Università Ca' Foscari – Venezia

L'ORGANIZZAZIONE FA LA DIFFERENZA?

IX Workshop dei Docenti e dei Ricercatori di Organizzazione Aziendale

7 – 8 Febbraio 2008

Track: Antecedenti, forme, meccanismi, dinamiche evolutive ed effetti sulle performance dei network e delle relazioni inter-organizzative

IL COORDINAMENTO ORGANIZZATIVO IN UN NETWORK DI SVILUPPATORI OS

FRANCESCO BOLICI

Università degli Studi di Cassino

francesco.bolici@eco.unicas.it

Introduzione

L'interazione mediata è l'essenza dell'organizzazione (Weick, 1995, p.99). Questo contributo affronta un aspetto specifico dell'interazione tra gli elementi dell'organizzazione, quello del coordinamento come strumento di miglioramento dell'efficienza organizzativa. Le forme di coordinamento sono quei meccanismi organizzativi che assicurano il regolare svolgimento di attività interdipendenti (intendendo in senso ampio tutte le situazioni in cui l'esecuzione di un'attività richieda la condivisione di risorse o lo svolgimento di più operazioni connesse) (Malone e Crowston, 1994).

L'obiettivo dell'analisi, primo passo di un più articolato programma di ricerca, è capire se le forme di coordinamento presenti nella letteratura classica -a partire dai contributi di March e Simon (1958) e Thompson (1967) fino a quelli di Crowston (1997, 2005) e Malone e Crowston (1994)- emergano o siano adottate anche dalle organizzazioni che operano in contesti dinamici e mutevoli. Infatti, pur essendo le definizioni, le classificazioni e i meccanismi tradizionali di coordinamento validi ed utili nei contesti di riferimento per cui sono stati pensati, più di un ricercatore pone dei dubbi sulla loro applicazione immediata e senza alcun adattamento a contesti estremamente differenti da quelli originali (Grandori 2000, p. 1; Kellogg et al. 2006, p.22). Senza presupporre che le forme di coordinamento siano necessariamente differenti per il semplice fatto che le strutture e le caratteristiche di alcune organizzazioni moderne differiscono da quelle classiche, riteniamo utile indagare quali siano le forme e i meccanismi di coordinamento più adatte ad operare all'interno di queste organizzazioni caratterizzate da team di lavoro temporanei, processi decisionali decentralizzati, potere di controllo distribuito, relazioni prevalentemente orizzontali e mediate dalla tecnologia, divisione del lavoro piuttosto dinamica e confini sfumati e permeabili.

Il nostro dominio di analisi si focalizza sulle comunità di sviluppo del software open source (OS) che riteniamo possedere tutte le caratteristiche delle organizzazioni che operano in un ambiente di riferimento dinamico, in continua evoluzione e technology intensive. Intendiamo approfondire l'analisi delle forme e dei meccanismi di coordinamento adatti ad operare in ambienti di network di sviluppatori OS: quali forme di coordinamento sono adottare nei progetti di

sviluppo? Esistono differenze nelle forme e nei meccanismi di coordinamento tra le attività all'interno di questi network organizzativi rispetto a quelle tradizionali? In caso affermativo, quali vantaggi e svantaggi comportano queste forme di coordinamento?

Il nostro lavoro intende contribuire a specificare meglio un problema di ricerca che riteniamo estremamente significativo sia dal punto di vista della ricerca che da quello manageriale, nel perseguire questo obiettivo incentreremo la nostra analisi su due specifiche ipotesi: 1) i processi di progettazione software possono essere scomposti secondo livelli di attività differenti tra loro (da quelli centrali a quelli periferici), a cui corrisponderanno delle logiche di gestione e coordinamento differenti; 2) il software (e nello specifico il kernel), assumendo le funzioni di un boundary object, diventa un fattore centrale attorno al quale si realizza il coordinamento sia delle più operative che di quelle più innovative.

Entrambi questi punti verranno analizzati criticamente all'interno del contributo nell'ottica di fornire una più approfondita trattazione del problema.

L'emergere della questione del Coordinamento in progetti Open Source

Il fenomeno open source ha subito destato molto interesse tra gli studiosi di diverse aree. I primi a riconoscere le grandi potenzialità di questo "laboratorio accelerato delle dinamiche organizzative"[1] sono stati i ricercatori che hanno indagato le spinte motivazionali degli sviluppatori a partecipare in progetti open source (ricordiamo tra gli altri Von Hippel e Von Krogh 2003), le dinamiche di creazione di innovazione (es. Kogut e Metiu, 2001), gli aspetti economici (Lerner e Tirole, 2002) e le tematiche più apertamente tecniche-ingegneristiche (es. Feller e Fitzgerald, 2000). Inizialmente non è tuttavia riscontrabile alcun tentativo strutturato di comprendere le dinamiche di coordinamento interne ai progetti di sviluppo software.

Una prima richiesta per indagini sulle forme di coordinamento in ambienti OS è quella che indirettamente emerge dalle parole della Grandori (2000, p. 5) quando propone le reti dei progetti OS come un caso estremo in cui le competenze e le conoscenze diventano fattori determinanti ed in cui l'apprendimento reciproco

[1] Pur non potendo citarlo da una fonte scritta, volevo esplicitamente riconoscere che questo modo di intendere il fenomeno open source e più in generale le tematiche legate all'ICT deriva dalle numerose discussioni con il Prof. Pontiggia e dall'influenza, oltremodo positiva, che esse hanno avuto su questo ed altri temi di ricerca.

diviene, oltre che obiettivo comune, l'elemento centrale intorno cui coordinare le attività. Esplicitamente la necessità di indirizzare l'analisi sui processi di coordinamento in contesti OS viene formulata nei lavori di Weber (2004, p.11-12), per cui i tradizionali meccanismi di coordinamento non sembrano essere una risposta sufficiente alle caratteristiche dei progetti OS, e in seguito anche di Crowston (2005) e Crowston e Howison (2005) che ribadiscono una certa assenza d'analisi dei meccanismi di coordinamento. Il recente articolo di Kellogg et. al (2006) è il primo che affronta espressamente questo tema investigando i meccanismi organizzativi che consentono il coordinamento in queste comunità. Un tentativo precedente era stato anche compiuto, anche se in via iniziale, dall'interessante lavoro di Iannacci (2005) sul coordinamento all'interno del progetto Linux.

Prima di iniziare l'analisi è importante chiedersi quali siano le caratteristiche degli ambienti OS significative per i processi di allineamento delle attività e degli attori. Nei processi di sviluppo software le attività possono essere più o meno strutturate a seconda delle metodologie per lo sviluppo software che vengono utilizzate anche se la tendenza è quella a concentrarsi su metodologie 'soft' che consentono una più facile gestione di interdipendenze più complesse rispetto a quelle semplicemente sequenziali tipiche degli approcci più rigidi. Oltre questa caratteristica, possiamo notare che gli attori che partecipano a varia natura in questi progetti possono essere raggruppati in prima analisi in due grandi categorie: gli sviluppatori e gli utenti. Una delle peculiarità di questi progetti è quella di operare in ambienti virtuali, quindi dispersi dal punto di vista geografico e con relazioni che si sviluppano principalmente in maniera principalmente asincrona. Queste caratteristiche rendono le interdipendenze tra attori e tra attività giù piuttosto complesse per essere anticipabili a priori. Questo diviene ancora più evidente se si considera l'alta frequenza con cui le attività degli sviluppatori vengono ad essere modificate sia dalle continue interazioni e richieste degli utilizzatori, sia dai contributi e suggerimenti degli altri developer che propongono nuovi requisiti, necessità, problemi o intuizioni emersi in corso d'opera e non anticipabili in precedenza. Per queste ragioni riteniamo che i progetti di sviluppo open source presentino generalmente una non strutturabilità delle interdipendenze.

La domanda a cui vogliamo cercare di dare una prima riposta che verrà poi approfondita nelle prossime fasi del programma di ricerca è: come si coordinano gli attori in un progetto di sviluppo software OS, dove le attività sono emergenti e in rapido cambiamento, il prodotto intangibile, i partecipanti sparsi geograficamente, la comunicazione sempre mediata e l'autorità non imposta da strutture organizzative pre-esistenti?

Sintesi dei Modelli Analitici di Coordinamento in Ambienti OS

In questa sezione richiamiamo brevemente alcuni recenti contributi che possono fornire dei riferimenti per l'analisi dei meccanismi di coordinamento in ambienti dinamici e mutevoli.

Il lavoro di Amin e Cohedent (2000) riprende la distinzione tra attività vitali per l'organizzazione e quelle periferiche. Ad attività di tipo core sono applicate logiche di coordinamento incentrate sulla conoscenza che permettano un allineamento dei processi cognitivi tra singoli attori e processi organizzativi, mentre ad attività sempre più periferiche si possono gradualmente applicare soluzioni sempre più tradizionali basate tipicamente sulla standardizzazione dei processi. In quest'ottica proponiamo la prospettiva, ormai ben conosciuta, delle comunità di pratica (Wenger 1998) e in particolare i processi combinati di reificazione e partecipazione come possibili mezzi per il superamento delle barriere organizzative e quindi per il coordinamento. Approfondendo l'analisi sulle soluzioni più innovative per le attività cruciali dell'impresa (e quindi a più alta intensità cognitiva) introduciamo i lavori di Carlile (2002, 2004) e quello più recente di Kellogg et al. (2006). Carlile classifica tre modalità attraverso cui si può scambiare e combinare la conoscenza tra unità o attori organizzativi: trasferimento, traslazione e trasformazione. Riferendosi a questo modello e cercando svilupparlo, Kellogg et al. (2006) propongono la classificazione di tre pratiche di coordinamento in grado di superare i confini organizzativi tra persone che partecipano ad uno stesso progetto: il 'displayed work' che consiste nel rendere disponibile agli altri il proprio lavoro; il 'represented work' che, attraverso l'uso di un "contenitore" comune a tutti, facilita la comprensione e l'interpretazione del proprio lavoro agli altri membri del progetto; l''assembled work' che consiste nel riutilizzo e rivisitazione del lavoro di altri partecipanti al

progetto allo scopo di ottenere un prodotto indipendente (collegato alla nozione di "collage effect" proposta da Giddens, 1991).

Riprendendo il concetto definito da Galison (1999), Kellogg et al. (2006) rivisitano l'idea di *trading zone* come modalità attraverso cui avviene il coordinamento locale di conoscenze ed azioni senza bisogno di creare prima un substrato di linguaggi, valori e interessi comuni. Punto chiave di questo concetto è il suo essere "luogo" del coordinamento senza richiedere alcun tipo di accordo preventivo o struttura sociale e interpretativa condivisa ex-ante. Le trading zone acquistano quindi una duplice connotazione di *dinamicità* --perché cambiano al cambiare degli attori, del tempo e dell'oggetto dello scambio-- e di *temporaneità* --perché non emergono come strutture sociali permanenti, ma vengono ad essere configurate ad ogni scambio a seconda delle specifiche condizioni--. La trading zone è quindi paragonabile al processo di negoziazione di significato (Wenger 2000), visto come un processo in continua evoluzione e composto da una negoziazione illimitata tra gli attori che ad ogni interazione reciproca modificano il significato che attribuiscono all'oggetto della negoziazione. Trading zone e negoziazione del significato possono, a questo livello, essere considerati come elementi utili per il coordinamento delle conoscenze, realizzabile attraverso l'allineamento delle attività e la condivisione e combinazione di capacità e conoscenze tra diverse comunità.

Interessante richiamare anche il lavoro di Hansen (1999) per cui la condivisione della conoscenza complessa si realizza efficientemente attraverso i legami forti (Granovetter, 1973) definiti sulla base della distanza e della frequenza della relazione sociale. Hansen afferma che i legami deboli siano preferibili, dal punto di vista dell'efficienza, per identificare e cercare la conoscenza attraverso i confini organizzativi, mentre quando «[…] complex forms of knowledge are considered, the instrumental benefit of weak ties are called into question. Weak ties may lead to search benefits in social network but they may also cause problems in transferring complex forms of knowledge» (p.83). La conclusione del lavoro è infatti che «Strong interunit ties provide the highest relative net effect […] when the knowledge is high complex, whereas weak interunit ties have the strongest positive effect on completion time when the knowledge is not complex» (p.105).

Sulla base di questo framework interpretativo sui meccanismi di coordinamento procederemo ora a studiare i meccanismi di coordinamento in ambienti di sviluppo software di tipo OS.

Domande di ricerca sulla base dei Modelli Analitici

Come si coordinano gli attori in un progetto di sviluppo software OS, dove le attività sono emergenti e in rapido cambiamento, il prodotto intangibile, i partecipanti dispersi geograficamente, la comunicazione sempre mediata e la gerarchia non imposta da strutture organizzative pre-esistenti?

Questa è la domanda di ricerca che iniziamo ad investigare da un punto di vista teorico attraverso questo contributo. L'obiettivo è fornire un'insieme di spunti di riflessione, analiticamente fondati, che costituiscano il framework teorico sulla base del quale condurre in futuro un'approfondita analisi empirica. Coerentemente con gli obiettivi ci soffermeremo sulle caratteristiche organizzative proprie dei diversi progetti OS (che emergono da una prima analisi esplorativa del repository SourceForge.net), domandandoci se le soluzioni per il coordinamento scelte (o emerse) nella realtà siano classificabili o meno attraverso le prospettive organizzative tradizionali. La prima ipotesi che proponiamo è la scomposizione dei progetti di sviluppo OS in diversi strati di attività, da quelli più centrali (es. attività di sviluppo delle parti cruciali del software, proprie dei developers esperti) a quelle più periferiche (es. feedback degli utenti finali). Riteniamo infatti le differenze dei diversi strati di attività comportino l'emergere o l'adozione di meccanismi di coordinamento specifici e diversi anche negli stessi domini OS.

La seconda ipotesi riguarda invece l'emergere di uno specifico boundary object, identificato nel software stesso (e in particolare nel kernel), come fattore centrale attorno al quale si realizza il coordinamento sia delle attività core che di quelle periferiche.

Ipotesi 1: differenziazione delle attività OS: una struttura a strati

La considerazione che i progetti di sviluppo software OS seguano una loro logica peculiare e distintiva, e spesso anche un po' naif, sembra ormai essere ampiamente condivisa. Studiosi ed addetti ai lavori hanno da subito apprezzato la ormai nota distinzione di Raymond (1998) tra la 'cattedrale' dei progetti proprietari ed il 'bazar' di quelli aperti. Questa definizione ha il vantaggio di

riuscire a sintetizzare, nella semplice dicotomia, delle differenze profonde dal punto di vista della struttura organizzativa (l'ordine delle cattedrali contrapposto al caos e/o organicità dei bazar), dei processi decisionali (accentrati vs distribuiti), dei meccanismi di comunicazione (verticali vs orizzontali) e così via.

Seguendo queste considerazioni, la maggior parte della letteratura riconosce alle comuni modalità organizzative dei progetti OS un'alta specificità (Lerner e Tirole 2001, Kogut e Metiu 2001, Bonaccorsi e Rossi 2002, Lakhani e von Hippel 2003, von Hippel e von Krogh 2003, Pontiggia e Bolici 2006, Kellog et al 2006) come ad esempio: team di lavoro temporanei, processi decisionali decentralizzati, controllo mutevole e distribuito, relazioni orizzontali e indipendenti, divisione del lavoro dinamico, confini sfumati e permeabili, processi di lavoro improvvisati, flessibili e partecipativi, composizione eterogenea.

Tuttavia, nonostante questo insieme di fattori piuttosto omogenei, è possibile notare una certa differenziazione tra le attività svolte all'interno dei progetti OS. In particolare le attività a più alto contenuto cognitivo e con interdipendenze più intense sono quelle che conservano una struttura organizzativa più organica, una divisione del lavoro più flessibile e un coordinamento più reciproco e dinamico[2]. Questo è il caso delle attività di sviluppo delle parti cruciali del software (es. kernel). In questo senso l'analisi di Amin e Cohendet (2000) per cui ad attività di tipo core vengono applicate logiche di coordinamento più aperte ed innovative, mentre ad attività sempre più periferiche si possono via via applicare soluzioni sempre più rigide. Notiamo che anche tra attività di sviluppo del codice vi può essere una scala di importanza, così la creazione del kernel diviene essenziale per il completamento del progetto, mentre un bug che riguarda la visualizzazione di un colore dell'interfaccia, per quanto spiacevole, non metterà in pericolo da solo la sopravvivenza del software. Vi è quindi una stratificazione delle attività da quelle più importanti a quelle periferiche, dove man mano che dalla zona centrale si va verso l'esterno le soluzioni di coordinamento adottate diventano più stringenti e meno flessibili. Questo è il caso delle attività di bugs fixing che nella maggior parte dei casi seguono un processo standardizzato, essendo richiesto ad

[2] Questo, come vedremo nelle conclusioni, risulta coerente anche con l'impostazione tradizionale thompsoniana per cui a forme di interdipendenza reciproca sono più adatti meccanismi flessibili come l'adattamento reciproco.

esempio di postare la segnalazione di errore in un certo luogo, rispettando una certa forma (es. indicando nell'oggetto della mail il tipo di problema riscontrato, attraverso una classificazione predeterminata dagli stessi sviluppatori "[openqrm - Installing openQRM] RE: qrm-configurator problem") e seguendo quindi un processo stabilito e standardizzato ex ante. Così ad un certo messaggio di errore ben determinato, si sa in anticipo chi sarà lo sviluppatore che deve dare una risposta e secondo quale modalità.

Le attività di programmazione possono quindi essere immaginate come dei cerchi concentrici, il cui centro rappresenta gli elementi essenziali del software sviluppato (es. kernel) e allontanandosi verso l'esterno gli elementi meno determinanti (si veda Fig. 1).

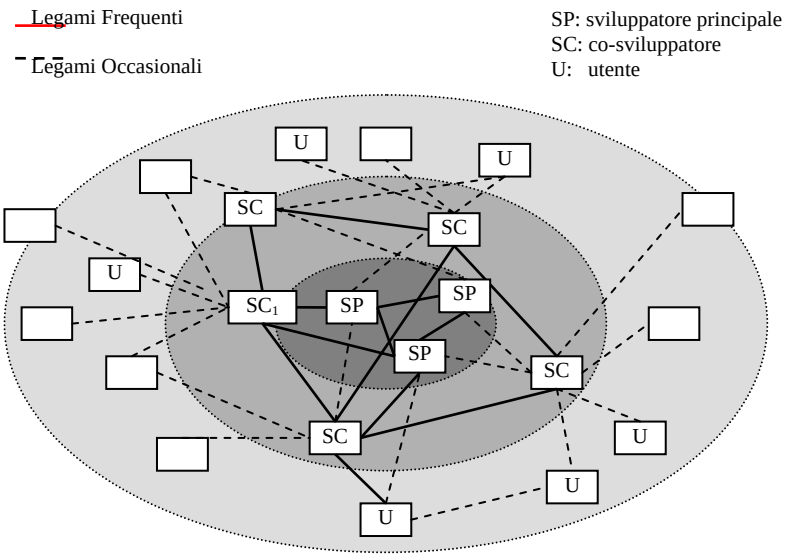

Figura 1 Struttura a strati delle attività di un progetto di sviluppo OS.

Interessante notare che negli strati di attività più esterni interagiscono un numero più elevato di attori, mentre all'avvicinarsi al centro, le persone che interagiscono diminuiscono di numero e soprattutto hanno sviluppato una maggiore fiducia reciproca dovuta anche alla frequenza e alla durata delle relazioni intessute. Le persone che si trovano al centro del progetto sono gli sviluppatori più impegnati (SP), che contribuiscono alla programmazione della maggior parte del codice e sovrintendono al disegno e all'evoluzione del progetto. Ad un livello più esterno troviamo altri sviluppatori (SC), il cui principale contributo è quello di creare e

sottomettere le pathces (modifiche e aggiunte al codice per risolvere un problema del software) che in alcuni casi verranno prima valutate dagli sviluppatori principali e quindi accettate o meno. L'anello più periferico rappresenta invece gli utenti (U) che non partecipano alla programmazione ma contribuiscono allo sviluppo del progetto cercando e riportando i bug, fornendo degli use-case, testando i nuovi rilasci del software e proponendo nuove idee. Infine vi è un'ulteriore strato in cui troviamo tutti gli utilizzatori passivi del software, quelle persone che pur downlodando e utilizzandolo non contribuiscono in alcun modo all'evoluzione del progetto.

Nel procedere con l'analisi dobbiamo tener presenti due considerazioni, sul rapporto tra strutture a rete e conoscenza, condivise da gran parte della letteratura (es. Kogut e Zander 1992, Brown e Duguid 1991, Kogut 2000,): i) più le attività sono centrali più diviene cruciale coordinare le conoscenze complesse , ii) per gestire processi di apprendimento organizzativo efficaci ed in grado di creare nuova conoscenza occorre connettere elementi ed attori dotati di un'ampia varietà (e quindi più facilmente riscontrabili al di fuori della propria comunità di riferimento). Individuiamo quindi una tensione tra spinte verso la centralizzazione e contemporaneamente decentralizzazione dei processi di coordinamento della conoscenza. Da un lato infatti abbiamo delle spinte verso la gestione di un numero limitato di soggetti (core-developers, mantainers, project managers, innovators) impegnato nelle attività centrali dei progetti che si caratterizzano per una numerosità e una frequenza delle comunicazioni tali da creare delle vere e proprie relazioni sociali in cui elementi quali la fiducia e la responsabilità divengono centrali[3]. Questo risponderebbe all'esigenza richiamata da Hansen (1999) di privilegiare la creazione di connessioni forti ('strong ties') là dove si abbia l'esigenza di trasferire la conoscenza in modo che le relazioni frequenti e stabili favoriscano la condivisione di elementi di natura sociale (quali ad esempio la fiducia). La spinta verso la decentralizzazione è invece rappresentata dalla necessità di favorire la creazione di nuova conoscenza andando ad interagire con altre comunità di sviluppatori o insiemi di utenti che possano garantire una

[3] Così prendendo ad esempio la struttura sociale propria di Linux"[] So that you end up with Linus [Torvalds] getting stuff that most of the time he doesn't have to work on very hard, because somebody he trusts has already filtered it" (Moody 2001, p.179).

maggiore varietà e quindi un'alta probabilità di innovazione. Sistemi a connessioni lasche sono più efficienti per quanto riguarda la ricerca e l'individuazione di conoscenze, capacità e idee nuove. La non ridondanza dei sistemi debolmente connessi, infatti, permette di gestire un maggior numero di relazioni con un minor dispendio di risorse, garantendo potenzialmente l'accesso ad una varietà più ampia di conoscenze non presenti originariamente all'interno del sistema stesso. Inoltre è molto più probabile che i processi di creazione di nuova conoscenza vengano innescati da un'idea che proviene da un attore ai margini del sistema originario e che quindi non presenta caratteristiche di omogeneità di comportamento, interpretazione e conoscenze. Se caliamo queste considerazioni nella realtà pratica dei progetti OS possiamo notare come risulti ancora valido il modello descritto in figura 1. Gli attori presenti nel nucleo centrale instaurano legami forti con i loro pari-livello e con alcuni degli sviluppatori dell'anello limitrofo che hanno una maggior vicinanza con le loro attività. A loro volta i co-sviluppatori presentano dei forti legami tra di loro e dei legami in genere (con alcune eccezioni) meno stretti con gli sviluppatori principali e gli utenti. Infine gli utenti si caratterizzano per delle relazioni di tipo lasco con gli sviluppatori periferici a cui sottomettono soprattutto richieste di aiuto, bugs, richieste per nuove funzionalità e feedback (soprattutto sulle nuove release). Notiamo che la divisione analitica presente in figura 1 diviene molto più sfumata nella realtà non essendoci confini "fisici" che separino ad esempio uno sviluppatore centrale da uno periferico, ma potendosi distinguere solamente sulla base delle loro attività e del credito e della fiducia che riscuotono all'interno del progetto. Così ad esempio l'attore SC_1 assume una posizione di confine tra la cerchia degli sviluppatori core e quella più esterna. In questo caso potremmo ipotizzare che lo sviluppatore stia divenendo sempre più importante per il progetto (si stanno infatti sviluppando anche legami forti con alcuni degli sviluppatori principali) e che quindi nel tempo potrebbe entrare a far parte degli sviluppatori più centrali (come pure si potrebbe ipotizzare la situazione diametralmente opposta). Questo è coerente con la considerazione che le strutture di un progetto sono dinamiche ed in evoluzione e che quindi i rapporti e le interazioni possono mutare anche significativamente nel tempo. Abbiamo quindi delle connessioni più

forti per quanto riguarda le attività cruciali per il processo di sviluppo software che consentono il coordinamento delle conoscenze, la loro gestione e trasferimento, mentre i legami più deboli verso l'esterno consentono di realizzare le attività di ricerca ed individuazione di nuove conoscenze ed idee in maniera più efficiente.

Altro esempio pratico è rappresentato dalla descrizione di Cox (1998) quando, riferendosi alla sua esperienza all'interno del progetto Linux, spiega che ad un certo punto gli sviluppatori avevano utilizzato un 'killfile' per mandare direttamente nel cestino tutti i messaggi postati nella mailing list del progetto provenienti da utenti esterni perché i developer consideravano il tempo dedicato a quelle relazioni e sottratto alla programmazione del codice una perdita di tempo. Lo stesso Cox ammette che quella fu una risposta sbagliata alla necessità di far avanzare speditamente il progetto e che, come avvenne in seguito, era impensabile per un progetto di sviluppo OS non stimolare e non permettere un'ampia discussione. Questo perché la varietà di conoscenze garantita dalle zone esterne è una risorsa difficilmente sostituibile, ed anche perché con le interazioni tra utenti meno esperti e quelli con più esperienza i primi possono imparare e migliorare le proprie capacità e quindi nel medio lungo termine contribuire con attività a maggior valore aggiunto alla riuscita del progetto.

In conclusione la nostra ipotesi riprende ed estende la considerazione proposta da Crowston e Howison (2005) per cui non è possibile stabilire a priori la natura decentralizzata delle comunicazioni all'interno dei progetti OS (per attività di bug fixing), ma essa dipende dalle caratteristiche dei diversi progetti. Secondo la nostra ipotesi, infatti, non solo la struttura organizzativa e sociale di un progetto OS varia da caso a caso --anche se rimarranno sempre presenti alcune caratteristiche proprie delle strutture eterarchiche (Kellog et al 2006)-- ma, anche all'interno di uno stesso progetto, attività di natura diversa sono coordinate e gestite secondo obiettivi e logiche differenti. Le attività più interdipendenti e complesse utilizzano meccanismi di coordinamento delle conoscenze (è il caso dello sviluppo del codice centrale del software), mentre i processi più periferici di raccolta di informazioni, commenti ed idee sono gestiti attraverso meccanismi di coordinamento più formalizzati (come avviene in alcuni progetti OS in cui le

attività di bug fixing sono standardizzate attraverso un processo prestabilito). Nel primo caso si adottano meccanismi di adattamento reciproco per la gestione di legami di tipo forte, mentre nel secondo i processi standardizzati permettono di gestire efficientemente il gran numero di relazioni deboli che il progetto ha verso l'ambiente più esterno.

Ipotesi 2: Meccanismi per il coordinamento in progetti OS: trading zone e boundary objects

La nostra seconda ipotesi riguarda gli strumenti per il coordinamento organizzativo in un network di attori OS. Rifacendoci ai concetti di negoziazione del significato e di trading zone intendiamo verificare analiticamente se le prospettive sulle comunità di pratica possano fornire degli strumenti per la gestione delle interdipendenze in progetti OS. Abbiamo in precedenza mostrato come i processi di creazione di conoscenza siano trasversali agli attori e agli strati di attività (è infatti dagli utenti che spesso proviene l'input che, portato in aree più "centrali" del progetto, innesca la creazione di nuova conoscenza) mentre il trasferimento e la gestione della conoscenza è più focalizzata sulle aree centrali.

La nostra domanda è quindi se può il meccanismo di negoziazione di significato (Wenger 1998), specificato nei processi di partecipazione e reificazione, costituire una valida modalità per il coordinamento delle attività e degli attori in un progetto di sviluppo software OS. La prima difficoltà che questo concetto incontra nella sua applicazione a contesti open source deriva dalla difficoltà di individuare delle comunità di pratica chiaramente definite. Nei progetti OS la rigida separazione tra sviluppatori e utilizzatori scompare. Oltre a questo notiamo anche che tutti coloro che, in misura diversa, partecipano ad un progetto OS condividono alcuni valori, idee e pratiche. Questi attori attribuiscono, infatti, un certo valore all'idea di sviluppare software che possa essere modificato e ridistribuito, sono disposti a contribuire con il proprio lavoro senza un ritorno economico diretto, sono, a livelli anche molto diversi, persone in grado di programmare e quindi dotate di specifiche competenze tecniche e accettano alcune regole di "condotta" della comunità virtuale. Per cui, nonostante le forti differenze che permangono ad esempio tra un utente-passivo e un amministratore di progetto, tutti i partecipanti

si trovano immersi in una struttura che richiede un minimo di condivisione del substrato sociale in cui sono virtualmente immersi.

Da qui la prima considerazione riguardante l'applicabilità del concetto di trading zone a questi progetti. Ricordiamo che le trading zone sono delle strutture di coordinamento che facilitano l'allineamento in condizioni di volatilità, temporaneità ed estrema rapidità senza richiedere ex ante la condivisione e omogeneizzazione di interpretazioni ed interessi. Quindi nel nostro caso specifico, esistendo già delle condizioni per una condivisione di valori e interessi tra tutti i membri del progetto, le trading zone possono essere adottate ed allo stesso tempo anche superate poiché sono ipotizzabili forme diverse di coordinamento basate sul riconoscimento reciproco e negoziazione del significato tra attori diversi. Riferendoci quindi anche allo schema presentato in figura 1 andiamo ad esaminare se e come i processi di negoziazione del significato possono supportare o facilitare il coordinamento delle conoscenze all'interno di un progetto OS.

Il processo di negoziazione del significato si scompone in due sotto-processi, quello di reificazione e quello di partecipazione. Il processo di reificazione è quello più facilmente individuabile all'interno dei progetti di sviluppo software open source. Il codice, il software stesso, è infatti un oggetto di confine tra diversi attori. Intorno a questo artefatto si vanno ad imperniare i processi di coordinamento tra i diversi attori, senza che questo implichi necessariamente che le prospettive dei diversi gruppi siano le stesse. Intorno al codice già sviluppato e visibile a tutti[4], i diversi programmatori si coordinano cercando di ritrovare attraverso una loro personale interpretazione come allineare le proprie idee e attività all'artefatto che hanno di fronte. Interessante richiamare e contestualizzare all'interno di progetti OS alcune delle caratteristiche che gli oggetti di confine devono avere --così come proposti da Star e Griesemer (1989) --: 1) *modularità*, la prospettiva di ogni attore e gruppo può essere presente all'interno dell'oggetto, e non è certo casuale che una delle caratteristiche principali dei progetti OS è che questi sono sviluppati seguendo proprio una logica modulare che sprona e stimola il lavoro dei singoli attori e comunità; 2) *astrazione*, ossia tutte le prospettive sono presentate insieme attraverso l'eliminazione delle caratteristiche specifiche di ogni

[4] Ritorna qui la distinzione proposta da Kellog et al (2006).

singola prospettiva. Questo in un progetto OS si realizza quando il codice proposto da uno sviluppatore viene vagliato, valutato ed eventualmente modificato da uno sviluppatore principale che si accerta della coerenza delle linee di codice con quanto già rilasciato e con gli obiettivi generali del progetto; 3) *adattamento*, gli oggetti di confine si prestano a più attività esattamente come succede per i software. A maggior ragione il software OS può essere adattato da ogni singolo utente alle proprie specifiche necessità e quindi rilasciato a tutta la comunità; 4) *standardizzazione*, le informazioni contenute nell'oggetto sono in una forma predeterminata, così che ogni attore sappia come utilizzare l'oggetto nel suo specifico contesto. Questo è valido sia per quanto riguarda le attività di programmazione, in cui vi è la condivisione del linguaggio e delle regole di sviluppo, che per le attività di bug reporting e fixing che sono generalmente standardizzate in dei processi condivisi e riportare in dei format facilmente leggibili e sempre uguali. Dalla descrizione di queste caratteristiche si evince che il codice stesso può rappresentare l'oggetto di confine principale per quanto riguarda il coordinamento delle attività e delle conoscenze tra gli attori. Anche senza bisogno di altri meccanismi di coordinamento un attore può allineare le sue attività di programmazione e combinare la propria conoscenza con quella di altri, semplicemente confrontandosi e dovendosi armonizzare con la parte centrale di codice (che può essere ad esempio il kernel[5]). Nella realtà di progetti OS sembra che questo meccanismo di coordinamento sia molto forte e gli sviluppatori allineino le proprie attività proprio grazie alla possibilità di raffrontarsi continuativamente con il codice già sviluppato, reso continuamente visibile e disponibile. In questo senso il software/boundary object svolge un doppio ruolo, essendo un fulcro centrale attorno a cui ruotano i meccanismi di coordinamento che servono sia a gestire le interdipendenze delle attività core che quelle periferiche, sia i legami di natura forte che quelli di natura debole. In questa sua capacità di esser strumento in grado di allineare sia le attività degli sviluppatori (si pensi a chi sviluppa nuove fetures che per forza di cose deve adattare la propria attività al codice già esistente in cui deve essere integrata la nuova funzione) che quelle degli utenti (è attraverso l'utilizzo del software che possono essere richieste

[5] Devo ringraziare Francesco Virili e Antonio Cordella per alcune illuminanti discussioni anche --e non solo-- su questo argomento.

nuovi feature o proposte delle idee innovative che risolvano un problema pratico e spesso personale).

L'utilizzo di oggetti di confine per il coordinamento delle conoscenze all'interno di un progetto presenta come svantaggio principale il fatto che il contenuto dell'oggetto può risultare ambiguo. Attraverso i processi di partecipazione si riescono però ad alleviare i problemi dovuti alle possibili deviazioni nell'interpretazione del codice. Per quanto riguarda i progetti open source la partecipazione si connota di una caratteristica ineluttabile: la non fisicità. La partecipazione quindi può avvenire solamente attraverso la mediazione di qualche tipo di tecnologia (il rischio in questo caso è quello di attenuare la forza dei meccanismi di partecipazione).

Ad una prima analisi teorica, risulta quindi interessante approfondire l'applicabilità dei processi di negoziazione del significato, attraverso il confronto imperniato sul codice sorgente e sui meccanismi di partecipazione, al coordinamento delle conoscenze e delle attività all'interno dei progetti di sviluppo software OS.

Conclusioni

In questo contibuto abbiamo analizzato il tema del coordinamento all'interno dei progetti di sviluppo software open source come primo passo di un più articolato programma di ricerca. In particolare è largamente riconosciuto come i processi di coordinamento interni ai progetti OS siano largamente inesplorati e ancora non compresi (Weber 2004, Crowston 2005, Crowston e Howison 2005, Kellogg et al. 2006).

Utilizzando i framework interpretativi e gli strumenti d'analisi basati sulle comunità di pratica abbiamo investigato le modalità attraverso cui si coordinano gli attori in un progetto di sviluppo software OS, dove le attività sono emergenti e in rapido cambiamento, il prodotto intangibile, i partecipanti sparsi geograficamente, la comunicazione sempre mediata e l'autorità non imposta da strutture organizzative pre-esistenti. Abbiamo quindi proposto di differenziare analiticamente le attività centrali da quelle periferiche e di associare a ciascuno strato di attività delle pratiche di gestione delle interdipendenze differenti. Le attività centrali (idea iniziale, programmazione delle sue componenti base,

principali attività di mantenimento, etc.) caratterizzate da legami forti tra gli attori, sono più efficacemente coordinate da meccanismi flessibili e dinamici. Al contrario, le attività più periferiche (bug fixing, attività di diffusione, etc.), in cui si cerca di creare un insieme di connessioni lasche tra una gran varietà di attori, sono coordinate secondo logiche di standardizzazione e pianificazione. Uno dei risultati di questa analisi è quello di riconoscere l'estremo vantaggio che la contemporanea presenza di entrambi questi tipi di legami in un sistema OS produce. Da una parte i legami forti permettono un migliore trasferimento e gestione della conoscenza, e una più efficiente gestione delle linee di sviluppo del progetto, dall'altra le connessioni deboli permettono la ricerca e l'identificazione di nuove idee e contributi tra un numero e una varietà più ampia di attori.

Inoltre emerge dalla nostra analisi l'importanza del software come boundary object a supporto e sostegno del coordinamento sia delle attività centrali che di quelle periferiche. Infatti sia le attività centrali più operative e cruciali gestite da attori immersi in un tessuto di relazioni forti, che i processi più periferici collegati da legami deboli al progetto e a più alto potenziale di innovazione, beneficiano del software già sviluppato e rilasciato come di un punto fermo attorno al quale innescare le proprie specifiche attività. Il kernel, o più in generale la struttura base del software, diviene così un elemento centrale a cui tutti le attività devono ricollegarsi e quindi spinge verso una coerenza e un implicito coordinamento di fondo che semplificano i complessi processi di gestione delle interdipendenze tra una grande varietà di attori.

In conclusione di questo lavoro vogliamo sottolineare come le risposte qui fornite necessitino di un ulteriore sforzo per assumere il valore di considerazioni conclusive e complete. Il presente articolo vuole fornire un contributo per la discussione e l'analisi dei meccanismi di coordinamento in ambienti OS che dovrà essere condotta attraverso una rigorosa analisi empirica. Con questo lavoro abbiamo voluto proporre un'interpretazione di come i meccanismi di coordinamento rivestano un ruolo fondamentale anche, e forse soprattutto, nei settori economici più innovativi e moderni. Speriamo che le conclusioni a cui siamo arrivati, per quanto indicative e non definitive, possano essere un ulteriore elemento di supporto per i prossimi lavori di ricerca su questo specifico tema.

References:

Amin, A. and P. Cohendet (2000). Organizational Learning and Governance Trough Embedded Practices. *Journal of Management and Governance*, 4, 93-116.

Bonaccorsi, A. e C. Rossi (2002). Why open source software can suceed? *LEM working paper series*, S'Anna, Pisa.

Brown, J.S. e P. Duguid (1991). Organizational learning and communities of practice. Toward a unified view of working, learning and innovation. *Organization Science* 2(1), 40-57.

Cox, A. (1998). Chattedrals, Bazaars and the Town Council, Slashdot (13 Oct), available at http://slashdot.org/features/98/10/13/1423253.shtml

Crowton, K. (1997). A coordination theory approach to organizational process design, *Organization Science*, Vol. 8, No. 2, 157-175.

Crowston, K. (2004). Electronic Communication and New Organizational Forms: a Coordination Theory Approach, *Syracuse FLOSS research working paper.*

Crowston, K. (2005). A coordination theory analysis of bug fixing in proprietary and free/libre open source software. Working paper, available at http://floss.syr.edu/StudyP/050328%20chapter.pdf.

Feller J., Fitzgerald B., "*Understanding Open Source software development*" Addison Wesley, Boston, 2002.

Galison, P. (1999). Trading zone: Coordinating action and belief. *The Science Studies Reader*, M. Biagioli (ed.), Routledge, NY, 137-160.

Grandori, A. (2000). Conjectures for a New Research Agenda on Governance. *Journal of Management and Governance*, 4, 1-9.

Granovetter, M.S. (1973). The Strenght of Weak Ties, *American Journal of Sociology*, 78, 1360-1380.

Hansen, M.T. (1999). The search-transfer problem. The role of weak ties in sharing knowledge across organization subunits. *Admin. Sci. Quart.*, 44(1), 82-11.

Iannacci, F. (2005). Coordination processes in open source software development: The Linux case study. *E:CO*, 7(2), 21-31.

Kellogg, K.C., W.J. Orlikowski e J. Yates (2006). Life in the Trading Zone: Structuring Coordination Across Boundaries in Postbureaucratic Organizations. *Organization Science*, 17(1), 22-44.

Kogut, B. (2000). The Network as Knowledge: Generative Rules and the Emergence of the Structure. *Strategic Management Journal*, 21, 405-425.

Kogut, B. e A. Metiu (2001). Open-Source Software Development and Distributed Innovation, *Oxford Review of Economic Policy*, 17(2), 248-264.

Kogut, B. e U. Zander (1992). Knowledge of the Firm, Combinative Capabilities and the Replication of Technology, *Organization Science*, 3(3), 383-397.

Lakhani, K.R. e E. von Hippel (2003). How open source software works: free user-to-user assistance, *Research Policy*, 32, 923-943.

Malone, T.W. e K. Crowston (1994). The Interdisciplinary Study of Coordination, *ACM Computing Surveys*, 26(1), 87-119.

March J.G. e H.A. Simon (1958). *Organizations*, New York: Wiley; ed. it. (1966) *Teoria dell'organizzazione*, Milano: Comunità.

Pontiggia, A. e F. Bolici (2006). Codice, Open Source e Diritti di Proprietà: Innovazione, Coordinamento e Diffusione, *atti del VII WOA*, Salerno, 2-3 Febbraio.

Thompson, J.D. (1967). *Organizations in actions*. New York, USA: McGraw-Hill. Edizione italiana, (2002) *L'azione organizzativa*, Torino, IT: ISEDI.

von Hippel, E., and G. von Krogh (2003). Open source software and the 'privatecollective' innovation model: Issues for organization science. *Organization Science* 14(2), 209-223.

Weber, S. (2004). *The success of open source*. Cambridge, MA: Harvard University Press.

Weick, K.E. (1995). *Sensemaking in Organizations*. Londra, U.K.: Sage Publications.

Wenger, K.E. (1998). *Communities of Practices. Learning, Meaning and Identity*. NY, J. Wiley & S.