POLITECNICO DI MILANO

Scuola di Ingegneria dell'Informazione

Corso di Laurea Specialistica in Ingegneria Informatica

Dipartimento di Elettronica e Informazione

IL RUOLO DEI SISTEMI DI GOVERNANCE NELLE MUTUE INFLUENZE DI COEVOLUZIONE DELLA RETE DI SVILUPPATORI E DEL PRODOTTO SOFTWARE: ANALISI EMPIRICA DELL'ESPERIMENTO NATURALE COMPIERE/ADEMPIERE

Relatore: Prof.ssa Chiara FRANCALANCI

Correlatore: Dr. Ing. Francesco MERLO

Tesi di Laurea Specialistica di:

Mirko CURTOLO Matricola: 740228

Marco FRAGNOLI Matricola: 749985

Anno Accademico 2010-2011

Sommario

Il lavoro di tesi propone una metodologia per la stima dell'impatto dell'organizzazione del processo di sviluppo sulla struttura del software. Lo scopo di tale metodologia è supportare le scelte in termini di gestione del team di sviluppo al fine di ottenere una maggiore qualità strutturale del prodotto software sviluppato.

L'obiettivo del presente lavoro è quello di studiare gli effetti di diversi tipi di *Governance* su una comunità di sviluppatori e sulla struttura del software che essi producono. Il lavoro si basa sull'elaborazione di modelli che descrivano le mutue influenze tra caratteristiche di interazione sociale interna ad una comunità di sviluppatori, rappresentate tramite una rete di collaborazioni, e caratteristiche del software descritte mediante metriche che esprimono la qualità della struttura interna del prodotto.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Il lavoro di tesi si propone di approfondire l'analisi dei mutui rapporti tra la struttura sociale che rappresenta le collaborazioni tra i membri di un team di sviluppo e la struttura interna del software che essi producono. L'obiettivo dello studio proposto è chiarire l'impatto di differenti metodi di gestione del processo di sviluppo sulle dinamiche che regolano tali rapporti, con lo scopo di individuare debolezze e punti di forza di ciascun metodo e renderne possibile il miglioramento.

In letteratura, diversi studi dimostrano come la struttura sociale di una comunità di sviluppatori che collaborano alla realizzazione di un software lasci un'impronta sulla struttura del prodotto. Tuttavia è stata osservata anche la relazione opposta, infatti, le caratteristiche dell'interazione tra gli sviluppatori sono influenzate dalla struttura del software. In quest'ottica è stato possibile formulare modelli di interazione che esprimono le mutue influenze tra questi due tipi di variabili.

Questo studio rivolge l'attenzione ad un particolare esperimento naturale che coinvolge due differenti comunità di sviluppatori, gestite secondo diverse filosofie, impegnate nello sviluppo di due prodotti software realizzati partendo da un punto comune e funzionalmente equivalenti.

Il progetto originale, gestito secondo la logica del Commercial Open Source, vede alcuni dei suoi sviluppatori insoddisfatti lasciare il gruppo con l'in-

tento di intraprendere un nuovo progetto, in cui sia prevista una gestione di tipo differente. Viene effettuato un fork, una copia integrale del codice sorgente, in modo da permettere l'avvio di un progetto parallelo. Il metodo di gestione del processo di sviluppo, utilizzato dalla comunità che si avvicina al nuovo progetto, aderisce al paradigma più classico dell'Open Source, il Community Open Source.

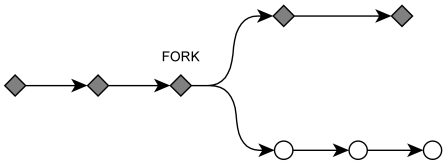


Figura 1.1: Fork tra due progetti di sviluppo software

Attraverso lo studio di questo caso è possibile mettere in evidenza il modo in cui diverse gestioni del processo di sviluppo hanno influenzato la struttura e la qualità di due prodotti software che hanno origine da una radice comune.

L'impatto delle differenti gestioni del processo di sviluppo è osservato tramite l'analisi delle reti di collaborazioni interne alle due comunità di sviluppatori, utilizzando come modello descrittivo la Social Network. Per quanto riguarda gli aspetti relativi alla struttura del software, si fa riferimento ad alcune metriche definite in letteratura, che permettono di descrivere il software sulla base della sua qualità strutturale interna.

L'obiettivo del presente lavoro è approfondire la conoscenza del legame tra proprietà della rete sociale che rappresenta le collaborazioni tra i membri di un team di sviluppo e caratteristiche strutturali del software sviluppato.

In questo lavoro vengono presentati diversi modelli che descrivono le dinamiche di mutua influenza tra caratteristiche della comunità di sviluppatori e qualità strutturali del software, ricavati dall'osservazione dei due casi esemplari forniti dall'esperimento. Lo studio viene integrato con un'analisi atta ad identificare quale, tra i due modelli di gestione del processo di sviluppo, abbia permesso la realizzazione di un prodotto software con qualità

strutturale interna migliore.

Questo documento è strutturato come segue: il Capitolo 2 presenta degli aspetti riguardanti lo stato dell'arte relativo alle tematiche trattate e fornisce le basi teoriche per la comprensione della metodologia utilizzata. Il Capitolo 3 presenta le ipotesi di ricerca e descrive i metodi impiegati per l'implementazione delle analisi effettuate spiegandone il fondamento teorico. Nel Capitolo 4 vengono mostrati e discussi i risultati delle analisi e vengono presentati i modelli di interazione ricavati da esse. Il Capitolo 5 presenta una panoramica conclusiva dei risultati ottenuti e descrive eventuali possibilità di approfondimento e sviluppo futuro dei modelli presentati.

Capitolo 2

Stato dell'arte

"La scienza è una disciplina nella quale la sciocchezza di questa generazione può oltrepassare il punto che ha raggiunto il genio dell'ultima generazione"

Max Gluckman

Da sempre l'ingegneria del software concentra i propri sforzi nel migliorare e affinare le tecniche di produzione di applicazioni informatiche. All'interno di tali processi, lo sviluppo software detiene un ruolo centrale e si inserisce come fattore chiave nel successo della produzione.

Ai fini di migliorare le caratteristiche del processo di sviluppo sono state introdotte tecniche di coordinamento degli sviluppatori e di misurazione della qualità del software sempre più elaborate e precise. Nel corso degli anni sono state elaborate metriche ben definite in grado di fornire accurate descrizioni delle proprietà dei fenomeni analizzati. Metriche e tecniche di misurazione sempre più accurate hanno permesso la creazione di modelli complessi in grado di descrivere con precisione i molti aspetti legati alla produzione del software. Le reti sono un modello spesso utilizzato per descrivere fenomeni legati al campo dell'ingegneria del software. Inoltre, il sempre maggiore interesse da parte dei più disparati ambiti scientifici nei confronti delle reti sociali ha portato a considerare con maggiore attenzione le dinamiche di interazione tra individui.

Il processo di sviluppo software è essenzialmente un processo che coinvolge un determinato numero di individui che collaborano. Da questo punto di vista le reti sociali hanno assunto una non trascurabile importanza anche nell'ambito dello studio di tale processo.

L'ascesa del paradigma Open Source unita a quella delle tecnologie basate su internet, è riuscita nell'impresa di permettere a sviluppatori situati in ogni parte del mondo di contribuire allo sviluppo di uno stesso progetto. Questi fenomeni hanno determinato la nascita di nuove forme di coordinamento basate sulle Social Network.

La possibilità di descrivere sia il software che le reti sociali tra gli sviluppatori che hanno collaborato a produrlo con modelli sempre più precisi e accurati ha fornito nuovi strumenti per le analisi delle relazioni che intercorrono tra processo di sviluppo software e prodotto finale. Unendo l'analisi delle metriche che esprimono la qualità della struttura software con un approccio di studio del processo di sviluppo legato alla teoria delle reti, è possibile analizzare le mutue influenze tra processo e prodotto nell'ambito dell'ingegneria del software.

In questo lavoro è presentato lo studio di un esperimento naturale che vede due diverse comunità di sviluppatori lavorare su un software inizialmente identico. La tesi si concentra sull'analisi delle corrispondenze tra struttura del software e struttura della rete degli sviluppatori, andando a considerare i software ottenuti dai due gruppi di sviluppatori.

In questo capitolo viene presentato lo stato dell'arte e vengono considerati i vari aspetti riguardanti le basi teoriche necessarie a comprendere la metodologia utilizzata per questo tipo di analisi.

Lo stato dell'arte è organizzato nel seguente modo. Nella Sezione 2.1 vengono illustrate le metriche per la valutazione della qualità di un prodotto software, nella Sezione 2.2 si prendono in considerazione i diversi aspetti riguardanti le reti in generale e la specifica applicazione di queste all'analisi del software. Vengono quindi esposte le metriche per l'analisi di una rete, si spiega come è possibile ottenere una rete (o grafo) partendo dal prodotto

software e viene introdotto il concetto di Social Network; nella Sezione 2.3 vengono considerate le influenze reciproche fra prodotto software e modello di interazione tra sviluppatori. La Sezione 2.4 è volta a riassumere tutti gli aspetti legati alla storia dell'esperimento naturale prestando particolare attenzione al caso dei software presi in analisi.

2.1 Qualità del software e metodi di misurazione

Un qualsiasi oggetto possiede una serie di proprietà che lo caratterizzano rispetto ad altri oggetti. L'esigenza di descrivere tali proprietà in modo formale aumenta con l'incremento della complessità dell'oggetto da descrivere. Non solo la definizione dei metodi per descrivere le proprietà ma anche i metodi di quantificarle o misurarle assume una rilevanza fondamentale ai fini di descrivere la qualità di un oggetto.

Una metrica è l'unità di misura di una determinata proprietà e necessita di definizioni precise sia in termini di significato che in termini di metodi di misurazione. Come suggerito da Coulange [25], occorre definire metriche che abbiano le seguenti caratteristiche:

- Semplici e definite in modo chiaro;
- Obiettive ed indipendenti da interpretazioni personali;
- Facili da misurare;
- Fortemente correlate alla proprietà che misurano;
- Robuste e quindi non soggette a grandi variazioni in relazione a piccole; variazioni delle proprietà dell'oggetto analizzato.

Per quanto riguarda l'ingegneria del software, l'adozione di metriche che fossero in grado di descrivere le caratteristiche del software ha richiesto tempo e la discussione su quali metriche adottare ha coinvolto molti studi diversi.

2.1.1 Metriche per la valutazione della qualità del software

La valutazione della qualità del software non è un processo semplice. Già a partire dagli anni '70/'80 furono introdotte alcune metriche per la valutazione della qualità del software [81], ma solo negli anni '90 raggiunsero precisione e complessità tali da essere adottate sistematicamente [68]. A seguito della cosiddetta Software Crisis [5], vennero adottati dei paradigmi abbastanza strutturati per definire le proprietà che una metrica deve possedere. Per quanto riguarda un software, è possibile definire una metrica come quantificazione di una qualche proprietà (intesa come attributo dimensionale o qualitativo) associata al software stesso o alle attività del processo di produzione ad esso correlato [68].

Tra i vari processi correlati alla produzione del software, ha un impatto decisamente predominante il processo di sviluppo. La definizione di metriche, nell'ambito dell'ingegneria del software, può essere maggiormente complessa rispetto a quanto accade in altri ambiti di produzione. Questo avviene soprattutto a causa della notevole complessità del prodotto. Nella produzione software, ad esempio, l'aumento della produttività non è sempre facilmente misurabile. In altri settori, soprattutto per quanto riguarda il manifatturiero, l'aumento della produttività coincide con la produzione di una quantità maggiore di output. Generalmente questo è accompagnato da un aumento dell'efficienza e quindi ad un risultato positivo in termini di processo di produzione. Un aumento della produttività, nel caso della produzione di software, non può essere un risultato legato solamente ad una maggiore capacità di produrre codice. Infatti le funzionalità implementate in un software non sono direttamente correlate al numero di righe di codice che lo compongono [41]. Occorre pertanto definire quale sia una possibile classificazione che permetta di mettere in corrispondenza le caratteristiche dell'implementazione con le funzionalità del software stesso. A tale fine è stata fatta una fondamentale distinzione tra metriche di processo e metriche di prodotto.

Le metriche di processo considerano quell'insieme di caratteristiche pro-

prie del processo di produzione del software: durata, tipologia di processo e altre statistiche legate alle attività del gruppo di sviluppo. Le metriche di prodotto sono invece associate al software vero e proprio, misurandone proprietà quali la complessità, la dimensione e la qualità.

In letteratura sono state definite diverse metriche di misura della complessità, tra le principali abbiamo la *Cyclomatic Complexity* [67], la *Software Science* [46], e l'*Information Flow Complexity* [48]. Ma le informazioni riguardanti la complessità del software non riescono a darci informazioni esaustive sulla qualità del prodotto analizzato.

Per cercare di ridurre la complessità progettuale (dovuta sia alle grandi dimensioni del software, sia alla maggiore criticità delle funzioni richieste [73, 101]) si è pensato ad un approccio alla progettazione che preveda una decomposizione funzionale [29, 75]. In particolare questa decomposizione prevede le divisione del software in parti funzionalmente meno complesse, dette moduli, che rappresentano concettualmente la porzione di software che si occupa di implementare un gruppo ben definito di funzionalità.

Quando i requisiti diventano più articolati, la divisione in moduli permette un approccio allo sviluppo del software più focalizzato a raggiungere un determinato obiettivo. A causa di questa suddivisione è necessario passare da metriche di complessità a metriche legate alla struttura del software; questo approccio si è rivelato particolarmente utile soprattutto dopo l'avvento del paradigma di programmazione Object Oriented (OO).

Molti lavori, in letteratura, hanno aiutato a definire le seguenti proprietà di struttura del software nei sistemi OO [22, 35, 93, 98]:

- *Coupling*: grado di interdipendenza di un modulo nei confronti di altri moduli facenti parte dello stesso software [102].
- *Cohesion*: la proprietà che esprime il grado unione, di aggregazione interna, di un modulo [102].
- *Inheritance*: possibilità di riutilizzo nel modulo in contesti differenti. Ovvero il grado di generalità delle funzioni implementate dal modulo

considerato.

- *Information hiding*: capacità di un modulo di mascherare il proprio funzionamento interno.

Coupling e *cohesion* sono state riconosciute come le proprietà principali per la descrizione della qualità del prodotto software in base al dibattito nato da alcuni importanti pubblicazioni [11, 25, 79, 49, 84].

Il *coupling* è una proprietà che esprime il grado di accoppiamento di un singolo modulo con gli altri moduli che compongono il sistema. Un oggetto A si considera accoppiato ad oggetto B se il primo utilizza funzionalità implementate nel secondo. Ad esempio, tra due oggetti, quello che possiede un livello di *coupling* più basso è generalmente considerato come avente una qualità strutturale interna migliore.

La *cohesion* di un software può essere pensata come una proprietà che indica quanto ciascuna parte che compone un modulo è correlata alle altre parti del modulo stesso. Un alto livello di *cohesion* è generalmente considerato indice di robustezza, leggibilità, affidabilità e riusabilità del codice che compone il modulo. Viceversa un basso livello di *cohesion* è spesso associato a codice complicato da testare e da mantenere.

Di notevole importanza è la considerazione fatta da Darcy et al. [28] che mette in evidenza come non sempre queste due metriche sono indipendenti ma possono influenzarsi a vicenda.

Nel corso degli anni sono state proposte metriche che misurano diversi aspetti legati alle proprietà sopracitate: le più utilizzate e conosciute sono quelle proposte da Chidamber e Kemerer [23] (WMC, DIT, NOC, CBO, RFC e LCOM) e quelle proposte da Brito e Abreu [31] (COF, PF, AIF, MIF, AHF e MHF). Di seguito verrà riportata una definizione per ognuna delle sei metriche proposte da Chidamber e Kemerer che prendono il nome di *Suite CK* [23]:

- **Weighted Methods per Class (WMC)**: WMC misura la complessità dei metodi di una classe ed è pari alla somma dei pesi dei singoli

metodi presenti al'interno della classe. Nel caso in cui un metodo avesse peso unitario, il WMC sarebbe pari al numero di metodi presenti nella classe. Tale metrica può essere vista come un indicatore dello sforzo necessario per sviluppare e mantenere una particolare classe.

- **Depth of InheritanceTree (DIT)**: DIT misura la massima profondità dell'oggetto nell'albero di ereditarietà. L'albero è costituito dalle relazioni padre-di e dalla sua inversa figlio-di che si vengono a costituire a causa dell'uso dei meccanismi di ereditarietà forniti dai linguaggi Object Oriented. Questa metrica misura la profondità dell'oggetto considerato all'interno dell'albero ovvero il numero dei suoi antenati.

- **Number Of Children (NOC)**: NOC misura il numero di classi che ereditano da quella considerata, ovvero il numero di discendenti nell'albero dell'ereditarietà.

- **Coupling Between Objects (CBO)**: Il CBO di una classe misura il numero di classi con cui è accoppiata e quindi il numero di dipendenze. Due classi si dicono accoppiate quando i metodi definiti in una classe usano metodi o istanze di variabili definiti nell'altra classe.

- **Response For a Class (RFC)**: Questa metrica misura il numero di metodi appartenenti al responseset di una classe, che si definisce come l'insieme di metodi che potenzialmente possono essere eseguiti in risposta ad una chiamata inviata ad un oggetto.

- **Lack of Cohesion Of Methods (LCOM)**: La metrica LCOM è una misura della mancanza di coesione tra i metodi di un oggetto. In generale un oggetto coeso esegue una sola funzione, mentre un oggetto non coeso tende ad eseguire più funzioni tra di loro diverse. Una LCOM alta è indice di bassa qualità del software.

Dopo aver parlato delle più importanti metriche riguardanti la complessità e la struttura del software è interessante prendere in considerazione una

metrica legata in particolare al processo di sviluppo del software. Tale metrica prende il nome di Function Points (FP) e si basa sull'idea di misurare le funzionalità implementate da un software identificando e quantificando le funzionalità che vengono offerte dal software all'utente.

I FP provvedono una stima della complessità del software nonché una stima della quantità di lavoro richiesto e, di conseguenza, dei costi di sviluppo. Danno quindi un'idea più precisa dello sforzo necessario per la realizzazione di un software [67], cosa non facile da definire, in quanto come detto precedentemente non è possibile far riferimento alle sole righe di codice per determinare l'efficienza di un processo di sviluppo.

L'analisi dei Function Point (FPA) valuta la complessità di un'applicazione misurando cinque tipi di elementi costituenti, detti *function Types*, legati all'interazione dell'applicazione con l'utente finale. Il numero totale di FP calcolati è pari alla somma pesata di queste cinque componenti.

Tabella 2.1: Function Types e relativi pesi nel calcolo dei Function Points

Elemento	**Peso**
Input (*external input*)	4
Output (*external output*)	5
Interrogazioni (*external inquiries*)	4
File (*internal logic files*)	10
Interface (*external interface files*)	7

La Tabella 2.1 mostra le cinque componenti con i rispettivi pesi associati. Il numero di input quantifica i dati che vengono richiesti dall'applicazione all'utente, mentre il numero di output rappresenta i dati che vengono forniti all'utente dall'applicazione. Il numero di interrogazioni quantifica le possibili richieste interattive che l'utente può effettuare all'applicazione, mentre il numero di file rappresenta la quantità di gruppi di informazioni correlate che derivano dagli input o dalle richieste effettuate dagli utenti. Infine, il numero di interfacce quantifica i punti di contatto dell'applicazione con

sistemi esterni, ad esempio altre applicazioni.

La somma pesata dei cinque function type descritti è detta *Unadjusted Function Point* (UFP), e deve essere ulteriormente raffinata in modo da considerare la complessità dell'applicazione stessa e del suo contesto: a tal fine la *function point analysis* prevede una serie di caratteristiche che devono essere pesate secondo una specifica scala di valori. La somma dei pesi assegnati a ciascuna di esse viene utilizzata per determinare un fattore di aggiustamento (*Complexity Adjustment Factor*, CAF). Il numero di Function Points implementati da un segmento di codice risulta quindi dall'espressione:

$$FunctionPoints = UFP * CAF$$

Il metodo per il calcolo dei FP è stato introdotto prima dell'affermazione della programmazione Object Oriented e quindi ha richiesto un adattamento della metodologia per questo paradigma di programmazione [23], confermando comunque la propria validità attraverso molti studi empirici [57, 63, 60, 13].

2.1.2 Relazione tra qualità del software e costi di mantenimento

Molti studi in passato sono stati concordi nell'affermare che i costi di mantenimento del software sono fortemente influenzati dalla qualità del software e in particolare da proprietà come leggibilità, affidabilità e testabilità del codice che lo compone [65, 107].

MacCormack et al. [1] proposero un modello di propagazione dei costi che forniva evidenza dell'impatto del *coupling* sui costi di mantenimento. Un altro studio, quello svolto da Tan e Mookerjee [100], analizzò l'impatto dell'entropia, che può essere vista come un modo di esprimere la qualità del software, sui costi di mantenimento.

Come suggerito da Slaughter et al. [85], si può assumere che massimizzandone la qualità, il software diviene un prodotto economicamente più conveniente. Questo anche a dispetto di un investimento iniziale più elevato.

Li e Henry [61] e Harter et al. [27] hanno dimostrato che i costi di mantenimento sono influenzati dalle metriche definite da Chidamber e Kemerer [23]. Un alto valore di queste metriche porta ad un maggiore sforzo legato alla manutenzione. Binkley e Scatch [14], Chidamber et al. [88] e Darcy et al. [28], hanno evidenziato la correlazione esistente tra *coupling* e difficoltà di sviluppo e mantenimento del software. Tuttavia è evidente che le difficoltà siano anche in relazione con la complessità delle funzionalità che il software implementa, Banker e Slaughter [9].

Slaughter et al. [85] hanno provato che è maggiormente conveniente investire nella qualità di progettazione del software molto presto nel processo di sviluppo. Questo in ordine di massimizzare il profitto derivante da una maggiore qualità.

Tutti questi studi dimostrano la forte connessione tra metriche di qualità di un software e i costi legati alla manutenzione dello stesso. Sottolineano quindi l'importanza di mantenere elevata la qualità del prodotto software.

La relazione tra qualità del software e costi di manutenzione è utilizzata all'interno di questo lavoro per rapportare ciascuna metrica ad un impatto reale sui costi di progetto. In tal modo è possibile stabilire dei livelli di qualità e quindi dei valori di metriche maggiormente desiderabili rispetto ad altri.

2.2 Reti

Negli ultimi anni, gli studi che riguardano le reti (o grafi) hanno ricevuto particolare attenzione soprattutto a causa dell'importanza assunta da fenomeni modellabili attraverso di esse. A titolo di esempio si può citare l'incredibile influenza esercitata al giorno d'oggi dalla rete Internet e dalle sue dinamiche. Di fondamentale rilevanza sono anche le reti sociali con le loro molteplici applicazioni. Parte del seguente lavoro di tesi è stata possibile proprio grazie ai risultati raggiunti in questo settore.

Dopo una breve introduzione sulle metriche più significative tra quelle che descrivono le proprietà delle reti, verrà offerta una panoramica dei di-

versi utilizzi delle reti come modelli che descrivono l'evoluzione di fenomeni nell'ambito dell'ingegneria del software. Infine verranno discusse le varie dinamiche riguardanti le reti sociali con particolare riferimento all'interazione sociale all'interno di una comunità di sviluppatori.

2.2.1 Metriche nelle reti

Alla fine degli anni '70 è stato sviluppato un metodo di descrizione quantitativo per le dimensioni principali di una rete e per le metriche caratteristiche che ne descrivono le proprietà.

Un grafo ci permette di studiare in modo formale fenomeni di differente natura negli ambiti più disparati, utilizzando metodi matematici. In letteratura sono stati scritti importanti lavori riguardanti le reti sociali [103], il World Wide Web [77], le reti neurali [55]. Tuttavia per descrivere proprietà e metriche riguardanti le reti è necessario un breve cappello introduttivo al fine di presentare quei concetti preliminari essenziali ad una descrizione accurata di una rete.

Si ricorda che convenzionalmente chiamiamo rete (o grafo) un insieme composto da archi e nodi. I nodi, o vertici, sono gli elementi della rete che modellizzano le entità fondamentali costituenti la rete e compongono un insieme $V = \{v_1, v_2, \ldots, v_n\}$. Le relazioni tra i nodi sono rappresentate dagli archi che possono essere orientati (l'ordine in cui compaiono i nodi è rilevante) o meno, cioè possono dirci se c'è influenza reciproca tra i due nodi o se solo uno dei due influenza l'altro.

L'insieme degli archi (in inglese *Edges*) si indica con la seguente simbologia: $E = \{e_1, e_2, \ldots, e_n\}$. Ogni arco è identificato univocamente dalla coppia dei nodi che unisce. Date le definizioni di nodi e archi è facile descrivere un grafo G con una notazione $G = < V, E >$, ovvero che un grafo è formato da un insieme di vertici V e da un insieme di archi E che unisce due vertici appartenenti a V.

In alcuni casi può essere comodo utilizzare più insiemi di vertici, ovvero vertici di tipi diversi. Nel caso delle reti a due modi ad esempio, la rete

è un insieme $G = < V, U, E >$ in cui V e U sono due insiemi di vertici di tipo differente (e che quindi descrivono oggetti di tipo differente) e E è un insieme di archi tali che: $E = (v, u)$ in cui v appartiene all'insieme V mentre u appartiene all'insieme U.

Utilizzando le reti a due modi è possibile descrivere situazioni molto utili nel ambito dell'ingegneria del software. Ad esempio un insieme di vertici può rappresentare gli sviluppatori ed un altro i progetti. In questi caso un arco associa ogni sviluppatore ad uno o più progetti e viceversa. In reti di questo tipo non hanno senso relazioni tra vertici dello stesso tipo.

Si definisce cammino tra due vertici v_1 e v_2, all'interno di una rete, una sequenza di archi che li congiunge, ovvero che ha come estremi i nodi v_1 e v_2. Un cammino minimo che congiunge due nodi è quindi il cammino tra di essi che coinvolge il minor numero possibile di archi, chiaramente questo è valido in un contesto in cui gli archi non siano pesati. Seguendo questa descrizione formale è possibile definire metriche chiare e robuste per le reti come si è fatto nella precedente sezione per il software. Uno dei concetti alla base dell'analisi delle reti sociali è quello di *centrality*, definita come importanza di un individuo all'interno della rete sociale (Social Network) in cui è inserito [39]. La *centrality* è fortemente collegata al potere di un nodo all'interno della rete.

Nelle reti sociali ogni nodo rappresenta una persona, o meglio un attore, e gli archi rappresentano una relazione tra questi. In queste reti la centralità si riferisce al concetto di potere sociale, cioè all'influenza o al prestigio che il singolo individuo ha all'interno della rete in cui è inserito [39].

Nel 1979 Freeman [39] ha introdotto quelle che poi sono diventate le metriche di centralità maggiormente diffuse: *degree centrality*, *betweenness centrality* e *closeness centrality*.

- **Degree Centrality**: è il più semplice indicatore della centralità [6] ed è definita come il numero di archi (o link) di un nodo, normalizzato per il numero di nodi presenti nella rete. Ovviamente un grado di connessione molto alto all'interno della rete è indice di un elevata

importanza del nodo nella rete stessa. Infatti, l'alto numero di connessioni permette di condizionare il comportamento della rete [80]. Nel caso in cui il grafo fosse orientato è chiaramente possibile definire una *InDegree centrality* (considerando solo gli archi entranti in un nodo) e una *OutDegree centrality* (considerando solamente gli archi uscenti).

- **Betweenness centrality**: questa metrica indica la frequenza media con cui un nodo è attraversato dal percorso più corto tra altri due generici nodi. Un nodo con alta *betweenness centrality* coinvolgerà e controllerà molti flussi informativi; questo è intuitivo se si pensa ad un nodo A che si trova sul percorso più corto che connette due generici nodi. Questi generici nodi per comunicare devono riferirsi ad A, o almeno anche a questo nodo. Questa metrica potrebbe quindi essere interpretata come la capacità di un nodo di raggiungere altri nodi della rete, essendo posizionato su un percorso obbligato tra loro. Per rendere più semplice il calcolo della *betweenness centrality* di un nodo, Newman [72] propone di calcolare tale metrica non in riferimento al percorso più corto tra due nodi generici, ma in riferimento ad un percorso casuale. In un lavoro più recente, Burt [17] suggerisce che il concetto di *betweenness centrality* è fortemente connesso ad un particolare caratteristica strutturale delle reti, ovvero la *Structural Holes*. Il concetto di *Structural Holes* è legato alla mancanza di compattezza nella rete, il che beninteso non è una caratteristica di per sé negativa. La presenza di queste strutture indica la caratteristica di una rete complessa di non essere omogeneamente connessa al proprio interno. Osservando alcune reti è possibile notare come esistono delle sotto-reti formate da nodi fortemente connessi tra loro e debolmente connessi con il resto della rete. La nozione di *betweenness centrality* è legata a quella di *Structural Holes* proprio perché un alto grado di *betweenness centrality* indica la capacità del nodo di far comunicare individui provenienti da diverse sottoreti. Un nodo che dispone del potere di mettere in relazione due diverse sottoreti, composte da un

numero elevato di individui, è un nodo che assume un grado elevato di importanza anche se dispone di poche connessioni, cioè se possiede un basso livello di *degree centrality*.

- **Closeness centrality**: è la terza metrica proposta da Freeman [39]. Esprime la lontananza di un nodo da tutti gli altri nodi che compongono la rete, basandosi sul calcolo dei cammini minimi. Questo tipo di metrica è meno intuitiva e più difficile da calcolare delle precedenti. Freeman ha osservato che il concetto di *closeness centrality* può essere associato all'idea di indipendenza di un nodo. Questo è principalmente dovuto al fatto che, a prescindere dal suo grado, se un nodo è relativamente vicino ad un altro nodo molto importante, con un alto valore di *degree* o *betweenness centrality*, è meno dipendente dagli altri nodi della rete. Ovvero pur essendo meno importante, è un nodo che può comunicare facilmente con tutta la rete proprio per la sua vicinanza con un nodo di grande importanza. Considerando un intera rete, è di grande interesse che il livello di *closeness centrality* medio non sia troppo basso perché indicherebbe una maggiore dispersione dei nodi che la compongono e quindi una diffusa difficoltà di comunicazione da parte di molti nodi.

Chiaramente ognuna delle metriche sopra citate non può essere calcolata per nodi non raggiungibili all'interno di una rete.

Altre metriche differenti sono state proposte recentemente; un esempio sono la *harmonic centrality*, proposta da Stephenson e Zelen [97], che considera la distanza armonica in sostituzione alla distanza minima, o il concetto introdotto da Bonanich [15] con il nome di *eigenvector centrality* che misura la centralità come il principale autovettore della matrice di adiacenza di tutta la rete. Quest'ultima metrica ha riscosso scarso successo negli anni a causa della sua complessità di calcolo. Ad ogni modo ci sono articoli come quello di Borgatti [16] che affermano ci sia un'affinità concettuale tra la nozione di *eigenvector centrality* e quella di *degree centrality*.

A prescindere dal tipo di centralità espresso dalle metriche, un nodo con un importanza molto elevata può essere considerato fortemente influente all'interno della rete stessa. La maggiore o minore uniformità di distribuzione dell'importanza all'interno delle reti è in grado di fornire indicazioni sul modo in cui l'informazione circola all'interno della rete e su chi controlla la maggior parte dei flussi informativi.

2.2.2 Software Network

Sempre restando all'interno del contesto riguardante le reti, è stato visto quanto tali strutture siano fondamentali per la comprensione di numerosi e diversi sistemi complessi, che vanno dallo studio delle proteine e dei geni allo studio del web [10].

Uno dei sistemi complessi che può essere modellizzato attraverso una rete è il software, ma prima di addentrarsi nella descrizione di come questo sia possibile è necessario spendere alcune parole sul concetto di Scale Free network.

Molte delle reti analizzate in questi studi presentano una struttura Scale Free. Tale struttura è caratterizzata dal fatto che il numero di connessioni tra i nodi che compongono la rete sembra seguire la distribuzione di Pareto ed è rappresentabile come una curva Power-law. In questo tipo di curve la coda cresce come una potenza negativa della variabile indipendente e la probabilità di trovare valori molto lontani dalla media non è trascurabile. All'interno di una rete di questo tipo esistono nodi con *degree* molto elevato rispetto alla media della rete, che prendono il nome di *hub*.

Le reti Scale Free sono molto diverse dalle reti casuali (Random Network) che furono le prime a esser studiate, già dagli anni 60 [36]. La principale differenza tra i due tipi di reti sta nella distribuzione del grado. Nelle reti casuali, infatti, tutti i nodi hanno in media lo stesso numero di archi (e quindi la distribuzione del grado è mediamente costante) mentre in quelle Scale Free, come è stato accennato, la distribuzione segue quella di Pareto. In altre parole una rete casuale sembra rispettare un principio più democratico

nella distribuzione degli archi rispetto ad una Power-law, in cui gli *hub* hanno un potere, all'interno della rete, molto superiore alla media [10].

Recentemente, alcuni studi hanno cercato di descrivere i sistemi software come reti Scale Free, in quanto le enormi dimensioni raggiunte dal questi software non sembrano sottostare alle dinamiche tipiche delle reti casuali [87].

Una rappresentazione schematica di questi complessi sistemi software può essere portata a termine andando a considerare le interazioni tra i moduli ai vari livelli di granularità (funzioni, classi, interfacce, package, etc.) e considerando queste come gli archi che uniscono le varie entità prese in considerazione.

Entrambi i software considerati in questo lavoro sono stati implementati con tecnologia Java che, come è noto, è uno dei più importanti linguaggi a seguire il paradigma Object Oriented. Le rappresentazioni, sotto forma di reti, di software realizzati con questo linguaggio di programmazione sono molteplici e di varia natura [40]. Tipicamente, in queste rappresentazioni l'unità fondamentale considerata è la classe Java, un oggetto che possiede degli attributi e dei metodi. Per attributo si intende una variabile che corrisponde ad una proprietà dell'oggetto e per metodo si intende una funzione che opera su un insieme di attributi.

Uno dei primi studi che ha evidenziato il ruolo predominante delle reti Scale Free nella rappresentazione di software realizzato in linguaggio Object Oriented è stato quello svolto da Mills nel 1988 [70]. Vi sono altri casi significativi, in letteratura, che riguardano l'analisi di grafi relativi a software di questo tipo in cui gli oggetti del sistema sono rappresentati da nodi e le relazioni tra i vari oggetti, determinate dalle chiamate che avvengono tra essi a *runtime*, sono rappresentate da links. Potanin et al. dimostrano come, rappresentando in questo modo le relazioni tra oggetti all'interno del software Object Oriented, si ottiene una rete di tipo Scale Free [4].

Altre dinamiche di questo tipo sono state individuate in grafi relativi ai grandi sistemi software Open Source Java, C e C++, tali risultati sono stati

confermati dagli andamenti dell'*InDegree* e dell'*OutDegree*, soggetti ad una distribuzione di Pareto [89].

2.2.3 Algoritmi di visualizzazione delle reti

Per rappresentare graficamente una rete complessa (come quella ricavata da un software), è necessario adottare degli algoritmi adatti. La parte della teoria dei grafi che si occupa di studiare e determinare i metodi di visualizzazione dei grafi si chiama *graph drawing*. Lo scopo principale di questi studi è determinare algoritmi che mostrino certe proprietà del grafo rappresentato [12].

Quando si tratta di visualizzare grafi ad elevata cardinalità (ad esempio le reti che rappresentano le connessioni tra le classi di un software) si può incorrere in problemi detti di *overplotting*, ovvero problemi che riguardano una visualizzazione che non permette di comprendere le proprietà strutturali del grafo [34].

Identificare le caratteristiche e quindi le regolarità all'interno di un grafo complesso è un obiettivo fondamentale della teoria dei grafi ed è una particolare tecnica di *Data Mining* associata a questo tipo di rappresentazioni.

Gli algoritmi di *graph drawing* hanno l'obiettivo di ottenere una rappresentazione grafica che soddisfi dei criteri, siano essi estetici, come rappresentare le simmetrie all'interno di un grafo, siano essi funzionali, come posizionare un'insieme di vertici vicini.

Particolare importanza all'interno di questo campo è stata assunta dagli algoritmi detti *Force Directed*. Questo tipo di algoritmi è di facile comprensione poiché rappresenta un grafo considerandolo come un sistema fisico di corpi con delle forze che agiscono tra di essi. Ciascuna disposizione dei corpi nello spazio relativamente a tutti gli altri corpi che compongono il sistema determina uno stato di energia che questo tipo di algoritmi tende a minimizzare. Questo significa che l'algoritmo dispone i vertici del grafo come fossero dei corpi che si respingono e considera le connessione tra i vertici come delle forze che li avvicinano. Gli algoritmi *Force Directed* minimizzano l'energia

del sistema nel senso che cercano di posizionare i corpi in modo che la forza agente su di essi sia nulla [12].

Un algoritmo di particolare interesse per lo scopo di questo lavoro è quello di Kamada-Kawai. In questo tipo di algoritmo ogni connessione è interpretata come un molla. Una connessione tra due vertici è quindi considerata come una forza di attrazione tra due corpi che prevede una costante elastica [56]

Il metodo tenta di disegnare i grafi in modo che la distanza euclidea tra due vertici tenda ad avvicinarsi al numero di archi che compongono il cammino minimo tra i vertici, cioè alla distanza geodesica. Lo scopo di questo algoritmo è trovare un disegno in cui, per ogni coppia $< u, v >$ di vertici, la distanza euclidea tra u e v sia proporzionale a quella geodesica tra tutte le coppie di nodi che compongono il sistema.

L'algoritmo Kamada-Kawai sceglie la costante elastica della forza che attrae due corpi in maniera tale che tra i vertici che hanno distanza geodesica più piccola esista una forza di attrazione maggiore, e quindi i nodi siano rappresentati più vicini nello spazio. L'obiettivo di questo algoritmo è raggiungere un punto di equilibrio che azzeri le forze agenti sui vari corpi rappresentati dai vertici del grafo. Tuttavia l'equilibrio deve rispettare anche altri due criteri:

- minimizzare il numero delle intersezioni tra archi, e
- distribuire i vertici e gli archi in modo uniforme.

L'algoritmo non giunge necessariamente ad un minimo globale della funzione di energia ovvero non annulla completamente le forze agenti sui corpi. Quella che viene raggiunta è una minimizzazione locale dell'energia. Trovare un minimo globale è difficile se lo spazio di ricerca è ampio, la strategia utilizzata da Kamada-Kawai è quella di trovare prima un minimo locale, in seguito i nodi vengono spostati in modo da minimizzare anche il livello di energia globale.

L'energia totale del sistema è ridotta risolvendo un'equazione differenziale per ogni vertice. Il riposizionamento viene ripetuto fino al raggiungimento di una livello di energia inferiore ad una certa soglia prefissata.

2.2.4 Social Network

Le relazioni tra diversi individui costituiscono quello a cui tipicamente ci si riferisce con il termine Social Network. Possono essere definite delle Social Network anche facendo riferimento al complesso rapporto di contratti sociali che, come accennato in precedenza, esistono tra persone che collaborano alla realizzazione di un progetto.

Le variabili umane come ad esempio il comportamento dei singoli individui che collaborano e la conseguente evoluzione delle relazioni che li legano, sono soggette ad incertezza, ovvero non possono essere completamente previste. Proprio in relazione alla difficoltà di modellare la loro evoluzione, le Social Network sono considerate sistemi complessi [103, 2].

Storicamente sono stati adottati due diversi approcci per quanto riguarda l'analisi delle Social Network [59]:

- **Approccio individualista**: è quello che è stato adottato maggiormente nelle scienze sociali. Si basa sull'assunzione che gli individui si comportino come se fossero isolati, ignorando il comportamento degli altri individui che compongono la rete. Di conseguenza solo le caratteristiche specifiche di ogni singolo individuo sono considerate. Questo permette di focalizzarsi sull'incremento dell'efficienza del lavoro del singolo mentre l'interazione con l'ambiente circostante non è tenuta in conto.

- **Approccio strutturale-relazionale**: Contrasta con l'approccio individualista ed è basato sull'assunzione che il comportamento del singolo individui è influenzato dal resto della rete. Per fare questo è necessario innanzitutto conoscere la struttura e le proprietà della rete sociale. È importante, poi, identificare metodi di misura di tali carat-

teristiche strutturali e formulare ipotesi sull'impatto che esse hanno sui comportamenti è quindi il fondamento di questo tipo di approccio.

Per quanto riguarda il lavoro che sarà svolto in questa tesi verrà adottato un approccio strutturale relazionale.

La struttura sociale è stata definita da White, Boorman e Brieger [45] come regolarità nei modelli relazionali tra entità concrete. L'approccio relazionale è quindi quello che si presta maggiormente ad identificare regolarità nella struttura di rapporti interpersonali implicati in un processo di sviluppo software.

Data la natura critica dello sviluppo software, che si presenta come fattore cruciale per il successo di un azienda che produce questo tipo di prodotti, il modello manageriale posto a controllo di tale processo assume un ruolo fondamentale. Ogni modello impone regole, politiche e comportamenti ben definiti alle persone che compongono il team di sviluppo.

2.2.5 Gestione del processo di sviluppo software e Network Governance

Il processo di sviluppo software è un'attività critica e, come ogni altro processo aziendale, è soggetta ad attività di gestione e controllo da parte di chi detiene la responsabilità della buona riuscita del processo.

Questa attività prende il nome di *Governance* e, non solo nel settore informatico, può essere descritta come il processo gestionale responsabile del successo del progetto. Secondo Renz [82], l'attività di Governance svolge diverse funzioni che includono in particolare:

- Stabilire regole, politiche e responsabilità legate al processo;
- Valutare e selezionare i progetti proposti in base alle risorse disponibili;
- Rendere operativi i progetti;
- Implementazione di un'organizzazione adatta al processo;

- Definire gli obiettivi;
- Il controllo dei processi in corso d'opera;
- Monitoraggio dei risultati raggiunti in relazione agli stakeholder;
- Valutazione dei risultati raggiunti;
- Risoluzione di problemi ed eccezioni relative ai processi controllati.

Sagers [90] afferma che la forma di Governance può essere influenzata dalle condizioni in cui essa viene applicata, tuttavia il suo obiettivo è sempre la minimizzazione dei costi di transazione [105]. In particolare per quanto riguarda i servizi, queste transazioni avvengono non solo tra diversi processi, ma anche all'interno dei processi stessi. La riduzione dei costi di tali scambi è un fattore chiave per il successo di un processo aziendale.

La scelta di un determinato processo di gestione, infatti, deve allinearsi alle caratteristiche della transazione. In questo modo non solo la Governance influenza il processo gestito, ma il prodotto finale a cui tale processo è rivolto, e le caratteristiche stesse del processo, influenzano il tipo di Governance.

Come accennato, le relazioni tra diverse persone costituiscono quella rete a cui tipicamente ci si riferisce con il termine Social Network. Possono essere definite delle Social Network anche facendo riferimento al complesso rapporto di contratti sociali che, come accennato in precedenza, esistono tra persone che collaborano alla realizzazione di un progetto.

Gestire un processo di sviluppo software implica, infatti, pianificare attività e organizzare risorse con l'obiettivo globale di raggiungere una serie di obiettivi di progetto [74]. La letteratura fornisce una quantità sempre crescente di prove delle relazioni tra struttura aziendale, ovvero l'organizzazione e il modello di comunicazione interna, con la struttura interna del software.

Lo sviluppo software è un processo molto differente da quelli di produzione che avvengono nel settore manifatturiero, in cui è possibile raggiungere ottimi risultati, ad esempio attraverso le tecniche di *Total Quality Managment*

[54] ampiamente adottate in Giappone. Un software è significativamente diverso da un semplice manufatto.

I processi di produzione del software sono incentrati sul lavoro creativo, impossibile da standardizzare e difficile da controllare, ed hanno caratteristiche peculiari che portano ad una significativa variazione delle performance da caso a caso [2].

La creazione di gerarchie organizzative e metodi nel campo del software non ha sempre l'effetto desiderato, cioè quello di aumentare le performance degli sviluppatori; in alcune situazioni potrebbe addirittura inibire la creatività degli sviluppatori causando un calo di produttività. Per questo motivo è interessante studiare quali siano i meccanismi di coordinamento più adatti per il processo di sviluppo di un software.

In un contesto Open Source le esigenze a cui è necessario rispondere sono molteplici. Oggi, il paradigma di sviluppo Open Source rappresenta una sfida al modo classico di sviluppare software. Migliaia di sviluppatori geograficamente lontani lavorano insieme, producendo prodotti software di alta qualità che in molti casi riescono a guadagnare consistenti quote di mercato, a scapito dei Closed Source concorrenti [104]. Per di più, le comunità Open Source sono dinamiche, gli sviluppatori possono liberamente entrare nella comunità o lasciarla, e l'assegnazione dei compiti è auto-determinata.

Molte persone (che, come detto, potrebbero trovarsi fisicamente in parti opposte del globo) collaborano in alcuni casi per sviluppare la stessa parte di un software [109, 71]. In questo caso è chiaramente necessario che i vari contributi vengano integrati al fine di generare un prodotto coerente [26]. Diversi lavori, in letteratura, hanno mostrato come la teoria dell'organizzazione sia meno efficace se applicata allo sviluppo del software [33, 69, 96, 58].

Recentemente uno dei paradigmi che ha preso particolarmente piede, rispetto a quelli classici, è quello della *Network Governance*. All'intero della teoria dell'organizzazione la *Network Governance* ha assunto una certa rilevanza soprattutto a causa della sua vasta applicazione nei progetti Open Source e per l'impatto che essa ha sul prodotto finale.

Una definizione di *Network Governance* è la seguente:

> La *Network Governance* coinvolge un determinato numero di attori che siano persistenti e soggetti ad un organizzazione strutturata. Questi attori collaborano per la creazione di un prodotto o un servizio sulla base di contratti impliciti che ne permettono il coordinamento. Questi contratti non sono vincoli legali ma possono essere considerati vincoli sociali [19].

Questo tipo di gestione ha recentemente iniziato ad essere considerata come un'alternativa ai più noti sistemi di coordinamento come Gerarchia e Mercato [18].

Facendo riferimento alla teoria dell'organizzazione classica, la forma di coordinamento denominata Gerarchia è, come suggerisce il nome, un metodo di gestione dei processi aziendali che mette in relazione i vari individui che collaborano per la realizzazione di un prodotto in termini di ruoli gerarchici. È quella più vicina al paradigma del Taylorismo e prevede che il potere decisionale sia mantenuto ai livelli più alti gerarchicamente. In questo modo le attività decisionali e quelle di controllo vengono eseguite da individui in alto nella scala gerarchica mentre i vari compiti legati alla produzione sono divisi in modo funzionale e delegati agli individui che si trovano in basso nella scala gerarchica.

Questo sistema prevede sostanzialmente che il coordinamento sia dettato dai ruoli manageriali.

Diversamente, i sistemi di coordinamento che si basano sul Mercato non prevedono unicamente ruoli gerarchici ma utilizzano come sistema di coordinamento la transazione.

Una transazione è una scambio di beni, servizi o informazioni tra diversi soggetti economici. Queste transazioni regolano i rapporti tra individui sia esternamente che internamente all'ambito aziendale e si basano sulle dinamiche di variazione qualità-prezzo. Questo meccanismo di mutuo coordinamento dei singoli individui attraverso un sistema di Mercato elimina

l'esigenza di un forte controllo gerarchico e prevede una maggiore uniformità dei ruoli dal punto di vista del potere decisionale.

Powell [76] cominciò a studiare le relazioni tra *Network Governance* e gli altri sistemi di coordinamento noti in precedenza, ovvero cercando di capire se questo meccanismo di coordinamento fosse da mettere in rapporto con Gerarchia e Mercato, posizionandosi in qualche modo tra questi due estremi, o fosse un meccanismo di coordinamento a sé stante.

Recentemente in letteratura [18], si è arrivati a considerare la *Network Governance* come una forma di coordinamento con caratteristiche peculiari in fatto di meccanismi di risoluzione dei conflitti e politiche di gestione.

Alla luce di questa considerazione della *Network Governance* come nuovo paradigma di coordinamento diviene interessante studiare quale ruolo hanno le caratteristiche della rete sul coordinamento ovvero sulla collaborazione tra i vari individui che compongono la rete sociale.

Un interessante studio condotto da Dorat e Delahaye [30], basandosi sulla teoria dei giochi, ha misurato il grado di cooperazione per il raggiungimento di un obiettivo comune che sorge tra individui che si relazionano attraverso diversi tipi di Social Network. Le evidenze empiriche apparse dagli esperimenti effettuati hanno suggerito che in una rete di tipo casuale (Random Network) la collaborazione emersa è maggiore di quella emersa nelle reti Scale Free. Questo studio suggerisce che le caratteristiche della rete su cui la *Network Governance* si basa sono fondamentali ai fini di un buon coordinamento degli attori in gioco. Chiaramente questo tipo di Governance è particolarmente interessante ai fini della comprensione delle dinamiche interne dei software Open Source considerati in questo lavoro.

2.3 Influenze tra prodotto software e sviluppatori

Recenti ricerche hanno messo in evidenza la presenza di una correlazione tra la struttura interna di un'organizzazione e la struttura dei prodotti che essa crea. In particolare anche nello studio del software si è cercato di capire

come la rete degli sviluppatori e quella estrapolata dal software presentassero simmetrie.

Tendenzialmente, in una struttura organizzata, alle persone vengono assegnati diversi compiti, ruoli e responsabilità, mentre la progettazione di un prodotto avviene attraverso l'allocazione delle funzionalità ai diversi componenti.

In questo contesto è chiaramente necessaria sia un'interazione tra i membri dell'organizzazione (che dovranno comunicare tra loro al fine di portare a termine i propri compiti) sia una correlazione tra le varie parti che compongono un prodotto e che insieme devono implementare le funzionalità finali.

Nel tempo è stato mostrato che, dovendo gli sviluppatori comunicare più intensamente quando i loro compiti sono correlati e dovendo avere i componenti del prodotto a loro volta delle interdipendenze, le connessioni tra i membri all'interno della struttura organizzativa rispecchieranno (saranno allineate) con quelle della struttura del prodotto [3, 94, 47, 8].

Gli studi sul software hanno mostrato come ci sia una sostanziale differenza nell'influenza che la struttura organizzativa degli sviluppatori ha sul prodotto e viceversa. Infatti nel caso della realizzazione di un prodotto Open Source la struttura della rete dei progettisti ha un forte impatto su quello che è il prodotto finale [37].

Nel caso di un organizzazione più legata al classico paradigma Closed Source la struttura del software influisce fortemente sulla struttura del team di sviluppo [37], ma, come venne evidenziato già a partire dalla fine degli anni 60 da Conway [24], l'organizzazione dei grandi progetti può infondere un'impronta sulla struttura del prodotto software.

Tuttavia la relazione mostrata da Conway non è verificata per ogni tipo di prodotto nel campo dell'ingegneria, Sosa et al. [64] mostrano, ad esempio, come non ci sia un totale allineamento tra la struttura di un motore aereo e la struttura del team di progettisti.

Questa considerazione è interessante anche alla luce di una precedente

osservazione di Sosa che metteva in evidenza come la struttura di un prodotto complesso influenzasse quella del team di sviluppo [64]. Il software presenta comunque caratteristiche peculiari in quanto è facilmente modificabile e basato su un'attività creativa umana (*human-centered*), quindi le relazioni tra struttura organizzativa degli sviluppatori e il design del prodotto possono essere molto più variegate e complesse rispetto ai più comuni ambiti industriali.

Inoltre i software Open Source presentano dinamiche sostanzialmente differenti rispetto a quelle dello sviluppo Closed Source esaminato da Conway. Come visto in precedenza, le strutture organizzative Open Source che presentano piccoli team distribuiti in tutto il mondo, utilizzano come principale meccanismo di comunicazione il prodotto stesso [32]. Avendo questa particolare caratteristica, il prodotto porta al suo interno una forte matrice legata alla rete degli sviluppatori (struttura organizzativa).

Come abbiamo visto in precedenza, Raymond [81] ha usato la famosa metafora della cattedrale e del bazaar per identificare le due strutture sociali rispettivamente di progetti Closed Source e Open Source; anche se naturalmente ci sono un certo numero di approcci intermedi tra i due paradigmi strutturali.

L'articolo di Raymond si inserisce all'interno di una serie di studi [51, 42, 86] che, negli ultimi anni, ha cercato di delineare l'organizzazione dello sviluppo dei prodotti Open Source. In seguito anche un articolo di Scacchi del 2002 ha proposto un'analisi volta alla comprensione del modo in cui gli sviluppatori nelle comunità coordinassero lo sviluppo del software in contesti diversi, con l'obiettivo di determinare pratiche e contesti organizzativi necessari per il successo dei progetti Open Source [106].

Alcuni studi in letteratura sostengono che aspetti particolari della nuova struttura sociale forniscano vantaggi alla produzione Open Source. Raymond afferma che l'approccio cooperativo allo sviluppo nell'Open Source porta alla realizzazione di un software più sicuro e meno soggetto ai bugs [81].

Un buon modo per comprendere le strutture di tali progetti consiste nel basarsi sulla centralità. In generale, gli individui che hanno più interazioni sono più centrali rispetto a coloro che ne hanno meno. Individui centrali, quindi, sono molto attivi e rappresentano un punto cruciale di collegamento per una rete sociale.

Nel contesto di sviluppo OS, è utile distinguere tra due forme di centralizzazione:

- **Development Centralization**: si riferisce alla produzione del codice;
- **Communication Centralization**: si riferisce alla coordinazione, alla socializzazione e allo scambio di informazioni.

Queste due diverse definizioni di centralità sono state fornite da Crowston e Howison [52] e utilizzate durante un'analisi di 120 progetti Open Source prelevati da *SourceForge.net*. I risultati hanno mostrato che sorprendentemente non ci sono delle particolari dinamiche legate alla centralizzazione comuni a tutti i progetti Open Source.

Tuttavia la struttura sociale dei progetti Open Source mostra in definitiva una struttura a cipolla, in cui ci sono quattro livelli principali, ognuno caratterizzato da particolari proprietà strutturali:

- **Sviluppatori *core***: hanno il grado massimo di centralità nella squadra e generano la maggior parte delle operazioni sul codice di base
- **Co-sviluppatori**: sono quegli sviluppatori che possono inviare *patch*, ma hanno un basso grado di centralità rispetto agli sviluppatori del core
- **Utenti attivi**: non contribuiscono al codice ma effettuano principalmente attività di bug reporting
- **Utenti passivi**: essi non interagiscono affatto con il team di sviluppatori, semplicemente utilizzano il software

Il grado di centralità di ogni persona nel progetto è stato anche analizzato da Slaughter et al. [86] attraverso lo studio dello scambio di e-mail negli archivi dell'Apache Software Foundation Web Server Project.

Questa analisi non solo ha confermato l'esistenza degli strati sopra elencati ma ha anche messo in evidenza dei sottogruppi all'interno di ogni strato: il solo core degli sviluppatori, composto da 22 persone, aveva infatti tre sottogruppi.

Un'altra importante conclusione raggiunta dallo studio [86] mostra come gli sviluppatori dello strato più interno avessero il massimo grado di transazioni con gli altri sviluppatori del team.

2.4 Storia dell'esperimento naturale e il caso Compiere/Adempiere

Un esperimento è un procedimento metodico che ha l'obiettivo di verificare o falsificare una o più ipotesi. Tali ipotesi possono riguardare la natura di un fenomeno studiato. Il fenomeno può essere condizionato dal contesto, ovvero dall'ambiente in cui si svolge l'esperimento, e le caratteristiche che influenzano il fenomeno vengono denominate variabili ambientali.

Lo sperimentatore può provocare variazioni controllate delle variabili ambientali al fine di studiare i diversi comportamenti che il fenomeno mostra come conseguenza delle variazioni. In tal caso di parla di esperimento controllato.

Tuttavia non è sempre possibile eseguire un esperimento controllato su un determinato fenomeno. Ad esempio nel campo dell'epidemiologia non è possibile eseguire esperimenti controllati per ovvi motivi.

Anche nello studio di fenomeni legati alla macroeconomia può risultare difficile agire sulle variabili d'ambiente che determinano lo svolgersi dei fenomeni analizzati. Per questi e per altri campi in cui la sperimentazione controllata non è possibile occorre eseguire esperimenti di tipo diverso.

L'esperimento naturale è uno studio osservazionale in cui lo sperimentatore non influisce sulle variabili d'ambiente [68]. Tuttavia questo non significa che l'esperimento sia privo di sollecitazioni ambientali ma solo che le sollecitazioni non vengono causate da chi sperimenta, bensì dalla natura stessa. Questo significa che chi sperimenta non condiziona le variabili ambientali, si limita, invece, a studiare il comportamento di un determinato fenomeno in situazioni di particolari interesse.

Gli esperimenti naturali sono significativi quando un cambiamento, ben definito e di grande importanza, avviene naturalmente e colpisce una porzione del fenomeno studiato. Questo deve provocare, per quanto concerne il fenomeno studiato, effetti plausibilmente attribuibili al cambiamento delle condizioni [68].

È chiaro quindi che lo studio degli esperimenti naturali deve avvenire su fenomeni colpiti da rilevanti cambiamenti e che presentano caratteristiche in un certo senso eccezionali.

Un esempio di esperimento naturale è quello discusso da Angrist ed Evans [5]. Gli autori hanno effettuato degli studi sull'effetto della grandezza di una famiglia, e quindi del numero dei figli, sulle possibilità di una madre di trovare lavoro. In questo caso il numero dei figli è la variabile d'ambiente e le possibilità di trovare lavoro il fenomeno osservato.

L'analisi di un numero elevato di casi forniva evidenza empirica del fatto che le famiglie con due figli dello stesso sesso erano più inclini a farne un terzo rispetto alle famiglie che avevano due figli di sesso opposto. In questo modo limitandosi ai campioni di famiglie con due figli dello stesso sesso, i due autori avevano maggiori possibilità di osservare un mutamento della variabile ambientale. I due autori, pur non avendo possibilità di modificare le variabili ambientali, ovvero il numero di componenti della famiglia, potevano osservare il mutamento della variabile che avveniva naturalmente e gli effetti sul fenomeno studiato. Si tratta quindi di un esperimento naturale.

Per quanto riguarda lo studio realizzato in questa tesi si vuole analizzare l'impatto di due diversi tipi di gestione del processo di sviluppo sulle

caratteristiche del software sviluppato. Per fare questo è necessario che due differenti comunità di sviluppatori, in cui la gestione del processo di sviluppo sia caratterizzata da un'interazione sociale differente, lavorino su un software funzionalmente equivalenti. Inoltre non è facile individuare gruppi di software con le stesse caratteristiche neppure concentrandosi su una stessa area di utilizzo. È difficile trovare due software che implementino le stesse funzionalità. L'estrema malleabilità del prodotto software permette agli sviluppatori di implementare le stesse funzionalità seguendo *concept* diversi.

Per questo motivo è auspicabile che l'attività di sviluppo da parte delle due differenti comunità, sia esercitata partendo da un software che possieda già delle macro-funzionalità ben definite e che sia per entrambe lo stesso software.

È evidente che realizzare un esperimento di questo tipo in modo controllato è un compito molto complesso. Il caso Compiere/ADempiere fornisce, però, un esperimento naturale utile agli scopi di questo lavoro. Compiere e ADempiere sono due progetti di sviluppo di software per la gestione delle risorse aziendali. La tipologia di questo software è denominata *Enterpise Resource Planning* (ERP).

2.4.1 Enterprise Resource Planning

Un ERP è un software rivolto alle aziende che implementa funzionalità che permettono il controllo dei flussi informativi interni ed esterni all'azienda. Questo avviene per mezzo del controllo delle informazioni provenienti da vari aspetti della gestione aziendale. Un software ERP offre il supporto per la gestione di:

- Finanza e Contabilità
- Gestione delle risorse umane
- Gestione dei rapporti istituzionali
- Produzione

- Vendite
- Marketing

Altre funzionalità più avanzate sono fornite solo da alcuni software ERP, citiamo ad esempio le funzionalità di *Customer Relationship Managment* (CRM), che si occupa di gestire il rapporto con i clienti, e *Supply Chain Managment* (SCM), che si occupa della gestione dell'approvvigionamento.

L'innovazione introdotta, in ambito aziendale, dall'utilizzo di un software ERP è la possibilità di raccogliere le informazioni provenienti dai diversi compartimenti aziendali e gestirle in maniera integrata. La gestione in un unico database di tutte le informazioni e la possibilità di accedere in modo condiviso alle stesse risorse permette una gestione più coerente dei dati ed evita fenomeni come la duplicazione e l'utilizzo di diversi standard all'interno di una stessa azienda.

La gestione di tutti questi aspetti della realtà aziendale avviene attraverso l'implementazione di moduli ben distinti che si occupano della raccolta e dell'elaborazione dei dati provenienti da ogni singolo settore. La modularità del software non deve però essere visibile all'utente che deve poter accedere all'intero inventario delle funzionalità attraverso una gestione unificata.

I generici sistemi software offrono sistemi di supporto per diversi tipi di attività:

- **Attività di tipo istituzionale**: ad esempio il pagamento degli stipendi, la gestione del bilancio e dei rapporti con la legge.
- **Attività di tipo operativo**: ad esempio la gestione degli ordini, delle operazioni e delle movimentazioni di materiale.
- **Attività di tipo direzionale**: come la definizione del posizionamento aziendale, l'analisi degli scenari e altri tipi di pianificazione strategica.

Un software ERP oltre ad integrare in un solo software tutti questi tipi di servizi offre interessanti funzionalità che coinvolgono più di una tipologia

di servizio. La gestione dei dati condivisa permette ad esempio di integrare i servizi istituzionali come la gestione della contabilità con le informazioni derivanti dalle attività svolte nel corso del processo di produzione. In questo modo è possibile allocare in tempo reale i costi legati alle singole attività e mantenere un maggiore controllo sulla spesa aziendale.

Un ERP è tipicamente un software altamente parametrizzabile, capace di adattarsi non solo al settore in cui un'azienda opera ma anche al contesto aziendale stesso. Dall'inizio degli anni '70 ad oggi il fenomeno ERP ha vissuto una costante crescita e l'utilizzo di tali software per la gestione aziendale è diventata fondamentale in più di un settore.

Il più diffuso software ERP in commercio è SAP utilizzato da oltre 176000 aziende. SAP è un software che viene rilasciato unicamente in licenza d'uso. È quindi un Closed Source. Entrambi i software presi in considerazione in questo lavoro sono, viceversa, software ERP sviluppati secondo il paradigma Open Source.

2.4.2 Community Open Source e Commercial Open Source

L'Open Source (OS) è nato come un movimento fortemente ideologico e principalmente supportato da sviluppatori volontari [81]. I progetti OS sono comunemente associati a particolari modelli di Governance che promuovono la trasparenza e la libertà degli sviluppatori [81, 38], in opposizione ai tradizionali modelli di Governance legati al software Closed Source che si concentrano sui bisogni degli utenti, sulla gerarchia aziendale e su una rigida pianificazione.

Raymond [81] paragona il classico processo di sviluppo di software al processo di progettazione e costruzione di una cattedrale mentre equipara le dinamiche che si stabiliscono durante lo sviluppo del software Open Source ad un bazaar che può essere realizzato attraverso diversi programmi e approcci.

Una concezione sbagliata dei progetti OS è che questi siano sostenuti solamente da volontari. Infatti tra i progetti Open Source esistono anche i

Commercial Open Source. Questi progetti sono solitamente portati avanti da una società, non sono tipicamente reperibili attraverso repository on line e vengono rilasciati secondo una licenza gratuita. Similmente a quanto avviene nel caso del Closed Source i processi di sviluppo del prodotto sono ben definiti e gli sviluppatori volontari sono relativamente pochi in quanto il team di progetto è composto quasi esclusivamente da dipendenti della società.

Viceversa, alle spalle degli sviluppatori che lavorano in un progetto Community Open Source non c'è nessuna azienda, nonostante essi possano lavorare per qualsiasi società al di fuori della loro attività nella community. Lo sviluppo sistematico risulta chiaramente più difficile se il gruppo dei volontari è distribuito geograficamente in diverse località, in quanto le classiche metodologie di sviluppo saranno poco adottabili [86, 53].

I Commercial Open Source hanno comunque come scopo principale quello di fornire un ritorno economico ai proprietari, in quanto (come si può chiaramente evincere dal nome) sono come una qualsiasi altra attività commerciale. Questi tipi di software offrono dei vantaggi sia per i clienti interessati ad usare il prodotto senza una licenza a pagamento, sia per quelli che pagheranno.

Gli utenti che sono solo interessati alla licenza gratuita possono effettuare, solitamente, il download online del software. La licenza a pagamento, tendenzialmente utile per le aziende, permette di avere una serie di potenziali vantaggi tra cui: la versione più avanzata del prodotto software, supporto tecnico (attraverso telefono o e-mail) e corsi di formazione per l'utilizzo del software. Inoltre un'azienda potrebbe essere interessata anche a stilare un contratto di assistenza con un'impresa che realizza software Commercial Open Source perché è molto probabile che, avendo quest'ultima un ritorno economico dal prodotto, sia interessata alla manutenzione e allo sviluppo del software.

Per questo tipo di progetti una percentuale di codice che varia tra il 92% e il 100% è sviluppata internamente all'azienda [21], discostandosi quasi totalmente dalle dinamiche di sviluppo del più classico dei modelli Open

Source, ma mantenendone il modello di distribuzione. Secondo gli studi condotti da Capra [20], in quasi il 40% dei progetti Commercial Open Source, il *testing* è pianificato ancor prima che l'implementazione abbia inizio. Inoltre, i piani di rilascio ed lo *scheduling* sono più affidabili, poiché il lavoro è quasi interamente fatto da persone assunte dalla società. Questa strada è stata già seguita in progetti Open Source, come il server HTTP Apache e PostgreSQL. In queste situazioni, chiunque può utilizzare liberamente il progetto Open Source e può acquisire il supporto per il software da una società che è nel *business* del supporto di quel progetto.

Un recente studio [21] ha presentato i risultati di un sondaggio condotto su entrambi i progetti, commerciali e comunitari per indagare sugli stili manageriali e sul ruolo delle imprese all'interno dei progetti.

Nel presente lavoro di tesi sono stati presi in considerazione due software ERP Open Source. Il primo, ADempiere, è un software che risponde al più classico dei paradigmi Open Source, il Community Open Source. Il secondo, Compiere, è invece un prodotto Commercial Open Source.

2.4.3 Open Source ERP

Un gran numero di progetti che riguardano lo sviluppo sviluppo di software Open Source nasce da esigenze particolari di utenti e sviluppatori che non sono soddisfatte dal software disponibile in commercio [108]. Chiaramente ciascun progetto è caratterizzato da un ben preciso livello di complessità che varia considerevolmente in base al campo di applicazione del software. Le persone che avviano progetti di questo tipo devono quindi possedere competenze specifiche che li rendano non solo in grado di affrontare le difficoltà tecniche intrinseche del processo di sviluppo, ma anche le difficoltà gestionali legate alla responsabilità del ruolo e al numero di collaboratori che partecipano al progetto.

Ma questo non è ancora sufficiente. Per avviare un progetto di sviluppo software complesso non basta avere le conoscenza tecniche e l'abilità gestionale, occorre possedere anche esperienza sul campo e conoscenza del domi-

nio applicativo del software che si intende sviluppare. Per i motivi citati, i progetti Open Source che riguardano software caratterizzati da un elevata complessità prevedono barriere all'ingresso molto difficili da superare.

I software ERP, se paragonati ad altre classi di applicativi, sono caratterizzati da una complessità certamente elevata e un grado di specificità non indifferente. Per questo motivo il fenomeno Open Source è stato lento ad abbracciare questa tipologia di software.

Sulla piattaforma *SourceForge.net*, il repository maggiormente utilizzato come supporto per i progetti Open Source, sono elencati 370 progetti di sviluppo di software ERP a fronte di 189.127 progetti totali, il che sottolinea la relativa giovinezza del fenomeno.

Compiere e ADempiere sono due progetti di sviluppo di software ERP che aderiscono al paradigma di Open Source. Per questo ed altri motivi, il caso Compiere/ADempiere assume una certa rilevanza.

2.4.4 Compiere

Compiere è un'applicazione ERP il cui codice sorgente è disponibile sia in licenza GPL (General Public License), sia in licenza CPL (Common Public License).

La licenza GPL è una licenza per il software libero. Con software libero si intende software utilizzabile liberamente, ovvero aperto non solo alla consultazione del suo codice sorgente ed al suo utilizzo, ma anche alla sua modifica. Questo tipo di licenza si contrappone alle licenze per software proprietario che viceversa hanno restrizioni sull'utilizzo, la distribuzione e la modifica.

La licenza CPL, invece, è una licenza per software Open Source che è stata pubblicata da IBM. Questo tipo di licenza ha lo scopo di promuovere la consultazione e lo sviluppo del codice sorgente di un software ma ne mantiene riservati i diritti per lo sfruttamento commerciale.

Lo sviluppo di Compiere inizia da zero nel 1999 ad opera di Jorg Janke, attuale responsabile nonché figura di riferimento per la comunità di svilup-

patori legati al progetto. Ad oggi il progetto viene gestito da una società, la Compiere Inc.

Compiere è un software ERP di notevole successo ed è attualmente considerato come uno dei software di riferimento nel panorama Open Source. Integra al proprio interno anche funzionalità generalmente considerate ERP-Extended come i servizi di *Customer Relationship Managment*(CRM) e di *Point Of Sale* (POS). Il software è rivolto a un utenza internazionale e pertanto è multilingua. Proprio a causa della propria utenza eterogenea, le funzionalità offerte da compiere tengono conto della diversa tassazione e fiscalità presente in ciascun paese. Al proprio interno, Compiere, modella i processi aziendali sotto forma di *workflow* che possono essere estesi e modificati, funzionalità che ne permettono l'integrazione anche con altri sistemi ERP e CRM.

L'architettura che è di tipo Client/Server ne permette quindi l'utilizzo su reti LAN o WAN e sulla rete Internet. Il software è sviluppato con tecnologia J2EE secondo un approccio orientato verso una forte parametrizzazione del funzionamento, questo permette una maggiore adattabilità del software ERP al contesto aziendale.

Uno dei punti di forza di Compiere è proprio il tipo di licenza sotto cui può essere rilasciato. Infatti, l'elevato costo di acquisizione medio di una licenza d'uso di un software ERP proprietario, è sempre stata un problema per le aziende appartenenti alla categoria della piccola-media impresa.

In linea teorica, Compiere può essere adottato gratuitamente da un'azienda. In realtà però la parametrizzazione di un software ERP è un processo complesso che richiede il supporto di sviluppatori ed esperti. Proprio fornendo questo tipo di servizi, Compiere Inc. ha potuto avere successo commerciale nonostante la concessione gratuita della licenza d'uso del proprio software. Grazie a queste caratteristiche, Compiere, è considerato come una valida soluzione ERP per le piccole-medie imprese.

La gestione del processo di sviluppo del software è orientato ad un paradigma considerato Commercial Open Source. Questo importante aspetto,

legato alla gestione del processo di sviluppo, ha avuto nel caso specifico un forte ruolo nella sua storia e, come vedremo, nell'evoluzione della sua comunità di sviluppatori.

Il modello di business del software Compiere prevede la libera distribuzione dell'applicazione che è disponibile sia in forma binaria, che sotto forma di codice sorgente. La distribuzione del software è però vincolata da accordi di natura commerciale. Compiere prevede l'esistenza di tre fasce di partnership (*Authorized Partner*, *Authorized Partner Silver*, *AuthorizedPartner Gold*).

Il livello di partnership è assegnato in base al pagamento di una tariffa di associazione, che va da un minimo di $ 5.000 ad un massimo di $ 10.000, e all'ottenimento di una certificazione della capacità professionale dell'aspirante partner, la cui erogazione è disposta dalla stessa Compiere Inc.

Contrariamente ad altri progetti Open Source, Compiere non concede libero accesso alla base di codice sorgente. Gli utenti comuni hanno la possibilità di accedere a release periodiche, gli *Authorized Partner* possono accedere a rilasci definiti(*milestone*, che hanno una frequenza maggiore), mentre gli *Authorized Partner Silver* e *Gold* possono avere accesso ai *daily snapshots* (frequenza giornaliera).

2.4.5 La nascita di ADempiere

Un processo sviluppo gestito in modalità bazaar [81] si basa sul lavoro di una comunità di utenti/sviluppatori che si crea, spesso autonomamente, attorno allo stesso. Le caratteristiche dei singoli e la qualità dell'interazione tra di essi determinano anche la crescita e il successo del progetto.

Sebbene Compiere sia un software Open Source, la modalità di gestione di tipo Commercial ha determinato nel tempo un malcontento nella comunità, che è sfociato in un fork. Un fork è una scissione, che prende l'avvio da una precisa release del programma originale. Il risultato è che il progetto si ramifica in due diverse direzioni.

Le caratteristiche di un fork su un progetto dell'importanza di Compiere possono essere diverse e possono portare a differenti scenari. Il nuovo progetto può ad esempio essere portato avanti da una nuova comunità di sviluppatori differente da quella del progetto originale. Oppure alcuni degli sviluppatori che appartengono alla comunità originale, per questioni di insoddisfazione, possono lasciare la prima comunità e formarne una seconda con nuovi elementi. Questa seconda comunità può avere caratteristiche differenti, nelle modalità di gestione del processo di sviluppo.

Le caratteristiche di questo secondo caso sono quelle che portano alla nascita di ADempiere, che prende avvio dalla versione *253b* di Compiere.

In data 1 Settembre 2006, un ristretto gruppo di membri della comunità di Compiere ha preso la decisione di aprire un forum [1]. All'interno di tale forum utenti e sviluppatori di Compiere hanno dato origine ad una discussione a proposito del futuro della comunità. Questa discussione era mirata a sondare il terreno per una scissione dal progetto originale. Il tutto si risolve nella decisione di effettuare un fork. Tra le motivazioni si può citare un malcontento diffuso a proposito della contribuzione esterna.

Secondo alcune lamentele infatti, Compiere Inc., non forniva chiare delucidazioni sull'effettivo stato della contribuzione interna con il rischio di una duplicazione del lavoro da parte dei contribuenti esterni. In aggiunta la poca chiarezza sui piani di lavoro impediva la possibilità di implementare nuove funzionalità.

Nella discussione viene evidenziata la necessità di disporre di un software ERP Open Source sviluppato attraverso politiche di gestione differenti. Pur mantenendo un rapporto amichevole con Compiere Inc., la nuova comunità decide di adottare forme di coordinamento e gestione differenti, omologabili al paradigma di Community Open Source.

Il preciso scopo di questo cambiamento era rendere più facilmente fruibile le informazioni che riguardavano lo stato di sviluppo del software e permettere una maggiore dinamicità e libertà nella scelta delle funzionalità

[1] http://red1.org/forum/viewtopic.php?t=931

da implementare. Diminuire la duplicazione del lavoro, rendere chiaro l'avanzamento e incentivare l'iniziativa erano i tratti fondamentali del nuovo tipo di gestione.

Sulla base dell'orientamento alla comunità da parte del gruppo, si è scelto di rilasciare il software su licenza GPL versione 3 della *Free Software Foundation*.

Il livello di partecipazione alla discussione tenuta all'interno del forum raggiunge il suo picco nel momento in cui viene deciso il fork. Questo ha l'effetto di attirare l'attenzione di altri sviluppatori che si associano al progetto.

Anche grazie a questo rinnovato entusiasmo, il modello di gestione di ADempiere si rivela fortunato. Attualmente ADempiere figura tra i progetti più attivi all'interno della piattaforma *SourceForge.net*.

2.4.6 Compiere e ADempiere dal fork in poi

I due software ERP su cui si concentra questo lavoro hanno storie differenti e, pur tenendo conto del fatto che il progetto di ADempiere prende il via da un fork di Compiere, possono aver compiuto scelte diverse per quanto concerne le funzionalità da implementare. Tali scelte possono influire sul numero e sul tipo di servizi offerti dal software sviluppato.

Ai fini di questo lavoro sarebbe utile poter studiare gli effetti del cambiamento nell'organizzazione degli sviluppatori sulle qualità strutturali del software sviluppato. Le scelte in tema di funzionalità da implementare e servizi da offrire all'utente possono anch'esse influenzare la struttura del prodotto software.

Per questo motivo è necessario che i due software, in seguito al fork, non abbiano seguito strade completamente diverse implementando tipi di servizi differenti.

Con lo scopo di effettuare un confronto di tipo funzionale tra i due diversi software, sono state redatte le liste dei servizi offerti all'utente da parte di

ciascun software[2].

Tabella 2.2: Lista delle funzionalità implementate dai software ERP Compiere e ADempiere

Funzionalità	**Compiere**	**ADempiere**
Gestione Contabilità	X	X
Gestione Ordini	X	X
Gestione Appalti	X	X
Gestione Inventario	X	X
Gestione della produzione	X	X
Gestione dei Partenr Finanziari	X	X
CRM	X	X
Gestione Vendite e Marketing	X	X
Gestione dei progetti	X	X
Gestione dei servizi	X	X
Workflow Engine	X	X
Supporto Database: Oracle XE/10g/11g	X	X
Supporto Database: PostgreSQL		X
Hot Deployment	X	X
Java Hot-swap Debug	X	X
Lato Server	X	X
Java Client	X	X
Interfaccia Web		X
Supporto Scripting		X

In Tabella 2.2 viene riportata una lista di funzionalità e viene segnalata l'effettiva implementazione da parte di ciascun progetto.

È possibile notare come la maggior parte delle funzionalità fondamentali per un ERP sia offerta da entrambi i software. ADempiere implementa un numero di funzionalità maggiori di quelle di Compiere. Tuttavia le differenze sono minime e riguardano semplicemente i tipi di Database supportati e alcune funzionalità WEB.

[2]http://www.humanflash.com/support/evaluation_guide.html

Per quanto riguarda le funzionalità tipiche di un ERP si può quindi affermare che i due software non hanno preso strade diverse e non ci sono differenze significative nel numero e nei tipi di macro-funzionalità offerte.

2.4.7 Particolarità del caso in esame

Diversi studi si sono occupati di analizzare le reciproche influenze tra struttura sociale che rappresenta i rapporti di collaborazione tra sviluppatori e struttura del software che essi producono.

Alcuni di questi lavori hanno evidenziato la capacità del modello di Governance utilizzato nella la gestione del processo di sviluppo di influenzare questo tipo di dinamiche.

Nell'attività di sviluppo software, soprattutto in ambito Open Source, il prodotto ha un ruolo fondamentale nel coordinamento di coloro che collaborano per realizzarlo. Molti dei più importanti software del panorama Open Source, sono sviluppati da individui che vivono in diverse parti del mondo.

Essendo il codice sorgente liberamente consultabile e modificabile, si è originato il fenomeno di intere comunità, composte da individui che non si conoscono, che collaborano alla realizzazione di un obiettivo comune, utilizzando il prodotto stesso come strumento di coordinamento.

In riferimento a questa importante caratteristica, lo studio della capacità dei diversi tipi di Governance di caratterizzare le influenze tra struttura sociale degli sviluppatori e struttura software, non può sottovalutare il ruolo della natura differente dei prodotti software considerati.

Studiando l'effetto di due differenti tipi di Governance sulle caratteristiche di due diversi software, in ambito Open Source, si trascura quindi il ruolo delle differenze strutturali tra i due software stessi.

Non è stato ancora effettuato, in letteratura, uno studio che mostri queste dinamiche su un caso particolare come quello di Compiere e ADempiere.

L'esperimento naturale osservato in questo lavoro mostra gli effetti di due modalità di Governance, profondamente differenti, all'interno di progetti che, ad un certo punto della loro storia condividono il medesimo codice

sorgente. Nel momento in cui viene deciso di effettuare il fork, il codice sorgente del progetto originale viene replicato. In quel preciso momento due diverse comunità di sviluppatori, gestite in maniera molto diversa, sono all'opera su un software identico.

Si può quindi, nell'osservare i due progetti, considerare trascurabile l'incertezza legata alla diversità dei software sulle dinamiche di mutua influenza oggetto dello studio. In questo modo è possibile osservare l'impatto di un tipo di Governance sul prodotto, raffrontandolo con quello di un altro tipo di Governance su un prodotto funzionalmente equivalente.

Capitolo 3

Metodologia di ricerca e ipotesi

"Un modello deve essere in parte sbagliato, altrimenti sarebbe la cosa stessa. Il trucco sta nel guardare laddove il modello è corretto."

Henry Bent

Questo capitolo ha lo scopo di introdurre le osservazioni che hanno portato alla formulazione delle ipotesi di ricerca e descrivere la metodologia che si intende utilizzare per realizzare le analisi atte a verificarle.

Il capitolo è strutturato come segue: nella Sezione 3.1 verranno presentate e discusse le ipotesi di ricerca. La Sezione 3.2 è divisa in due parti. Nella prima verranno illustrate le metodologie di raccolta dei campioni di software analizzati e le scelte effettuate nel calcolo delle metriche che descrivono la qualità strutturale interna del software. La seconda parte mostra i metodi con cui sono stati ricavati i modelli di interazione sociale relativi a ciascuna comunità di sviluppatori e illustra le tecniche di calcolo delle metriche che esprimono le caratteristiche strutturali di tali modelli di interazione.

3.1 Ipotesi di ricerca

Il presente lavoro si colloca nel contesto dello studio dei mutui rapporti tra caratteristiche dei processi di produzione e caratteristiche del prodotto.

Si è visto come il modo in cui le gestione del processo di sviluppo influenza la qualità del design del software sviluppato sia un tema di fondamentale importanza all'interno dell'ingegneria del software.

La teoria dell'organizzazione classica è in grado di fornire eccellenti metodi di coordinamento per ogni tipo di processo produttivo ma non è in grado di tenere in conto delle variabili legate all'incertezza connessa alla natura del compito. La notevole complessità del prodotto software implica l'esigenza di conoscenze tecniche legate alla natura stessa delle funzionalità che tale software deve implementare.

Con l'avvento del paradigma Open Source sono nati nuovi modelli di coordinamento che prevedono diverse dinamiche di adattamento del processo di sviluppo alle caratteristiche del design del software da sviluppare.

Lo straordinario impatto del sistema di sviluppo Open Source ha caratterizzato la nascita di differenti modalità di gestione del processo di sviluppo anche all'interno del paradigma stesso. La distinzione tra Community Open Source e Commercial Open Source rappresenta un'ulteriore divisione all'interno del panorama.

Il caso Compiere/ADempiere rappresenta, da questo punto di vista, un'occasione per lo studio di due differenti tipi di Governance. Sebbene entrambi i progetti considerati abbiano aderito al paradigma Open Source, vi sono delle significative differenze nella modalità di Governance adottate per gestire il team di sviluppo. Differenze evidenziate dall'esigenza di una parte della comunità originale di effettuare un fork e procedere con lo sviluppo di un progetto parallelo, gestito in modalità differente.

Compiere è un progetto che ha scopi commerciali e, sebbene il codice sorgente del software sia disponibile per chiunque, le modalità di partecipazione allo sviluppo sono fortemente controllate. Il processo di sviluppo ruota attorno ad una figura molto forte, Jorg Janke, che è fondatore e responsabile

del progetto nonché uno dei maggiori contributori per quanto riguarda l'implementazione del codice. Attorno a tale figura ruotano un ristretto numero di project manager, formalmente associati a Compiere Inc., che coordinano vari aspetti del processo di sviluppo.

Il suo fork, ADempiere, nasce dall'esigenza di una parte della comunità di stravolgere le modalità di contribuzione e i metodi di condivisione delle informazioni sullo stato del progetto. Inoltre viene considerato importante aumentare la possibilità da parte degli sviluppatori di prendere iniziativa. Il nuovo gruppo di sviluppatori che si avvicina ad ADempiere adotta metodi di coordinamento diversi, molto più orientati alla comunità.

In riferimento a questo tipo di considerazioni si può affermare che i due progetti sono gestiti secondo differenti modelli di Governance. Tali differenze nelle modalità di gestione del processo di sviluppo hanno un peso sulle caratteristiche dell'interazione tra i membri delle comunità di sviluppatori. Nel presente lavoro si analizzano, quindi, gli effetti di diverse gestioni del processo di sviluppo su prodotti software funzionalmente equivalenti.

Dato il ruolo fondamentale del prodotto software come mezzo di condivisione delle informazioni in ambito Open Source, non si può non considerare l'impatto che avrebbe la diversità del prodotto software sulle caratteristiche dell'interazione sociale interna alle due comunità.

Considerando che al momento del fork i due progetti disponevano di un software identico si può affermare, quindi, che l'influenza delle caratteristiche del prodotto, almeno inizialmente, è la stessa su entrambe le comunità. In questo modo è possibile escludere una variabile di incertezza dal modello. Con preciso riferimento a questa particolarità del caso in esame, si è considerato trascurabile l'effetto delle differenze funzionali dei software analizzati.

Parlando di influenze tra struttura del software e caratteristiche dell'interazione tra i membri di una comunità di sviluppatori si fa riferimento al concetto di Social Network. Diversi studi hanno contribuito alla formulazione di modelli che esprimono le mutue influenze tra caratteristiche della Social

Network, espresse da metriche di centralità, e caratteristiche del software, descritte da metriche di qualità della struttura interna [64, 47, 3, 8, 94].

In particolare alcuni lavori hanno identificato modalità di interazione peculiari a seconda del paradigma utilizzato per la gestione del processo di sviluppo. Studi empirici hanno dimostrato come le dinamiche di interazione siano diverse nel caso in cui si parli di software Open Source o di software Closed Source [37].

Nel caso dell'Open Source è stato osservato un allineamento tra struttura delle rete sociale, che rappresenta l'interazione tra coloro che producono il software, e struttura del prodotto finale. In particolare si è notato come le caratteristiche della Social Network siano in grado di lasciare un'impronta sulla struttura del software.

I metodi di coordinamento prevalenti in ambito Open Source si distaccano nettamente da altri tipi di gestione del processo di sviluppo, maggiormente utilizzati nel contesto aziendale, all'interno di cui si osserva più spesso un coordinamento di tipo gerarchico.

I metodi di coordinamento utilizzati in alcuni progetti Commercial Open Source, però, presentano delle analogie con tale organizzazione gerarchica. Per questo motivo si è portati a credere che i metodi di Governance utilizzati all'interno di questo paradigma si discostino da quelli osservati nei classici progetti Open Source, quelli di tipo Community.

In particolare la capacità delle caratteristiche di una struttura gerarchica di lasciare un'impronta sulla struttura del software è necessariamente diversa da quella di altri tipi di struttura sociale.

Alla luce di queste osservazioni si può formulare la prima ipotesi di ricerca:

> **Ipotesi H1**: Differenti modelli di Governance (Commercial Open Source e Community Open Source) implicano differenti relazioni tra la struttura interna del software e la struttura della rete di sviluppatori.

Esaminando il caso Compiere/ADempiere si può affermare che il fork ADempiere, nato dall'esigenza di un parte della comunità di una migliore condivisione delle informazioni, sia quello più aderente al paradigma Open Source. Per questo motivo è lecito aspettarsi di osservare dinamiche in cui la struttura del software è influenzata dalla struttura della rete sociale degli sviluppatori, come spesso osservato nei progetti aderenti a questo paradigma.

Le caratteristiche di un flusso di informazioni più controllato, un minore incentivo all'iniziativa ed in generale una gestione del processo di sviluppo più centralizzata, osservate nel progetto Compiere, sono invece più vicine al concetto di sviluppo software all'interno di un contesto aziendale.

Una maggiore disparità nei ruoli, all'interno di una comunità, può essere vista come un tipo di organizzazione più orientata alla gerarchia. Il fatto che la contribuzione sia regolata da contratti e autorizzazioni fornite da Compiere Inc. accentua questo tipo di caratteristica.

Avvicinandosi ad un coordinamento di tipo gerarchico e allontanandosi da quella forma di gestione che potrebbe essere considerata come *Network Governance*, il peso delle decisioni, e quindi delle scelte in fase di progetto, non è più sulle spalle dell'intera comunità ma è fortemente condizionata da una struttura gerarchica.

Se nel paradigma Community Open Source, come si è detto, le caratteristiche delle rete sociale degli sviluppatori lasciano un'evidente impronta sulla struttura del software che essi producono, in ambito Commercial Open Source la situazione potrebbe non essere la stessa. In questo caso si ipotizza, infatti, che la struttura sociale interna alla comunità si rifletta sulla struttura del software con minore intensità, in quanto la struttura del software è anche fortemente condizionata dalle scelte effettuate in fase di progettazione da coloro che gerarchicamente sono preposti a questo compito.

Viceversa, una volta progettato il software, lo sviluppo dovrà modellarsi attorno ad esso. Una volta effettuate le scelte in termini di struttura del software, un gruppo di sviluppatori dovrà essere adattato a tale struttura,

ovvero i compiti implementativi saranno suddivisi sulla base delle caratteristiche del software. È evidente che un gruppo di sviluppatori che lavora ad un sottoinsieme funzionale del software, ad esempio un modulo, sarà rappresentato da un gruppo di nodi maggiormente connessi tra loro all'interno di una Social Network.

Pertanto la rete sociale si modellerà attorno alle caratteristiche strutturali del software. Questo tipo di osservazioni ha contribuito alla formulazione di due ulteriori ipotesi di ricerca:

> **Ipotesi H2A**: In un progetto di tipo Community Open Source, la struttura della rete di sviluppatori influenza la struttura interna del software, ma non viceversa.

> **Ipotesi H2B**: In un progetto di tipo Commercial Open Source, la struttura della rete di sviluppatori influenza la struttura interna del software, e viceversa.

Concentrando l'attenzione sul modo in cui le caratteristiche della rete sociale degli sviluppatori influiscono sulla qualità del software design ci si aspetta di osservare che talune caratteristiche influenzino positivamente la qualità mentre altre influiscano negativamente.

Le caratteristiche della rete sociale, che in questo lavoro sono descritte attraverso le metriche di centralità *closeness* e *betweennes*, possono avere un ruolo positivo o negativo a seconda della loro capacità di avere impatto sulla qualità del software. Ci si aspetta che le reti sociali interne alle due comunità, nei due diversi paradigmi di Open Source rappresentati da Compiere e ADempiere, mostrino sostanziali differenze in termine di metriche di centralità.

È interessante a questo punto studiare quali caratteristiche di interazione sociale hanno impatto positivo sulla qualità, quali hanno un impatto negativo e quali fondamentalmente non esercitano nessuna influenza. Per delineare tale scenario si possono formulare alcune ipotesi.

Closeness e *betweennes* sono caratteristiche della rete sociale che influiscono sulla capacità dei singoli nodi di comunicare con il resto della rete. Ci si aspetterà, quindi, che un elevato valore medio di queste metriche di centralità indichi una capacità più diffusa, tra i vari nodi di una rete, di comunicare con il resto della rete stessa.

Una maggiore capacità di comunicare può essere causa di una migliore condivisione delle informazioni all'interno della rete stessa.

Se il modello adottato dalla comunità di ADempiere per favorire una maggior condivisione delle informazioni e premiare l'iniziativa ha avuto successo, ci si aspetta che le metriche di centralità delle reti sociali influiscano positivamente sulle caratteristiche di qualità del software design.

Questo da origine ad altre due ipotesi di ricerca:

> **Ipotesi H3A**: In un progetto Community Open Source, maggiore è il valore delle metriche di centralità degli sviluppatori, migliore è la qualità della struttura interna del software.

D'altra parte una minore condivisione delle informazioni e una maggiore centralizzazione delle decisioni possono avere un impatto sulla struttura del software non facile da prevedere. Infatti, se il numero di decisori all'interno della rete rimane relativamente basso, l'iniziativa e il controllo rimane prerogativa di un limitato numero di sviluppatori all'interno della rete. Questo significa che le decisioni e le attività di controllo, facendo riferimento alla Social Network, sono irradiate da quei nodi che hanno un alto valore di centralità verso il resto della rete.

Come risultato, il coordinamento del processo di sviluppo non è più opera della comunità intera ma di pochi decisori all'interno di essa. Ci sia aspetta quindi che non siano più le caratteristiche della rete sociale ad avere impatto sulla qualità del software design, ma altri tipi di variabile.

> **Ipotesi H3B**: In un progetto Commercial Open Source, le metriche di centralità degli sviluppatori non hanno un impatto rilevante sulla qualità della struttura interna del software.

In riferimento al modo in cui le metriche che descrivono la struttura sociale di un team di sviluppo influenzano quelle che riguardano la qualità strutturale del software si possono fare osservazioni di tipo comparativo.

I due progetti si differenziano per il tipo di Governance adottata per la gestione del processo di sviluppo. Ciascun tipo di Governance ha un peso nella capacità della rete sociale di influenzare la qualità strutturale del prodotto.

Paragonando le due applicazioni analizzate con l'ausilio di metriche legate alla struttura del software è quindi possibile valutare quale dei due tipi di gestione ha esercitato un maggior controllo sulla qualità del proprio software.

Se in un progetto Community Open Source la struttura della rete sociale degli sviluppatori si riflette in quella del software sviluppato, si può ipotizzare che sia più facile per una comunità di questo tipo imprimere una buona struttura sul proprio prodotto.

Diversamente, se in un progetto Commercial Open Source è molto più forte la caratteristica della struttura sociale della comunità di adattarsi a quella del software, sarà il prodotto ad avere il ruolo principale nelle dinamiche. Per questo motivo si può ipotizzare che in tali progetti ci sia una leva di controllo minore, da parte del team di sviluppo, sulla qualità della struttura interna del software prodotto.

Tenuto conto delle precedenti considerazioni è possibile formulare un'ulteriore ipotesi di ricerca:

> **Ipotesi H4**: La Governance di tipo Community Open Source permette lo sviluppo di software con qualità strutturale maggiore rispetto a quella ottenibile con la Governance di tipo Commercial Open Source.

3.2 Metodologia di ricerca

In questa sezione verrà presentata la metodologia utilizzata per l'acquisizione e l'analisi del campione di dati. Nella prima parte verranno descritti gli aspetti metodologici riguardanti le analisi della struttura software; nella seconda parte verranno descritti invece gli aspetti riguardanti l'analisi dei modelli di interazione sociale.

3.2.1 Struttura del software

In questa sezione sezione verrà descritto il procedimento tramite cui è stato ricavato il campione di dati utilizzato successivamente per le analisi. Andremo inoltre ad esaminare i vari metodi di calcolo adoperati per ricavare le metriche che esprimono la qualità della struttura interna del software, anch'esse descritte nel capitolo precedente.

Campione analizzato

Il campione analizzato è stato ricavato prelevando le release dei software Compiere e ADempiere dalla piattaforma *SourceForge.net*.

Per quanto riguarda il software Compiere sono state prelevate 30 versioni, rilasciate nel corso dei circa nove anni compresi tra la fine del 2001 e la prima metà del 2010. Come avviene per la maggioranza dei software, le versioni non sono state rilasciate con cadenza regolare e si concentrano soprattutto tra il 2001 e l'inizio del 2006, quando avviene il fork da cui nasce il progetto ADempiere. Va registrata una minore attività negli anni seguenti, che porta al rilascio di solo 7 versioni in 4 anni.

Per quanto riguarda ADempiere la prima versione è stata rilasciata appunto nel mese di Ottobre del 2006. Il progetto è molto attivo fino alla fine del 2008, periodo in cui vengono rilasciate 14 delle 16 versioni analizzate. Le ultime due versioni vengono rilasciate con cadenza pressoché annuale. Sono quindi state disponibili per le analisi 46 differenti versioni di software.

Operazionalizzazione delle variabili

Una volta disponibili le 30 release di Compiere e le 16 di ADempiere si è potuto procedere all'analisi del codice. Sono state ricavate quindi alcune metriche utilizzate, poi, per descrivere le caratteristiche strutturali del software in termini di qualità del design.

Le metriche che esprimono la qualità della struttura interna del software sono state ottenute in due fasi:

- sono state ricavate le metriche della Suite CK riferite ai singoli oggetti.
- sono state utilizzate le metriche della Suite CK per il calcolo di metriche in grado di esprimere la qualità degli interi sistemi, cioè le varie versioni di software analizzate, viste come insiemi di oggetti.

La misurazione delle metriche è stata effettuata sui due differenti software, Compiere e ADempiere, considerando ognuna delle versioni prelevate da *SourceForge.net* come un sistema a sé stante.

Tale misurazione è stata eseguita tramite un tool per l'analisi statica, sviluppato appositamente, che utilizza le librerie di *Spoon*, un compilatore aperto ed estensibile che si appoggia al compilatore del linguaggio Java (Javac).

Spoon si basa sulla costruzione di un meta-modello del codice compilato che viene reso accessibile sia in lettura che in scrittura agli utenti. Tale meta-modello può essere utilizzato per analisi statiche e permette quindi di analizzare il codice senza doverlo eseguire.

Il tool, dopo aver analizzato il meta-modello generato da ciascun software analizzato, fornisce il valore delle metriche che saranno presentate in seguito e riferite ai singoli oggetti, ovvero alle singole classi che compongono ciascuna versione. Dopo l'applicazione del tool si ottengono tanti insiemi di metriche quante sono le versioni dei software analizzati.

Le principali metriche calcolate sono quelle descritte da Chidamber e Kemerer [23] note come Suite CK. La Suite CK è un insieme di metriche adatte

a descrivere sistemi inerenti ad un contesto ad oggetti. Di essa fanno parte sei metriche: *Weighted Methods per Class* (WMC), *Depth of Inheritance Tree (DIT)*, *Number Of Children* (NOC), *Coupling Between Objects* (CBO), *Response For a Class* (RFC), *Lack of Cohesion Of Methods* (LCOM).

Di seguito vengono fornite, per quanto riguarda questo lavoro, descrizioni e metodi di calcolo di ciascuna di queste sei metriche:

Weighted Methods per Class: Questo tipo di metrica misura la complessità di un oggetto, come ad esempio una classe, basandosi sul conteggio dei metodi di quest'ultima. È possibile proporre una metrica WMC che si basi anche esclusivamente sul numero di metodi implementati all'interno di una classe ma, in questa tesi, è stato considerato un altro tipo di calcolo. In particolare ciascun metodo viene prima pesato, assegnando la *Ciclomatic Complexity* come peso del singolo metodo.

Dato un Oggetto O e detta $\boldsymbol{M}$ la lista dei suoi metodi, la WMC è calcolata come segue:

$$WMC(O) = \sum_{M_i \in \boldsymbol{M}} g(M_i), \tag{3.1}$$

dove $g(\bullet)$ è la funzione che restituisce il peso di un metodo. È chiaro che un oggetto che presenta una WMC alta è un metodo molto complesso e tendenzialmente più specifico. Il livello di riusabilità dell'oggetto può quindi non essere molto elevato.

Depth of Inheritance Tree: Questa metrica esprime il grado di profondità nell'albero dell'ereditarietà, ovvero il numero di antenati di un oggetto. Nei linguaggi a eredità multipla, invece, il calcolo di questa metrica è più complesso dato che non esiste un unico cammino tra un oggetto padre ed uno dei suoi eredi. In tal caso occorre trovare il percorso più lungo.

Questa metrica esprime quanto complesse siano le caratteristiche che un oggetto eredita dai suoi antenati. Un oggetto con un numero elevato di antenati, oltre ad ereditare molte caratteristiche, è soggetto più facil-

mente a cambiamenti nel corso dell'evoluzione del software dato che i suoi cambiamenti sono strettamente correlati a quelli dei suoi antenati.

Number Of Children: Concettualmente simile a DIT ma misura il numero dei figli di un oggetto e quindi il numero di oggetti che erediteranno i suoi metodi.

Un valore elevato di NOC implica che l'oggetto sia molto influente. Questa caratteristica può essere positiva, poiché un oggetto con molti eredi è tipicamente molto utilizzato, ma possono sorgere dei problemi durante le fasi di test a causa della forte influenza su un gran numero di oggetti.

Coupling Between Objects: Questa metrica misura il grado di dipendenza di un oggetto dagli altri oggetti appartenenti allo stesso progetto.

Dato un oggetto O si definisce accoppiato ad un generico altro oggetto O_j, se tra i due oggetti esiste una qualsiasi azione compiuta da un metodo di O che fa riferimento a un metodo o a un attributo dell'oggetto O_j.

In pratica il valore di CBO misura il numero di oggetti accoppiati nel sistema in esame. Considerando quindi un sistema S composto da una lista di oggetti $\partial = O_1, O_2, ..., O_n$, il valore di CBO del un generico oggetto O_i viene calcolato in questo modo:

$$CBO(O_i) = \sum_{O_j \in \partial} connection(O_i, O_j) \tag{3.2}$$

Dove la funzione $connection(A, B)$ assume valore 1 se l'oggetto A invoca un metodo o utilizza un attributo dell'oggetto B, e valore 0 altrimenti.

Un alto valore di CBO, tra gli oggetti che appartengono ad un sistema software, è controproducente. Innanzitutto è indice di scarsa modularità nel design del software: un oggetto con un altro valore di *coupling* è difficilmente riutilizzabile in altri contesti. Inoltre una modifica effettuata su un oggetto con *coupling* elevato ha impatto su un gran numero di altri oggetti rendendo in questo modo complicato effettuare le operazioni di test sull'oggetto considerato.

Sahraoui et al. [44] affermano che un alto valore di CBO è altamente indesiderabile, e altri studi dimostrano che un valore basso di CBO è desiderabile anche per questioni di *information hiding* [75].

Response For a Class: Per RFC si intende la cardinalità dell'insieme dei metodi che possono essere eseguiti in seguito all'utilizzo di un determinato oggetto. Tale insieme, dato l'oggetto O, è definito come l'unione tra l'insieme dei suoi metodi e l'insieme dei metodi del generico oggetto O_i tale per cui esiste una relazione del tipo $M_O \rightarrow M_{O_r} \rightarrow ... \rightarrow M_{O_i}$ in cui M_{O_i} è un metodo dell'oggetto O_i e la freccia indica una relazione di invocazione tra metodi.

La metrica viene ottenuta percorrendo l'albero delle chiamate ed aggiungendo, di volta in volta, all'insieme tutti i metodi degli oggetti invocati fino al raggiungimento di metodi senza invocazioni o di un punto fisso (ovvero nel momento in cui aggiungendo all'insieme tutti i metodi di un oggetto invocato, l'insieme non subisce variazioni).

Questo tipo di metrica esprime il potenziale accoppiamento tra un oggetto considerato e altri oggetti appartenenti allo stesso sistema. Mentre metriche come il CBO si riferiscono a relazioni effettivamente rilevate, RFC tiene conto anche di relazioni potenziali che possono nascere nel corso dell'evoluzione del software. Un oggetto con un basso valore di CBO ma con un alto valore di RFC è comunque potenzialmente molto complesso e su di esso testing e debugging possono risultare più complicati.

Lack of Cohesion Of Methods: La metrica LCOM indica la mancanza coesione interna di un oggetto. Nel caso in cui oggetto abbia coesione alta i suoi metodi eseguiranno concettualmente la stessa funzione.

Per ottenere una migliore coesione, un oggetto poco coeso, dovrebbe essere scomposto in oggetti più semplici, le cui funzionalità implementate abbiano un maggior grado di somiglianza.

L'assunzione che viene fatta per il calcolo di questa metrica è che due metodi, implementati nello stesso oggetto, sono tanto più simili quanto simile è l'insieme di attributi su cui operano.

Avere molti metodi che operano sullo stesso insieme di variabili aumenta la coesione di un oggetto. Avere metodi che fanno uso di tanti attributi differente fa, viceversa, diminuire il valore di coesione.

Questa metrica è storicamente la più controversa della Suite CK e diverse pubblicazioni nel corso degli anni hanno espresso critiche [43, 50, 49, 66, 98], in alcuni casi proponendo delle modifiche [7, 50, 62]. In questo lavoro viene utilizzata la ridefinizione di LCOM proposta da Chidamber e Kemerer [23].

Facendo riferimento alla definizione di Chidamber e Kemerer, dato un oggetto O, LCOM viene calcolata in questo modo:

$$LCOM(O) =\mid P \mid - \mid Q \mid \tag{3.3}$$

Dove P è l'insieme delle coppie di metodi appartenenti all'oggetto O, tali per cui considerando gli insiemi di attributi su cui operano, non esista alcun attributo che appartenga ad entrambi gli insiemi. L'insieme Q contiene invece le coppie di metodi entrambi operanti su almeno uno stesso attributo. La differenza delle cardinalità di questi due insiemi indica appunto la mancanza di coesione, ovvero il numero di coppie di metodi poco coesi in eccesso al numero dei metodi coesi.

Per chiarire questo concetto è possibile utilizzare un esempio: considerando dei metodi simbolici A, B e C, e due attributi simbolici x, y, immaginiamo di trovarci nella seguente situazione: *A(x)*, *B(x,y)*, *C(y)* in cui *A(x)* è da leggere come il metodo A opera sull'attributo x. In questo caso ipotetico il calcolo di LCOM è fatto in questo modo:

$$LCOM(O) = \left|\left\{\{A, C\}\right\}\right| - \left|\left\{\{A, B\}, \{B, C\}\right\}\right| = 1 - 2 = -1 \tag{3.4}$$

Per quanto concerne questo tipo di metrica, è preferibile avere una coesione più alta poiché tale caratteristica agevola l'*information hiding*. Un basso livello di coesione è sintomo di un design inadeguato e di maggiore complessità dell'oggetto.

Calcolo delle metriche normalizzate per i Function Points

Per lo scopo che questo lavoro si propone di realizzare, è necessario possedere informazioni sull'intero sistema e non solamente sulle parti che lo compongono. Questo significa calcolare il valore di metriche che esprimano le proprietà di un'intera versione dei software analizzati e non quelle delle singole classi.

Per ottenere metriche significative a livello di sistema non è sufficiente, quindi, effettuare operazioni statistiche, come ad esempio il calcolo della media o della varianza, sull'insieme di oggetti che compongono un sistema, ma occorre avere idea del peso specifico all'interno del sistema stesso di ciascun oggetto analizzato.

Le metriche della *Suite CK* sono riferite all'oggetto. I sistemi software analizzati sono composti da un insieme di oggetti, di classi, le cui metriche possono essere misurate come spiegato nelle sezioni precedenti. Tuttavia ciascuna classe analizzata possiede caratteristiche proprie sia in termini di dimensioni, ovvero di righe di codice che la compongono, sia a livello di funzionalità implementate.

Ciascun valore di metrica misurato va quindi riferito alle caratteristiche della classe a cui viene attribuito. Una classe di grandi dimensioni o particolarmente importante dal punto di vista delle funzionalità implementate, possiede un peso specifico più elevato all'interno del sistema.

Nel presente lavoro sono stati stimati i Function Points implementati da ciascuna classe. Utilizzando i Function Points è possibile avere un'idea della complessità funzionale dell'oggetto analizzato e quindi del suo peso specifico all'interno di un'applicazione che implementa un certo numero di funzionalità.

Molti studi hanno dimostrato empiricamente che il rapporto tra righe di codice e Function Points implementati varia in base al linguaggio di programmazione. Esistono diversi criteri di conversione del valore LOC (numero di righe di codice) in FP per ciascun linguaggio di programmazione. Tali criteri sono in grado di fornire fattori di conversione adatti a stimare il numero di FP relativi ad un singolo modulo software. I software Compiere e ADempiere sono stati implementati sfruttando il linguaggio J2EE (Java 2 Enterprise Edition).

In questo lavoro ci si è basati sulle tabelle di conversione fornite da IFPUG (International Function Point Users Group) che presentano i fattori di conversione $LOC \rightarrow FP$. Il fattore indicato per la conversione $LOC \rightarrow FP$ misurato empiricamente su un numero molto elevato di applicazioni implementate in linguaggio J2EE è 57. Occorre chiarire che tale fattore di conversione rappresenta una scelta adottata in maniera convenzionale.

Il valore di FP stimato è stato utilizzato, in questo lavoro, per il calcolo di metriche impiegate per analisi di tipo comparativo e pertanto l'errore introdotto non è stato considerato significativo.

Partendo dalle metriche delle *Suite CK* sono stati definite altre sei metriche, calcolate come segue:

$$WMC_{FP}(O) = \frac{WMC(O)}{FP_O}, \tag{3.5}$$

$$DIT_{FP}(O) = \frac{DIT(O)}{FP_O}, \tag{3.6}$$

$$NOC_{FP}(O) = \frac{NOC(O)}{FP_O}, \tag{3.7}$$

$$COU(O) = \frac{CBO(O)}{FP_O}, \tag{3.8}$$

$$RFC_{FP}(O) = \frac{RFC(O)}{FP_O}, \tag{3.9}$$

$$COH(O) = -\frac{LCOM(O)}{FP_O}, \tag{3.10}$$

in cui FP_o è la stima dei Function Point implementati dall'oggetto O.

Le metriche della *Suite CK* esprimono qualità degli oggetti relative al rapporto di tali oggetti con il resto del sistema. Le nuove metriche calcolate rappresentano il concetto più esteso di misure che tengono conto della dimensione dei singoli oggetti.

La media del valore di una di queste metriche, effettuata su tutti gli oggetti che compongono un sistema software, è, grazie all'utilizzo dei Function Points, in grado di tenere conto del peso di ciascun oggetto in rapporto alle funzionalità implementate.

Tramite questo metodo, ciascun oggetto che compone il sistema contribuisce alla media del sistema in rapporto al suo peso specifico all'interno di esso.

È importante sottolineare che tale peso non è dipendente dalle dimensioni del sistema ma è una stima della rilevanza dell'oggetto in senso assoluto. Un oggetto che implementa un alto numero di Function Points non è solo importante all'interno del sistema, ma è generalmente considerato un oggetto di dimensione rilevante.

3.2.2 Struttura della Social Network

All'interno di un team di sviluppo possono crearsi, spontaneamente o meno, diversi modelli di interazione sociale. In ambito Open Source accade spesso che l'interazione non avvenga tramite un contatto personale ma attraverso un qualche canale di comunicazione.

Con l'avvento delle tecnologie legate ad internet sono nate diverse piattaforme che offrono il supporto per lo sviluppo di applicazioni anche per sviluppatori che vivono in differenti parti del mondo. Questi sistemi di supporto, qualora offrano un accesso libero e completo a tutti i dati in es-

si contenuti, costituiscono una grossa opportunità di studio dei modelli di interazione sociale.

Campione analizzato

Un supporto largamente utilizzato per lo sviluppo di software sono i sistemi di controllo versione. Il controllo versione viene utilizzato nei processi di sviluppo software come base per la gestione dei dati riguardanti l'evoluzione del software stesso. Questi sistemi devono essere in grado di memorizzare tutte le modifiche che vengono effettuate sul codice sorgente di un software nel corso del processo di sviluppo.

Quando ad un unico processo di sviluppo contribuisce una comunità di sviluppatori, qualsiasi siano i metodi di coordinamento utilizzati, la modifica del codice è connessa a rischi di incompatibilità con ogni altra modifica effettuata da altri sviluppatori. Il conflitto può nascere anche da un punto di vista strettamente legato alla sincronizzazione delle modifiche. Un sistema di controllo versione deve quindi tener conto di ciascuna modifica, sincronizzarla con le altre e fornire il supporto alle decisioni qualora le modifiche risultino incompatibili.

Tali sistemi non si occupano solo di numerare in maniera sequenziale ciascuna modifica effettuata ma anche di registrare informazioni su chi la effettua. In questo modo è possibile tenere traccia di tutti gli sviluppatori che eseguono modifiche su uno stesso modulo, ad esempio una stessa classe.

Proprio in relazione al fatto che la modifica concorrente su uno stesso oggetto da parte di diversi sviluppatori sia un'attività critica, l'esigenza di coordinamento è elevata. Basandosi su questa esigenza coloro che collaborano, con le proprie modifiche, all'evoluzione di uno stesso oggetto devono relazionarsi. La collaborazione di due sviluppatori alla realizzazione di uno stesso oggetto, ad esempio alla scrittura del codice di una stessa classe, evidenzia un rapporto di esigenza informativa tra i due attori, esigenza che si deve risolvere in un meccanismo di coordinamento.

Con riferimento al già citato meccanismo di contratti sociali tipici della Network Governance si può vedere l'esigenza di coordinamento tra i due attori come una connessione. All'interno di una comunità, secondo questo modello, una connessione implica che due sviluppatori abbiano lavorato ad uno stesso progetto e quindi si siano dovuti mutuamente coordinare, in qualche modo, per la realizzazione di uno scopo comune.

Una mancanza di connessione indica invece l'indipendenza informativa tra i due sviluppatori. Considerando quindi ogni oggetto, ogni classe, come un singolo progetto è possibile analizzare l'interazione sociale di un'intera comunità partendo dall'analisi dell'intero software su cui essa è all'opera, ovvero sull'insieme di oggetti che compongono il sistema.

Se un singolo oggetto permette di identificare un determinato numero di connessioni tra sviluppatori, un intero sistema permette di identificare un'intera rete di connessioni.

Su questa assunzione si basano alcuni studi effettuati sui modelli di interazione sociale in ambito Open Source [32, 37].

Un repository SVN è una tecnologia basata su Subversion, un sistema di controllo versione molto diffuso. Generalmente un repository Svn è dedicato ad un solo progetto e mantiene informazioni sull'evoluzione del software nel corso del processo di sviluppo.

Attraverso questo tipo di tecnologia è possibile avere accesso ad una base di dati in grado di fornire molte informazioni che riguardano il software.

Il repository è generalmente organizzato in diverse sezioni dedicate:

- **Trunk**: Contiene tutti i file che compongono il progetto principale, numerati per versione ed aggiornati allo stato corrente dei lavori

- **Tags o Release**: Contiene una struttura di cartelle etichettate con il nome di una versione del progetto rilasciata. All'interno di ognuna di queste cartelle vi sono tutti i file che componevano il progetto principale al momento di una data release. Questa sezione può essere vista come un insieme di fotografie che memorizzano lo stato del progetto

nel preciso momento in cui il software viene rilasciato. Tipicamente questa sezione viene realizzata prelevando, al momento del rilascio, tutti i file contenuti nella sezione *trunk* e spostandoli nella cartella etichettata con il nome della versione.

- **Branches**: Questa sezione contiene tutti i progetti secondari legati a quello principale. Questi progetti secondari, nel corso dell'evoluzione del software, possono anche rientrare nel progetto originale sotto forma di nuove funzionalità. In tal caso il progetto entra a far parte della sezione *trunk*.

Il tutto è realizzato tramite tecnologie client/server che permettono agli utenti di ottenere accesso al repository, scaricare l'intera base di dati, aggiornarla, ed eseguire modifiche sui file.

La sincronizzazione delle modifiche avviene tramite l'utilizzo di due operazioni, *update* e *commit*. Con l'operazione di *update*, è possibile ottenere una versione aggiornata del repository e quindi sincronizzare la propria versione con le ultime modifiche effettuate da parte degli altri utenti. L'operazione di *commit* permette invece di inviare al server le modifiche effettuate sulla propria copia del repository.

Ogni volta che uno sviluppatore effettua una modifica significativa su uno dei file o sulla struttura di cartelle che compone il repository, esegue un'operazione di *commit*. In seguito all'operazione, il sistema effettua una copia del file e la numera. Insieme alla copia del file vengono salvate informazioni che riguardano l'identificativo dello sviluppatore che effettua la modifica, data e ora, il tipo di modifica effettuata (creazione,eliminazione, modifica di un file) e un commento da parte dell'autore della modifica.

Nel presente lavoro saranno analizzati due diversi repository, uno che contiene informazioni riguardanti Compiere [1] e un altro che contiene quelle riguardanti ADempiere [2].

[1] http://svn.compiere.org/core

[2] https://adempiere.svn.sourceforge.net/svnroot/adempiere

Un repository è una base di dati generalizzata che coinvolge tutti gli aspetti del progetto, aspetti che non riguardano solo il codice sorgente di un'applicazione.

All'interno di un repository vi può essere qualsiasi tipo di file necessario per il funzionamento del software nonché file utili agli sviluppatori come supporto al processo di sviluppo.

Per l'analisi che questo lavoro si propone di realizzare non è necessario possedere informazioni su tutti i file che compongono il software ma solo sui file che contengono il codice sorgente. Dato che entrambi i software analizzati sono stati realizzati con tecnologia Java, i file di interesse sono file con estensione .java che contengono il codice sorgente dei software.

Un repository SVN non è però una base di dati che si presta ad interrogazioni di tipo analitico. Un repository registra dati su un'insieme di eventi distribuiti ed è orientato a garantire proprietà come l'atomicità delle operazioni, l'integrità dei dati e la persistenza. È una base di dati orientata alla transazione. I repository SVN non offrono, dunque, possibilità di effettuare interrogazioni di tipo analitico come quella necessaria a questo lavoro.

Per poter effettuare queste interrogazioni è stato appositamente sviluppato un tool in grado di estrarre dal repository le informazioni necessarie e le memorizzarle in una base di dati adatta ad interrogazioni di tipo analitico.

Il tool, implementato anch'esso con tecnologia Java, sfrutta i metodi della classe *SVNRepository* per prelevare informazioni dal repository e inserirle all'interno di una base di dati SQL. Tale software è pensato per scaricare informazioni partendo dalla *repository root*, ovvero dalla cartella principale di tutto il progetto contenente le sezioni *Trunk*, *Tags* e *Branches*.

Inoltre sono state implementate funzioni orientate a consentire una navigazione selettiva all'interno del repository che permettesse di scaricare tutte le versioni di un file, anche in situazioni in cui un file fosse stato spostato all'interno del repository.

In questo modo il tool non si limita a scaricare informazioni dai file con estensione .java da tutta la base di dati, ma permette, selezionando uno

specifico percorso, di scaricare informazioni su ogni file contenuto in esso e seguire a ritroso la sua evoluzione in ogni posizione precedentemente occupata. Questa funzionalità è importante per escludere dall'analisi tutti quei file che appartengono a progetti secondari non inclusi nelle versioni rilasciate, ma di includere i file che appartengono a progetti secondari che sono stati in seguito inclusi nel progetto principale. Il tool è poi stato eseguito sia sul repository di Compiere che a quello di ADempiere.

Avendo costruito una base di dati contenente tutte le versioni numerate dei file che compongono le varie release di ciascun software e avendo le informazioni sugli autori di tutte le modifiche effettuate nel corso del processo di sviluppo, è quindi stato possibile effettuare interrogazioni di tipo analitico che hanno permesso di estrarre una rete di collaborazioni. Tale rete può essere utilizzata come modello di interazione sociale all'interno della comunità, con preciso riferimento al software e alla versione analizzata.

In questo modo si possono ottenere tanti modelli di interazione sociale quante sono le versioni di software analizzate. Dato che le versioni di software sono rilasciate nel corso degli anni e rappresentano una vera e propria evoluzione del prodotto, dall'analisi delle collaborazioni tra sviluppatori nei periodi compresi tra i vari rilasci di versione, è possibile ricavare dati sull'evoluzione dei modelli di interazione sociale nel corso del tempo.

Operazionalizzazione delle variabili

Non è sufficiente disporre di una serie di reti sociali per descrivere adeguatamente l'evoluzione dei modelli di interazione sociale all'interno di una comunità di sviluppatori.

Per raggiungere tale scopo occorre identificare i parametri che descrivono le caratteristiche di tali reti ed analizzare il modo in cui essi variano nel tempo.

Una tipico approccio all'analisi delle reti sociali è rappresentato dalle metriche di centralità. Il concetto di centralità è associato ad un singolo nodo della rete e rappresenta la sua importanza all'interno della rete stessa.

Un nodo che ha un grado di centralità elevata è un nodo che ha una posizione di rilievo all'interno della rete, dove con posizione di rilievo si intende una posizione di potere. Il potere, all'interno di una rete sociale, è strettamente legato al concetto di capacità di comunicazione con il resto della rete.

Per un singolo nodo, in un complesso reticolato di contratti sociali come le reti in analisi, vi possono essere diversi modi di essere importante. Un nodo che ha molte connessioni influenza molti altri nodi e di conseguenza uno sviluppatore che collabora con molti altri sviluppatori è in una posizione di potere, in quanto il suo operato influenza un gran numero di altri sviluppatori: questo può avvenire in diverse maniere.

Uno sviluppatore con molte connessioni ad altri individui della rete dimostra di aver lavorato a molti progetti o perlomeno di aver lavorato a diversi progetti importanti. Il concetto di progetto importante è legato al numero di sviluppatori che vi hanno partecipato. Tuttavia, trascurando il numero e l'importanza dei progetti a cui uno sviluppatore ha lavorato, avere molte connessioni indica la capacità di comunicare con una parte importante della comunità e quindi di avere potere di influenzarla. Ma avere molte connessioni non è l'unico modo di avere potere all'interno di una rete sociale. All'interno di una rete che possiede una struttura complessa possono formarsi delle sottoreti. Per sottorete si intende un insieme di nodi fortemente connessi tra loro ma collegati alla rete principale solo attraverso un numero limitato di connessioni, ovvero solo tramite un numero relativamente basso di altri nodi.

Questo tipo di struttura, una comunità all'interno di una comunità, indica che nel progetto principale possono essersi venuti a creare altri sotto-progetti o gruppi di sotto-progetti. Un gruppo di sotto-progetti può essere visto come un singolo modulo, un sottoinsieme funzionale dell'intero software, come può essere ad esempio il modulo CRM all'interno di un software ERP extended.

Il fatto che un gruppo di sviluppatori sia fortemente impegnato nello sviluppo di un modulo ma scarsamente impegnato nell'implementazione di

funzionalità che non riguardano quel modulo, può risultare nella formazione di una di queste sotto-strutture. Al di là del numero di connessioni di un singolo nodo, essere un nodo di connessione tra una di queste comunità interne ed il resto della rete è una posizione di importanza.

Uno sviluppatore responsabile del processo di sviluppo di un modulo importante, a cui partecipa un buon numero di sviluppatori, può anche essere connesso a pochi di essi ma costituisce l'unico legame fra gli sviluppatori di quel modulo e il resto della rete. Questo significa che ha una posizione molto importante poiché ogni informazione tra la comunità interna e il resto della rete deve necessariamente passare attraverso di lui.

C'è ancora un altro modo in cui un nodo può essere importante nella rete. Un nodo con un basso numero di connessioni e che non sia collegamento tra nessuna sottorete può comunque influenzare gran parte della rete. Infatti un nodo molto vicino ad un centro di comunicazione, ad esempio vicino ad un nodo che è di collegamento tra due comunità interne, non può comunicare direttamente con molti nodi ma è relativamente vicino ad un nodo che può farlo.

In questo modo può comunque influenzare gran parte del resto della rete. Allo stesso modo uno sviluppatore che collabora spesso con un capo progetto, è vicino ad un centro di passaggio di molte informazioni ed acquista gradualmente importanza all'interno del progetto. Inoltre uno sviluppatore che sia vicino ad un centro di comunicazione può più facilmente collaborare con il resto della comunità.

Il grado di centralità di un nodo è indice dell'importanza dello sviluppatore che quel nodo rappresenta. Per questo motivo le metriche di centralità dei nodi della rete possono fornire importanti informazioni sulle caratteristiche dell'interazione all'interno della comunità che la rete rappresenta. In questo lavoro si fa particolare riferimento a due tipi di metriche: *closeness* e *betweennes*.

La metrica *closeness* esprime un rapporto di vicinanza di un nodo rispetto al resto della rete. Questa vicinanza è calcolata in base al cammino

minimo tra il nodo esaminato e tutti gli altri nodi della rete. Un nodo con un'elevata *closeness* riesce a comunicare con il resto della rete con facilità maggiore di un nodo con *closeness* più bassa.

Il valore di CLO del generico nodo N_i è calcolato come segue:

$$CLO(N_i) = \frac{N-1}{\sum_{i=1}^{N} d(N_i, N_j)}; \tag{3.11}$$

dove N è il numero dei nodi della rete, $N_i \neq j$ e la funzione $d(N_i, N_j)$ restituisce la distanza geodesica tra i due nodi considerati.

La metrica *betweennes* esprime la proprietà di un nodo di essere un centro di comunicazione. Questa metrica è basata sul numero di percorsi minimi tra due generici nodi della rete che attraversano il nodo in esame.

Il valore di BET relativo ad un generico nodo N_iè calcolato come segue:

$$BET(N_i) = \frac{2}{(N-1)(N-2)} \sum_{j<k} \frac{g_{jk}(N_i)}{g_{jk}}; \tag{3.12}$$

dove N è il numero di nodi della rete, g_{jk} rappresenta il numero di percorsi minimi tra i nodi N_j eN_k mentre $g_{jk}(N_i)$ restituisce il numero di percorsi minimi tra N_j e N_k che passano per N_j.

I valori di CLO e BET legati a ciascun nodo sono stati calcolati utilizzando il software per l'analisi di reti *Pajek*.

Le metriche fino ad ora descritte si riferiscono alla centralità dei singoli nodi. Per ottenere valori che siano espressioni delle caratteristiche dell'intera rete è stata calcolata la media sull'insieme di nodi che rappresentano gli sviluppatori di una stessa comunità. Ogni rete ricavata da una versione di software differente è stata considerata una comunità differente o meglio, una differente fotografia di una comunità, scattata in un dato momento della sua evoluzione.

Una volta ottenuti i valori della metriche descritte, è possibile ottenere un andamento nel tempo delle proprietà corrispondenti relative ai modelli di interazione sociale.

Analizzando il variare di tali metriche è possibile ottenere informazioni sull'evoluzione della struttura di contratti sociali all'interno della comunità e quindi sulle caratteristiche di coordinamento tra gli attori implicati nel processo di sviluppo.

Capitolo 4

Analisi dei risultati

"La realtà è ciò che è, non ciò che dovrebbe essere."

Lenny Bruce

In questo capitolo verranno presentati e discussi i risultati delle analisi descritte nel Capitolo 3. Nella Sezione 4.1 verranno illustrati alcuni modelli che rappresentano la struttura software dei campioni analizzati e saranno discusse le caratteristiche di questi modelli dal punto di vista delle metriche che esprimono la qualità strutturale interna di un'applicazione; nella Sezione 4.2 verranno commentati i modelli di interazione sociale relativi alle due differenti comunità di sviluppatori e analizzate le evoluzioni delle metriche relative alle strutture delle reti sociali ricavate. Nella Sezione 4.3 saranno descritti gli strumenti di analisi utilizzati per ricavate i modelli che descrivono le dinamiche di mutua influenza tra la struttura sociale relativa ad un team di sviluppo e la struttura interna del software. Verranno inoltre discusse le capacità di ciascun modello di Governance di influire sulle dinamiche descritte da tali modelli. Infine nella Sezione 4.4 verranno presentati i test di ipotesi e le analisi qualitative effettuati sulle metriche che esprimono le caratteristiche della struttura del software con lo scopo di valutare quale modello di Governance abbia permesso la realizzazione di un prodotto contraddistinto da una qualità strutturale interna migliore.

4.1 Analisi della struttura software

Come discusso nel Capitolo 2 esistono diverse rappresentazioni di software che prevedono l'utilizzo delle reti. Rappresentare un software come una rete è un modo per studiarne la struttura. L'obiettivo di questo lavoro è, appunto, lo studio delle mute influenze tra struttura sociale che regola le collaborazioni all'interno di una comunità di sviluppatori e struttura del software sviluppato.

Sia un modello di interazione sociale, che la configurazione degli elementi che compongono un software, possono essere raffigurati tramite una rete. Studiando le proprietà delle reti che descrivono tali modelli è possibile ricavarne elementi di somiglianza.

La struttura interna del software può essere descritta tramite un modello che esprime il grado di interrelazione tra gli oggetti più semplici che lo compongono, ad esempio i moduli software.

In particolare nei linguaggi ad oggetti è possibile scomporre l'intero software in parti funzionalmente più semplici, dette classi, che descrivono le proprietà e i metodi degli oggetti del linguaggio. Analizzando le relazioni tra le classi è possibile esprimere le relazioni tra gli oggetti che sono istanze delle stesse.

In questo lavoro è stata adottata una rappresentazione in cui parti che compongono un'applicazione, in questo caso le classi Java, sono rappresentate dai nodi di una rete. Le connessioni tra i diversi nodi rappresentano una relazione di tipo funzionale. Una classe è connessa ad un'altra se almeno un metodo della prima effettua chiamate a metodi implementati nella seconda. È stato scelto di rappresentare il sistema attraverso una rete con archi bidirezionali e quindi le relazioni tra le classi presentano un verso.

Il concetto di relazione tra le classi è quindi il medesimo utilizzato per la definizione della metrica COU e quindi per il calcolo del *coupling* del sistema.

Esiste dunque una relazione tra le metriche che esprimono la qualità del software e la struttura dello stesso, rappresentata tramite una rete. Per

ciascuna delle versioni analizzate è stata estratta una rete che rappresenta le connessioni tra le classi, tali connessioni sono rilevate attraverso l'analisi delle chiamate a *runtime* tra i vari metodi.

L'evoluzione di tali reti è interessante in quanto direttamente collegata con il livello di accoppiamento del sistema e, quindi, ai costi legati al processo di sviluppo. Utilizzando il metodo Kamada-Kawai per le visualizzazione di grafi abbiamo potuto rappresentare graficamente le relazioni tra le varie classi che compongono i sistemi software analizzati, sottolineando le caratteristiche strutturali legate al *coupling*.

In Figura 4.1 sono rappresentate alcune reti che sono significative per mostrare l'evoluzione della struttura dei software Compiere e ADempiere in diversi momenti della storia dei due progetti.

Si può innanzitutto osservare che la rete estratta dalla versione 244 di Compiere presenta una struttura molto irregolare. I nodi che rappresentano le classi del software sono sparsi e piuttosto distanti tra loro. Si ricorda che una connessione tra due nodi, nell'algoritmo di visualizzazione Kamada-Kawai, ha l'effetto di una forza attrattiva che si oppone alla forza repulsiva che tutti i nodi si esercitano reciprocamente. Il fatto che i nodi siano rappresentati in modo sparso sul piano significa, quindi, che non si rilevano forti gradi di accoppiamento tra nessun gruppo di nodi della rete. Il valore di *coupling* misurato su tale versione di software è, infatti, il più basso tra quelli misurati.

L'iniziale dispersione dei vari oggetti del sistema si trasforma, con la maturazione del prodotto software, fino a diventare un nucleo fortemente connesso. Esso rappresenta il cuore del prodotto ed è attorniato da un sottile anello di oggetti meno connessi al sistema. Questa forma indica un aumento del livello di accoppiamento tra la maggior parte delle classi che compongono Compiere, un fenomeno rilevato anche dall'analisi delle metriche misurate sul sistema.

Le reti ricavate dalla versione 253b di Compiere (l'ultima prima del fork) e dalle versione 310 di ADempiere (la prima dopo il fork) presentano una

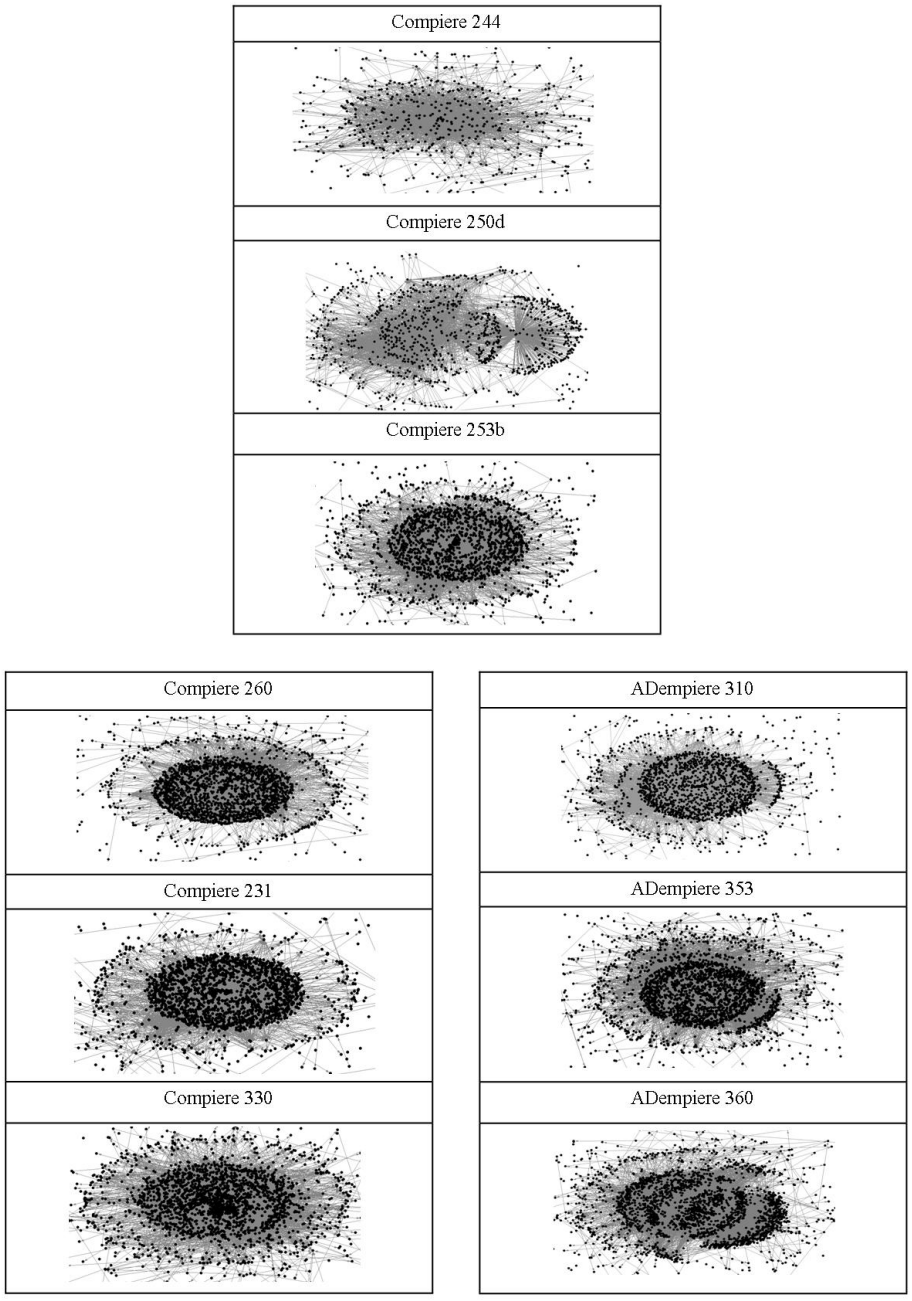


Figura 4.1: Alcune "fotografie" delle reti che rappresentano il software Compiere e ADempiere in diversi momenti della loro evoluzione

topologia molto simile. Questo, insieme ai valori misurati per le diverse metriche analizzate, è un ulteriore prova che la struttura di ADempiere ricalca, inizialmente, molto da vicino quella di Compiere.

Tuttavia, osservando le caratteristiche delle reti che rappresentano l'ultima versione di Compiere (330), e l'ultima versione di ADempiere (360), si può notare come i due progetti abbiamo infine preso strade diverse.

Il progetto ADempiere sembra avere un accoppiamento meno uniforme. Il nucleo centrale si spezza in cinque parti maggiormente accoppiate fra loro. Questo comportamento, pur non avendo un effetto chiaramente rilevabile tramite la misurazione delle metriche, può indicare un tipo di accoppiamento tra le classi più selettivo.

Essendo le applicazioni ERP una classe di software fortemente modularizzati, un accoppiamento alto solo tra i vari moduli che compongono il sistema potrebbe essere una caratteristica più desiderabile di un accoppiamento uniforme tra tutti gli oggetti.

La relazione tra livello di *coupling* di un sistema e connessioni tra nodi nella rete che lo rappresenta, non è l'unica relazione tra le metriche di qualità e le caratteristiche topologiche della struttura software.

In questo lavoro è stato dato un peso altrettanto importante al livello di coesione di un sistema. Tale caratteristica è descritta dalla metrica COH che, come già detto chiarito, descrive il concetto di misura di quanto i singoli moduli di un sistema siano coesi al proprio interno.

Immaginando di scomporre ogni singolo modulo che compone un sistema software in parti ulteriormente più semplici, si potrebbe ottenere una rappresentazione della struttura interna di ciascuna classe analizzata.

Sebbene molto più difficile da rappresentare graficamente, anche la struttura interna di un singolo modulo possiede caratteristiche topologiche. Il grado di correlazione tra la metrica COH e la struttura interna delle singole classi è lo stesso di quello tra metrica COU e la rete di connessioni tra i vari oggetti che compongono un sistema software.

All'interno di questo lavoro le metriche COU e COH sono di fondamen-

tale importanza per descrivere proprietà strutturali della rete software. Per questo motivo sono state utilizzate nell'analisi dell'influenza tra la struttura sociale interna di una comunità di sviluppatori e struttura del software sviluppato.

4.2 Analisi della struttura sociale

Lo studio effettuato in questo lavoro ha lo scopo di analizzare gli effetti di due diversi tipi di gestione del processo di sviluppo sul medesimo prodotto. Questo non sarebbe possibile senza analizzare l'impatto della gestione del processo di sviluppo sull'organizzazione sociale degli sviluppatori.

In questo lavoro, ciascun modello di organizzazione sociale tra sviluppatori è rappresentato tramite una Social Network. Tale Social Network è ricavata da una rete di collaborazioni, ovvero di modifiche concorrenti sulle stesse unità software, ed è quindi una stima del modello di interazione sociale all'interno della comunità.

In quanto software aderente al paradigma del Community Open Source, ADempiere dovrebbe essere caratterizzato da un maggiore orientamento alla comunità. Compiere, viceversa, è per sua stessa natura un prodotto commerciale. Pur essendo Open Source, il suo orientamento a generare profitto, un maggiore controllo sulla condivisione delle informazioni, una gestione centralizzata delle decisioni e modalità di contribuzione regolamentate da contratti, fa di questo progetto qualcosa di più vicino al paradigma di Governance aziendale.

Ci si aspetta che tali differenze siano evidenziate da una diversa struttura all'interno della Social Network che rappresenta la comunità di sviluppatori. In mancanza di una differenza nelle caratteristiche del modello di interazione sociale non si potrebbe sostenere di trovarsi effettivamente di fronte a due tipi di gestione differente.

Essendo entrambi i progetti di tipo Open Source (ed essendo quindi il codice sorgente dei due software disponibile interamente) si potrebbe obiettare che le caratteristiche di gestione delle informazioni non abbiano influenza sul

resto. Ovvero si potrebbe obiettare che, se all'interno del paradigma Open Source la comunicazione si modella sul prodotto, e quindi sul software, allora il fatto di fornire pieno accesso al codice sorgente dovrebbe fornire un canale sufficiente per la condivisione delle informazioni.

Se così fosse, essendo il prodotto software il medesimo per entrambi i progetti, le caratteristiche di interazione sociale tra gli sviluppatori delle due diverse comunità non dovrebbe presentare differenze.

Viceversa, dato che i due modelli di interazione sociale risultano sostanzialmente differenti, si potrebbe concludere che, in questo caso, il codice sorgente del prodotto non è il solo canale di condivisione delle informazioni. Questo significa che esiste un'interdipendenza informativa, tra i vari sviluppatori di una comunità, le cui esigenze devono essere assolte attraverso qualche altro tipo di mezzo.

Un forte controllo dei canali ufficiali, ad esempio i forum di progetto e i blog degli sviluppatori, una regolamentazione delle modalità di contribuzione attraverso un sistema di contratti e un minore sostegno all'iniziativa nei confronti dei membri della comunità con contratti di livello inferiore, possono risultare determinanti per quanto concerne la condivisione delle informazioni.

Tali differenze rappresenterebbero quindi un aspetto fondamentale della gestione del processo di sviluppo e renderebbero evidente la necessità di discriminare sulla natura dei progetti anche all'interno del panorama Open Source.

Analizzando il repository SVN di ADempiere sono state estratte 16 reti corrispondenti alle 16 versioni rilasciate. Ciascuna rete corrisponde alle collaborazioni tra diversi sviluppatori, collaborazioni che hanno avuto luogo nel periodo di tempo intercorso tra il rilascio di due versioni differenti.

Accostando tra loro Social Network ricavate dalla stessa comunità, ma in momenti diversi della sua storia, si può in un certo senso rappresentare l'evoluzione della comunità nel corso del tempo.

In Figura 4.2 sono mostrate alcune delle Social Network ricavate dal

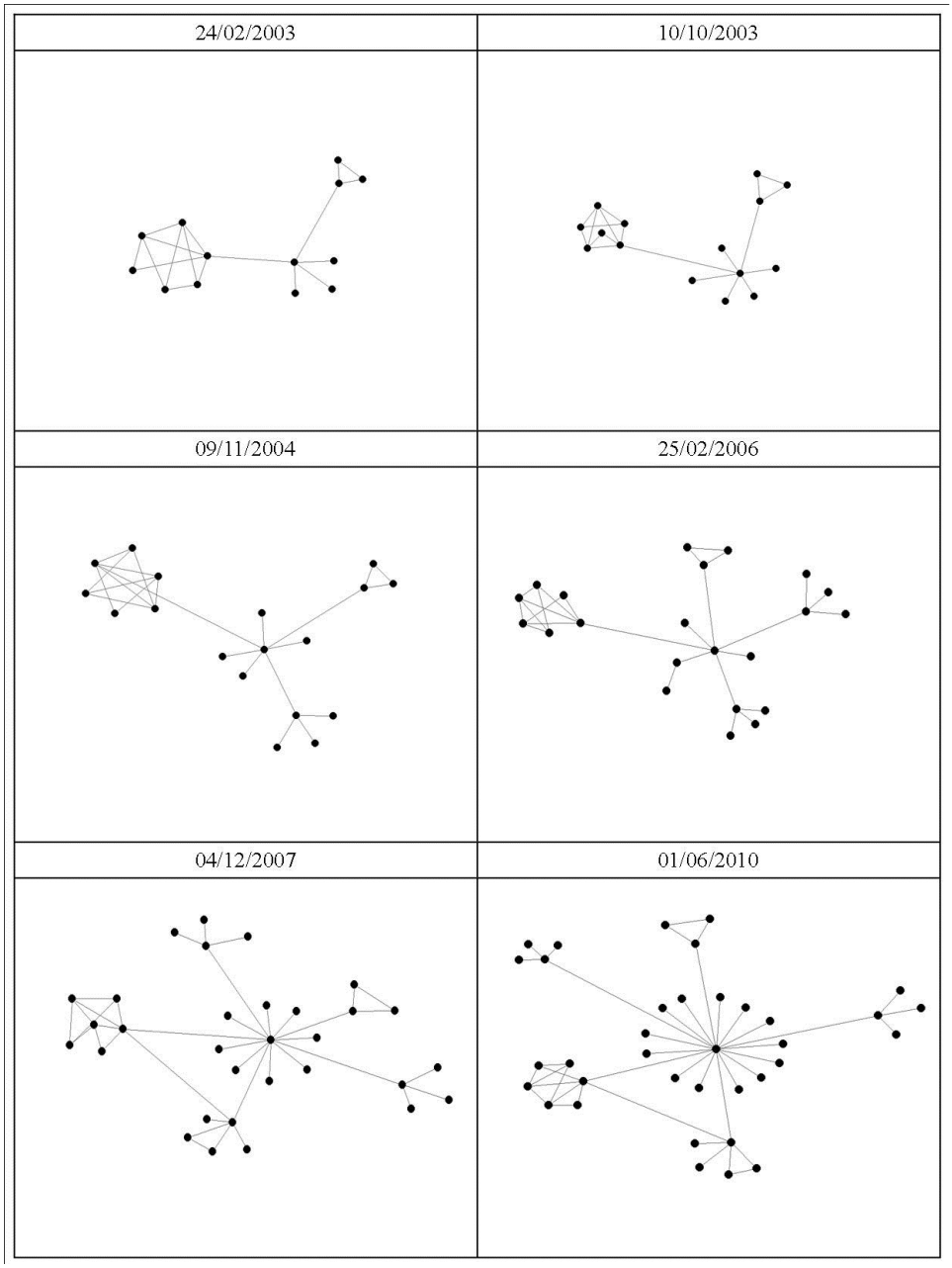


Figura 4.2: Alcune delle reti che rappresentano la comunità di sviluppatori di Compiere in diversi momenti della sua storia

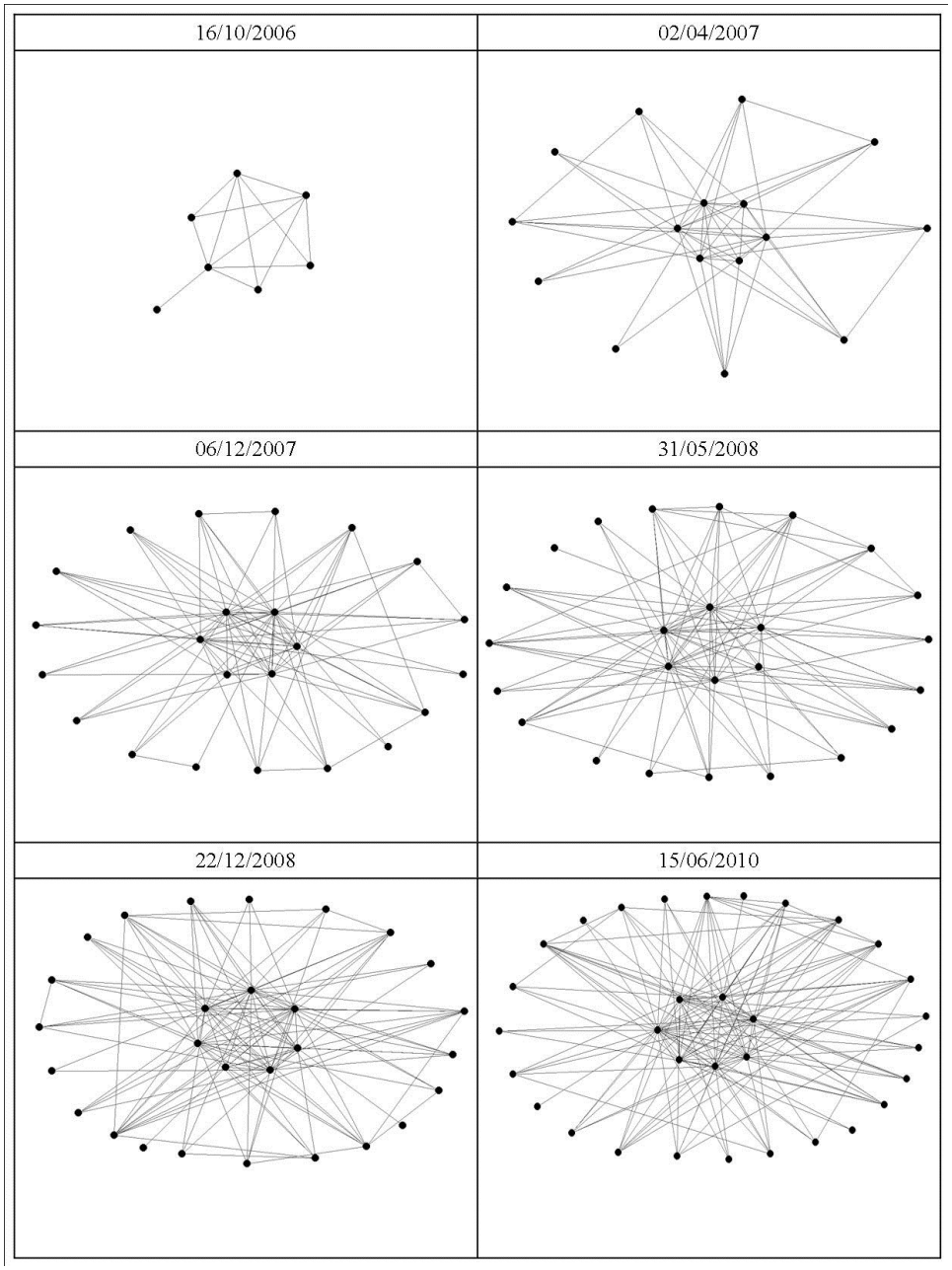


Figura 4.3: Alcune delle reti che rappresentano la comunità di sviluppatori di ADempiere in diversi momenti della sua storia

repository SVN di Compiere. La Figura 4.3 riporta, invece, alcune delle reti sociali estratte dal repository SVN del progetto ADempiere.

Appare evidente una sostanziale differenza tra i modelli di interazione sociale ricavati nei due diversi progetti. Le reti sociali ottenute dalla comunità di ADempiere presentano generalmente più connessioni e mostrano una distribuzione più omogenea delle stesse.

Le analoghe reti sociali estratte da Compiere sono caratterizzate da una distribuzione delle connessioni meno uniforme. È possibile rilevare un analogia tra le reti ricavate da Compiere e le reti Scale Free. La presenza di pochi nodi con un grado molto elevato rispetto alla media della rete è una caratteristica classica di questo tipo di reti.

Questo tipo di struttura è quella che ci si aspetta quando, all'interno di una comunità, ci sono nodi di grande importanza che caratterizzano il flusso informativo di tutto il resto della rete. La struttura delle reti sociali ricavate da Compiere sembra essere la conferma di una gestione molto più centralizzata rispetto a quella di ADempiere.

D'altra parte la distribuzione più omogenea del grado rilevata nelle reti di ADempiere indica una maggiore uniformità tra i ruoli degli sviluppatori rappresentati dai vari nodi.

Le differenze nella struttura delle reti sociali ricavate dalle due diverse comunità riflettono alcune caratteristiche dei diversi modelli di Governance adottati nei progetti.

Tali differenze sono osservabili concretamente tramite l'analisi delle metriche legate alla centralità dei singoli nodi che compongono le varie reti.

Le metriche CLO e BET rappresentano i valori medi del grado di centralità dei nodi in una rete. Tali metriche esprimono la capacità media dei singoli nodi di influenzare il flusso informativo interno in una rete sociale.

Osservando le variazioni dei valori delle metriche è possibile notare come per entrambe le comunità i valori medi di centralità tendano a crescere con il tempo. Questa tendenza può essere spiegata con il costante aumento del numero di nodi nelle reti sociali, e quindi con l'ingresso di nuovi sviluppatori

nelle due comunità, unita alla creazione di un maggior numero di connessioni tra i nodi già presenti nelle reti.

È possibile ipotizzare che il costante aumento del livello di *coupling* dei corrispondenti sistemi software, rilevato nella sezione precedente, abbia un ruolo in questo tipo di dinamica. Ma si demanda alla Sezione 4.3 questo tipo di considerazioni.

Per quanto concerne l'analisi dell'interazione sociale si può rilevare che le reti sociali riguardanti la comunità di ADempiere hanno valori di BET molto inferiori a quelli osservati nella rete sociale di Compiere. Questa caratteristica indica una maggiore uniformità nell'importanza dei nodi della rete. Infatti, il significato di un più alto valore medio dei valori di *betweennes centrality* è che in una rete ci sono nodi che hanno un elevata capacità di mettere in comunicazione differenti parti della rete. Per contro, questo significa che un gran numero di informazioni devono necessariamente passare per uno di questi nodi di grande importanza. Questa caratteristica della rete sociale indica chiaramente che la comunità di Compiere è caratterizzata da un determinato numero di sviluppatori molto importanti che interconnettono tutti gli altri.

Questa è una caratteristica tipica delle reti Scale Free: pochi nodi con un livello estremamente alto (rispetto alla media della rete) di *betweenness* e con molte connessioni.

Diversamente, le reti sociali riguardanti la comunità di ADempiere mostrano un livello di *betweennes* inferiore. Questa caratteristica segnala una maggiore uniformità nell'importanza dei singoli nodi della rete. Caratteristica in linea con un forte orientamento alla comunità tipica dell'Open Source.

In altre parole un alto valore della metrica BET può indicare che, nella rete, le informazioni passano soprattutto attraverso un ristretto numero di nodi. Caratteristica che indica come la comunicazione interna sia fortemente influenzata da pochi nodi, molto importanti nella rete. Viceversa, un valore di BET più basso indica una minore presenza di *Structural Holes* e

quindi una struttura di rete più omogeneamente connessa. In una rete di questo tipo l'informazione circola in modo meno prevedibile e quindi i flussi informativi risultano certamente meno controllati.

La rete sociale della comunità di Compiere mostra un livello medio di *closeness centrality* dei nodi che la compongono piuttosto elevato. Questo indica che i vari sviluppatori della comunità sono tutti molto vicini, ovvero che per comunicare con un altro sviluppatore all'interno della comunità non è necessario passare per l'intermediazione di molti altri individui.

Questa vicinanza, unita ad un livello di *beetweennes* medio più elevato, può essere interpretata come indizio di una forte centralizzazione. Infatti, se all'interno di una comunità principale sono presenti tante comunità interne, collegate fra loro da un numero limitato di nodi con valore di BET elevato, la vicinanza o meno dei nodi appartenenti a queste comunità interne è dettata dalle relazioni tra questi nodi di importanza elevata. Se i nodi ad elevata importanza sono distanti tra loro, la distanza tra le comunità interne rimarrebbe comunque elevata. Se invece tutti i nodi di importanza elevata sono collegati ad un unico nodo centrale in grado di interconnetterli tutti, la distanza tra le varie comunità interne sarebbe notevolmente ridotta.

In Figura 4.4 vengono mostrate le caratteristiche di reti con diverse combinazioni di livelli alti e bassi di BET e CLO.

Un più alto valore sia della metrica BET che della metrica CLO, può essere indizio di un struttura fortemente influenzata dai nodi con il valore più alto di *betweenness*. Unitamente all'osservazione delle varie reti sociali relative alla comunità di Compiere, i valori delle metriche di centralità medi possono essere considerati prova della centralizzazione della comunità di Compiere, caratteristica tanto osteggiata dai membri della comunità di ADempiere che hanno voluto e ottenuto il fork dal progetto principale.

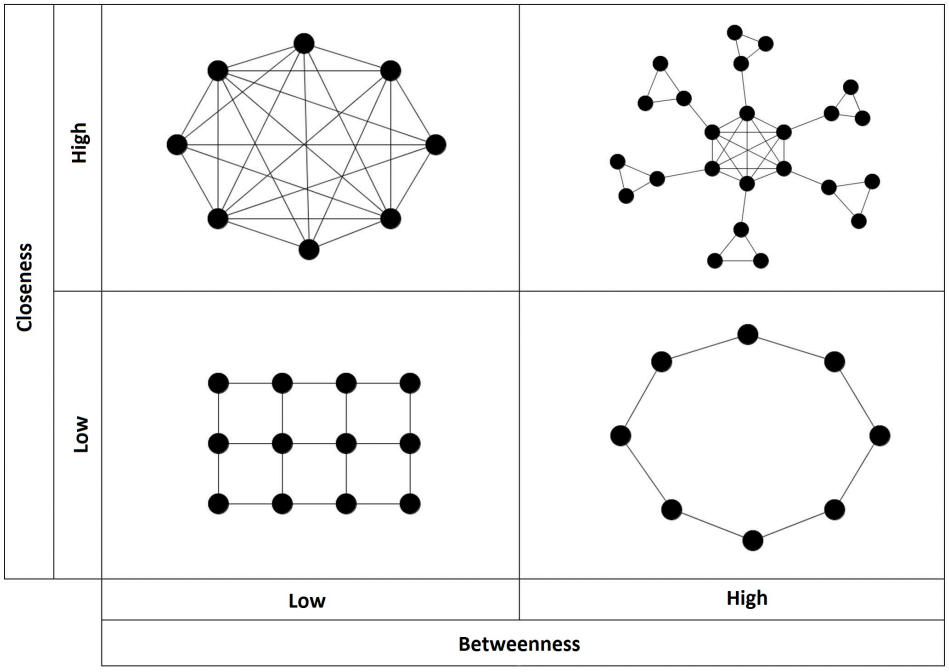


Figura 4.4: Reti caratterizzate da diversi livelli di betweennes e closeness centrality

4.3 Mutue influenze tra struttura del software e rete sociale degli sviluppatori

Molti studi hanno analizzato le mutue influenze tra caratteristiche del processo di produzione e caratteristiche del prodotto [3, 94, 47, 8, 64].

In particolare per quanto riguarda lo sviluppo software, sono stati osservati diversi tipi di simmetria tra la struttura sociale che regola i rapporti di collaborazione tra un gruppo di sviluppatori e la struttura del software che essi producono.

Disponendo di metriche che esprimono caratteristiche strutturali di una rete software (COU e COH) e di metriche che esprimono caratteristiche strutturali di una Social Network (CLO e BET), si può quindi misurare il grado di correlazione tra i valori che assumono nel tempo.

Sia le metriche che esprimono la qualità della struttura software, sia quelle che esprimono le caratteristiche delle reti sociali degli sviluppatori,

sono riferite a versioni di software che sono state rilasciate periodicamente nel corso di un certo numero di anni.

Essendo tali metriche associabili a ciascuna versione di software analizzato, ed essendo le versioni collocabili in un preciso contesto temporale, è possibile costruire delle serie temporali composte dai valori misurati.

Queste serie temporali esprimono l'andamento di ciascuna metrica nel tempo e quindi descrivono l'evoluzione della proprietà ad essa correlata.

Utilizzando alcuni test statistici è possibile analizzare queste serie temporali con lo scopo di trovare il modo in cui il valore delle metriche ad un certo istante, in un dato punto della serie temporale, è influenzato dai valori passati di tutte le altre serie considerate.

Uno degli strumenti utilizzati in questo lavoro è la Vector Autoregression Analysis. Questo tipo di analisi permette di ottenere, attraverso metodi di regressione lineare, dei modelli detti VAR. Tali modelli furono introdotti da Christopher Sims [95] ed hanno dimostrato nel tempo di avere una capacità predittiva notevole. Il modello usato per l'analisi in questo lavoro è il seguente:

$$\begin{aligned} CLO(t) = c_{CLO} + \beta_{1,CLO} CLO_{t-1} + \beta_{2,CLO} BET_{t-1} + \\ + \beta_{3,CLO} COU_{t-1} + \beta_{4,CLO} COH_{t-1} + \varepsilon_{CLO} \end{aligned} \tag{4.1}$$

$$\begin{aligned} BET(t) = c_{BET} + \beta_{1,BET} CLO_{t-1} + \beta_{2,BET} BET_{t-1} + \\ + \beta_{3,BET} COU_{t-1} + \beta_{4,BET} COH_{t-1} + \varepsilon_{BET} \end{aligned} \tag{4.2}$$

$$\begin{aligned} COU(t) = c_{COU} + \beta_{1,COU} CLO_{t-1} + \beta_{2,COU} BET_{t-1} + \\ + \beta_{3,COU} COU_{t-1} + \beta_{4,COU} COH_{t-1} + \varepsilon_{COU} \end{aligned} \tag{4.3}$$

$$\begin{aligned} COH(t) = c_{COH} + \beta_{1,COH} CLO_{t-1} + \beta_{2,COH} BET_{t-1} + \\ + \beta_{3,COH} COU_{t-1} + \beta_{4,COH} COH_{t-1} + \varepsilon_{COH} \end{aligned} \tag{4.4}$$

Questo tipo di analisi permette di ricavare i coefficienti $\beta_{i,MET}$ associati a ciascuna delle equazioni appena descritte. Ogni coefficiente rappresenta il contributo sul valore della generica metrica MET del valore di ciascuna altra metrica all'istante precedente.

È necessario quindi stabilire un passo per determinare cosa si considera come istante precedente. Nel presente lavoro è stato utilizzato un passo, anche detto ritardo τ, di valore unitario. Questo significa che ciascun passo di regressione è effettuato tenendo in conto solamente il valore delle metriche all'istante precedente. Nel presente lavoro un passo corrisponde ad un salto di versione, ovvero la regressione lineare su una metrica viene fatto tenendo conto del valore di tutte le altre metriche nella versione di software precedente.

I valori misurati da ciascuna versione di software possono essere ordinati attraverso le date di rilascio. Dall'analisi delle serie temporali è possibile descrivere, tramite i coefficienti calcolati nel modello VAR, in quale modo le variazioni osservabili in ciascuna serie temporale influenzano le altre.

Lo scopo di un modello VAR è rappresentare un modello di variazione congiunta delle variabili analizzate. Attraverso un modello VAR è, cioè, possibile ricavare le equazioni che descrivono i contributi di ciascuna variabile su tutte le altre che compongono il modello. In ogni equazione, ciascun coefficiente è associato ad un intervallo di confidenza e ad un valore di *p-value*.

Esistono altri test che sfruttano questo tipo di strutture. In particolare il Granger Causality Test, utilizzato anch'esso in questo lavoro, è in grado determinare se una serie temporale sia utile per prevedere l'evoluzione di un'altra serie temporale [91].

All'interno di tale test, una serie temporale X influenza (causa) una serie temporale Y, se è possibile dimostrare con strumenti statistici (t-test) che i valori ritardati di X forniscono informazioni significative sui valori futuri di Y.

Per fare questo tipo di analisi si sfruttano equazioni simili a quelle de-

scritte per la Vector Autoregression Analysis. La differenza con i modelli VAR è che in ciascun test uno dei coefficienti relativo ad una serie temporale al tempo precedente viene posto a zero. Ad esempio nell'equazione 4.1, per testare se il valore BET_{t-1} della serie temporale BET sia utile alla previsione del valore di CLO_t viene azzerato il coefficiente $\beta_{2,CLO}$.

In questo modo si cerca di testare l'ipotesi nulla che una serie temporale non sia utile per la previsione di un'altra considerata nel modello. Se il livello di significatività minimo (*p-value*), ottenuto dal Granger Causality Test, è ritenuto sufficientemente basso, l'ipotesi nulla viene scartata e ciò significa che la serie temporale BET influenza (causa) la serie temporale CLO.

Grazie a questo tipo di test è possibile tracciare un grafo delle influenze che ciascuna serie temporale esercita sulle altre e avere quindi utili informazioni sulle mutue influenze che caratterizzano il modello. Inoltre il test di Granger è stato utilizzato per fornire una conferma delle evidenze empiriche segnalate dal modello VAR.

Dato che sia la Vector Autoregression Analysis che il Granger Causality Test utilizzano metodi di regressione lineare, il presupposto a questo tipo di analisi è che la distribuzione dei campioni di dati sia gaussiana.

Se, per ciascuna metrica, il numero di campioni estratti dal software Compiere è 30, un numero sufficiente ad ipotizzare una distribuzione di tipo normale, per quanto riguarda il progetto ADempiere sono disponibili 16 versioni e quindi solamente 16 campioni per ogni metrica.

Per questo motivo è stato eseguito, per ciascun campione considerato, il test di buon adattamento alla distribuzione normale descritto da D'Agostino et al. [78]. I livelli di kurtosi (allontanamento dalla normalità distributiva) evidenziati dal test sono molto bassi per tutte le distribuzioni campionarie delle misurazioni effettuate. Hanno fornito riscontri positivi anche i valori di asimmetria e l'Omnibus K^2 test.

La Vector Autoregression Analysis è stata utilizzata per esaminare innanzitutto il grado di correlazione tra metriche di centralità della rete sociale degli sviluppatori e metriche di qualità della struttura del software.

Tale correlazione può essere espressa formulando un modello che descriva l'evoluzione congiunta di tutte le metriche di sistema analizzate.

Ciascun coefficiente calcolato è caratterizzato da un livello di significatività, espresso da un *p-value*, che indica il grado di evidenza empirica della relazione che il coefficiente esprime. Il significato di un valore di *p-value* superiore ad una certa soglia è che il calcolo del coefficiente di influenza della metrica corrispondente, all'interno dell'equazione considerata, è fortemente soggetto a un fattore di casualità. Questo tipo di risultato indica che non sono state rilevate evidenze empiriche forti dell'impatto del valore della variabile all'interno dell'equazione.

I livelli di significatività sono un parametro che non è sempre fissato a priori. In molti lavori di questo tipo si preferisce fissare i livelli di significatività considerati accettabili solo in seguito all'analisi dei risultati.

Nel presente lavoro si è scelto di utilizzare soglie di significatività considerate convenzionali. I valori di significatività utilizzati sono:

- *P-value* < 0.02 (2%): forte evidenza empirica
- $0.02(2\%) <$ *p-value* $< 0.05(5\%)$: debole evidenza empirica
- *P-value* > 0.1 (10%): nessuna evidenza empirica

In questo modo, anche solo limitandosi ad analizzare le significatività dei coefficienti, si possono ricavare informazioni sull'impatto del valore ad un certo istante di una serie temporale, su quello all'istante successivo di qualsiasi altra.

Se l'impatto è notevole, il *p-value* calcolato attraverso la Vector Autoregression Analysis sarà inferiore ai livelli di significatività selezionati. Se, viceversa, l'impatto è poco influente, il test non fornirà una sufficiente evidenza empirica della correlazione tra i due valori.

Date le sostanziali differenze tra la storia di Compiere e la storia di ADempiere sia in termini di evoluzione del progetto, sia dal punto di vista delle modalità di gestione del processo di sviluppo, una prima serie di analisi è stata effettuata separatamente sui due diversi software.

Facendo riferimento alle ipotesi formulate nel Capitolo 3, si voleva dimostrare che la correlazione tra la struttura della rete sociale degli sviluppatori e la struttura del software fosse influenzata dalla modalità di Governance utilizzata nel progetto.

In particolare come enunciato nell'ipotesi **H1** si vuole provare che, nonostante entrambi i software aderiscano nominalmente al paradigma Open Source, le significative differenze in termini di modalità di gestione del processo abbiano un effetto su tale correlazione.

I risultati prodotti dalla Vector Autoregression Analysis sulle 16 versioni del software Compiere, con ritardo τ pari a uno, sono elencati in Tabella 4.1.

Analizzando il grado di significatività, espresso nella quarta colonna della tabella, si può notare come ciascuna metrica risulti fortemente influenzata dal valore di se stessa all'istante t-1 ($CLO_{t-1} \rightarrow CLO_t$, coef=2.05, *p-value*=0.001; $BET_{t-1} \rightarrow BET_t$, coef=-1.64, *p-value*=0.031; $COU_{t-1} \rightarrow COU_t$, coef=0.88, *p-value*=0.000; $COH_{t-1} \rightarrow COH_t$, coef=0.86, *p-value* = 0.000). Questa caratteristica può essere considerata come indicatore di un'evoluzione regolare della valore delle metriche considerate.

Dai risultati esposti in Tabella 4.1 si può rilevare evidenza empirica dell'impatto delle metriche di centralità delle Social Network sulle caratteristiche di qualità strutturale del software ($CLO_{t-1} \rightarrow COU_t$, coef=-0.01, *p-value*=0.015; $CLO_{t-1} \rightarrow COH_t$, coef=-1.64, *p-value*=0.000; $BET_{t-1} \rightarrow COU_t$, coef=0.06, *p-value*=0.001 ; $BET_{t-1} \rightarrow COH_t$, coef=5.84, *p-value* = 0.000).

Tuttavia non si rileva alcuna evidenza empirica della relazione inversa, ovvero dell'impatto delle caratteristiche strutturali del software sulle metriche di centralità della Social Network.

Per verificare gli esiti della Vector Autoregression Analysis è stato effettuato il Granger Causality Test sulle serie temporali derivanti dalle metriche di ADempiere. Il test identifica le relazioni di causalità esistenti tra le diverse serie temporali considerate nel modello. La significatività del test è

riportata dal *p-value* sulla colonna più a destra.

Anche in questo caso si utilizzano le soglie di significatività discusse

Tabella 4.1: Risultati della Vector Autoregression Analysis applicata alle serie temporali ricavate da ADempiere

	Coef.	**Std. Err.**	**z**	***p-value***	**[95% Conf.**	**Interval]**
$\mathbf{CLO_t}$						
CLO_{t-1}	2.05177	0.61102	3.36	0.001*	0.85418	3.24935
BET_{t-1}	-4.7672	2.29235	-2.08	0.038**	-9.2601	-0.2743
COU_{t-1}	-5.7031	25.6424	-0.22	0.824	-55.961	44.5551
COH_{t-1}	-0.1306	0.39638	-0.33	0.742	-0.9075	0.64633
cons	4.87164	25.1128	0.19	0.846	-44.349	54.0919
$\mathbf{BET_t}$						
CLO_{t-1}	0.63604	0.20413	3.12	0.002*	0.23595	1.03613
BET_{t-1}	-1.6474	0.76584	-2.15	0.031**	-3.1484	-0.1464
COU_{t-1}	-8.5807	8.56672	-1	0.317	-25.371	8.20976
COH_{t-1}	-0.0537	0.13242	-0.41	0.685	-0.3132	0.20585
cons	7.76329	8.38978	0.93	0.355	-8.6804	24.207
$\mathbf{COU_t}$						
CLO_{t-1}	-0.0115	0.00471	-2.44	0.015*	-0.0207	-0.0023
BET_{t-1}	0.0602	0.01765	3.41	0.001*	0.0256	0.09479
COU_{t-1}	0.88988	0.19744	4.51	0.000*	0.50291	1.27684
COH_{t-1}	-0.0053	0.00305	-1.73	0.084***	-0.0113	0.0007
cons	0.08028	0.19336	0.42	0.678	-0.2987	0.45926
$\mathbf{COH_t}$						
CLO_{t-1}	1.63824	0.36391	4.5	0.000*	0.925	2.35149
BET_{t-1}	-5.8341	1.36526	-4.27	0.000*	-8.51	-3.1583
COU_{t-1}	-55.69	15.2719	-3.65	0.000*	-85.622	-25.758
COH_{t-1}	0.86481	0.23607	3.66	0.000*	0.40211	1.3275
cons	57.3943	14.9565	3.84	0.000*	28.0801	86.7085

*	**Forte evidenza empirica**
**	**Debole evidenza empirica**
***	**Evidenze empiriche minori**

precedentemente. Un basso valore di *p-value* va considerato come evidenza empirica del fatto che il valore all'istante precedente della serie temporale a cui il valore si riferisce ha una forte rilevanza nell'equazione.

Tabella 4.2: Risultati del Granger Causality Test applicato alle serie temporali ricavate da ADempiere

Equation	**Excluded**	**chi2**	**df**	***p-value***
CLO_t	BET_{t-1}	4.3247	1	0.038**
CLO_t	COU_{t-1}	.04947	1	0.824
CLO_t	COH_{t-1}	.10849	1	0.742
CLO_t	ALL	11.364	3	0.010*
BET_t	CLO_{t-1}	9.7082	1	0.002*
BET_t	COU_{t-1}	1.0033	1	0.317
BET_t	COH_{t-1}	.16445	1	0.685
BET_t	ALL	21.818	3	0.000*
COU_t	CLO_{t-1}	5.9618	1	0.015*
COU_t	BET_{t-1}	11.632	1	0.001*
COU_t	COH_{t-1}	2.9953	1	0.084
COU_t	ALL	20.521	3	0.000*
COH_t	CLO_{t-1}	20.266	1	0.000*
COH_t	BET_{t-1}	18.261	1	0.000*
COH_t	COU_{t-1}	13.297	1	0.000*
COH_t	ALL	27.65	3	0.000*

*	**Forte evidenza empirica**
**	**Debole evidenza empirica**
***	**Evidenze empiriche minori**

I risultati sono riportati in Tabella 4.2.

Si può osservare che le considerazioni effettuate sull'analisi precedente trovano sostanzialmente conferma. La metrica CLO risulta influenzata da BET e viceversa. Le metriche COU e COH sono influenzate da tutte le altre metriche del modello.

L'evidenza dell'impatto della Social Network sul software è un risultato

già riscontrato in diversi studi effettuati sul software Open Source. I risultati esposti in Tabella 4.2 possono essere visti come una conferma empirica della legge di Conway.

Tabella 4.3: Risultati della Vector Autoregression Analysis applicata alle serie temporali ricavate da Compiere

	Coef.	**Std. Err.**	**z**	***p-value***	**[95% Conf.**	**Interval]**
CLO_t						
CLO_{t-1}	0.33681	0.69498	0.48	0.628	-1.0253	1.69894
BET_{t-1}	1.71651	1.98972	0.86	0.388	-2.1833	5.61629
COU_{t-1}	18.3108	6.95315	2.63	0.008*	4.6829	31.9387
COH_{t-1}	0.21543	0.20583	1.05	0.295	-0.188	0.61884
cons	-9.0559	5.48926	-1.65	0.099***	-19.815	1.70289
BET_t						
CLO_{t-1}	-0.1495	0.226	-0.66	0.508	-0.5925	0.29342
BET_{t-1}	1.33194	0.64703	2.06	0.04**	0.0638	2.60009
COU_{t-1}	6.92249	2.26106	3.06	0.002*	2.4909	11.3541
COH_{t-1}	0.08059	0.06693	1.2	0.229	-0.0506	0.21178
cons	-3.5863	1.78502	-2.01	0.045**	-7.0848	-0.0877
COU_t						
CLO_{t-1}	0.0291	0.0175	1.66	0.096***	-0.0052	0.06339
BET_{t-1}	-0.0659	0.05009	-1.32	0.189	-0.164	0.03231
COU_{t-1}	0.13447	0.17504	0.77	0.442	-0.2086	0.47755
COH_{t-1}	-0.0136	0.00518	-2.62	0.009*	-0.0237	-0.0034
cons	0.39962	0.13819	2.89	0.004*	0.12877	0.67046
COH_t						
CLO_{t-1}	-0.8071	0.61029	-1.32	0.186	-2.0033	0.38903
BET_{t-1}	2.09822	1.74725	1.2	0.23	-1.3263	5.52277
COU_{t-1}	3.26253	6.10585	0.53	0.593	-8.7047	15.2298
COH_{t-1}	0.54954	0.18075	3.04	0.002*	0.19529	0.9038
cons	-4.8997	4.82035	-1.02	0.309	-14.347	4.54804

*	**Forte evidenza empirica**
**	**Debole evidenza empirica**
***	**Evidenze empiriche minori**

Nella Tabella 4.3 sono riportati i risultati ottenuti dalla Vector Autoregression Analysis su 30 versioni del software Compiere. Dall'analisi dei risultati ottenuti è possibile notare un gran numero di differenze con quelli ottenuti sul software ADempiere. Innanzitutto non vi è più una forte evidenza empirica dell'impatto della Social Network sulla struttura del software. In secondo luogo è possibile notare come la metrica COU, ovvero il livello di *coupling* del sistema, abbia un ruolo centrale nel modello. Si può rilevare una forte evidenza empirica dell'impatto di COU sulle metriche di centralità ($COU_{t-1} \rightarrow CLO_t$, Coef=18.31, *p-value*=0.008; $COU_{t-1} \rightarrow BET_t$, Coef=6.92, *p-value*=0.002). COH, la metrica che esprime il grado di coesione dei moduli software, è influenzata solo da se stessa all'istante precedente ($COH_{t-1} \rightarrow COH_t$, Coef=0.54, *p-value*=0.002).

Questo lascia pensare che la capacità da parte dell'intero sistema, ed in particolare delle metriche di centralità della Social Network, di influenzare la metrica COH sia molto bassa.

La Tabella 4.4 mostra i risultati del Granger Causality Test effettuato sulle versioni di Compiere.

Si può notare come anche da questo test la metrica COU risulti influenzare fortemente le metriche legate alla Social Network degli sviluppatori. Lo stesso discorso vale per la metrica COH che, secondo il modello di causalità fornito dal test, è indipendente dal resto del sistema. Le considerazioni effettuate precedentemente sono quindi avvalorate dal test di Granger.

Il numero di relazioni rilevanti è inferiore al numero di quelle individuate per il software ADempiere. C'è una forte evidenza empirica dell'impatto della coesione sul *coupling* del sistema e del *coupling* sulle metriche di centralità delle Social Network.

In pratica si può affermare che l'evidenza empirica dell'impatto della struttura del software sulla rete sociale degli sviluppatori è molto forte. Viceversa, l'impatto della Social Network sulle caratteristiche del software è di debole intensità. Un'evidenza empirica superiore al 90% è osservabile solamente tra CLO e COU ($CLO_{t-1} \rightarrow COU_t$, Coef=0.03, *p-value*=0.096).

Le metriche CLO e COU appaiono non correlate con il valore assunto nella versione precedente. Questo ha il significato che la variazione delle metriche non ha un andamento regolare ed è molto influenzata dalla variazione di altre metriche.

Il fatto che la metrica COU abbia un evoluzione poco controllata da variabili interne al modello ma che abbia una forte capacità di influenzare il modello stesso, lascia pensare che il livello di *coupling* sia una caratteristica molto importante nella progettazione di questo software e la sua evoluzione si fortemente condizionata da variabili non considerate.

Tabella 4.4: Risultati del Granger Causality Test applicato alle serie temporali ricavate da Compiere

Equation	**Excluded**	**chi2**	**df**	***p-value***
CLO_t	BET_{t-1}	.74424	1	0.388
CLO_t	COU_{t-1}	6.9351	1	0.008^{*}
CLO_t	COH_{t-1}	1.0954	1	0.295
CLO_t	ALL	7.5548	3	0.056^{***}
BET_t	CLO_{t-1}	.43777	1	0.508
BET_t	COU_{t-1}	9.3735	1	0.002^{*}
BET_t	COH_{t-1}	1.4499	1	0.229
BET_t	ALL	9.5114	3	0.023^{**}
COU_t	CLO_{t-1}	2.7659	1	0.096^{***}
COU_t	BET_{t-1}	1.7293	1	0.189
COU_t	COH_{t-1}	6.8439	1	0.009^{*}
COU_t	ALL	24.859	3	0.000^{*}
COH_t	CLO_{t-1}	1.749	1	0.186
COH_t	BET_{t-1}	1.4421	1	0.230
COH_t	COU_{t-1}	.28551	1	0.593
COH_t	ALL	3.9581	3	0.266

*	**Forte evidenza empirica**
**	**Debole evidenza empirica**
***	**Evidenze empiriche minori**

Questi risultati possono essere spiegati ipotizzando che, avendo un minor controllo del livello di *coupling* di un sistema da parte della comunità degli sviluppatori, la metrica COU sia influenzata da altre caratteristiche della gestione del processo di sviluppo.

Una forte influenza sulla struttura del software può essere esercitata da decisioni prese dai capi progetto nonché dalle caratteristiche funzionali del software stesso. Come si è visto, il processo di sviluppo di Compiere ha la caratteristica di un maggiore accentramento della decisioni. Tali decisioni non dipendono solo da caratteristiche della Social Network degli sviluppatori ma, in maniera anche più accentuata, da fattori non inclusi nel modello.

L'altra metrica apparentemente non correlata con il proprio valore all'istante precedente è CLO, che dipende fortemente da COU. Per il tipo di modello di interazione sociale scelto per questo lavoro, due sviluppatori sono connessi se collaborano ad uno stesso progetto. Inoltre maggiore è il grado di connessione tra progetti differenti, maggiore sarà il livello di *coupling* del sistema. È chiaro quindi che la connessione tra la metrica COU e la metrica CLO rappresenta un risultato non inatteso.

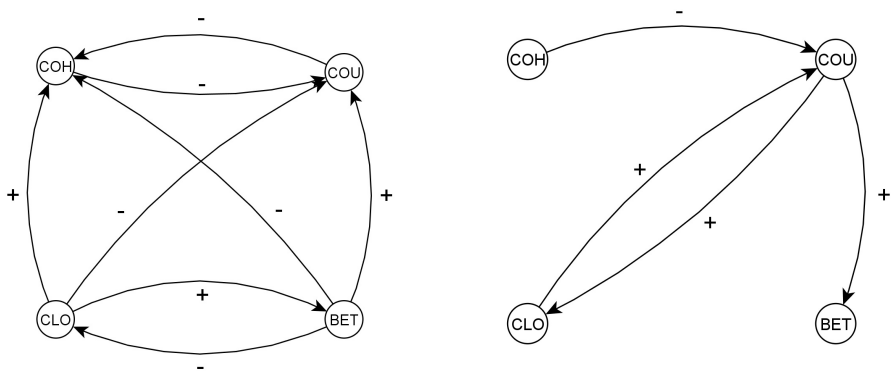
Le variazioni del livello di *coupling*, che risultano poco correlate con il resto del sistema, finiscono per influenzare fortemente le variazioni del livello di *closeness* nella rete sociale degli sviluppatori.

Alla luce di queste considerazioni non rappresenta una sorpresa che le metriche COU e CLO, nel modello ricavato dall'analisi del software Compiere, siano le uniche che si influenzano vicendevolmente.

Il minor controllo da parte del sistema sui valori di COH e, più in generale, dinamiche invertite rispetto a quanto osservato per il software ADempiere lasciano pensare che le differenze nella gestione del processo di sviluppo, da parte delle due comunità, abbia avuto un impatto fondamentale.

In Figura 4.5 sono messi a confronto i grafi ricavati dal Granger Causality Test che esprimono le mutue influenze esercitate tra le varie metriche considerate.

I due grafi mostrano dinamiche profondamente discordanti. Entrambi i



(a) ADempiere (b) Compiere

Figura 4.5: Grafi che mostrano le mutue influenze tra metriche legate alla qualità strutturale del software e metriche di centralità nelle reti sociali nei progetti ADempiere e Compiere

software aderiscono al paradigma dello sviluppo Open Source ma manifestano delle significative differenze nelle relazioni tra metriche legate alla Social Network e metriche legate alla struttura del software.

D'altra parte le differenze osservabili non sono imputabili a differenti caratteristiche funzionali del prodotto software sviluppato, perlomeno non in maniera rilevante. La data del fork tra i due progetti costituisce il momento in cui due diverse comunità di sviluppatori sono al lavoro sullo stesso prodotto. Le differenze che i due prodotti assumono da quel momento in poi sono comunque riferibili al cambiamento nelle caratteristiche di gestione del processo di sviluppo.

Pertanto le differenti dinamiche osservate sono una conseguenza dei modelli di Governance profondamente diversi adottati nei due progetti.

L'ipotesi **H1** afferma che i due modelli di Governance relativi ai paradigmi Community e Commercial Open Source, abbiano un impatto fondamentale sulla comunità di sviluppatori e siano in grado di condizionare il modo in cui struttura sociale della comunità e struttura del prodotto si influenzano vicendevolmente.

Per quanto osservato dalle analisi effettuate sui due software si può af-

fermare che l'ipotesi **H1**, per quanto riguarda i campioni di Commercial e Community Open Source osservati in questo lavoro, è verificata.

ADempiere, che nasce dal fork di Compiere, aderisce al paradigma di Community Open Source e quindi alle caratteristiche già citate di un'elevata condivisione delle informazioni e di una maggiore libertà di iniziativa da parte di ciascun membro della comunità.

Il grafo ricavato da ADempiere, ovvero dall'analisi delle metriche per tutte le 16 versioni considerate, mostra dinamiche di interazione tra Social Network degli sviluppatori e caratteristiche strutturali del software tipiche di un prodotto Open Source. Una forte dipendenza delle metriche COU e COH dalle metriche CLO e BET indica che le caratteristiche del modello di interazione della comunità di sviluppatori lasciano un'impronta sulla struttura del software.

L'ipotesi **H2A** afferma che all'interno di un progetto Community Open Source la struttura sociale che caratterizza la rete di collaborazioni interna ad una comunità di sviluppatori si riflette sulla struttura del software sviluppato.

Le analisi hanno mostrato una forte evidenza empirica dell'impatto delle metriche di centralità sulle metriche che esprimono la qualità strutturale del software. Per questo motivo l'ipotesi **H2A**, per quanto concerne il campione analizzato, può considerarsi verificata.

Compiere è certamente un software Open Source ma è catalogabile come orientato al paradigma del Commercial Open Source. Il forte controllo esercitato sul processo di sviluppo da parte di Compiere Inc., una minore condivisione delle informazioni e in generale un minor impatto della scelte della comunità sul processo di sviluppo, fa di Compiere un software solo nominalmente un progetto Open Source. L'evidenza empirica fornita dalle analisi effettuate fornisce una prova in tal senso.

Il controllo operato dalla società Compiere Inc. sul processo di sviluppo influisce, come è stato mostrato, sulle modalità di interazione tra gli sviluppatori che appartengono alla comunità di Compiere. La struttura della

Social Network ricavata dall'analisi delle collaborazioni interne tra gli sviluppatori del software Compiere ha caratteristiche diverse dalle analoghe ricavate dal progetto ADempiere.

Come effetto di queste differenze si osserva un minore impatto della struttura sociale della comunità su quella del software. Viceversa, le caratteristiche strutturali del software hanno un peso nel condizionare le modalità di interazione interne alla comunità.

L'ipotesi **H2B** afferma che nell'ambito del Commercial Open Source, la struttura del software influenza quella della rete sociale degli sviluppatori e viceversa.

Dato che i modelli estratti da Compiere mostrano un notevole impatto della metrica COU su entrambe le metriche di centralità, è possibile affermare che il livello *coupling* del software condiziona le caratteristiche della Social Network che rappresenta la comunità di sviluppatori. Tuttavia è possibile osservare anzhe la relazione inversa. La metrica CLO influenza, anche se debolmente la metrica COU.

Alla luce di queste considerazioni si può affermare che l'ipotesi **H2B** ottiene un riscontro positivo dalle analisi effettuate.

Approfondendo i risultati ottenuti dalla Vector Autoregression Analysis si possono indagare, oltre che le mutue influenze esercitate tra ciascuna metrica considerata, anche le modalità di tale interazione.

In particolare per quanto riguarda il software ADempiere, in cui l'impatto della Social Network sulle caratteristiche di qualità del software è evidente, è interessante studiare quale tipo di influenza è esercitata dalla prima sulle seconde.

Per le considerazioni che seguiranno è importante tenere presente che i valori di *coupling* e *cohesion*, all'interno di una sistema, hanno conseguenze ben precise sui costi connessi al processo di sviluppo e di mantenimento del software che sono state dimostrate da un gran numero di altri studi già citati. In particolare è desiderabile per un software avere livelli di *coupling* non elevati e livelli di *cohesion* più alti possibile.

Con questo presupposto è interessante osservare il segno dei coefficienti ottenuti attraverso la Vector Autoregression Analysis. Considerando il progetto ADempiere, si può osservare l'impatto della metrica CLO, che ricordiamo esprimere la capacità media dei nodi di mantenere una bassa distanza da tutti gli altri nodi all'interno di una rete, sulle metriche COU e COH.

La crescita della metrica CLO influisce sulla decrescita della metrica COU ($CLO_{t-1} \rightarrow COU_t$, Coef = -0.01). È importante sottolineare che il basso valore del coefficiente non è rilevante, esso esprime solo la differenza tra le due metriche in termini di valore assoluto. Di maggiore interesse è il segno di tale interazione, nella fattispecie che sia negativo. Questo significa che la crescita della *closeness* nella rete sociale degli sviluppatori di ADempiere ha conseguenze benefiche sul livello di *coupling* del sistema software sviluppato, che è desiderabile mantenere basso.

Per quanto riguarda l'influenza della metrica CLO sulla metrica COH si può notare che essa ha segno positivo ($CLO_{t-1} \rightarrow COH_t$, Coef=1.63). Questo significa che l'aumento del livello di *closeness* della rete sociale ha impatto positivo sulla coesione dei moduli che compongono il software, che è desiderabile mantenere elevata.

Dai risultati analizzati appare evidente che un incremento della metrica CLO porta a risultati benefici in termini di qualità strutturale del software. Un alto valore di *closeness*, nella rete che esprime l'interazione sociale della comunità di sviluppatori di ADempiere, è quindi desiderabile.

Spostando l'attenzione sull'interazione tra la metrica BET e le metriche di qualità del software si può notare una situazione differente. L'incremento del valore di BET ha segno positivo sul valore della metrica COU nella versione successiva ($BET_{t-1} \rightarrow COU_t$, Coef=0.06) e segno negativo sul corrispondente valore di COH ($BET_{t-1} \rightarrow COHt$, Coef=-5.83). La situazione è opposta a quella osservata per la metrica CLO.

La crescita del livello di *betweennes* all'interno della rete sociale degli sviluppatori causa quindi un peggioramento qualitativo della struttura del software.

Cercando di dare un'interpretazione a questi risultati occorre fare un discorso legato alla storia dei due software analizzati in questo lavoro. La metrica BET è legata al valor medio della *betweennes* di ogni singolo nodo che compone una rete. Nodi con un alto valore di *betweennes* sono nodi attraverso cui passano un gran numero di informazioni a causa dell'elevata capacità di tali nodi di mettere in relazione varie parti della rete. Per sua stessa natura questa caratteristica esprime una disomogeneità dei flussi informativi all'interno della rete. Questa disomogeneità dei flussi informativi è un chiaro segnale della disparità dei ruoli dei singoli nodi all'interno della rete. Valori molto alti o molto bassi di questa metrica sono entrambi indicatori di una non uniforme distribuzione delle informazioni all'interno della comunità che la rete sociale rappresenta.

La migliore condivisione delle informazioni e una maggiore spinta all'iniziativa nei confronti di tutti i membri della comunità è stata nei propositi degli sviluppatori di ADempiere fin dalla nascita del progetto. Questa caratteristica si rispecchia nei valori delle metriche relative alla struttura sociale misurate sul progetto. Una *closeness* media più alta e un minor livello di *betweennes* all'interno della rete sociale è una possibile risposta alle esigenze della comunità appena descritte. Il fatto che tali caratteristiche influenzino le metriche di qualità del software in maniera positiva è un possibile indizio di una corretta gestione del processo di sviluppo in relazione alle esigenze del progetto.

Tuttavia nell'ipotesi **H3A** si affermava che un valore alto delle metriche di centralità, all'interno della rete sociale che rappresenta le collaborazione tra sviluppatori di un progetto Community Open Source, avesse effetti positivi sulla qualità della struttura software.

Alla luce dei risultati ottenuti si può dire che tale ipotesi non è stata confermata. Tuttavia cercando di trarre informazione dall'analisi del modello ottenuto si possono fare proposizioni meno generali:

- **Proposizione P1**: Per quanto osservato dalle analisi effettuate sul campione Community Open Source, la crescita della *closeness centra-*

lity media dei nodi all'interno della rete sociale degli sviluppatori ha un impatto positivo sulla qualità della struttura software.

- **Proposizione P2**: Per quanto osservato dalle analisi effettuate sul campione Community Open Source, la crescita della *betweenness centrality* media dei nodi all'interno della rete sociale degli sviluppatori ha un impatto negativo sulla qualità della struttura software.

Queste due proposizioni servono a rendere più chiare le dinamiche di interazione tra metriche di centralità della rete sociale e struttura del software evidenziate dai risultati delle analisi.

Per le considerazioni fatte precedentemente riguardo alle similitudini osservate con le dinamiche tipiche dell'Open Source, i risultati ottenuti possono essere visti come una conferma empirica del forte impatto della rete sociale sulla struttura del software osservata in altri studi all'interno di questo tipo di paradigma.

Viceversa, dalle analisi effettuate sul software Compiere vediamo come il contributo delle metriche di centralità sulle metriche di qualità del software sia molto modesto.

Per quanto riguarda il software Compiere è possibile considerare il limitato apporto delle metriche di centralità all'evoluzione delle altre metriche del modello come una prova empirica a supporto dell'Ipotesi **H3B**. Tale ipotesi afferma che, per quanto riguarda il software Compiere, le caratteristiche della rete sociale degli sviluppatori non influiscono in modo rilevante sulla qualità strutturale del software.

L'evidenza empirica rivela che le metriche che riguardano la struttura del software non dipendono fortemente dalle caratteristiche della rete sociale bensì accade il contrario. Sono le metriche di qualità del software a influenzare la struttura della rete sociale.

Concludendo, dalle analisi effettuate sul campione di Commercial Open Source, abbiamo rilevato dinamiche simili a quelle riscontrate in altri studi per i software Closed Source.

Per completezza è interessante osservare che la metrica COU influenza le metriche di centralità CLO e BET con segno positivo. A questo risultato si può dare il significato che un maggiore accoppiamento tra i vari oggetti che compongono il sistema software è in grado di influenzare il numero di collaborazioni all'interno della comunità.

Questo tipo di dinamica può essere considerata una caratteristica peculiare della rete che rappresenta la comunità di Compiere. Per le considerazioni fatte nella Sezione 4.2 tale rete ha delle analogie con una rete Scale Free. Questo tipo di rete sociale è caratterizzata dalla forte presenza di *Structural Holes*, ovvero la formazione di comunità più piccole all'interno della comunità principale. Ad esempio se un software è fortemente modularizzato e la suddivisione dei lavori, all'interno della comunità, è basata su tale modularizzazione, ci si aspetta di veder sorgere tante comunità interne quanti sono i moduli del software. Tali comunità interne sono legate fra loro da nodi con un alto valore di *betweennes centrality*.

Data l'elevata presenza di *Structural Holes* all'interno delle rete sociali di questo tipo, un maggior accoppiamento tra i vari moduli del software, rappresentata da un aumento della metrica COU, ha l'effetto di avvicinare le varie comunità interne mediante la formazione di nuove connessioni.

Una maggiore vicinanza tra le comunità interne ha l'effetto di un innalzamento dei valori medi di centralità delle reti.

È difficile considerare tale avvicinamento come un effetto voluto. Tenendo conto delle considerazioni precedentemente effettuate sull'evoluzione della metrica COU, si può ipotizzare che queste dinamiche siano legate alla natura del progetto. Si può supporre che rappresentino una naturale deriva di un progetto in cui le caratteristiche in termini di interazione sociale della comunità di sviluppatori non siano in grado di esercitare un sufficiente controllo sulle caratteristiche strutturali del software.

La Vector Autoregression Analysis e il Granger Causality Test sono stati effettuati su 30 versioni del software Compiere. Di queste, 18 sono state rilasciate prima della nascita di ADempiere. Successivamente al fork, il

progetto ADempiere è stato generalmente più attivo arrivando a rilasciare, fino all'agosto 2011, 16 versioni del suo software ERP.

Essendo il numero di versioni di Compiere precedenti al fork comparabile con il numero di versioni di ADempiere rilasciate in seguito al fork si è ritenuto interessante provare a studiare l'evoluzione di un ipotetico progetto costituito congiuntamente dalle 18 versioni di Compiere pre-fork e dalle 16 di ADempiere che seguono la scissione.

Analizzando l'evoluzione di un progetto di questo tipo è possibile ricavare un modello che sia caratterizzato da due qualità particolari:

- Continuità dal punto di vista della struttura software
- Discontinuità dal punto di vista della gestione del processo di sviluppo

Queste due caratteristiche e il numero relativamente simile di versioni considerate per entrambi i software, permettono di studiare la capacità di ciascuna delle due modalità di sviluppo software di caratterizzare il prodotto finale.

Estraendo dalle analisi un modello di evoluzione congiunta di metriche legate alla Social Network e metriche legate alla qualità del software e conoscendo le modalità di interazione osservate in ciascun progetto separatamente, è possibile valutare quali dinamiche abbiano avuto una maggiore influenza. Se le dinamiche osservate in Compiere hanno un impatto maggiore, ci si aspetta di osservare un modello in cui la Social Network non influisce particolarmente sulla struttura software.

Se viceversa le dinamiche osservate per il software ADempiere hanno avuto un impatto maggiore, il modello estratto da questo nuovo ipotetico progetto mostrerà una maggiore capacità delle metriche legate alla Social Network di influire sulle metriche di qualità del software.

In Tabella 4.5 sono riportati i risultati della Vector Autoregression Analysis sulle versioni che costituiscono il progetto congiunto.

Si può osservare come, al contrario di quanto osservato per Compiere, non ci sia alcuna evidenza dell'impatto delle metriche di qualità della

Tabella 4.5: Risultati della Vector Autoregression Analysis applicata alle serie temporali ricavate dall'unione delle misurazioni effettuate sulle 18 versioni di Compiere pre-fork e sulle 16 di ADempiere post-fork.

	Coef.	**Std. Err.**	**z**	***p-value***	**[95% Conf.**	**Interval]**
$\mathbf{CLO_t}$						
CLO_{t-1}	1.07589	0.51905	2.07	0.038**	0.05858	2.09321
BET_{t-1}	-1.0225	1.40428	-0.73	0.467	-3.7749	1.7298
COU_{t-1}	-0.3357	15.6833	-0.02	0.983	-31.074	30.4029
COH_{t-1}	-0.1984	0.50406	-0.39	0.694	-1.1863	0.78957
cons	5.19767	14.2253	0.37	0.715	-22.683	33.0787
$\mathbf{BET_t}$						
CLO_{t-1}	0.03656	0.18717	0.2	0.845	-0.3303	0.40342
BET_{t-1}	0.64226	0.5064	1.27	0.205	-0.3503	1.63479
COU_{t-1}	-0.8534	5.65557	-0.15	0.88	-11.938	10.2313
COH_{t-1}	-0.0875	0.18177	-0.48	0.63	-0.4437	0.26879
cons	1.89254	5.1298	0.37	0.712	-8.1617	11.9468
$\mathbf{COU_t}$						
CLO_{t-1}	0.01633	0.00438	3.73	0.000*	0.00774	0.02491
BET_{t-1}	-0.047	0.01185	-3.97	0.000*	-0.0702	-0.0238
COU_{t-1}	0.39781	0.13234	3.01	0.003*	0.13844	0.65718
COH_{t-1}	-0.012	0.00425	-2.83	0.005*	-0.0204	-0.0037
cons	0.43328	0.12003	3.61	0.000*	0.19802	0.66854
$\mathbf{COH_t}$						
CLO_{t-1}	-0.1988	0.14711	-1.35	0.177	-0.4871	0.0895
BET_{t-1}	0.49925	0.39799	1.25	0.21	-0.2808	1.2793
COU_{t-1}	4.07262	4.44485	0.92	0.36	-4.6391	12.7844
COH_{t-1}	0.71996	0.14286	5.04	0.000*	0.43997	0.99996
cons	-7.7986	4.03164	-1.93	0.053**	-15.7	0.1033

*	**Forte evidenza empirica**
**	**Debole evidenza empirica**
***	**Evidenze empiriche minori**

struttura software COU e COH, sulle metriche relative alla centralità dei nodi nelle Social Network, CLO e BET. Questo indica che, statisticamente, l'impatto della gestione centralizzata del processo di sviluppo non ha inciso sufficientemente sul software da risultare nel modello estratto.

È possibile osservare che la metrica COU è fortemente influenzata da tutte le altre metriche considerate ($CLO_{t-1} \rightarrow COU_t$, Coef=0.01, *p-value* = 0.000; $BET_{t-1} \rightarrow COU_t$, Coef= -0.04, *p-value* = 0.000; $COU_{t-1} \rightarrow COU_t$, Coef= 0.39, *p-value*=0.003; $COH_{t-1} \rightarrow COU_t$, Coef= -0.01, *p-value*=0.005).

Tabella 4.6: Risultati del Granger Causality Test applicato alle serie temporali ricavate dall'unione delle misurazioni effettuate sulle 23 versioni di Compiere pre-fork e sulle 16 di ADempiere post-fork.

Equation	**Excluded**	**chi2**	df	*p-value*
CLO_t	BET_{t-1}	.53021	1	0.467
CLO_t	COU_{t-1}	.00046	1	0.983
CLO_t	COH_{t-1}	.15488	1	0.694
CLO_t	ALL	1.6167	3	0.656
BET_t	CLO_{t-1}	.03815	1	0.845
BET_t	COU_{t-1}	.02277	1	0.880
BET_t	COH_{t-1}	.2316	1	0.630
BET_t	ALL	.27878	3	0.964
COU_t	CLO_{t-1}	13.899	1	0.000*
COU_t	BET_{t-1}	15.746	1	0.000*
COU_t	COH_{t-1}	8.0145	1	0.005*
COU_t	ALL	18.464	3	0.000*
COH_t	CLO_{t-1}	1.8267	1	0.177
COH_t	BET_{t-1}	1.5736	1	0.210
COH_t	COU_{t-1}	.83953	1	0.360
COH_t	ALL	1.9015	3	0.593

*	**Forte evidenza empirica**
**	**Debole evidenza empirica**
***	**Evidenze empiriche minori**

Il Granger Causality Test eseguito sulle serie temporali composte dai valori delle metriche ricavati dalle 22 versioni di Compiere e dalle 16 di ADempiere ha fornito i risultati riassunti in Tabella 4.6.

Ancora una volta il test di Granger conferma i risultati ottenuti dalla Vector Autoregression Analysis identificando la metrica COU come quella più influenzata dai valori, nella versione di software precedente, di tutte le altre metriche.

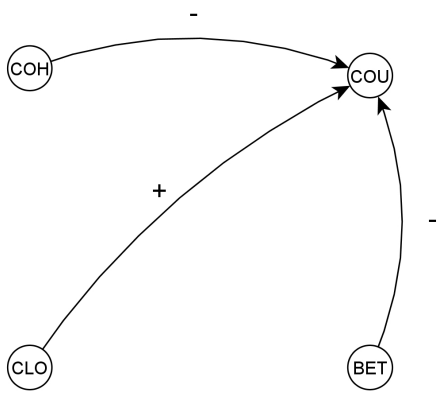


Figura 4.6: Grafico riassuntivo delle mutue influenze tra metriche ricavate dal Granger Causality Test sulle 18 versioni di Compiere pre-fork e sulle 16 di ADempiere post-fork

I risultati sono sintetizzati dal grafo in Figura 4.6. Questo risultato si allinea con quanto osservato dall'analisi del software ADempiere. L'impatto delle 16 versioni di ADempiere testate, per quanto riguarda l'interazione tra COU e il resto delle metriche, è decisamente più rilevante.

D'altra parte però la metrica COH, che influenza COU, non sembra influenzata considerevolmente dalle altre metriche. Questo risultato si allinea con quanto visto all'interno del progetto Compiere.

Il modello estratto in questa analisi conserva al proprio interno sia tratti osservati nel campione di Community Open Source, sia tratti osservati nel campione di Commercial Open Source.

In definitiva nessuna delle due parti della storia del progetto congiunto prevale.

Si può sostanzialmente affermare che l'impatto della gestione del processo di sviluppo tipica dei prodotti Community Open Source abbia il pregio di mantenere un forte controllo sul livello di *coupling* del sistema. Tuttavia la perdita di controllo sulla metrica COH, suggerisce che l'intero progetto eredita dalla prima parte della propria storia, quella riguardante una modalità di gestione di tipo Commercial, una forte incapacità di controllo sulla coesione degli oggetti appartenenti al sistema software.

4.4 Impatto dei diversi modelli di Governance sulla qualità del software

Appurato che i due metodi di Governance osservati nei progetti Compiere e ADempiere hanno un impatto fondamentalmente diverso sulla comunità di sviluppatori e di conseguenza sul software sviluppato, è interessante approfondire quali siano le conseguenze in termini di qualità del prodotto finale dei due tipi di gestione differenti.

L'analisi della variazioni delle metriche nel tempo permette di studiare, per un dato fenomeno, l'evoluzione delle proprietà ad esse correlate. In particolare dalle metriche che esprimono la qualità della struttura software si possono ricavare importanti informazioni sulle caratteristiche del progetto e sui mutamenti a cui il prodotto sviluppato è soggetto nel corso del processo di sviluppo.

Le metriche della *Suite CK*, di cui si è ampiamente parlato, si riferiscono alle proprietà dei singoli oggetti che compongono un sistema software. Ogni metrica esprime un differente aspetto della struttura dell'oggetto legato alla sua complessità.

È stato dimostrato il legame tra la complessità di un oggetto, espressa dalle metriche della *Suite CK*, e alcuni costi legati allo sviluppo software. L'elevata complessità di un oggetto ne mina la riusabilità all'interno di altre parti del codice, rende l'oggetto difficile da testare e rende complicati gli

interventi di manutenzione sul software in generale. L'effetto è un aumento dei costi di produzione e mantenimento del software.

Nel presente lavoro sono stati utilizzati i Function Points per rapportare il valore delle metriche, misurato su ciascun oggetto, al numero di funzionalità che questo implementa all'interno del software e quindi alla sua importanza relativa all'interno del sistema.

La media del valore di ciascuna metrica calcolata su una determinata versione di software esprime la complessità media degli oggetti che ne fanno parte. Questa caratteristica, pur non essendo un'esatta espressione della complessità dell'intero sistema, fornisce informazioni sulla qualità dei singoli oggetti che lo compongono.

Le seguenti figure (Figura 4.7, Figura 4.8, Figura 4.9, Figura 4.10, Figura 4.11, Figura 4.12) mostrano gli andamenti comparati delle metriche descritte nel Capitolo 3 per quanto riguarda i due software analizzati.

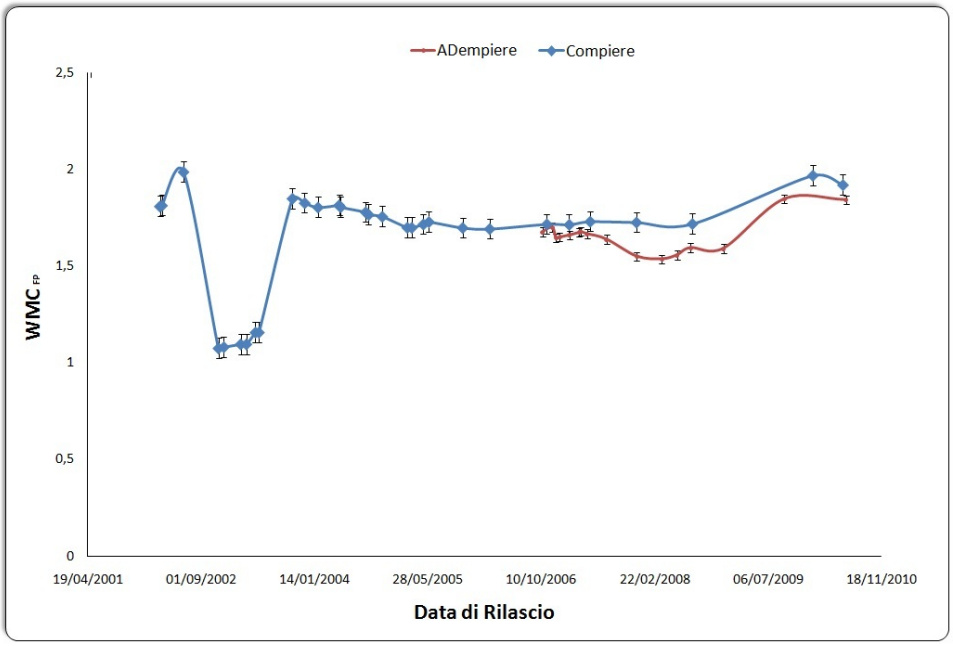


Figura 4.7: Andamento comparato del valore della metrica WMC_{FP} misurata su tutte le versioni di Compiere e ADempiere analizzate

Per tutte le metriche di qualità considerate, le prime versioni di ADem-

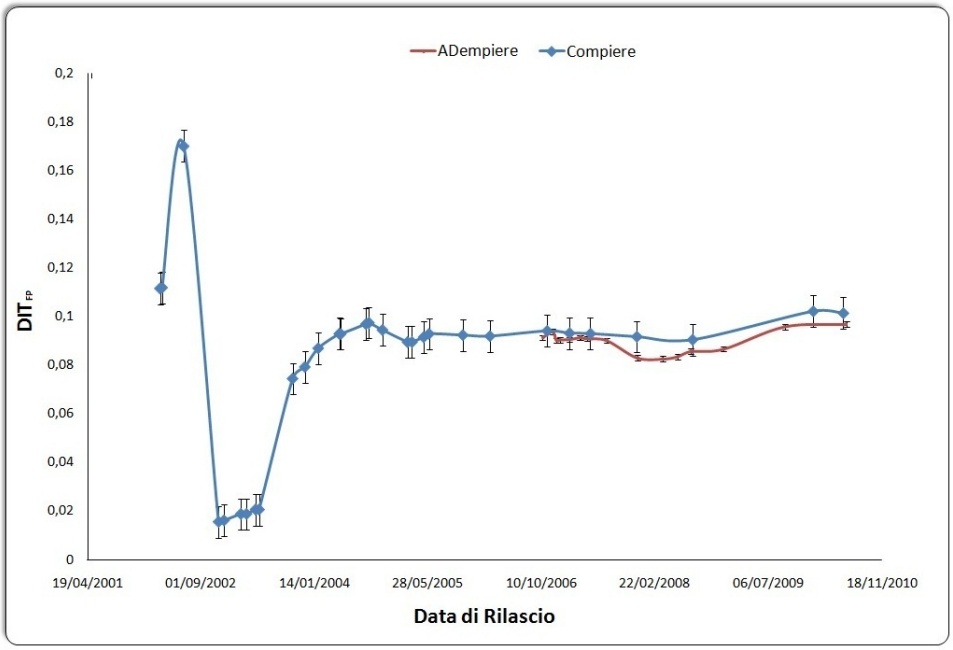


Figura 4.8: Andamento comparato del valore della metrica DIT_{FP} misurata su tutte le versioni di Compiere e ADempiere analizzate

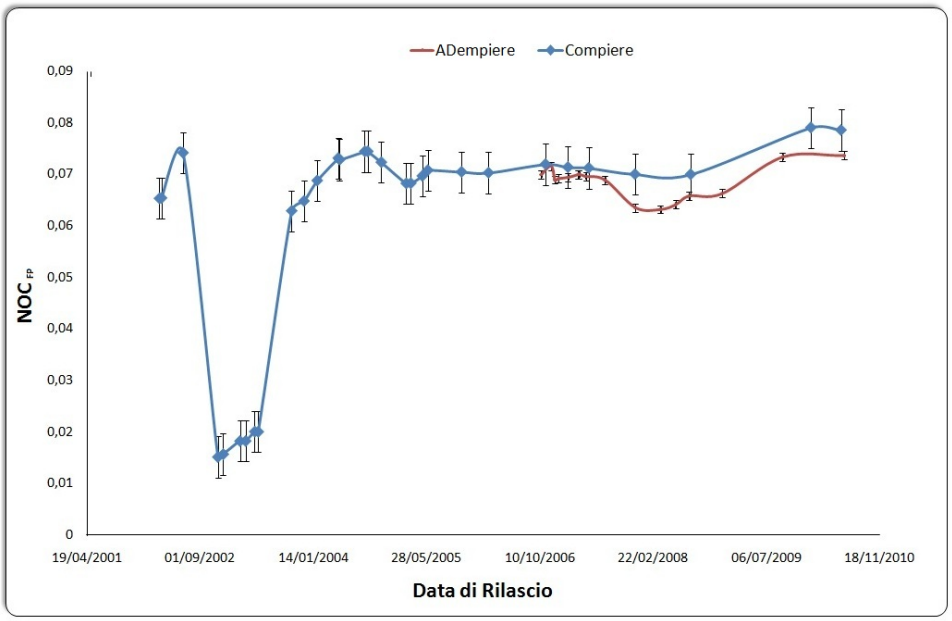


Figura 4.9: Andamento comparato del valore della metrica NOC_{FP} misurata su tutte le versioni di Compiere e ADempiere analizzate

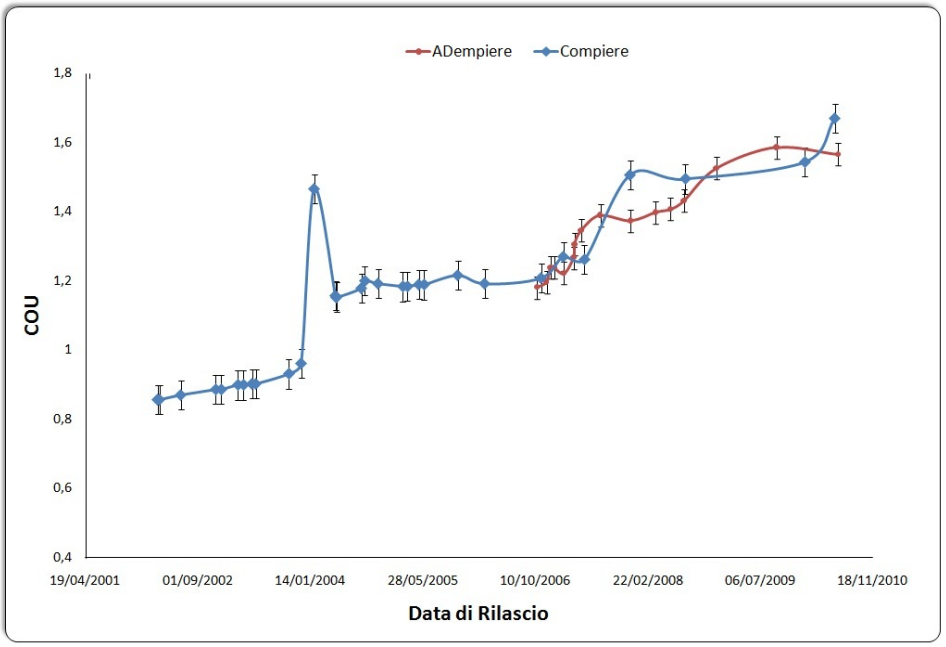


Figura 4.10: Andamento comparato del valore della metrica COU misurata su tutte le versioni di Compiere e ADempiere analizzate

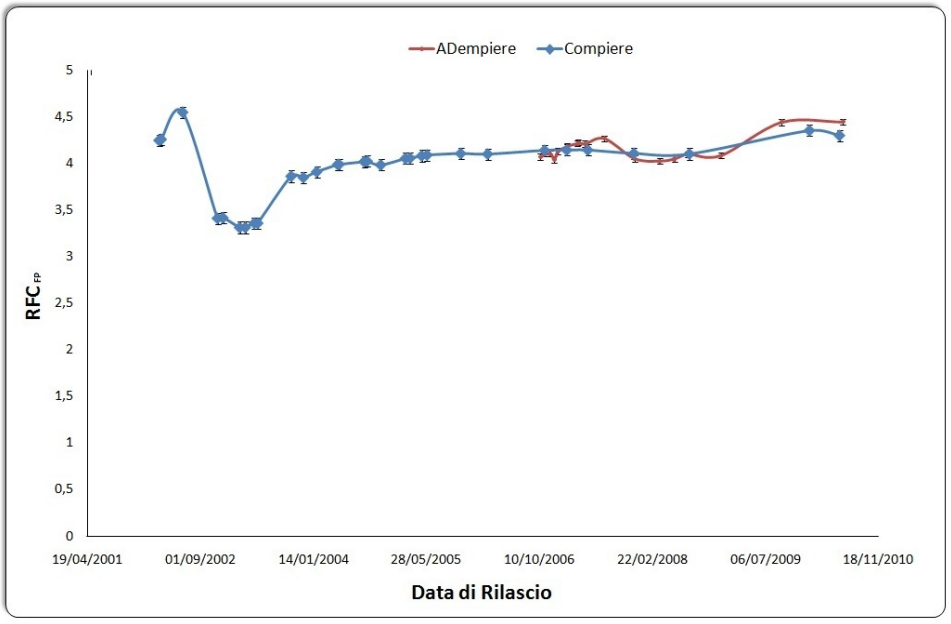


Figura 4.11: Andamento comparato del valore della metrica RFC_{FP} misurata su tutte le versioni di Compiere e ADempiere analizzate

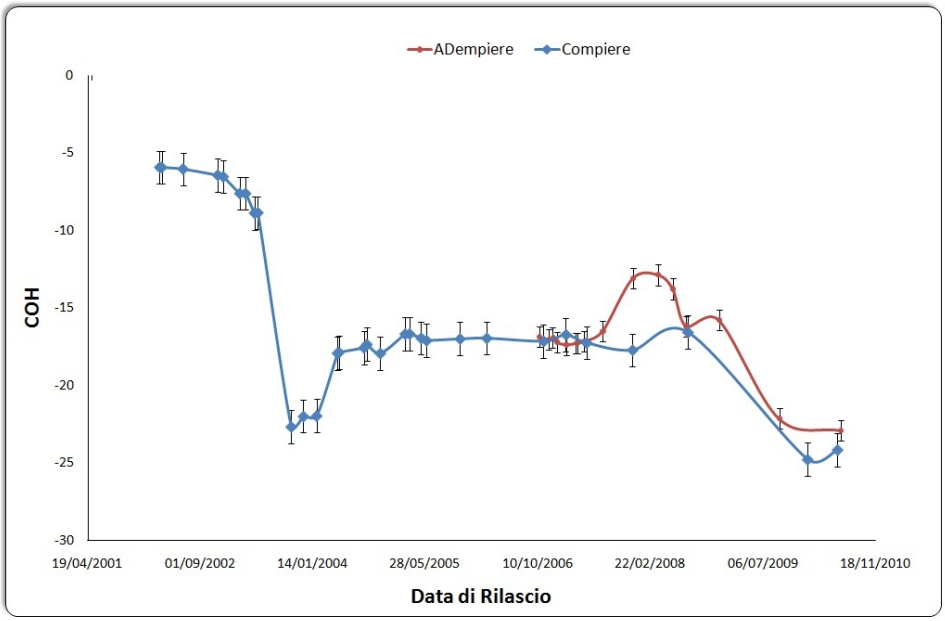


Figura 4.12: Andamento comparato del valore della metrica COH misurata su tutte le versioni di Compiere e ADempiere analizzate

piere rilasciate in seguito al fork mostrano valori che rappresentano una continuità con quelli misurati per Compiere. Questa caratteristica è importante ed indica che nelle prime versioni che seguono il fork i due software hanno una struttura molto simile.

La similitudine è essenziale per affermare che le differenze osservabili in seguito siano conseguenza delle diverse modalità di gestione del processo di sviluppo e non di altre variabili legate alla natura del software.

Tutte le metriche esprimono diversi aspetti della complessità del singolo oggetto. Per quanto riguarda WMC_{FP}, DIT_{FP}, NOC_{FP}, COU, DIT_{FP}, RFC_{FP}, a valori più bassi corrisponde generalmente una maggiore qualità del software. Un discorso a parte va fatto per la metrica COH che, per uniformità con l'uso tipico di questa metrica descritto in letteratura, ha segno opposto ed ad un valore maggiore, corrisponde una maggiore qualità dell'oggetto descritto.

Analizzando il grafico in Figura 4.7 si può notare come tutti i valori

della metrica WMC_{FP} misurati sulle versioni del software ADempiere che seguono il fork rimangono al di sotto di quelli misurati sul software Compiere, nello stesso periodo anche in caso di considerare il massimo errore standard.

Questa osservazione lascia intuire che i singoli oggetti che compongono il software ADempiere abbiano una qualità media migliore di quelli che fanno parte di Compiere.

Un discorso simile può essere fatto in riferimento al grafico in Figura 4.8. Per quanto riguarda DIT_{FP} però l'errore standard di alcuni valori sul software Compiere sono troppo ampi e non permettono di affermare con certezza che ADempiere presenti una qualità più alta di Compiere in tutte le versioni post-fork.

Il grafico in Figura 4.9 mostra una situazione analoga. La metrica DIT_{FP} indica che livelli di complessità media degli oggetti che compongono ADempiere è inferiore a quelli che fanno parte di Compiere. Anche in questo caso, l'alta variabilità dei valori misurati su Compiere introduce un errore standard che non permette di fare affermazioni certe sul confronti di alcune versioni.

Osservando il grafico in Figura 4.10 si nota, con il passare del tempo e quindi con il succedersi delle versioni, un aumento costante del valore di COU misurato su entrambi i software. Ad un incremento della metrica COU si associa un grado di accoppiamento maggiore fra gli oggetti che compongono il sistema.

Nella storia dei due software si alternano fasi in cui gli oggetti che compongono Compiere sembrano avere una qualità media migliore di quelli che fanno parte di ADempiere e fasi in cui avviene l'opposto. Questo andamento alternato non permettere di affermare che una delle due applicazioni abbia una qualità strutturale migliore dell'altra.

Va osservato però che per quanto riguarda i valori misurati sul software ADempiere si registra una maggiore stabilità della metrica. Nelle 16 versioni del software il livello di *coupling* sale da un valore di 1.181 ad una valore si 1.566, quindi con un differenziale di 0.375 e un incremento medio di 0.023

per versione.

Il valore della metrica COU più basso rilevato sul software Compiere è pari a 0.898. Il valore della stessa metrica nell'ultima versione del software analizzata è 1.668. In 25 versioni il valore è incrementato di 0.77 con un incremento medio di 0.03.

Considerando che un eccessivo accoppiamento tra gli oggetti che compongono il sistema software è una caratteristica indesiderabile, l'evoluzione della metrica COU osservata sul software ADempiere è indizio di maggior controllo sulla qualità della struttura software rispetto a quella di Compiere. Risultato in linea con quanto mostrato nella Sezione 4.4.

Il valore della metrica COU misurato sull'ultima versione di ADempiere è inferiore a quello misurato in Compiere anche considerando l'intervallo di confidenza evidenziato dall'errore standard.

Per quanto riguarda la metrica RFC_{FP}, il cui andamento può essere osservato in Figura 4.11 si può notare una situazione diversa. L'andamento della metrica nel tempo, per entrambi i software, presenta ancora delle fasi alternate ma in questo caso è ADempiere a mostrare, nella sua ultima versione, valori della metrica superiori e quindi una qualità media più bassa.

Analizzando l'andamento della metrica COH, in Figura 4.12 si può notare che, il valore iniziale della prima versione considerata della metrica COH per il software Compiere, è -7.627 mentre quello registrato sull'ultima versione è -24.164. Il differenziale è quindi -16.537 con decremento medio di 0.662.

Osservando invece l'evoluzione delle metriche sul software ADempiere, si registra un valore di coesione iniziale pari a -16.887 e un valore finale pari a -22.911. Il differenziale registrato è -6.024 con un decremento medio di 0.3765.

All'interno di un sistema software è desiderabile mantenere una maggiore coesione tra i vari moduli software. Un aumento del valore della metrica COH esprime un incremento della coesione all'interno dei singoli moduli che compongono il sistema. Si può concludere che, anche in questo caso, il

progetto ADempiere sembra avere un maggior controllo sulla qualità della struttura software.

L'ipotesi **H4** afferma che il modello di Governance di un progetto Community Open Source permetta di ottenere un software con qualità strutturale migliore di quello ottenuto attraverso il tipo di Governance adottato in un progetto Commercial Open Source.

Tuttavia dall'osservazione dei grafici non è possibile ottenere prove certe di una migliore qualità strutturale da parte di uno dei due software.

Disponendo dei valori delle metriche misurate sulle 16 versioni di ADempiere e sulle 8 di Compiere che seguono il fork, è possibile fare un'analisi sui dati dal punto di vista statistico.

Sono stati effettuati dei test di ipotesi per ciascuna delle metriche descritte.

Essendo le versioni di Compiere successive al fork solo 8, e non avendo informazioni riguardo la distribuzione dei nostri campioni, è stato effettuato un test di Wilcoxon-Mann-Whitney.

L'ipotesi nulla dei test effettuati è, per ciascuna metrica, che i valori misurati sul software Compiere e quelli misurati sul software ADempiere appartengano alla stessa popolazione.

Questo tipo di test è stato eseguito per fornire una valenza statistica alle misurazioni effettuate indicando che le differenze tra i valori misurati non siano dovute al caso. Escludendo l'ipotesi nulla, il test conferma che i campioni di dati ottenuti da Compiere siano estratti da una popolazione con caratteristiche (ad esempio media e varianza) diverse dalla popolazione da cui sono stati estratti i campioni di ADempiere.

Il test di Wilcoxon-Mann-Whitney fornisce anche il valore della probabilità di osservare valori più alti su una o sull'altra popolazione.

Gli interessanti risultati ottenuti grazie a questo test sono riassunti nella Tabella 4.7.

Il test di omogeneità utilizzato per confrontare le due diverse popolazioni mostra, per quanto riguarda la metrica WMC_{FP}, una forte evidenza

empirica contro l'ipotesi nulla e suggerisce che, mediamente, i valori di tali metrica per il software Compiere siano maggiori di quelli di ADempiere.

Questo risultato indica che con buona probabilità ($P(Comp\_WMC_{FP} > Ademp\_WMC_{FP}) = 0.9$) la qualità strutturale delle versioni di ADempiere che seguono il fork, per quanto riguarda gli aspetti legati alla complessità descritti dalla metrica WMC_{FP}, è genericamente migliore di quelle rilasciate da Compiere nello stesso periodo.

Risultati analoghi sono stati ottenuti dai test sulle metriche DIT_{FP} e NOC_{FP}. Per entrambi i test, infatti, l'ipotesi nulla viene scartata. Inoltre i rispettivi valori delle probabilità $P(Comp\_DIT_{FP} > Ademp\_DIT_{FP}) = 0.83$ e $P(Comp\_NOC_{FP} > Ademp\_NOC_{FP}) = 0.84$, assieme all'andamento qualitativo osservato sui grafici, fanno supporre che, anche per quanto riguarda gli aspetti descritti da DIT_{FP} e NOC_{FP}, il software ADempiere possegga una qualità strutturale migliore di Compiere nel periodo che segue il fork.

I test che riguardano le altre tre metriche considerate, COU_{FP}, RFC_{FP} e COH_{FP}, non forniscono una sufficiente evidenza empirica che permette di scartare l'ipotesi nulla.

Questo risultato è dovuto sia ad una maggiore variabilità dei valori

Tabella 4.7: Risultati del test di ipotesi di Wilcoxon-Mann-Whitney sulle medie dei valori delle metriche WMC_{FP}, DIT_{FP}, NOC_{FP}, COU, RFC_{FP} e COH

H0 : Comp_X = ADemp_X			
X	**z**	***p-value***	**P(Comp_X > Ademp_X)**
WMC_{FP}	3.123	0.0018	90%
DIT_{FP}	2.572	0.0101	83%
NOC_{FP}	2.694	0.0071	84%
COU	0.367	0.7133	55%
RFC_{FP}	0.796	0.426	60%
COH	-1.347	0.1779	33%

misurati, sia al numero relativamente basso di campioni considerati.

Sebbene i 16 campioni utilizzati di ADempiere siano sufficienti per un'analisi di questo tipo, le sole 8 versioni di Compiere che seguono il fork non permettono di disporre dei dati sufficienti a fornire una chiara evidenza empirica in casi di un determinato livello di indecisione.

Considerare nei test anche le versioni di Compiere che precedono il fork avrebbe probabilmente fornito dati sufficienti anche ai test che non hanno fornito risultati. Tuttavia tale analisi non avrebbe rispettato i presupposti di questo lavoro e sarebbe stata, quindi, poco significativa per la verifica delle ipotesi proposte.

Tre dei test effettuati suggeriscono, con buona significatività, che la qualità strutturale di ADempiere è migliore rispetto a quella di Compiere. Tuttavia gli altri test effettuati non forniscono alcuna conferma a riguardo.

Alla luce dei risultati dei test, l'ipotesi **H4** può considerarsi verificata solo parzialmente ma, dalle evidenze qualitative, ci sono forti indizi che indicano che ADempiere ha caratteristiche strutturali migliori rispetto a Compiere.

Il campione di Commercial Open Source considerato sembra mostrare una qualità inferiore, nelle versioni post-fork, della sua controparte Community Open Source.

Capitolo 5

Conclusioni e sviluppi futuri

"È il destino comune delle nuove verità cominciare come eresie e finire come superstizioni."

Thomas Henry Huxley

Come discusso nel Capitolo 2, il software è un prodotto particolare sia per la sua complessità sia per il fatto di essere il risultato di un'attività creativa incentrata sul lavoro umano. Dato che la produzione del software è generalmente un'attività svolta da più di un individuo, esige modalità di coordinamento che devono assolvere alla necessità di diversi individui di collaborare sullo stesso prodotto.

La difficoltà di automatizzare il processo di produzione software rende particolarmente importante lo studio delle caratteristiche di interazione sociale tra i vari attori coinvolti in tale processo. Pertanto, un gruppo di individui che collabora per la realizzazione di un prodotto software deve organizzarsi o essere organizzata secondo metodi ben precisi. Per questo motivo si parla di gestione del processo di sviluppo.

È stato dimostrato che, per quanto riguarda la produzione di software, la struttura della rete sociale che rappresenta le collaborazioni tra individui appartenenti a un team di sviluppo è in grado di influenzare la struttura del prodotto e viceversa.

La struttura del software assume quindi un ruolo fondamentale nell'analisi di tale dinamiche.

Nell'andare ad analizzare l'impatto della struttura della comunità di sviluppatori sulla qualità del software non si può trascurare il ruolo della sostanziale differenza tra i prodotti analizzati. Questo proprio perché il prodotto stesso ha il ruolo di meccanismo di condivisione delle informazioni e quindi influenza l'interazione tra gli sviluppatori.

Il caso preso in esame in questo lavoro ha permesso di studiare l'impatto di due differenti gestioni del processo di sviluppo su prodotti software funzionalmente equivalenti. Tale equivalenza è dimostrata non solo dalla storia dei software, che hanno un origine comune, ma anche da tutte le misurazioni effettuate relative alla loro struttura interna.

Dai risultati delle analisi eseguite, risulta che i due tipi di gestione osservati abbiano una capacità ben differente di incidere sulle dinamiche che descrivono le mutue influenze tra struttura sociale del team di sviluppo e struttura del software.

Il progetto originale è di tipo Commercial Open Source, ha scopi commerciali ed il suo processo di produzione è fortemente controllato da una società. Gli sviluppatori più importanti all'interno della comunità sono soci dell'azienda. L'informazione sullo stato dei lavori e sulle direzioni di progetto non è uniformemente distribuita tra i membri della comunità. Vi è un forte accentramento del potere decisionale.

I modelli estratti da questo progetto mostrano che le caratteristiche dell'interazione sociale tra gli sviluppatori non incidono in modo forte sul software in termini di qualità della struttura interna. Viceversa mostrano che è l'interazione sociale a dipendere fortemente dalle caratteristiche strutturali del prodotto.

Il progetto che nasce dal fork è di tipo Community Open Source e adotta una politica più orientata alla comunità. L'informazione circola liberamente, l'iniziativa non è contrastata e le figure di riferimento non sono imposte ma acquisiscono importanza grazie al grande livello di partecipazione.

I modelli di interazione estratti da questo prodotto mostrano come le caratteristiche di interazione tra sviluppatori influenzano decisamente la qualità della struttura del software. Questa può essere considerata come una maggiore capacità di controllo da parte del team di sviluppo sulla qualità del software sviluppato.

I due tipi di Governance analizzati, con le loro caratteristiche peculiari, hanno effettivamente un forte peso nelle dinamiche di mutua influenza tra struttura del team di sviluppo e struttura software.

I risultati ottenuti mostrano che una maggiore e uniforme condivisione delle informazioni tra gli sviluppatori appartenenti ad una comunità ha un effetto positivo sulla qualità del prodotto. Viceversa, un maggiore accentramento dei poteri decisionali, un'organizzazione più gerarchica e un flusso di informazioni più controllato, hanno l'effetto di un calo della qualità strutturale del software.

In riferimento a queste osservazioni è quindi possibile aspettarsi che una gestione del processo di sviluppo di tipo Community Open Source permetta di ottenere un software con qualità strutturale migliore di uno ottenuto attraverso la gestione osservata nel progetto Commercial Open Source.

Dai test effettuati sulle versioni rilasciate dopo la separazione dei due progetti tale ipotesi può considerarsi verificata solo parzialmente. Tuttavia attraverso analisi qualitative è possibile affermare che il software prodotto attraverso la gestione orientata alla comunità, tipica del Community Open Source, mostra livelli di qualità superiore a quello prodotto attraverso un'organizzazione più tipicamente gerarchica osservata nel Commercial Open Source.

Dai risultati ottenuti appare evidente che la condivisione delle informazioni sullo stato dei lavori sia un processo critico all'interno dell'attività di sviluppo software e rappresenti una sostanziale differenza tra i metodi di coordinamento tipicamente aziendali e quelli più legati al classico ambiente di sviluppo Open Source.

Un possibile sviluppo futuro di questo lavoro è rappresentato dallo stu-

dio dei canali di condivisione delle informazioni utilizzati in ambito Open Source. Studiando l'impatto del controllo dei canali di informazione sulle caratteristiche dell'interazione interna ad team di sviluppo, si può studiare indirettamente l'impatto della condivisione delle informazioni sulla qualità strutturale del software e quindi sui costi di sviluppo e mantenimento. Un tale studio potrebbe essere utile per integrare il quadro delle mutue influenze agenti tra struttura sociale che rappresenta le collaborazioni tra i membri di un team di sviluppo e struttura del software.

Bibliografia

[1] J. Rusnak A. MacCormack and C. Y. Baldwin. Exploring the structure of complex software designs: an empirical study of open source and proprietary code. *Management Science*, vol. 52 no. 7:pp. 1015–1030, 2008.

[2] R. Albert and A. L. Barabasi. Emergence of scaling in random networks. *Science*, vol. 286 no. 5439:pp. 509–12, 1999.

[3] T. J. Allen. *Managing the flow of technology.* MIT Press, Cambridge., 1998.

[4] M.R. Frean A.Potanin, J. Noble and R. Biddle. Scale-free geometry in oo programs commun. *ACM*, vol. 48 n. 5:99–103, 2005.

[5] L. J. Arthur. *Measuring programmer productivity and software quality.* John Wiley, 1985.

[6] T. S. Raghu B. Choi and A. Vinze. An empirical study of standards development for e-business: a social network perspective. *Proceedings of Annual Hawaii International Conference on SystemSciences*, 2006.

[7] L. L. Constantine B. Henderson-Sellers and I. M. Graham. Coupling and cohesion (towards a valid metrics suite for object-oriented analysis and design). *Object-Oriented Systems*, vol. 3:pp. 143–158, 1996.

[8] C. Y. Baldwin and K. B. Clark. *Design Rules (vol. 1): The power of modularity.* MIT Press, Cambridge, 2000.

[9] R. D. Banker and S. A. Slaughter. A study on the effects of software development practices on software maintenance cost. *Proceedings of International Conference on Software Maintenance*, 1996.

[10] A.L. Barabási and E. Bonabeau. Scale-free networks. *Scientific American*, vol. 288:pp. 60–69, 2003.

[11] V. R. Basili, L. Briand, S. Condon, Y. M. Kim, and W. L. Melo. Understanding and predicting the process of software maintenance releases. *Proceedings of 18th International Conference on Software Engineering (ICSE'96)*, vol. 1:pp. 464–474, 1996.

[12] G. Di Battista. Graph drawing: the aesthetics-complexity trade-off (invited lecture). *Operations Research Proceedings pages 92-94, 1999*, Springer-Verlag:pp.92–94, 1999.

[13] V. Del Bianco and L. Lavazza. An empirical assessment of function point-like object-oriented metrics. *Proceedings of 11th International Software Metrics Symposium (METRICS'05)*, vol. 1, 2005.

[14] A. B. Binkley and S. R. Scatch. Validation of the coupling dependency metric as a predictor of runtime filures and maintenance measures. *Proceedings of international Conference on Software Engineering,*, 1998.

[15] P. Bonacich. Factoring and weighting approaches to status scores and clique identification. *Journal of Mathematical Sociology*, vol.2:pp. 113–120, 1972.

[16] S. P. Borgatti. Centrality and aids. *Connections*, vol. 18, no. 1:pp. 112–115, 1995.

[17] R. S. Burt. A note on social capital and network content. *Social Networks*, vol. 19, no. 4:pp. 355–373, 1997.

[18] K. Fladmoe-Lindquist C. Jones, W. S. Hesterly and S. P. Borgatti. Professional service constellations: how strategies and capabilities influence collaborative stability and change. *Organization Science*, vol. 9 no. 3:pp. 396–410, 1998.

[19] W. S. Hesterly C. Jones and S. P. Borgatti. A general theory of network governance: exchange conditions and social mechanisms. *Academy of Management Review*, vol. 22 no. 4:pp. 911–945, 1997.

[20] E. Capra. Software design quality and development effort: an empirical study on the role of governance in open source projects. *PhDthesis, Politecnico di Milano, Dipartimento di Elettronica e Informazione*, 2008.

[21] E. Capra and A. I. Wasserman. Evaluating software engineering processes in commercial and community open source projects. *Proceedings of International Workshop on Emerging Trends in FLOSS Research and Development*, 2007.

[22] J. Y. Chen and J. F. Lu. A new metric for object oriented design. *Journal of information Systems and Software Technology*, vol. 35 no. 4:pp. 232–240, 1993.

[23] S. R. Chidamber and C. F. Kemerer. A metric suite for object-oriented design. *IEEE Transactions on Software Engineering*, vol. 20:pp. 476–493, 1994.

[24] M. E. Conway. How do committees invent? *Datamation*, vol. 14 no. 4:pp. 28–31, 1968.

[25] B. Coulange. *Software reuse*. Springer Verlag, London, UK, 1998.

[26] J. J. Elaman D. B. Walz and B. Curtis. Inside a software designteam: knowledge acquisition, sharing and integration. *Communications of the ACM*, vol. 36 no. 10:pp. 63–77, 1993.

[27] M. S. Krishnan D. E. Harter and S. A. Slaughter. Effects of process maturity on quality, cycle time and effort in software product development. *Management Science*, vol. 46 no. 4:pp. 451–466, 2000.

[28] S. A. Slaughter D. P. Darcy, C. F. Kemerer and J. E. Tomayko. The structural complexity of software: an experimental test. *IEEE Transactions on Software Engineering*, vol. 31 no. 11:pp. 982–995, 2005.

[29] E. W. Dijkstra. Letter to the editor: Goto statement considered harmful. *Communications of the ACM*, vol. 11 no. 3:pp. 147–148, 1968.

[30] R. Dorat and J. P. Delahaye. Networks of communities and evolution of cooperation. *I. J. Bifurcation and Chaos (IJBC)*, vol. 18 n. 7:2123–2131, 2008.

[31] F. Brito e Abreu. The mood metrics set. *In Proceedings of ECOOP Workshop on Metrics*, 1995.

[32] F. Merlo E. Capra, C. Francalanci and C. Rossi Lamastra. A survey on firms' participation in open source community projects. *Proceedings of International Conference on Open Source Systems*, 2009.

[33] D. K. Das E. G. Flamholtz and A. S. Tsui. Towars an integrative framework of organizational control. *Accounting, Organizations and Society*, vol. 10 no. 1:pp. 587–598, 1985.

[34] S. G. Eick and A. F. Karr. Visual scalability. Technical report, Technical Report Number 106, National Institute of Statistical Sciences (NISS), Research Triangle Park, NC 27709-4006, June, 2000.

[35] T. J. Emerson. A discriminant metric for module comprehension. *Proceedings of International Conference on Software Engineering*, 1984.

[36] P. Erdõs and A. Rényi. On random graphs. *Publ. Math.*, pages pp. 290–297, 1959.

[37] S. A. Slaughter F. Merlo and C. Francalanci. The co-evolutionof social networks and software structures: a study of open and closed source projects. *Proceedings of Academy of Management Annual Meeting*, 2009.

[38] B. Fitzgerald. The transformation of open source software. *Management Information Science Quarterly*, vol. 30 no. 2:pp. 587–598, 2006.

[39] L. C. Freeman. Centrality in social networks: a conceptual clarification. *Social Networks*, vol. 1 no. 3::pp. 215–239, 1979.

[40] S. Pinna G. Concas, M. Marchesi and N. Serra. Power-laws in a large object-oriented software system. *IEEE Trans. Software Eng.*, vol.33 n.10:pp. 687–708, 2007.

[41] C. Ghezzi, M. Jazayeri, and D. Mandrioli. *Fundamentals of Software Engineering.* Prentice Hall, 2003.

[42] G.Robles and J. M. Gonzales-Barahone. Geographic location of developers at sourceforge.net. *In Proceedings of International Workshop on Mining Software Repositories*, 2006.

[43] B. S. Gupta. *A critique of cohesion measures in the object-oriented paradigm.* PhD thesis, Michigan Technological University, 1997.

[44] R. Godin H. A. Sahraoui and T. Miceli. Can metrics help bridging the gap between the improvement of oo design quality and its automation? Technical report, Technical report, Montreal University, 2000.

[45] S. A. Boorman H. C. White and R. L. Brieger. Social structure from multiple networks. *American Journal of Sociology*, no. 81:pp. 730–780, 1976.

[46] M. H. Halstead. *Elements of software science*. Elsevier North-Holland, Amsterdam, 1977.

[47] R. Henderson and K. Clark. Architectural innovation. *Admin. Sci. Quarterly*, vol. 35 n. 1:pp. 9–30, 1990.

[48] S. Henry and D. Kafura. Software structure metrics based on information flow. *IEEE Transactions on Software Engineering*, vol. 7 no. 5:pp. 510–518,, 1981.

[49] M. Hitz and B. Montazeri. Chidamber and kemerer's metrics suite: a measurement theory perspective. *IEEE Transactions on Software Engineering*, vol. 22 no. 4:pp. 267–271, 1996.

[50] M. Hitz and B. Montazeri. Measuring coupling and cohesion in object oriented systems. Technical report, Institut Angewandte Informatik und Systemanalyse, University of Vienna, October 1995.

[51] J. Howison and K. Crowston. The social structure of free and open source software development. *First Monday*, vol. 10 no. 2, 2005.

[52] J. Howisone and K. Crowston. The social structure of free and open source software development. *First Monday*, vol. 10 no. 2, 2005.

[53] J. Roberts I. Hann and S. A. Slaughter. Mativations, participationand performance in open source software development. *Management Science*, vol. 52 no. 7:pp. 984–999, 2006.

[54] K. Ishikawa. *Introduction to quality control*. Union of Japanese Scientists and Engineers Press, 1989.

[55] M. Stinchcombe K. Hornik and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, vol. 2:359–366, 1989.

[56] T. Kamada and S.Kawai A. n algorithm for drawing general undirected graphs. *Information Processing Letters*, vol 31:pp. 7–15, 1989.

[57] C. F. Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, vol. 30 no. 5:pp. 416–429, 1988.

[58] L. J. Kirsch. The management of complex tasks in organizations: controlling the systems development process. *Organization Science*, vol. 7, no. 1:pp. 1–21, 1996.

[59] D. Knoke and S. Yang. *Social network analysis, Second Edition*. Sage Publications, 2008.

[60] S. Kusumoto, M. Imagawa, K. Inoue, S. Morimoto, K. Matsusita, and M. Tsuda. Function point measurement from Java programs. *Proceedings of International Conference on Software Engineering (ICSE'02)*, vol. 1:pp. 576–582, 2002.

[61] W. Li and S. Henry. Maintenance metrics for the object oriented paradigm. *Proceedings of International Software Metrics Symposium*, 1993.

[62] W. Li and S. Henry. Object-oriented metrics that predict maintainability. *The Journal of Systems and Software*, vol. 23:pp. 111–122, February 1993.

[63] G. C. Low and D. R. Jeffery. Function points in the estimation end evaluation of the software process. *IEEE Transactions of Software Engineering*, vol. 16:vol. 16, 1990.

[64] S. D. Eppinger M. E. Sosa and C. M. Rowles. Identifying modularand integrative systems and their impact on design team interactions. *Journal of Mechanical Design*, vol. 125 no. 2:pp. 240–252, 2003.

[65] J. Bant W. Opdyke M. Fowler, K. Beck and D. Roberts. *Refactoring: improving the design of existing code*. Addison-Wesley, 2001.

[66] T. Mayer and T. Hall. A critical analysis of current oo design metrics. Technical report, Centre for Systems and Software Engineering (CSSE), South Bank University, London, UK, 1999.

[67] T. J. McCabe. A complexity measure. *Proceedings of International Conference on Software Engineering*, 1976.

[68] W. Melo, L. Briand, and V. Basili. Measuring the impact of reuse on quality and productivity in object oriented systems. Technical report, University of Maryland, Department of Computer Science, 1995.

[69] K. A. Merchant. Progressing toward a theory of marketing control: a comment. *Journal of Marketing*, vol. 52:pp. 40–44, 1988.

[70] H.D. Mills. Strategic imperatives in software engineering education. *CSEE 1988*, pages 9–19, 1988.

[71] M. Hadass N. Ahituv and S. Neumann. A flexible approach to information systems development. *Management Information ScienceQuarterly*, vol.8 no. 2:pp. 69–78, 1984.

[72] M. E. J. Newman. A measure of betweenness centrality based on random walks. *Social Networks*, vol. 27, no. 1:pp. 39–54, 2005.

[73] E. I. Oviedo. Control flow, data flow and programmers complexity. *Proceedings of International Computer Software and Applications Conference*, 1980.

[74] M. Page-Jones. *Practical project management.* Dorset House Publishing Company, 1985.

[75] D. L. Parnas. On the criteria to be used in decomposing a system into modules. *Communications of the ACM*, vol. 15 no. 12:pp. 1053–1058, 1972.

[76] W. W. Powell. *Research in organizational behavior. Chapter: Neither-market nor hierarchy: Network forms of organization, pp.295-336.* JAI Press, Greenwich, 1990.

[77] H. Jeong R. Albert and A. L. Barabási. The diameter of the www. *Nature*, vol. 401 (6749):pp. 130–131, 1999.

[78] A. Belanger R. B. D'Agostino and R. B. J. D'Agostino. A suggestion for using powerful and informative tests of normality. *The American Statistician*, vol. 44 n.4:pp. 316–321, 1990.

[79] S. Counsell R. Harrison and R. Nithi. An evaluation of the mood set of object oriented software metrics. *IEEE Transactions on Software Engineering*, vol. 24 no. 6:pp. 491–496, 1998.

[80] S. J. Wayne R. T. Sparrowe, R. C. Liden and M. L. Kraimer. Social networks and the performance of individuals and groups. *The Academy of Management Journal*, vol. 44, no. 2:pp. 316–325, 2001.

[81] E. S. Raymond. *The cathedral and the bazaar*. First Monday, `http://www.firstmonday.org/issues/issue3_3/raymond/`, 2004.

[82] P. S. Renz. *Project governance: implementing corporate governance and business ethics in nonprofit organizations*. SpringerPhysica-Verlag, 2007.

[83] G. Robles. A software engineering approach to libre software. `http://www.opensourcejahrbuch.de/2004/pdfs/III-3-Robles.pdf`, 2004. WWW Document.

[84] L. H. Rosenberg. Applying and interpreting object oriented metrics. Technical report, NASA Software Assurance QUality Center, 1998.

[85] D. E. Harter S. A. Slaughter and M. S. Krishnan. Evaluating the cost of software quality. *Communications of the ACM*, vol. 41 no. 8:pp. 67–73, 1998.

[86] J. Roberts S. A. Slaughter and I. Hann. Communication networks in an open source software project. *Proceedings of International Conference on Open Source Systems*, 2006.

[87] M. Marchesi S. Focardi and G. Succi. A stochastic model of software maintenance and its implications on extreme programming processes. 2000.

[88] D. P. Darcy S. R. Chidamber and C. F. Kemerer. Managerial use of metrics for object oriented software: an exploratory analysis. *IEEE Transactions on Software Engineering*, vol. 24 no. 8:pp. 629–639, 1998.

[89] R. Ferrer S. Valverde and R.V. Solé. Scale free networks from optimal design. *Europhys. Lett.*, vol. 60:pp. 512–517, 2002.

[90] G. W. Sagers. The influence of network governance factors on success in open source software development projects. *Proceedingsof International Conference on Information Systems*, 2004.

[91] Anil Seth. Granger causality. *Scholarpedia*, vol. 2 n. 7:1667, 2007.

[92] S. Shankland. Open source produces best results. online, February 2003.

[93] R. C. Sharble and S. S. Cohen. The object oriented brewery: a comparison of two object oriented development methods. *ACMSIGSOFT Software Engineering Notes*, vol. 18 no. 2:pp. 60–73, 1993.

[94] H. A. Simon. The architecture of complexity. *Proc. American Philosophical Society*, vol. 106 n. 6:pp. 467–482, 1962.

[95] C. A. Sims. Macroeconomics and reality. *Econometrica*, vol. 48 n. 1:pp. 1–48, 1980.

[96] S. A. Snell. Control theory in strategic human resource management: the mediating effect of administrative information. *TheAcademy of Management Journal*, vol. 35, no. 2:pp. 297–327, 1992.

[97] K. A. Stephenson and M. Zelen. Rethinking centrality: methods and examples. *Social Networks*, vol. 11:pp. 1–37, 1989.

[98] R. Subramanyam and M. S. Krishnan. Empirical analysis of ck metric for object oriented design complexity: implications for software defects. *IEEE Transactions on Software Engineering*, vol. 29 no. 4:pp. 297–310, 2003.

[99] R. Ferenc T. Gyimothy and I. Siket. Empirical validation of object oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*, vol. 31 no. 10:pp. 897–910, 2005.

[100] Y. Tan and V. S. Mookerjee. Comparing uniform and flexible policies for software maintenance and replacement. *IEEE Transactions on Software Engineering*, vol. 31:pp. 238–255, 2005.

[101] D. Troy and S. Zweben. *Software engineering metrics I: measuresand validations.* McGraw-Hill, 1993.

[102] G. Myers W. Stevens and L. Constantine. Structured design. *IBM Systems Journal*, vol. 13:pp.115–139, 1974.

[103] S. Wasserman and K. Faust. *Social network analysis: methods and applications.* Cambridge University Press, 1994.

[104] D. A. Wheeler. Why open source software / free software? look at the numbers! `http://www.dwheeler.com/oss_fs_why.html`, 2003. WWW Document.

[105] O. E. Williamson and S. G. Winters. *The nature of the firm, SecondEdition.* Oxford University Press, 1993.

[106] W.Scacchi. Software development practices in open source communities: Acomparativecase study(. *Position Paper, at http://opensource.ucc.ie/icse2001/scacchi.pdf, accessed 2 February 2004.*, 2002.

[107] H. Andou Y. Kataoka, T. Imai and T. Fukaya. A quantitative evaluation of maintainability enhancement by refactoring. *Proceedings of International Conference on Software Maintenance*, 2002.

[108] L. Zhao and S. G. Elbaum. Quality assurance under the open source development model. *Journal of Systems and Software (JSS)*, vol. 66 n. 1:pp. 65–75, 2003.

[109] R. W. Zmud. Management of large software development efforts. *Management Information Science Quarterly*, vol.4 no. 2:pp. 44–55, 1980.