si chiama Zope.

Quando Digital Creations decise di cercare un capitale di rischio, l'esperto convocato valutò la futura nicchia di mercato dell'impresa, il suo personale e i suoi strumenti, per poi consigliare alla Digital Creations di portare Zope all'open source.

Stando agli standard tradizionali dell'industria del software, questa sembrerebbe una mossa completamente folle. La saggezza convenzionale, acquisita alla scuola di economia, ci insegna che una proprietà intellettuale vitale come Zope rappresenta la punta di diamante di un'impresa e che non deve essere venduta in alcun caso. Ma l'esperto aveva due argomenti a suo sostegno: il primo, che il vero patrimonio di Zope, in realtà, erano i cervelli e le abilità delle persone in esso impegnate; il secondo, che probabilmente, Zope avrebbe prodotto più valore nel costruire un mercato che come strumento segreto.

Per capire tutto questo, paragonate le due situazioni. In quella convenzionale, Zope rimane l’arma segreta della Digital Creations. Ammettiamo pure che sia molto efficace. Il risultato è che l’impresa sarà in grado di ottenere una qualità superiore in tempi brevi, ma nessuno lo saprà. Sarà facile soddisfare i clienti, ma più difficile sarà iniziare con l’accattivarsi una clientela di base.

L’esperto, invece, capì che uno Zope open source avrebbe rappresentato una pubblicità importantissima per il vero patrimonio della Digital Creations: le persone. Ipotizzò che i clienti, valutando le prestazioni di Zope, avrebbero considerato più efficace contattare gli esperti, piuttosto che sviluppare una gestione interna di Zope.

Da allora, uno dei rappresentanti di Zope conferma pubblicamente che la strategia open source ha "aperto molte porte che non si sarebbero trovate altrimenti". I potenziali clienti non mancano di rispondere alla logica sottesa a questa situazione e, come previsto, Digital Creations è in auge.

Un altro esempio recentissimo è fornito da e-smith, inc.. Questa società vende contratti di assistenza per un server Internet pronto all’uso e open source, un adattamento di Linux. Uno dei rappresentanti, descrivendo la diffusione dei software e-smith scaricati gratuitamente, ha affermato: "Per la maggioranza delle imprese, questa sarebbe pirateria del software: per noi, è libero mercato."

In questo modello, si vendono accessori per il software open source. Allo scalino più basso, tazze da tè e magliette; a quello più alto, documentazione redatta e prodotta a livello professionale.

La O'Reilly Associates, editrice di molti manuali eccellenti sul software open source, è un ottimo esempio di impresa fornitrice di accessori. Per la precisione, O'Reilly impiega famosi hacker operanti nell'open source (tra cui Larry Wall e Brian Behlendorf), come metodo per costruirsi una reputazione nel mercato di sua scelta.

In questo modello, si rilascia il software in file binario e codice sorgente con una licenza commerciale, ma contenente una data di scadenza della non disponibilità del codice sorgente. Ad esempio, si potrebbe scrivere una licenza che permetta la libera ridistribuzione, proibisca l'uso commerciale senza pagamento di contributi e garantisca che il software rientri nei termini della licenza GPL un anno dopo il rilascio o in caso di cessata attività del produttore.

Con questo modello, i clienti hanno la certezza che il prodotto sia personalizzabile secondo i bisogni, in quanto ne hanno il codice sorgente. Il prodotto è ad alto margine di sicurezza: la licenza garantisce che una comunità open source possa rilevare il prodotto, qualora l'impresa originale si estingua.

Poiché il prezzo e il volume delle vendite si basano su queste aspettative del cliente, l'impresa originale dovrebbe vedere aumentare i propri profitti grazie a un prodotto di questo tipo, più che con una licenza esclusivamente commerciale. Inoltre, dal momento che il codice originale è sotto licenza GPL, sarà comunque passato in rassegna seriamente da più persone, subirà correzioni e altri aggiustamenti che solleveranno l'autore da una parte del 75% di oneri di manutenzione.

Questo modello è stato seguito con successo da Aladdin Enterprises, gli autori del famoso programma Ghostscript (un interprete PostScript in grado di tradurre nel linguaggio originale di molte stampanti).

Lo svantaggio principale di questo modello è che le clausole commerciali tendono a inibire la revisione reciproca, nonché la partecipazione, proprio agli inizi del ciclo di produzione, quando ce n'è più bisogno.

Questo modello rappresenta una strategia aziendale in prospettiva. Si procede all'open sourcing di una tecnologia software, mantenendo una sequenza di prove o insieme di criteri di compatibilità, per poi vendere agli utenti un marchio che certifichi la compatibilità dello strumento tecnologico in loro possesso con tutti gli altri della stessa marca.

(È così che Sun Microsystems dovrebbe gestire Java e Jini.)

Questo modello, ancora una volta, rappresenta una strategia aziendale in prospettiva. Immaginiamo una specie di servizio di borsa telematica in abbonamento. Il valore non è da ricercare né nel software client, né nel server, bensì nell'oggettiva affidabilità delle informazioni fornite. Quindi, si rende disponibile il codice sorgente di tutti i software e si vendono abbonamenti per accedere ai contenuti. Via via che gli hacker aprono il client a nuove prestazioni e lo migliorano in vari modi, il mercato si espande automaticamente.

(Ecco perché AOL dovrebbe effettuare l'open sourcing del suo software client.)

Dopo aver passato in rassegna alcuni modelli di business che sostengono lo sviluppo del software open source, possiamo affrontare il quesito più generale: quando è ragionevole, in termini economici, utilizzare l'open source e quando è preferibile il software commerciale. Innanzitutto, occorre chiarire i vantaggi di ciascuna strategia.

L’approccio commerciale consente di raccogliere profitti dai propri bit segreti; d’altra parte, chiude le possibilità di un’autentica revisione reciproca e autonoma. L’approccio open source pone le condizioni per la revisione autonoma da parte di più individui, ma non fa riscuotere alcun profitto per i bit segreti.

Il vantaggio di tenere segreti i bit si comprende facilmente: per tradizione, le strategie aziendali centrate sul software sono state costruite intorno a esso. Fino a poco tempo fa, invece, il vantaggio della revisione autonoma non era di facile comprensione. Il sistema operativo Linux, tuttavia, ci insegna una lezione che probabilmente avremmo dovuto imparare già anni fa dalla storia del core software di Internet e da altre branche dell’ingegneria: la revisione reciproca consentita dall’open source è l’unico metodo misurabile per raggiungere alta fedeltà e qualità.

In un mercato competitivo, perciò, una clientela alla ricerca di alta fedeltà e qualità premierà i produttori di software che scelgono l’open source scoprendo come mantenere un flusso di profitti basato sui servizi, sul valore aggiunto e su altri mercati complementari a quello del software. Questo è il fenomeno che sta dietro il sorprendente successo di Linux che, nato dal nulla nel 1996 è giunto alla fine del 1998 a oltre il 17% del mercato dei server aziendali e sembra sulla buona strada per dominare il mercato nel giro di due anni (all’inizio del 1999, IDC prevedeva che Linux sarebbe cresciuto più velocemente di tutti gli altri sistemi operativi messi insieme fino al 2003).

Un altro vantaggio dell’open source, di importanza quasi pari al primo, è la sua utilità come metodo per propagare standard "liberi" e costruire mercati intorno a essi. La fortissima crescita di Internet deve molto al fatto che nessuno possiede il protocollo TCP/ IP; nessuno, cioè, è in possesso di un "lucchetto" con cui controllare i protocolli di base di Internet.

L’effetto rete innescato dal successo dei protocolli TCP/ IP e di Linux è abbastanza chiaro e si riduce, in ultima analisi, a una questione di fiducia e simmetria: le potenziali parti coinvolte in un’infrastruttura comune hanno ragione di fidarsi maggiormente, se possono vedere come funziona in tutte le sue applicazioni e preferiranno un’infrastruttura in cui tutte le parti abbiano pari diritti a una in cui una sola parte goda di una posizione privilegiata da cui trarre profitti ed esercitare il proprio controllo.

Tuttavia, non è necessario ricorrere all'effetto rete per rilevare l'importanza delle questioni di simmetria per i consumatori di software. Razionalmente, nessun consumatore di software sceglierà di chiudersi in un circuito monopolistico controllato da un fornitore, divenendo così dipendente dal software commerciale, se ha a disposizione un'alternativa open source di qualità accettabile. Questo argomento si rafforza via via che il ruolo del software diventa decisivo per l'attività del consumatore: e più vitale è, meno tollerante sarà il consumatore nei confronti di un controllo da parte di terzi.

Infine, un vantaggio importante offerto dal software open source ai clienti, e collegato alla questione della fiducia, è il suo margine di sicurezza. Se il codice sorgente è disponibile, il cliente ha una via d'uscita in caso di fallimento del produttore. Ciò si rivela particolarmente importante per il debugging degli elementi di interfaccia, poiché l'hardware tende ad avere cicli di vita brevi, ma l'effetto è più generico e si traduce in un maggior valore del software open source.

Quando i profitti ottenuti dai bit segreti sono maggiori del rendimento da open source, è ragionevole, in termini economici, scegliere la soluzione commerciale. Quando invece è il rendimento da open source a superare i proventi dei bit segreti, è ragionevole, in termini economici, scegliere l'open source.

In sé, questa è un'osservazione banale. Ma non è più banale, se teniamo conto che il rendimento dell'open source è più difficile da calcolare e da prevedere rispetto ai profitti derivanti dai bit segreti: inoltre, tale rendimento è ampiamente sottovalutato, più spesso che sopravvalutato. Anzi, prima che il mondo del business iniziasse a ripensare le proprie premesse in seguito alla diffusione del codice sorgente di Mozilla, all'inizio del 1998, i vantaggi dell'open source erano considerati, a torto ma assai diffusamente, equivalenti a zero.

Come valutare, dunque, i vantaggi dell'open source? È una domanda difficile, in generale, ma la si può affrontare come qualsiasi altro problema di natura predittiva. Si può iniziare da alcuni casi in cui l'approccio open source abbia trionfato o fallito. Si può poi cercare di generalizzare, pervenendo a un modello che dia almeno un'idea qualitativa dei contesti in cui l'open source sia la formula vincente per l'investitore o per l'impresa che cerchi di massimizzare i rendimenti. Infine, si può tornare ai dati e tentare di raffinare il modello.

Dall'analisi presentata in [CatB], ci si potrebbe aspettare che l'open source abbia un alto rendimento quando (a) affidabilità, stabilità e misurabilità siano centrali e (b) la correttezza del design e dell'implementazione non siano efficacemente verificabili, se non attraverso la revisione reciproca e autonoma. (Il secondo criterio si ritrova in pratica nella maggior parte dei programmi non banali).

Il desiderio razionale da parte del consumatore di non ritrovarsi chiuso nel circuito monopolistico di un fornitore farà aumentare il suo interesse nei confronti dell'open source (e, di conseguenza, il valore sul mercato competitivo della scelta open source da parte dei fornitori) via via che il ruolo del software diventa centrale per il consumatore stesso. Perciò, un terzo criterio (c) spinge verso l'open source qualora il software sia un bene capitale di vitale importanza per l'impresa (come avviene, per esempio, nei dipartimenti aziendali di gestione informatica).

Per quanto riguarda l'area di applicazione, abbiamo osservato sopra come un'infrastruttura open source crei fiducia e simmetria che, nel tempo, tenderanno

ad attrarre nuova clientela e a vincere nella competizione contro il software commerciale; e, spesso, è meglio avere una quota più piccola di questo mercato in rapida espansione piuttosto che una quota più grande di un mercato esclusivamente commerciale e in stagnazione. Di conseguenza, per l'infrastruttura software, una strategia open source che miri all'ubiquità ha più probabilità di rivelarsi remunerativa nel lungo periodo rispetto a una strategia commerciale che miri all'ottenimento dei diritti di proprietà intellettuale.

In effetti, la capacità da parte dei potenziali clienti di ragionare sulle future conseguenze delle strategie del produttore e la loro riluttanza ad accettare il monopolio di un fornitore implica una limitazione più stringente: senza godere ancora di un indiscusso potere sul mercato, si può scegliere o una strategia di ubiquità open source o una strategia che punti a un guadagno diretto derivante dal software commerciale, ma non entrambe. (Esempi analoghi di questo principio sono riscontrabili altrove, per esempio sui mercati elettronici, in cui, spesso, i clienti rifiutano di acquistare profili costituiti dal solo codice sorgente.) L'esempio può anche essere presentato in termini meno negativi: dove domina l'effetto rete (gli effetti positivi delle cosiddette "esternalità di rete"[1]), è probabile che l'open source sia la formula vincente.

Potremmo riassumere questa logica osservando che l'open source sembra raggiungere il massimo dell'efficacia nell'aumentare i profitti rispetto al software commerciale, per software che (d) istituiscano o abilitino una comune infrastruttura di calcolo e comunicazioni.

Infine, si può evidenziare il fatto che i fornitori di servizi unici o anche solo altamente differenziati hanno più ragioni di temere la copia dei loro metodi da parte della concorrenza, rispetto ai produttori di servizi i cui algoritmi centrali e conoscenze di base siano largamente noti. Di conseguenza, l'open source ha più possibilità di prevalere quando (e) i metodi di punta (o equivalenti funzionali) fanno parte delle comuni conoscenze ingegneristiche.

Il core software di Internet, Apache e l'implementazione Linux dell'Application Program Interface (API) Unix con standard ANSI rappresentano esempi canonici di tutti e cinque i criteri. Il cammino verso l'open source nell'evoluzione di tali mercati è bene illustrato dal ritorno di convergenza dei dati su reti TCP/IP a metà degli anni Novanta, dopo quindici anni di tentativi falliti di costruire imperi con protocolli commerciali come DECNET, XNS, IPX e simili.

D’altra parte, la scelta dell’open source è assai meno sensata per le imprese che abbiano in loro possesso esclusivo una tecnologia software generante valore (cioè che soddisfi strettamente il criterio (e)) che sia (a) relativamente insensibile ai guasti; (b) facile da verificare con mezzi diversi dalla revisione reciproca e autonoma; e che non sia (c) centrale per l’impresa. Inoltre, il suo valore non deve poter aumentare in modo significativo (d) per l’effetto rete o per l’ubiquità.

Un esempio di questo caso estremo risale agli inizi del 1999, quando mi fu chiesto "Ci consiglia di passare all’open source?" da una società di software per il calcolo di schemi di taglio per segherie, che desiderava estrarre la maggior metratura possibile di assi dai tronchi di legname. La mia conclusione è stata "No". L’unico criterio che questo prodotto soddisfaceva, e neanche completamente, era il (c); ma al limite, un operatore esperto sarebbe anche in grado di realizzare gli schemi di taglio a mano.

Un punto importante è che la posizione di un prodotto su questa scala può variare nel tempo, come vedremo nel seguente caso di studio.

Riassumendo, le seguenti discriminanti fanno propendere per l’open source:

- (a) affidabilità/ stabilità/ misurabilità sono fondamentali;
- (b) la correttezza del design e dell’implementazione non possono essere verificate efficacemente se non tramite la

  revisione reciproca e indipendente;

- (c) il software è di vitale importanza per il controllo dell’impresa da parte dell’utente;
- (d) il software istituisce o abilita una comune infrastruttura di calcolo e comunicazioni;
- (e) i metodi di punta (o loro equivalenti funzionali) fanno parte delle comuni conoscenze ingegneristiche.

Note

1. ↑ Esternalità di rete: il termine "esternalità" si riferisce, nel lessico del business, agli effetti indiretti esercitati da un utente su altri utenti della stessa

rete. Il principio su cui si basa il concetto di "effetto rete" con tutti i suoi corollari, è quello secondo cui il valore di un bene aumenta con il numero dei suoi utenti. In base a questo principio, il valore di Internet o di un software, aumenta proporzionalmente al numero di utenti della Rete o del software. *(N.d.T)*

La storia del best seller fra i giochi della famiglia Id Software illustra alcuni modi in cui la pressione del mercato e l'evoluzione del prodotto possano modificare radicalmente l'entità dei vantaggi del software commerciale nei confronti dell'open source.

Quando Doom fu lanciato per la prima volta, alla fine del 1993, la sua animazione in prima persona e in tempo reale lo rendeva unico nel suo genere (l'antitesi del criterio (e)). L'impatto visivo della tecnica non era soltanto sorprendente: addirittura, per molti mesi, nessuno riuscì a capacitarsi di come fosse stato ottenuto, con i microprocessori scarsamente potenti di quei tempi. Questi bit segreti erano in grado di fruttare moltissimo. Inoltre, il rendimento potenziale da open source era basso. Come gioco individuale, il software: (a) comportava costi tollerabili in caso di guasto, (b) non era terribilmente difficile da sottoporre a verifica, (c) non era centrale per l'attività dei consumatori, (d) non beneficiava dell'effetto rete. Quindi, era razionale, in termini economici, che Doom fosse commerciale.

Tuttavia, il mercato intorno a Doom non rimase fermo. La futura concorrenza inventò equivalenti funzionali delle sue tecniche di animazione e cominciarono a comparire altri giochi "first-person shooter", come Duke Nukem. Via via che questi giochi intaccavano la quota di mercato di Doom, diminuiva il valore dei bit segreti.

D'altra parte, i tentativi di espandere la quota di mercato si scontrarono con nuove sfide tecniche: maggiore affidabilità, più schemi di gioco, aumento dell'utenza di base e piattaforme multiple. Con l'avvento del "deathmatch" multiplayer e dei servizi di simulazione Doom, il mercato ha iniziato a mostrare un significativo effetto di rete. Tutto ciò impegnava il tempo dei programmatori, che Id avrebbe preferito dedicare al gioco successivo.

Tutte queste tendenze accrebbero i vantaggi della disponibilità del codice sorgente. A un certo punto, le curve dei guadagni si incontrarono e, per Id, divenne economicamente razionale portare Doom a open source, per poi spostarsi, per fare profitti, su mercati secondari, come quello delle antologie di schemi di gioco. E qualche tempo dopo, tutto questo è successo veramente. Il codice sorgente completo di Doom fu diffuso alla fine del 1997.

Doom rappresenta un interessante caso di studio, in quanto non è né un sistema operativo, né un software di comunicazioni in rete. Pertanto, è assai lontano dagli esempi ovvi e canonici di successo nel settore open source. Anzi, il ciclo vitale di Doom, compreso il cambiamento di categoria, potrebbe assurgere a esempio del ciclo vitale delle applicazioni software nell'odierna ecologia del codice: un ambiente in cui le comunicazioni e il calcolo distribuito creano seri problemi di resistenza, affidabilità e misurabilità affrontabili soltanto con la revisione reciproca, ma anche frequenti passaggi di categoria sia tra vari ambienti tecnici, sia tra attori in concorrenza (con tutte le relative questioni di fiducia e simmetria).

Doom si è evoluto da gioco individuale a deathmatch. L'effetto rete si identifica sempre più con l'elaborazione. Tendenze simili sono sempre più evidenti anche nelle più consistenti applicazioni aziendali, come i pacchetti gestionali integrati, via via che le aziende costruiscono reti più fitte con i fornitori e con la clientela: e sono implicite, naturalmente, nell'intera architettura del World Wide Web. Ne consegue che quasi ovunque, i vantaggi dell'open source sono in crescita costante.

Se si manterranno le tendenze attuali, la sfida centrale della tecnologia del software e della gestione dei prodotti, nel prossimo secolo, sarà comprendere quando sia il momento buono per agire, cioè quando consentire al codice commerciale di passare all'infrastruttura open source, per sfruttare gli effetti della revisione reciproca e attrarre maggiori guadagni nel settore dei servizi e sui mercati secondari.

Sono ovvi gli incentivi economici dell'indovinare il punto critico per il cambio di categoria, non superandolo, né precedendolo di troppo. Oltre a questo, attendere troppo a lungo comporta un serio rischio di opportunità: quello di essere battuti da un concorrente che opti per l'open source nella stessa nicchia di mercato.

La ragione per cui si tratta di una questione seria è che, per qualsiasi categoria di prodotto, la risorsa rappresentata dalla clientela, così come le risorse di talento disponibili per il reclutamento nella cooperazione open source, sono limitate e il reclutamento tende a mantenersi nel tempo. Se due produttori sono rispettivamente il primo e il secondo a diffondere due codici sorgente in competizione tra loro e con funzioni pressoché uguali, il primo ha più possibilità

di attrarre il maggior numero di utenti e la maggioranza dei co-sviluppatori, nonché i più motivati. Il secondo dovrà accontentarsi degli avanzi. Il reclutamento tende a mantenersi, anche perché gli utenti prendono dimestichezza col prodotto e gli sviluppatori investono il loro tempo nel codice stesso.

La comunità open source si è organizzata in modo tale da tendere ad amplificare gli effetti di produttività dell'open source stesso. Nel mondo di Linux, in particolare, è significativo, in termini economici, il fatto che ci siano più distributori di Linux, in concorrenza tra loro, che costituiscono una categoria a parte rispetto agli sviluppatori.

Gli sviluppatori scrivono il codice e lo rendono disponibile su Internet. Ogni distributore seleziona un certo sottoinsieme del codice disponibile, lo integra, costruisce i pacchetti, gli dà un nome, per poi venderlo ai clienti. Gli utenti possono scegliere il tipo di distribuzione e possono anche completare una distribuzione scaricando il codice direttamente dai siti degli sviluppatori.

L'effetto di questa separazione di categorie è di creare un mercato assai fluido, che lascia spazio al miglioramento. Gli sviluppatori competono tra loro per l'attenzione di distributori e utenti, sulla qualità dei loro software. I distributori competono per ottenere il denaro degli utenti, sull'appropriatezza delle loro politiche di selezione, nonché sul valore che possono aggiungere al software.

Un effetto di primo ordine, in questa struttura interna del mercato, è che nessun nodo della rete è indispensabile. Gli sviluppatori possono recedere dall'attività e, anche se la loro porzione del codice di base non viene rilevata direttamente da un altro sviluppatore, la competizione per l'attenzione tenderà a generare rapidamente alternative funzionali. I distributori possono fallire senza danneggiare o compromettere la base comune del codice open source. L'ecologia, tutto sommato, ha una risposta più rapida alle esigenze di mercato e maggiori capacità di resistere ai traumi e di rigenerarsi, di quanto non potrà mai raggiungere una qualsiasi struttura monolitica che tratti sistemi operativi commerciali.

Un altro effetto importante è la diminuzione delle spese aggiuntive, accompagnata da un aumento dell'efficienza, dovuto alla specializzazione. Gli sviluppatori non subiscono le pressioni che compromettono abitualmente i progetti commerciali tradizionali, trasformandoli in veri e propri gineprai: niente liste insensate e fuorvianti di obblighi di marketing, niente ordini da parte della direzione di utilizzare linguaggi o ambienti fuori luogo e sorpassati, niente richieste di riscoprire l'acqua calda nei modi più assurdi e disparati in nome della differenziazione del prodotto o della tutela della proprietà intellettuale e (cosa più

importante) niente scadenze. Niente tour de force per lanciare la versione 1.0 prima ancora che sia pronta: e questo (come osservano DeMarco e Lister nella loro trattazione dello stile gestionale denominato "svegliatemi quando è finita", in [DL]), generalmente, conduce non soltanto a un incremento della qualità, ma anche alla massima rapidità nella distribuzione di risultati veramente operativi.

I distributori, d’altra parte, finiscono per specializzarsi nei compiti che possono svolgere nel modo più efficace. Liberi dalla necessità di sostenere progetti consistenti e continuativi per lo sviluppo del software soltanto per rimanere competitivi, possono concentrarsi sull’integrazione dei sistemi, sulla creazione di pacchetti, sulle garanzie di qualità e sui servizi.

Sia i distributori, sia gli sviluppatori si attengono ai principi di correttezza, costantemente interpellati e controllati come sono dalla loro utenza: anche questo è parte integrante del metodo open source.

La Tragedia dell’area comune potrebbe non essere applicabile allo sviluppo open source così come lo vediamo oggi, ma questo non significa che non sussistano i presupposti per chiedersi se l’attuale vigore della comunità open source sia sostenibile. I giocatori di punta abbandoneranno la cooperazione con la crescita dei guadagni?

Sono molti i livelli a cui si può porre questo quesito. Il nostro contro-racconto della "Commedia dell’area comune" si basa sul presupposto che il valore del contributo individuale all’open source sia difficile da tradurre in moneta. Ma questo presupposto è assai meno solido per le aziende (per esempio, i distributori di Linux) che godono già di un flusso di profitti legati all’open source. Il loro contributo viene già tradotto quotidianamente in moneta. Il loro ruolo cooperativo, quindi, può considerarsi stabile?

L’esame di questo quesito ci condurrà a un’analisi interessante dell’economia del software open source nel mondo reale contemporaneo e di ciò che implica, per l’industria del software del domani, un vero e proprio paradigma da industria dei servizi.

A livello pratico, applicata alla comunità open source come la conosciamo oggi, questa domanda viene formulata, solitamente, in due modi diversi. Il primo: Linux si frammenterà? Il secondo: di converso, Linux finirà per diventare un attore dominante, quasi monopolistico?

L’analogia storica che molti adottano nell’ipotizzare che Linux si frammenti è il comportamento dei produttori di versioni di Unix proprietarie negli anni Ottanta. Nonostante infinite discussioni sugli standard open source, nonostante numerose alleanze, consorzi e accordi, lo Unix proprietario è andato in pezzi. Il desiderio da parte dei produttori di differenziare i loro prodotti, aggiungendo o modificando le applicazioni del sistema operativo, si è dimostrato maggiore dell’interesse ad aumentare il volume totale del mercato Unix, mantenendo la compatibilità (e di conseguenza abbassando sia i vincoli d’ingresso agli sviluppatori autonomi di software, sia il costo totale del diritto di proprietà per i consumatori).

È abbastanza improbabile che questo avvenga a Linux, per il semplice motivo che tutti i distributori sono costretti a lavorare a partire da una base comune di codice open source. Per nessuno di loro è veramente possibile

mantenere la differenziazione, perché le licenze sotto cui ricade il codice di Linux li obbligano, in effetti, a condividere il codice con tutte le parti coinvolte. Nel momento in cui un distributore sviluppa una caratteristica, tutti i concorrenti sono liberi di clonarla.

Poiché tutte le parti comprendono questo concetto, nessuno pensa anche lontanamente di intraprendere il genere di manovre che hanno frammentato lo Unix proprietario. Al contrario, i distributori di Linux sono vincolati a competere con modalità che apportano benefici al consumatore e al mercato in generale. Devono, cioè, competere sui servizi e sull'assistenza, e la scommessa del loro design consiste nell'individuare le interfacce che portano alla facilità di installazione e di utilizzo.

La base comune del codice sorgente preclude anche la possibilità di monopolio. Quando gli operatori di Linux se ne preoccupano, solitamente si mormora il nome "Red Hat", quello del più grande e fortunato dei distributori (con una quota di mercato stimata a circa il 90% negli Stati Uniti). Ma è degno di nota il fatto che, pochi giorni dopo l'annuncio, nel maggio 1999, dell'attesissimo lancio di Red Hat 6.0 (prima ancora che i CD-ROM di Red Hat venissero distribuiti in quantità significative), le immagini dei CD-ROM appena rilasciati, create a partire dal sito FTP pubblico di Red Hat, venivano già pubblicizzate da un editore e da diversi altri distributori di CD-ROM, a prezzi inferiori rispetto al presunto listino della Red Hat.

La Red Hat non ha fatto una piega, perché i fondatori capiscono molto chiaramente di non possedere e di non poter possedere i bit del loro prodotto: le norme sociali della comunità Linux lo proibiscono. In un recente commento alla famosa affermazione di John Gilmore secondo cui Internet interpreta la censura come un danno e aggira l'ostacolo, è stato detto giustamente che la comunità hacker che si occupa di Linux interpreta i tentativi di controllo come un danno e le aggira. Se Red Hat avesse protestato per la clonazione in anteprima del suo ultimissimo prodotto, la sua possibilità di ottenere la cooperazione della comunità degli sviluppatori, in futuro, ne sarebbe uscita seriamente compromessa.

L'elemento forse più importante, al giorno d'oggi, è che le licenze per i software che esprimono queste regole comunitarie in forma vincolante a livello legale, proibiscono esplicitamente a Red Hat di monopolizzare il codice sorgente su cui si basa il loro prodotto. L'unica cosa che possono vendere è una relazione marca/servizio/assistenza con persone che scelgano liberamente di pagarsela. Non

si tratta di un contesto in cui la possibilità di un monopolio predatore incomba pericolosamente.

Esiste anche un altro senso in cui l'afflusso di denaro sta cambiando il mondo dell'open source. Sempre più, i migliori talenti della comunità scoprono di poter essere pagati per quello che desiderano fare, anziché coltivare l'open source come un hobby finanziato da un altro lavoro diurno. Le società come Red Hat, O'Reilly Associates e VA Linux Systems stanno costituendo un vero e proprio sistema per la ricerca semi-indipendente con contratti per assumere e mantenere talenti open source in pianta stabile.

Ciò ha senso, in termini economici, solo se il costo pro capite della gestione di un laboratorio del genere può essere facilmente assorbito dai guadagni presunti che l'azienda otterrà, facendo crescere più velocemente il mercato. O'Reilly può permettersi di stipendiare i principali autori di Perl e Apache per la gestione dei loro prodotti, perché si aspetta che il loro operato consenta di vendere più libri su Perl ed Apache. VA Linux Systems può investire nei suoi laboratori perché i miglioramenti apportati a Linux fanno aumentare il valore d'uso delle workstation e dei server che vende l'impresa. Infine, Red Hat finanzia i Laboratori di Sviluppo Avanzato Red Hat per aumentare il valore della sua offerta Linux e attrarre nuova clientela.

Per gli strateghi dei settori più tradizionali dell'industria del software, cresciuti in culture che considerano la proprietà intellettuale protetta da un brevetto (o dal segreto commerciale) come la punta di diamante di una società, questo comportamento può sembrare (nonostante il suo effetto di crescita sul mercato) inesplicabile. Perché mai finanziare una ricerca di cui ogni singolo concorrente è libero (per definizione) di appropriarsi a costo zero?

Sembrano esserci due ragioni dominanti. La prima è che, finché queste imprese restano leader nella loro nicchia di mercato, possono prevedere di aggiudicarsi la parte del leone dei proventi di un progetto di Ricerca e Sviluppo open source. D'altronde, l'utilizzo di Ricerca e Sviluppo per attrarre nuovi profitti non è certo un'idea originale: ciò che è interessante, piuttosto, è il calcolo implicito secondo cui i presunti guadagni futuri saranno sufficientemente ampi da permettere alle imprese di sopportare tranquillamente il free riding.

Se questa analisi basata sul presunto valore futuro è necessaria in un mondo di capitalisti agguerriti con gli occhi puntati sulla redditività del capitale investito, tuttavia non rappresenta la spiegazione più interessante del reclutamento di

talenti, in quanto le aziende stesse propongono un’analisi più nebulosa. Interrogate, vi risponderanno che stanno soltanto facendo la cosa giusta per la comunità da cui provengono. Ma il vostro umile scrittore conosce abbastanza i vertici di tutte e tre le imprese citate sopra per confermare che queste solenni affermazioni non possono essere bollate come imbrogli. Anzi, io stesso fui assunto da VA Linux Systems alla fine del 1998, con la funzione esplicita di consigliare loro "la cosa giusta", e li trovai assai ben disposti ad ascoltarmi.

Un economista ha il diritto di chiedere quale guadagno sia in gioco. Se siamo d’accordo che il discorso di fare la cosa giusta non è frutto di una vana dissimulazione, poi dovremo scoprire qual è l’interesse per l’azienda a fare "la cosa giusta". E la risposta, in sé, non è né sorprendente né difficile da verificare, a patto che si ponga la domanda giusta. Come altri comportamenti apparentemente altruistici in altre industrie, ciò che queste imprese contano di acquistare, in effetti, è la simpatia nei confronti dell’azienda e dei suoi prodotti.

Neanche l’idea di lavorare per guadagnarsi simpatia e valutarla come un bene in grado di predire i futuri guadagni sul mercato, è nuova. Ciò che è interessante è l’alta considerazione in cui le imprese tengono questa simpatia, come indica il loro comportamento. È dimostrabile che esse sono disposte ad assumere talenti, anche costosi, per progetti che non generano profitti diretti, perfino nelle fasi in cui c’è più bisogno di capitali, come nei periodi immediatamente precedenti l’offerta primaria di azioni.

I vertici di queste imprese sono abbastanza espliciti circa le ragioni per cui ritengono particolarmente importante la simpatia da parte della clientela. Contano molto sul lavoro volontario della clientela di base, sia per lo sviluppo dei prodotti, sia come strumento di marketing informale. Il loro rapporto con la clientela di base è profondo e spesso fa leva su legami di fiducia personale tra individui all’interno e all’esterno di una data impresa.

Queste osservazioni corroborano una lezione che abbiamo appreso in precedenza, da una diversa linea di ragionamento. La relazione tra Red Hat, VA, O’Reilly, la loro clientela e i loro sviluppatori non è tipica dell’impresa industriale. Ci mostra, piuttosto, un caso estremo e interessante che riproduce le caratteristiche delle industrie di servizi basate sul capitale intellettuale. Spaziando al di fuori delle industrie tecnologiche, riscontriamo questi parametri (per esempio) negli studi legali associati, nelle aziende mediche e nelle università.

In effetti, potremmo rilevare che le aziende di open source assumono hacker di primo piano più o meno per la stessa ragione per cui le università assumono studiosi di primo piano. In entrambi i casi, abbiamo un procedimento simile, per meccanismi ed effetti, al sistema del mecenatismo, che finanziava le belle arti fino alla Rivoluzione Industriale: si tratta di una somiglianza di cui alcune parti in causa sono perfettamente consapevoli.

I meccanismi di mercato che consentono di finanziare (e di trarne profitti!) lo sviluppo open source sono ancora in rapida evoluzione. I modelli aziendali che abbiamo passato in rassegna in questo saggio non saranno probabilmente gli ultimi a essere inventati. Gli investitori stanno ancora pensando attentamente alle conseguenze della reinvenzione dell'industria del software come struttura esplicitamente concentrata sui servizi, anziché sulla proprietà intellettuale esclusiva, e continueranno ancora per un bel po'.

Questa rivoluzione concettuale avrà dei costi sui vecchi profitti di chi investe nel 5% dell'industria che fa ancora leva sul valore di vendita: storicamente, le imprese nei servizi sono meno remunerative di quelle industriali (anche se, come qualsiasi medico o avvocato potrà confermarvi, i redditi degli addetti ai lavori, solitamente, sono più alti). Ma tutti questi profitti persi saranno più che colmati dai vantaggi in termini di spesa, visto che i consumatori di software si assicurano fortissimi risparmi e maggiore efficienza, con l'open source. (Si può istituire un parallelo, qui, con gli effetti che la sostituzione di Internet alla tradizionale rete telefonica vocale sta esercitando ovunque).

La promessa di tanto risparmio ed efficienza è di creare un'opportunità di mercato che imprenditori e investitori si stanno preparando a sfruttare. Mentre era in preparazione la prima bozza di questo saggio, la più prestigiosa società d'investimento in capitale di rischio della Silicon Valley ha acquisito quote di maggioranza nella prima società di inserimento specializzata nell'assistenza tecnica a Linux ventiquattro ore su ventiquattro, sette giorni alla settimana. Ci si aspetta che, prima della fine del 1999, verranno lanciate diverse offerte primarie di azioni relative a Linux e ad altri progetti open source e che riporteranno un discreto successo.

Un altro sviluppo assai interessante è rappresentato dai primi tentativi sistematici di costruire mercati specifici per lo sviluppo open source.
SourceXchange e CoSource rappresentano due tentativi leggermente diversi di applicazione del modello di "asta inversa" ("reverse auction")[1] al finanziamento dello sviluppo open source.

Le tendenze generali sono chiare. Abbiamo già citato le proiezioni dell'IDC secondo cui Linux crescerà più velocemente di tutti gli altri sistemi operativi messi insieme fino al 2003. Apache è al 61% della quota di mercato ed è in

crescita costante. L'utilizzo di Internet è in crescita esponenziale e i sondaggi come Internet Operating System Counter mostrano che Linux e altri sistemi operativi open source sono già maggioritari tra gli host che navigano Internet e in costante recupero rispetto ai sistemi commerciali. La necessità di utilizzare infrastrutture Internet open source condiziona sempre più non soltanto il design di altri software, ma anche le strategie d'impresa e gli schemi di utilizzo/acquisto di software per tutte le società esistenti. Queste tendenze, in linea di massima, sembrano orientate ad accelerare.

Note

1. ↑ Reverse auction: neologismo coniato nel 1996 da Hal Segal, fondatore dell'azienda TravelBid. Si tratta di un'asta in cui i compratori elencano gli articoli o prodotti (anche finanziari) che intendono acquistare, mentre i venditori ne propongono i prezzi. Si tratta di un tipo di asta che lascia più spazio all'acquirente rispetto alle tradizionali aste all'inglese e all'olandese, in quanto è il compratore a stabilire gli esatti requisiti del prodotto di suo interesse, anziché basare il proprio acquisto, come avviene nelle aste tradizionali, sulle caratteristiche presentate dal venditore *(N.d.T)*.

Che aspetto avrà il mondo del software una volta completata la transizione all'open source?

Per rispondere a questa domanda, sarà utile suddividere i software in base al grado di completezza con cui i loro servizi sono descrivibili secondo gli standard tecnici dell'open source, il che collima perfettamente col grado di mercificazione che ormai caratterizza il servizio di base.

Quest'asse corrisponde abbastanza bene a quello che si immagina generalmente, quando si pensa ad "applicazioni" (niente affatto mercificate, con standard tecnici open source deboli o addirittura inesistenti), "infrastrutture" (servizi mercificati, con standard rigidi) e "prodotti middleware" (parzialmente mercificati, con standard tecnici efficaci ma incompleti). I casi canonici di oggi, nel 1999, potrebbero essere un word processor (applicazione), uno stack TCP/IP (infrastruttura) e un motore di banca dati (prodotto middleware).

La precedente analisi del vantaggio in termini di guadagno lasciava intendere che infrastrutture, applicazioni e prodotti middleware si sarebbero trasformati in modi diversi e avrebbero presentato diverse combinazioni di equilibrio tra open source e software commerciale. Lasciava intendere altresì, lo ricordiamo, che la prevalenza dell'open source in un determinato settore del software sarebbe stata una funzione dell'entità dell'effetto rete, dei costi per eventuali guasti e del grado di importanza del software in qualità di bene capitale per l'azienda.

Possiamo azzardare alcune previsioni, applicando questi criteri euristici non già ai singoli prodotti, ma a interi settori del mercato del software. Cominciamo subito.

Le infrastrutture (Internet, Web, sistemi operativi e i livelli più bassi del software di comunicazione che deve attraversare i confini tra parti in competizione) saranno quasi tutte open source, mantenute in cooperazione tra consorzi di utenti e strutture di distribuzione/servizi a scopo di lucro con un ruolo come quello attualmente ricoperto da Red Hat.

Le applicazioni, d'altra parte, avranno una tendenza prevalente a restare commerciali. Si presenteranno circostanze in cui il valore d'uso di un algoritmo o di una tecnologia non divulgati sarà abbastanza alto (e in cui i costi associati all'inaffidabilità saranno abbastanza bassi e i rischi associati al monopolio del

fornitore sufficientemente sopportabili) per far sì che i consumatori continuino a pagare per un software commerciale. L'ambiente in cui il permanere di questa situazione è più probabile sono le applicazioni indipendenti sui mercati verticali, dove l'effetto rete è debole. L'esempio già citato delle segherie è uno di questi: è altamente probabile che un altro sia rappresentato dai software di identificazione biometrica, secondo le ultime previsioni del 1999.

I prodotti middleware (come i database, gli strumenti di sviluppo e gli strati personalizzati più elevati degli stack di protocolli delle applicazioni) rappresenteranno una categoria più mista. La tendenza delle categorie middleware a presentarsi come commerciali o open source dipende, probabilmente, dai costi associati ai guasti, dove costi maggiori esercitano pressioni di mercato per una maggiore disponibilità del codice sorgente.

Per completare la descrizione, comunque, occorre far notare che né le "applicazioni" né i "prodotti middleware" sono categorie stabili. Nel capitolo dal titolo "Capire quando è il momento", abbiamo visto che le singole tecnologie software sembrano attraversare un vero e proprio ciclo di vita naturale che le porta da razionalmente commerciali a razionalmente open source. La stessa logica si applica su grande scala.

Le applicazioni tendono a ricadere nella categoria dei prodotti middleware, via via che si sviluppano tecniche standardizzate e che alcune porzioni del servizio vengono mercificate. (I database, per esempio, sono diventati prodotti middleware da quando SQL ha separato l'interfaccia dal motore). Più i servizi si mercificano, più tenderanno a ricadere nelle infrastrutture open source, una transizione che vediamo già in atto nei sistemi operativi.

In un futuro che comprenda la competizione con l'open source, possiamo prevedere che, prima o poi, il destino di qualsiasi tecnologia software sarà o morire o divenire anch'essa parte dell'infrastruttura open source. Se questa non è proprio una bella notizia per gli imprenditori che sperano di raccogliere profitti dai software commerciali per sempre, ci lascia comune intendere che l'industria del software, nel suo insieme, rimarrà imprenditoriale, con nuove nicchie che si apriranno in continuazione nella categoria applicativa più alta e con una speranza di vita limitata per i monopoli IP chiusi, le cui categorie di prodotto ricadranno nelle infrastrutture.

Alla fine, naturalmente, questo equilibrio sarà ideale perché siano i consumatori di software a condurre il processo. Sempre più software di alta qualità diventeranno stabilmente disponibili all'utilizzo e al perfezionamento, anziché andare fuori produzione o essere chiusi nel solaio da qualcuno. Per concludere, la pentola magica di Ceridwen è una metafora troppo debole: perché il cibo si consuma o si deteriora, mentre i codici sorgente del software, teoricamente, durano per sempre. Il libero mercato, nel suo senso libertario più ampio, a includere ogni attività non coercitiva, sia di commercio, sia di dono, può produrre abbondanza di software per sempre e per tutti.

- [CatB] *The Cathedral and the Bazaar* [Tr. it La cattedrale e il bazaar]
- [HtN] *Homesteading the Noosphere* [Tr. it Colonizzare la noosfera]
- [DL] De Marco, Lister, *Peopleware: Productive Projects and Teams* (New York, Dorset House, 1987; ISBN 0932633-05-6)
- [SH] Shawn Hargreaves ha scritto una buona analisi dell'applicabilità dei metodi open source ai giochi; http://www.talula.demon.co.uk/games.html Playing the Open Source Game].

Diverse discussioni stimolanti con David D. Friedman mi hanno aiutato a raffinare il modello dell'"area comune all'inverso" applicato alla cooperazione open source. Devo ringraziare anche Marshall van Alstyne per avermi fatto notare l'importanza concettuale delle informazioni che danno accesso a un bene competitivo. Ray Ontko dell'Indiana Group mi ha aiutato con le sue critiche. Mi hanno aiutato anche molte persone che hanno assistito alle mie conferenze tra giugno 1998 e '99; se siete tra loro, sapete di chi sto parlando.

Un'ulteriore dimostrazione del modello open source è che questo saggio ha raggiunto miglioramenti significativi grazie ai consigli che ho ricevuto via e-mail, pochi giorni prima della pubblicazione. Lloyd Wood mi ha fatto notare l'importanza del margine di sicurezza dell'open source e Doug Dante mi ha dato l'idea del modello aziendale "Liberare il futuro". Una domanda di Adam Moorhouse ha portato alla trattazione dei vantaggi dell'esclusione. Lionel Oliviera Gresse mi ha fornito una denominazione migliore per uno dei modelli aziendali. Stephen Turnbull mi ha canzonato per il trattamento semplicistico dell'effetto free riding.

I produttori di periferiche hardware (schede Ethernet, schede di controllo del disco, schede video e simili) sono storicamente restii a passare all'open source. La situazione sta cambiando, attualmente, con imprese quali Adaptec e Cyclades che iniziano a diffondere specifiche e codice sorgente dei driver dalle loro schede. Ciononostante, si trova ancora una certa resistenza. In questa appendice, tenteremo di chiarire diversi equivoci economici che ne sono causa.

Se siete produttori di hardware, potreste temere che l'open source riveli dettagli importanti sul funzionamento del vostro hardware e che la concorrenza possa copiarlo, riportando ingiustamente un vantaggio competitivo. Al tempo in cui i cicli di produzione duravano dai tre ai cinque anni, questa era un'obiezione valida. Oggi, il tempo che impiegherebbero gli ingegneri della concorrenza per copiare e capire la copia occuperebbe una parte considerevole del ciclo di produzione e questo tempo non lo impiegherebbero per innovare o differenziare i loro stessi prodotti. Il plagio è quindi una trappola in cui a voi fa piacere che la vostra concorrenza cada.

In ogni caso, questi dettagli non rimangono segreti a lungo, al giorno d'oggi. I driver hardware non sono come i sistemi operativi, né come le applicazioni: sono piccoli, facili da disassemblare e semplici da clonare. Potrebbero farlo anche dei ragazzini, novelli programmatori: e spesso lo fanno.

Sono letteralmente migliaia i programmatori Linux e FreeBSD che hanno sia le capacità, sia la motivazione per costruire driver per una nuova scheda. Per molte classi di dispositivi con interfacce relativamente semplici e standard ben conosciuti (come le schede controller e di rete), questi appassionati hacker riescono spesso a prototipare un driver quasi alla stessa rapidità in cui può farlo il vostro laboratorio, anche senza documentazione e senza disassemblare un driver esistente.

Anche per dispositivi problematici come le schede video, non si può fare granché per contrastare un programmatore abile armato di un disassemblatore. I costi sono contenuti e le barriere legali penetrabili: Linux è un'opera internazionale, e ci sarà sempre una giurisdizione in cui il reverse engineering sarà legale.

Per avere prove concrete della verità di queste affermazioni, vi basterà esaminare la lista di dispositivi supportati dal kernel di Linux o contenuti negli

indici di siti come Metalab riguardanti i driver e notare la frequenza con cui ne appaiono di nuovi.

Il messaggio? Mantenere segreto il vostro driver sembra una buona idea a breve scadenza, ma probabilmente, si rivelerà cattiva, alla lunga (ovviamente, qualora siate in concorrenza con altri produttori che abbiano già adottato l'open source). Ma se proprio siete costretti a farlo, memorizzate il codice in una ROM sulla scheda. Poi pubblicatene l'interfaccia. Adottate l'open source per quanto vi è possibile, per costruire il vostro mercato e dimostrare alla potenziale clientela che credete nelle vostre capacità di superare la concorrenza per idee e innovazioni, quando serve.

Se mantenete la soluzione commerciale, normalmente otterrete il peggio del peggio: i vostri segreti verranno divulgati, non otterrete assistenza gratuita allo sviluppo e non avrete fatto perdere tempo ai vostri concorrenti più stupidi per clonarvi. Ma la cosa più grave è che avrete perso l'occasione, per i vostri driver, di un'adozione diffusissima e veloce. Un mercato ampio e importante (formato dai gestori dei server che fanno funzionare tutta Internet e più del 17% delle banche dati delle imprese) metterà da parte la vostra azienda, giustamente, considerandola incompetente e paurosa, perché voi non avrete compreso queste cose. Poi, compreranno le vostre schede da qualcun altro che invece le avrà capite.

Quella presentata è la revisione n. 1.14
Le versioni non descritte in questa sede contengono aggiustamenti editoriali e tipografici.

- 20 maggio 1999, versione 1.1: bozza iniziale
- 18 giugno 1999, versione 1.2: prima versione privata a scopo di revisione
- 24 giugno 1999, versione 1.5: prima pubblicazione
- 24 giugno 1999, versione 1.6: aggiustamenti: tentativo di definizione di "hacker"
- 24 giugno 1999, versione 1.7: aggiustamenti: delucidazione del criterio (e)
- 24 giugno 1999, versione 1.9: "margine di sicurezza", modello "Liberare il futuro" e aggiunta di una nuova sezione sui vantaggi dell'esclusione
- 24 giugno 1999, versione 1.10: migliore denominazione del modello delle "Lame da rasoio"
- 25 giugno 1999, versione 1.13: correzione al 13% della dichiarazione sui guadagni di Netscape; aggiunta di un'esposizione più completa dell'effetto free riding, correzione della lista di protocolli commerciali
- 25 giugno 1999, versione 1.14: aggiunta di e-smith, inc.
- 9 luglio 1999, versione 1.15: nuova appendice sui driver hardware e miglioramenti della spiegazione dei beni competitivi, con la consulenza di Rich Morin.
- 30 settembre 1999 - traduzione in italiano di Sabrina Fusari.

Informazioni su questa edizione elettronica:

Questo ebook proviene da Wikisource in lingua italiana[1]. Wikisource è una biblioteca digitale libera, multilingue, interamente gestita da volontari, ed ha l'obiettivo di mettere a disposizione di tutti il maggior numero possibile di libri e testi. Accogliamo romanzi, poesie, riviste, lettere, saggi.

Il nostro scopo è offrire al lettore *gratuitamente* testi liberi da diritti d'autore. Potete fare quel che volete con i nostri ebook: copiarli, distribuirli, persino modificarli o venderli, a patto che rispettiate le clausole della licenza Creative Commons Attribuzione - Condividi allo stesso modo 3.0 Unported[2].

Ma la cosa veramente speciale di Wikisource è che **anche tu** puoi partecipare.
Wikisource è costruita amorevolmente curata da lettori come te. Non esitare a unirti a noi.

Nonostante l'attenzione dei volontari, un errore può essere sfuggito durante la trascrizione o rilettura del testo. Puoi segnalarci un errore a questo indirizzo: http://it.wikisource.org/wiki/Segnala_errori

I seguenti contributori hanno permesso la realizzazione di questo libro:

- Aubrey
- Kronin
- Qualc1
- Candalua
- ProtectoBot
- IPork
- OrbiliusMagister
- Carlomorino
- Xavier121
- Zhuyifei1999
- Siebrand
- Maat
- Skalman
- Ftiercel
- Marc

Il modo migliore di ringraziarli è diventare uno di noi :-)

A presto.

---

1. ↑ http://it.wikisource.org
2. ↑ http://www.creativecommons.org/licenses/by-sa/3.0/deed.it