Qualità nel software: strategie a sorgente libero e aperto

Alessio Maria Braccini (abraccini@luiss.it)
Centro di Ricerca sui Sistemi Informativi – LUISS Guido Carli

Keywords: open source, free software, qualità

Abstract

Il software è una componente cruciale in molti prodotti: l'ampia diffusione delle tecnologie dell'informazione lo ha portato ad operare su elaboratori elettronici ed altri apparati che integrano al loro interno microprocessori. Questi dispositivi, definiti intelligenti arricchiscono il funzionamento di molti prodotti o servizi esistenti e contribuiscono a crearne di innovativi.

Nei confronti dello sviluppo del software esistono una serie di problematiche note da tempo, collegate alle caratteristiche di questo prodotto/servizio.

La natura ibrida (il software può essere considerato sia un prodotto che un servizio) e la peculiarità del suo ciclo di sviluppo (prevalentemente brain intensive, quindi difficilmente meccanizzabile o automatizzabile) sono alla base di tali problemi. E' facile notare un certo grado di difettosità nel software e, contrariamente a tutti gli altri prodotti, anziché essere evitato viene spesso tollerato o giustificato.

I difetti nel software si riscontrano sotto forma di malfunzionamenti, interruzioni di servizio, comportamenti inattesi o imprevedibili che rendono di fatto impossibile o molto difficoltosa l'erogazione del servizio che offre all'utente finale. Questi non si riscontrano solamente nelle applicazioni distribuite con finalità commerciale, ma anche in ambienti mission critical.

I sostenitori del Free/Open Source Software ritengono che i software realizzati e distribuiti con questa modalità siano competitivi, se non migliori, dal punto di vista qualitativo, rispetto ai loro alter ego commerciali. Oggetto di questo paper è l'analisi del modello organizzativo dello sviluppo e della letteratura alla base del software Free/Open

Source al fine di delineare quante più caratteristiche possibili possano contribuire a confermare o a confutare l'affermazione precedentemente formulata.

1. Introduzione

In seguito all'importanza dell'informazione come fattore produttivo nei prodotti e nei servizi attuali, il software è una componente cruciale in molti processi e attività. Il software è utilizzato per programmare il comportamento di microprocessori situati in computer, telefoni, televisori, centraline elettroniche ed altri apparati definiti intelligenti. L'importanza del software in un sistema informatico è oggi più rilevante rispetto a 30 anni fa: le economie di scala hanno continuamente contribuito ad incrementare la potenza e la capacità dell'hardware a parità costo. La criticità, in termini di costo, di una soluzione informatica si è quindi spostata dall'hardware al software (Pressman 1997).

Alcune caratteristiche del processo di sviluppo del software rendono gli aspetti qualitativi ad esso collegati più delicati rispetto ad altri prodotti o servizi. Il software può essere considerato sia come un servizio che come un prodotto: se viene distribuito assieme ad un altro bene si parla di software *embedded* e può essere assimilato ad un prodotto, al tempo stesso però, può essere impiegato per trarre vantaggio dalle potenzialità di un prodotto essendo così assimilato ad un servizio. In entrambi i casi il software è un componente intangibile e la sua immaterialità rende più difficile la misurazione di aspetti quantitativi che possono essere la base per la definizione di elementi quali standard di produttività o tassi di difettosità. In ultimo, il software è poi il risultato di un processo di sviluppo prevalentemente *brain intensive*, difficilmente meccanizzabile e automatizzabile.

Sebbene esistano metodologie e strumenti a supporto del processo di sviluppo software che prendono in considerazione questi aspetti, non è infrequente notare difetti e malfunzionamenti nei software di utilizzo quotidiano, ma anche in applicazioni *mission critical*. Il software è pieno di difetti che sono scoperti in continuazione, alcuni rimangono ignoti per un tempo variabile, forse per sempre (Viega, McGraw 2002).

I sostenitori del software Open Source ritengono che questa modalità organizzativa di sviluppo riesca a produrre software in grado di competere con le alternative commerciale e, in alcuni casi, di raggiungere risultati migliori. Queste affermazioni si basano sull'efficacia del processo di peer review attuato dai volontari che contribuiscono ad un progetto di sviluppo Open Source.

In questo paper, dopo aver riassunto le principali tematiche connesse alla qualità del software (§ 3) si descriverà l'organizzazione di un processo di sviluppo Open Source valutandone gli impatti qualitativi (§ 4.1, § 4.1.1) e infine si illustreranno le principali esperienze in termini di misurazioni di qualità effettuate su software Open Source (§ 4.2.1, § 4.2.2).

2. Termini e definizioni

In questa sezione viene fornita una definizione dei principali termini utilizzati.

Il termine Open Source viene spesso accomunato, in maniera erronea, al termine Free Software. Sebbene i due non facciano riferimento allo stesso fenomeno, in questo paper si parlerà solo di software Open Source, intendo però comprendere anche i software Free. Difatti l'aspetto ritenuto interessante alla luce delle considerazioni che saranno formulate riguarda la libera distribuzione del codice sorgente e che questa è garantita in entrambe i casi.

Circa la definizione di software adottata questa comprende le istruzioni (i programmi), le strutture dati che permettono al programma di elaborare adeguatamente l'informazione e la documentazione che descrive l'uso e l'operato del programma (Pressman, 1997).

3. Qualità nello sviluppo del software

Gli aspetti connessi alla qualità nel software risentono delle peculiarità di questo prodotto/servizio nonché delle caratteristiche del suo processo di sviluppo. Il tasso di difettosità di un prodotto (hardware), sulla base del tempo, ha un andamento decrescente che lo porta ad un livello minimo durante la sua vita utile di esercizio, questa termina al momento in cui il tasso di difettosità assume un andamento crescente che ne richiede la sostituzione con un componente alternativo o con uno nuovo dello stesso tipo (Illustrazione 1).

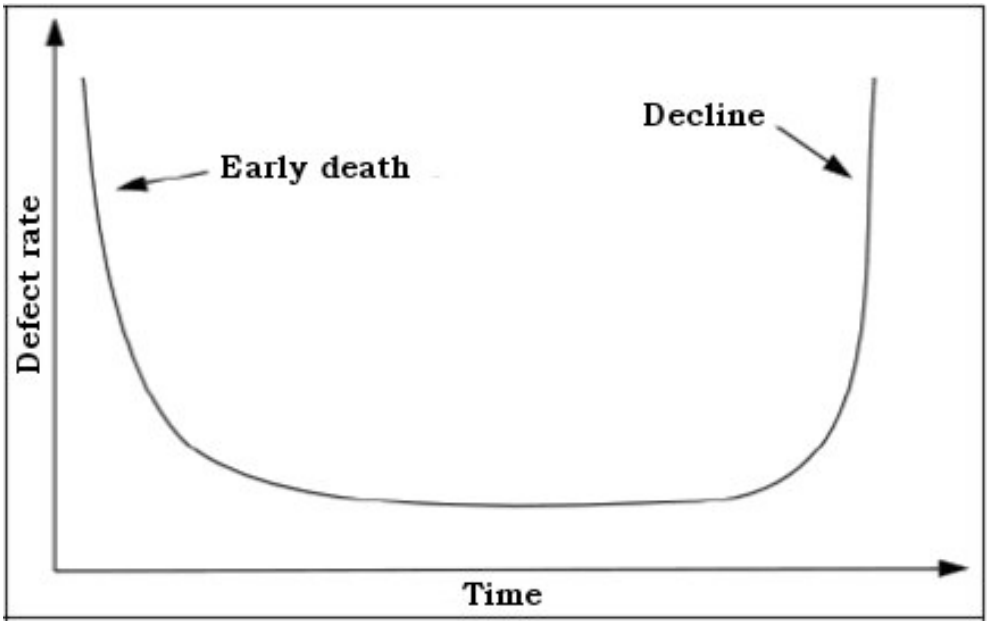

Illustrazione 1: Tasso di difettosità dell'hardware (Pressman 1997)

Nel caso del software non è possibile pensare ad una sostituzione perché per molti componenti software non esiste alternativa e tutte le copie del software sono identiche. In secondo luogo nel caso del software è necessario iterare la curva del tasso di difettosità per ogni intervento di manutenzione dal momento che l'output è necessariamente un software

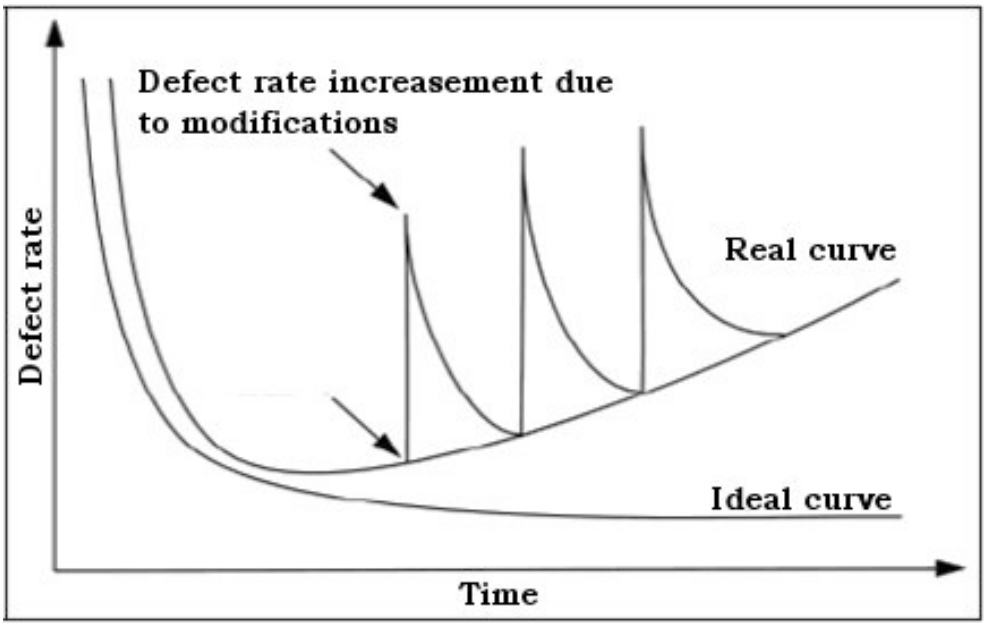

Illustrazione 2: Tasso di difettosità del software (Pressman 1997)

diverso dal precedente[1] (Illustrazione 2).

[1]Anche in seguito ad una modifica minimale non è possibile garantire che il tasso di difettosità sia rimasto inalterato o diminuito. Nell'estate del 1991 si verificò un guasto al software di segnalazione del sistema di telefonia locale della California. Il guasto ebbe origine in seguito ad un intervento di manutenzione che richiese la modifica di tre linee di codice e, in seguito alla quale, nessuno ritenne necessario sottoporre il sistema a nuovi test (Joch, nd).

Altri aspetti che hanno forti impatti qualitativi sul software riguardano la definizione del processo di sviluppo e il suo grado di maturità: esistono difatti standard internazionali utilizzabili per la definizione del processo e per l'assessment del suo grado di maturità (quali ad esempio il Capability Maturity Model del Software Engineering Institute). Da questo punto di vista il problema principale riguarda la necessità definire il processo, eventualmente ricorrendo all'utilizzo di questi standard come forma di guida, cercando di abbandonare, quanto più possibile, sistemi di produzione di tipo artigianale o artistico per avvicinarsi maggiormente a forme di tipo industriale.

4. Open Source e Qualità

La principale tematica connessa alla qualità nel processo di sviluppo del software Open Source riguarda l'efficenza delle *peer review* rese possibili dalla libera distribuzione del codice sorgente.

Nei paragrafi seguenti sarà descritta l'organizzazione di un processo di sviluppo Open Source cercando di evidenziarne gli impatti qualitativi (§ 4.1, 4.1.1) e le principali esperienze di misurazione di qualità in prodotti e processi Open Source (§ 4.2.1, § 4.2.2).

4.1. Organizzazione dello sviluppo

Le licenze di distribuzione del software Open Source consentono la circolazione del codice sorgente assieme al codice eseguibile ed offrono la possibilità agli utenti finali di effettuare modifiche, integrazioni, correzioni o altri interventi sui software. Questo aspetto non è neutrale nei confronti dell'organizzazione del processo di sviluppo.

I progetti di sviluppo Open Source si strutturano in modalità organizzative che consentono e favoriscono la partecipazione esterna da parte di utenti/sviluppatori che possono contribuire con correzioni, integrazioni, scrittura di documentazione, assistenza agli utenti o altre attività. Attorno ad un progetto di sviluppo si forma così una *comunità* di soggetti che sono al tempo stesso utenti e sviluppatori del software al quale fanno riferimento. L'insieme di singole comunità collegate ad uno specifico software instaurano rapporti di network formandone una più grande ed estesa definita come la comunità di sviluppo del software Open Source.

Ogni progetto di sviluppo viene gestito da un ristretto nucleo di persone denominato *core team* (Srinarayan, Vijayan, Balaji 2002 e von Krogh, Spaeth, Lakhani 2003) che ha il

compito di gestire lo sviluppo nella sua interezza, anche nel caso di assenza di contributi esterni. Questo core team si interfaccia poi con il resto della comunità e ne gestisce le relazioni.

I rapporti tra i membri che partecipano allo sviluppo vengono gestiti in maniera trasparente e danno luogo, alternativamente, a forme organizzative gerarchiche o forme più aperte (Garzelli, Galoppini 2003) Le dinamiche che regolano la partecipazione e la crescita all'interno dei progetti sono basate su un principio di natura motivazionale e meritocratica (Haruvy, Prasad, Sethi 2003). La meritocrazia è alla base della carriera all'interno di una comunità di sviluppo. La crescita viene spesso segnalata dall'acquisizione di maggiori ruoli e responsabilità: il livello massimo corrisponde all'acquisizione del diritto di commit nel server CVS, cioè la possibilità di apportare modifiche al software senza dover passare senza alcun filtro. La motivazione è invece il fondamento sul quale si basa la contribuzione esterna e che rende stabile e duratura nel tempo la collaborazione tra un gruppo di persone che sono distanti dal punto di vista temporale e territoriale e che hanno contatti solo di natura telematica. A volte all'interno dello sviluppo di un progetto Open Source intervengono anche aziende che possono svolgere sia il ruolo di contributori di spessore, sia veri e propri promotori del progetto.

Dal punto di vista organizzativo i progetti di sviluppo Open Source possono essere classificati in tre grandi macro-categorie (Garzelli, Galoppini 2003):

- Corporate;
- Ibridi;
- Volontari.

Nei progetti di sviluppo di tipo *Corporate* è forte la presenza di una azienda che gestisce il processo nella sua interezza, svolgendone il 100% delle attività. Progetti di questo tipo hanno solitamente una scarsa interazione con la comunità.

I progetti di tipo *Volontario* invece interagiscono fortemente con i membri della comunità e il processo di sviluppo viene gestito al 100% da risorse completamente volontarie che coordinano i loro sforzi nell'ambito di obiettivi e piani d'azione definiti dalla comunità stessa.

Una via intermedia è rappresentata dai progetti di natura *Ibrida* che prevedono la presenza di una azienda con il ruolo di promotrice dell'iniziativa di sviluppo, il cui processo però interessa anche la comunità, con la quale intrattiene delle relazioni. Lo sviluppo in questo caso avviene nell'ambito di obiettivi e piani decisi dall'azienda ma che

possono essere influenzati/influenzabili anche dalle richieste della comunità. L'assegnazione di ruoli e responsabilità, nonché delle attività da svolgere è quindi più flessibile rispetto all'organizzazione di tipo Corporate, dal momento che anche ai dipendenti dell'azienda deve essere lasciata la libertà di poter scegliere su quale attività focalizzarsi.

4.1.1. Impatti qualitativi dell'organizzazione di un processo di sviluppo Open Source

L'aspetto dell'organizzazione di un processo di sviluppo Open Source che potenzialmente può avere un maggiore impatto dal punto di vista della qualità del prodotto finito e che spesso viene citato proprio in tale contesto, consiste nella possibilità che gli utenti hanno di effettuare delle *peer review* sul software che viene sviluppato (Benkler 2002). La libera disponibilità del codice sorgente rende possibili interventi manutentivi anche a persone che non fanno stabilmente parte del *core team*, ma che sono solo dei contributori esterni. Dal punto di vista qualitativo questo può contribuire alla correzione dei difetti e degli errori individuati. Questo aspetto è citato da Raymond come la "legge di Linus" (Raymond 2001), sintetizzato nella frase "*given enough eyeballs all bugs are shallows*", ovvero, "dato un numero sufficiente di occhi (che guardano il codice) tutti i bug (errori) vengono a galla". Questa legge si basa sull'assunto che gli utenti esterni possono contribuire allo sviluppo segnalando la presenza di errori, ed eventualmente correggerli.

Tale comportamento è stato anche sintetizzato in un modello matematico il quale descrive (Challet, Le Du 2003) come il software a sorgente aperto è in grado di evolvere più rapidamente verso lo *zero defect state* anche se la qualità media dei programmatori che contribuiscono allo sviluppo è più bassa rispetto ad una soluzione di tipo tradizionale.

4.2. Misurazioni di qualità in software Open Source

Nei paragrafi seguenti saranno illustrate le principali misurazioni di qualità effettuate su software (§ 4.2.1) e processi (§ 4.2.2) Open Source.

4.2.1. Misure di prodotto

La tematica della qualità nel software a sorgente aperto è stata spesso presa in considerazione nell'ambito della ricerca. In particolare sono disponibili numerosi case study che cercano di fornire un giudizio sintetico sulla qualità di un software Open Source rapportando il numero dei difetti riscontrati a misure quali il numero di linee di codice.

Un primo gruppo di case study sono stati realizzati da Reasoning Inc, azienda che offre servizi di code inspection a supporto di controlli di qualità sul software, la quale ha analizzato con strumenti automatici alcuni software Open Source (Linux 2.4.19, il DBMS MySQL 4.0.16 e il server web leader di mercato Apache 2.1) paragonando il tasso di difettosità riscontrato con il relativo tasso stimato per gli alter ego commerciali.

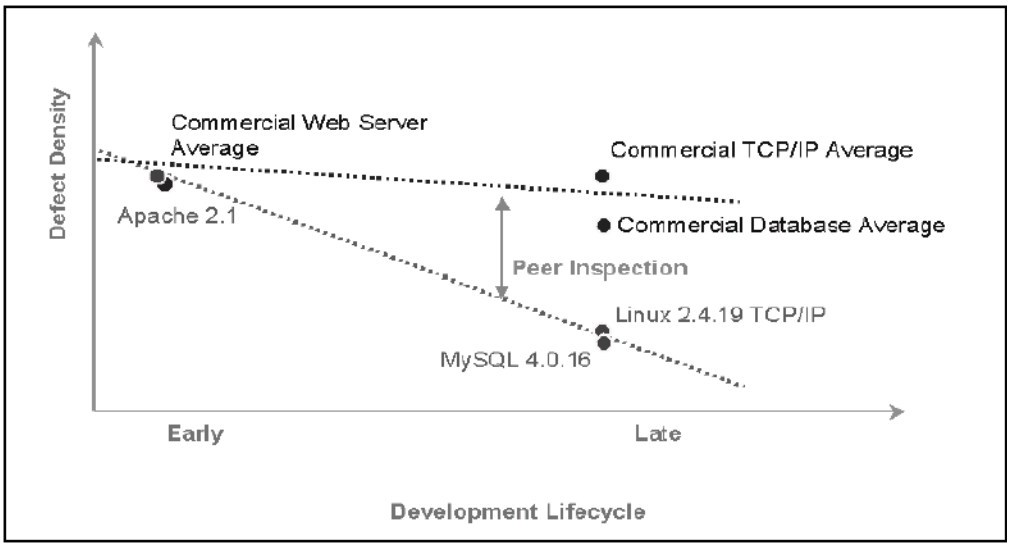

Illustrazione 3: Tasso di difettosità (num difetti/KLOC) e stadio del ciclo di vita (maturo/immaturo) (Reasoning, 2003)

Il risultato di queste misurazioni è riassunto dall'illustrazione n. 3 dalla quale si ricava che il tasso di difettosità di una applicazione Open Source che si trova nella prima fase del ciclo di vita è superiore rispetto alla media commerciale. Questo tasso denota però un trend decrescente diventando inferiore a quello del software commerciale per fasi avanzate del ciclo di vita.

Un altro case study è stato realizzato da Coverity Inc, società che offre servizi di supporto al controllo di qualità nello sviluppo del software che ha sottoposto a code inspection 32 applicazioni Open Source differenti, ricavando una media di 0,434 difetti per KLOC. Sempre sulla base di questi risultati, lo stack LAMP (ovvero l'insieme dei software

Linux, Apache, MySQL, Php, alla base di molti servizi web) ha un valore ancora inferiore (0.290) (Chelf 2006).

Nell'ambito di uno studio più completo volto ad analizzare il processo di sviluppo di Apache, è stata anche effettuata una misurazione del tasso di difettosità di questa applicazione (Mochus, Fielding, Herbsleb, 2000). La metodologia adottata in questo caso considera il numero di difetti per mille linee di codice aggiunte (contrariamente ai due casi precedenti che considerano solo il numero complessivo di linee di codice) e per migliaia di cambiamenti (cioè interventi correttivi sul codice). Il risultato di questa misurazione viene paragonato a quattro software commerciali indicati con le seguenti sigle:

- A: software di gestione di un dispositivo wireless;
- C, D ed E: applicazioni varie per la gestione di attività operative, amministrative e di mantenimento.

Measure	**Apache**	**A**	**C**	**D**	**E**
Post-release Defects/KLOCA	2,64	0,11	0,1	0,7	0,1
Post-release Defects/KDelta	40,8	4,3	14	28	10
Post-feature test Defects/KLOCA	2,64	*	5,7	6,0	6,9
Post-feature test Defects/KDelta	40,8	*	164	196	256

Tabella n. 1 – Tasso di difettosità di Apache (Mochus, Fielding, Herbsleb, 2000)

La misurazione prende in considerazione i difetti percepibili dagli utenti finali (post-release) e i difetti registrati prima dell'attività di test (post-feature). Nel caso di Apache, non esiste differenza tra queste misure (dal momento che essendo un software Open Source viene distribuito più frequentemente rispetto ad un software commerciale). In relazione al software commerciale, invece, la difettosità del codice prima del test è sensibilmente inferiore nel caso di Apache.

4.2.2 Misure di processo

Altro aspetto connesso alla misurazione di qualità in progetti Open Source riguarda il processo di sviluppo. La tematica relativa al processo di sviluppo è di particolare interesse nel caso del software Open Source perché, alla luce delle caratteristiche organizzative precedentemente descritte, viene spesso criticato per l'adozione di

metodologie produttive più vicine ad attività artistiche o artigianali piuttosto che industriali.

Il Capability Maturity Model misura in maniera sintetica la maturità di un processo di sviluppo software su una scala di cinque livelli: iniziale, ripetibile, definito, gestito, ottimale. Per ciascuno di questi livelli (ad eccezione del primo) sono definite delle attività chiave (Key Process Area) sulle quali l'organizzazione deve focalizzarsi al fine di migliorare la maturità del processo e giungere allo step successivo (Paulk, Curtis, Chirssis, Weber 1993). Il Capability Maturity Model è quindi sia uno strumento per la misurazione del grado di maturità di un processo di sviluppo, sia per il miglioramento dello stesso.

Incrociando i dati disponibili relativi al processo di sviluppo Open Source con i requisiti del Capability Maturity Model si riscontra un grado di maturità medio basso, ma con una elevata variabilità interna (Zhao, Elbaum 2003, Zhao, Elbaum 2000). In particolare viene segnalato che molte Key Process Area non sono applicabili ai processi di sviluppo Open Source (definizione e gestione del processo, sub-contracting e altre) e che molti processi di sviluppo Open Source coprono invece Key Process Area di livello elevato (gestione della configurazione, tracciabilità di progetto) che risultano implementati in maniera abbastanza diffusa.

5. Conclusioni

In merito alle caratteristiche qualitative del processo di sviluppo Open Source possono essere formulate alcune considerazioni. In primo luogo va fatto notare che la maggior parte dei progetti Open Source nasce sulla base di un bisogno non soddisfatto (Zhao 2003) e spesso con la diretta collaborazione degli utenti finali. Da un punto di vista concettuale, quindi, il software Open Source può avere una maggiore probabilità di raggiungere la customer satisfaction. In ottica contrattuale, l'integrazione del cliente finale nel processo di sviluppo può inoltre permettere di instaurare relazioni con un minore grado di conflittualità.

In secondo luogo la peer review può contribuire ad una effettiva riduzione del tasso di difettosità, ma la libera distribuzione del codice sorgente non garantisce che questo effettivamente avvenga. La contribuzione esterna non è un fenomeno automatico né scontato: un progetto Open Source potrebbe anche non stimolare l'interesse della comunità, in primo luogo, e la contribuzione esterna richiede una conoscenza tecnica

(dell'applicazione, del dominio applicativo e della tecnologia utilizzata) che non è alla portata di chiunque.

L'assenza di difetti nel codice sorgente e la relativa correzione sono solamente un aspetto della qualità del software. Prendendo in considerazione una definizione di qualità nel software, come quella fornita dallo standard ISO/IEC 9126, la presenza/assenza di difetti impatta solamente in maniera marginale. Il tasso di difettosità, misurato come rapporto tra il numero di difetti riscontrati ed una misura quantitativa del software (solitamente il numero di linee di codice) è quindi un indicatore di qualità solamente parziale.

Le considerazioni relative alla migliore qualità dei software Open Source formulata sulla base di ispezioni automatizzate effettuate sul codice sorgente andrebbero quindi riformulate sulla base di queste affermazioni. La densità di difetti in rapporto al numero di linee di codice risente delle diverse metodologie adottabili per la sua misurazione e del grado di maturità del ciclo di sviluppo. Questo rende tali misure tra di loro difficilmente comparabili: anche se fosse adottata la stessa metodologia su un campione selezionato di software Open Source, i diversi stati di ciclo di vita dei singoli progetti selezionati potrebbero influenzare il risultato finale, per cui questi aspetti devono essere presi in adeguata considerazione.

Se il tasso di difettosità non può essere adottato come indicatore assoluto di qualità la ricerca circa l'effettivo potenziale di un processo di sviluppo Open Source in ottica qualitativa dovrebbe adottare una prospettiva di customer satisfaction, anche al fine di poter valutare l'impatto e l'importanza di altri fattori (quali la presenza di standard de facto, l'esistenza di servizi di formazione e assistenza o altro) accessori al semplice codice sorgente.

Va precisato che le peculiarità di questa metodologia di sviluppo contribuiscono a definire un modello produttivo che si distingue profondamente dalle tradizionali modalità organizzative al punto da poter essere definito *sui generis*. Il fenomeno dell'Open Source è estremamente eterogeneo per cui al suo interno è possibile rintracciare realtà diverse tra di loro, sia dal punto di vista organizzativo che del software che queste realizzano. Notevole sembra essere quindi il contributo relativo alla classificazione dei progetti di sviluppo Open Source in tre macro-categorie.

Potrebbe essere interessante rileggere alcune esperienze di ricerca sinora maturate nell'ambito del software Open Source utilizzando questa classificazione come base per la

stratificazione del campione selezionato o per la selezione dei casi da analizzare. Dividendo tra progetti volontari, ibridi e corporate, alcune delle conclusioni dei casi esposti in questo paper potrebbero non essere più valide.

Alla luce di quest'ultimo aspetto, le considerazioni formulate circa la maturità del processo di sviluppo Open Source e l'inapplicabilità di alcuni dei requisiti del Capability Maturity Model appaiono quindi senza fondamento, dal momento che un processo di tipo ibrido o corporate può essere gestito in maniera più simile a quella di un processo tradizionale.

Bibliografia

Benkler Yochai, "Coase's Penguin, or, Linux and The Nature of the Firm", The Yale Law Journal 2002 vol. 112 n. 3 p. 369-446

Challet Damien, Le Du Yann, "Closed source versus open source in a model of software bug dynamics", 2003, http://arxiv.org/pdf/cond-mat/0306511

Chelf Ben – Coverity Inc, "Measuring software quality – A Study of Open Source Software", 2006, http://coverity.com/html/library.php

Elliot Margaret S., Scacchi Walt, "Free Software: A Case Study of Software Development in a Virtual Organizational Culture", Institute for Software Research – University of California (Irvine), 2003, http://www.ics.uci.edu/%7Ewscacchi/Papers/New/Elliott-Scacchi-GNUe-Study-Report.pdf

Garzelli Giamapolo, Galoppini Roberto, "Capability Coordination in Modular Organization: Voluntary FS/OSS Production and the Case of Debian GNU/Linux", 2003, http://econpapers.repec.org/paper/wpawuwpio/0312005.htm

Haruvy e., Prasad A., Sethi S. P., "Harvesting Altruism in Open-Source Software Development", Journal of Optimization Theory and Applications 2003 vol. 118 n. 2, p. 381-416

Joch Alan, "How software doesn't work", nd, http://www.byte.com/art/9512/sec6/art1.htm

Mockus Audris, Fielding Roy T., Herbsleb James, “A Case Study of Open Source Software Development: The Apache Server”, ICSE 2000 Proceedings

Paulk Mark C., Curtis Bill, Chrissis Mary Beth, Weber Charles V., “The Capability Maturity Model for Software”, 1993, IEEE Software n. 4

Pressman Roger S., “Software engineering. A practitioner's approach”, 1997, ISBN 0072853182

Raymond Eric S., “The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary”, 2001, O'Reilly

Reasoning Inc, “How Open Source and Commerical Software Compare: MySQL 4.0.16 – A qualitative analysis of database implementations in commercial software and in Mysql 4.0.16” - “How Open Source and Commercial Software Compare – A qualitative Analysis of TCP/IP implementations in commercial software and in the Linux Kernel” - “Automated software inspections: a new approach to increased software quality and productivity”, 2003

Srinarayan Sharma, Vijayan Sugumara, Balaji Rajagopalan, “A framewokr for creating hybrid-open source software communities”, Info Systems Journal 12, 2002, p. 7-25

Viega John, McGraw Gary, “Software sicuro”, Apogeo 2002

von Krogh Georg, Spaeth Sebastian, Lakhani R. Karim, “Community, joining, and specialization in open source software innovation: a case study”, Research Policy 32, p. 1217-1241, Elsevier

Zhao Luyin, Elbaum Sebastian, “A survey on software quality related activities in open source”, ACM SIGSOFT Software Engineering Notes 25 (3), p. 54-57

Zhao Luyin, Elbaum Sebastian, “Quality assurance under the open source development model”, The Journal of Systems and Software 66, 2003, p. 65 – 75