UNIVERSITA' DEGLI STUDI DELL'INSUBRIA

FACOLTA' DI SCIENZE MATEMATICHE, FISICHE E NATURALI

CORSO DI LAUREA IN SCIENZE E TECNOLOGIE DELL'INFORMAZIONE

TESI DI LAUREA

ANALISI DELLE CARATTERISTICHE DEI PROGETTI OPEN SOURCE

Relatore:
Prof. Sandro Morasca

Correlatore:
Dott. Davide Taibi

Laureando:

MICHELE PROTO

Anno Accademico 2006 - 2007

Sommario

Introduzione

Il filone di attività della tesi ricalca gli interessi comuni del progetto QualiPSo (Quality Platform for Open Source Software), al fine di contribuire alla crescita del mondo del software Open Source definendo e implementando tecnologie e procedure inerenti tali software.

Scopo del lavoro è stato quello di individuare ed analizzare i parametri fondamentali utilizzati per valutare i prodotti open source presenti all'interno del repository di SourceForge. Lo studio utilizza le caratteristiche degli attuali metodi di valutazione ampliate con altre mai introdotte in precedenza e che in seguito verranno inserite all'interno di metodi costruiti per automatizzare il processo di valutazione dei progetti open source. L'analisi viene eseguita inizialmente sull'intero database del repository, per passare successivamente ad osservare un intervallo di sei mesi, raggruppato per settimane (scelta arbitraria ma comunque rappresentativa di un andamento su un periodo sufficientemente lungo) al fine di comprendere quali variazioni hanno nel tempo le caratteristiche studiate.

Da queste fasi preliminari sono state estrapolate le caratteristiche dei prodotti studiati: bugs, patches, release, documentazione, sviluppatori, donazioni, nascita di nuovi progetti e download.

Lo studio inizialmente si è scontrato con l'impossibilità di poter utilizzare i dati di FLOSSmole (progetto open source che mette a disposizione alcuni dati del CVS Sourceforge) e per questo si è deciso di usare le informazioni contenute del database di Notre Dame (stessa finalità di FLOSSmole ma con dati più pertinenti).

L'analisi ha mostrato immediatamente una scelta infelice nella progettazione del meccanismo di tracking di SourceForge, perché di default i bugs, le patches, le release etc, se non espressamente specificato, sono assegnate al livello di priorità cinque (SourceForge utilizza nove livelli di priorità), facendo crescere di molto le componenti di questa categoria. Ma ciò, nel caso dei bugs, espone i prodotti ad un controllo "superficiale", perché gli sviluppatori, non potendo discriminare "a vista d'occhio" i problemi critici (quelli con i livelli più alti), potrebbero preferire continuare lo sviluppo del progetto piuttosto che controllare periodicamente la scoperta di problemi critici.

Lo studio dei bugs ha permesso di capire che questi ultimi sono distribuiti in ordine crescente rispetto all'aumentare della stabilità dei prodotti (SourceForge classifica i prodotti in cinque livelli

prealpha, alpha, beta, stabile e maturo) fino al livello stabile, mentre i progetti maturi presentano mediamente un numero di bugs molto inferiore. Di contro, analizzando le patches rilasciate, abbiano notato che i progetti ricevono anche loro un numero di miglioramenti crescenti finché non si raggiunge l'ultimo livello di stabilità (maturo) i quali non ricevono più un cospicuo numero di componenti aggiuntivi, ma, essendo ormai i prodotti consolidati, ricevono maggiormente solo le correzioni dei bugs scoperti.

Inoltre, continuando lo studio dei bugs, è emerso che gli sviluppatori tendono a risolvere prima i problemi (bugs) sui progetti con una stabilità maggiore (comportamento che ci aspettavamo) e successivamente si "dedicano" a sviluppare soluzioni per i software appena nati.

Un risultato opposto a quello che ci si aspettava è stato ottenuto studiando il tempo medio di soluzione dei bugs, infatti è stato evidenziato che non appena i progetti vengono dichiarati maturi, subiscono una sorta di abbandono e il tempo medio di soluzione aumenta in misura maggiore rispetto agli altri.

Un altro risultato inatteso si è avuto nello studio della documentazione, perché è emerso che all'interno di SourceForge è inserita poca documentazione. Infatti, in valore assoluto i documenti presenti confrontati con il numero di progetti gestiti da

SourceForge (circa 130.000 al momento in cui si è stato compiuto lo studio) rappresentano appena il 10%. C'è da dire (ad onor del vero), che un'analisi fatta fuori da SourceForge, ha mostrato che molte comunità forniscono la documentazione del proprio progetto direttamente dal proprio sito, usando il CVS espressamente per lo sviluppo ed il download.

Questo lavoro di tesi ci ha permesso di capire che la strada intrapresa è quella giusta, infatti una prima sensibilizzazione potrebbe essere fatta verso le comunità che sviluppano i sistemi di tracking al fine di evitare i problemi visti con quello di SourceForge (si ricordi infatti la cattiva gestione di assegnamento alla priorità intermedia). Inoltre i problemi riscontrati ci hanno permesso di comprendere quali caratteristiche migliorare per effettuare analisi più pertinenti, come ad esempio la centralizzazione dei dati in un unico database e l'accesso diretto allo stesso.

Struttura della Tesi

Nel primo capitolo, si introduce il mondo open source descrivendo, tramite le licenze e i metodi di distribuzione dei software, la filosofia che si trova dietro questo mondo e i problemi etici che ne hanno permesso la nascita. Viene inoltre sottolineata la differenza tra il software libero e l'open source, fino a giungere alla nascita dei CVS e dei repository, illustrandone le funzionalità e le potenzialità messe a disposizione.

Nel secondo capitolo, viene introdotto il problema della scelta dei software open source e come questi vengono valutati al fine di trovare quello più adatto alle proprie esigenze. Vengono inoltre presentati e analizzati i principali metodi di valutazione esistenti.

Il terzo capitolo mostra le caratteristiche che saranno analizzate suddividendole in tre grosse macro aree: la valutazione delle qualità esterne, l'attività delle comunità ed il supporto a breve termine.

Il quarto capitolo analizza in primis il CVS di SourceForge, e le sue peculiarità, successivamente si occupa dell'approccio ai dati del database di FLOSSmole prima e quelli di Notre Dame dopo, spiegando nel dettaglio il perché non è stato possibile utilizzare gli

elementi di FLOSS. Questi due database seppur con approcci diversi e ampiamente spiegati nel capitolo, mettono a disposizione delle comunità scientifiche i dati del CVS SourceForge. Vengono inoltre illustrati i problemi riscontrati anche con il database di Notre Dame e le sue incongruenze.

Nel quinto capitolo invece si mostrano i risultati dell'analisi di tutte le caratteristiche studiate, le scoperte ottenute e i risultati inattesi che si sono avuti. Lo studio viene presentato con l'ausilio di grafici che permettono di illustrare le differenze tra le varie componenti esaminate.

Il sesto capitolo è dedicato infine alle conclusioni a cui siamo giunti e agli sviluppi futuri di cui questo lavoro di tesi può essere il principio. Si riportano le appendici con le query sviluppate per reperire i dati, i valori di timestamp usati per l'analisi sul semestre e le tabelle del database di Notre Dame che posseggono i dati di questo lavoro.

Capitolo Primo

1.1. PREMESSA

In questi ultimi anni si sente sempre più parlare di software Open Source, software free, proprietario etc…, confondendo spesso il software e la sua licenza con il modo in cui viene distribuito. Non è difficile, infatti, sentir parlare di software Open Source allo stesso modo in cui si parla di software Freeware, ma questo è errato. Proviamo a fare un po' di chiarezza.

Il software nel 1984 viene definito dall'OMPI[1] come "*Espressione di un insieme organizzato e strutturato di istruzioni (o simboli) contenuti in qualsiasi forma o supporto (nastro, disco, film, circuito), capace direttamente o indirettamente di far eseguire o far ottenere una funzione, un compito o un risultato particolare per mezzo di un sistema di elaborazione elettronica dell'informazione*".

[1] OMPI Organizzazione Mondiale della Proprietà Intellettuale;

Ma il software è un'opera di ingegno e, pertanto, un bene immateriale, infatti "*il valore del software, anche sotto il profilo giuridico, non sta nel supporto su cui è registrato, ma nel suo contenuto ideativo e il pericolo che corre il suo autore non è tanto che gli sia sottratto quel supporto, ma che sia plagiato indebitamente da altri quel contenuto*".[2]

Quindi, a differenza di quanto asserito dall'OMPI che utilizza una definizione riduttiva, il software è un prodotto dell'ingegno e come tale deve essere tutelato. Per tale motivo ogni prodotto software viene, quindi, distribuito associandolo ad una licenza d'uso, ed è questa licenza che stabilisce come può essere utilizzato ed eventualmente ridistribuito.

1.2. CATEGORIE DI LICENZE

La licenza del Software Libero[3] rispetta le caratteristiche richieste dalla Free Software Foundation, la quale prevede che un software può essere utilizzato, studiato, adattato, migliorato e ridistribuito liberamente. La disponibilità del codice sorgente deve essere un

[2] R. Borruso, *La tutela giuridica del software. Diritto d'autore e brevettabilità*, Milano, 1999, pag. 3
[3] http://www.fsf.org/licensing/essays/free-sw.html

prerequisito. Questo tipo di licenza ha una maggiore valenza etica, in quanto valorizza la libertà del software e di chi lo produce. Questo non significa che produrre software libero implichi la non commercializzazione, infatti nulla vieta di scrivere software utilizzando questa licenza ma commercializzando il prodotto finito. Ovviamente nel corso della storia questa licenza è stata criticata ferocemente soprattutto dalle grandi industrie costruttrici di software che utilizzano software proprietario.

Simile è la licenza del software Open Source che soddisfa le condizioni della Open Source Definition[4] elaborata dall'Open Source Initiative[5]. Tale definizione si avvicina a quella espressa dalla Free Software Foundation, ma è stata pensata per motivi e destinatari diversi. Il software viene lasciato nella disponibilità degli sviluppatori che vorranno modificarlo, in modo tale che, con la collaborazione libera (ma nulla vieta che possa essere pagata), il prodotto finale possa raggiungere una complessità maggiore rispetto a quella che potrebbe ottenere un singolo gruppo di programmatori.

Altro tipo di licenza è quella del Software Copylefted[6], in cui le condizioni di distribuzione non permettono ad eventuali altri

[4] http://opensource.org/docs/osd
[5] http://opensource.org
[6] http://www.gnu.org/copyleft/copyleft.html

ridistributori di apporre restrizioni al momento della ridistribuzione o modifica del software. Ciò implica che ogni copia del software, anche se modificata, deve essere rilasciata come software libero e con la stessa licenza, quindi il software derivato deve essere ancora libero. La licenza Copylefted più diffusa è la GPL[7] (General Public License).

Ispirata a principi totalmente differenti è la licenza del Software Proprietario che concede all'utente solo ed esclusivamente l'utilizzo del prodotto, e sotto condizioni e restrizioni dettate dal proprietario stesso. Queste restrizioni possono essere di tipo tecnico rendendo pubblico solo il codice binario e non divulgando il codice sorgente. In questi casi la modifica del software risulta molto difficile, ed ottenibile solo con metodi di "reverse engineering" ovviamente vietati dalla licenza stessa. Altre restrizioni sono invece di tipo giuridico e consistono nella protezione del software con brevetti e copyright.

Un altro termine utilizzato per indicare solitamente il software proprietario è: Closed Source Software, che si pone in diretta contrapposizione con il concetto di "Open Source Software". Tale termine è utilizzato anche per indicare quel software distribuito con il codice sorgente, ma senza la possibilità di modificarlo.

[7] http://www.gnu.org/copyleft/gpl.html

Altro esempio di licenza è quella del Software semi-libero che attribuisce esclusivamente ai privati il permesso di usare, copiare, distribuire e modificare il software purché ciò avvenga senza scopo di lucro (incluse le versioni distribuite con modifiche). Infatti i vincoli maggiori vengono posti proprio sulla vendita.

Molto più generica è la definizione di Software non libero, in quanto tratta in generale quel software la cui licenza non soddisfa tutte le richieste della definizione di software libero.

Il software di pubblico dominio, invece, è quel software privo di proprietario e di copyright. E' un caso speciale di software libero senza permesso d'autore: chi lo utilizza gode della maggior parte dei diritti del software libero, ma non ha nessuna garanzia che tali diritti permangano, in quanto chiunque può appropriarsene e rendere proprietarie le versioni modificate, inserendo vincoli sulle modifiche e distribuendo il prodotto come software proprietario.

1.3. DISTRIBUZIONE DEL SOFTWARE

Dopo aver esaminato i diversi tipi di licenze è necessario scongiurare un'altra confusione che di solito si fa in questo campo: confondere la licenza che accompagna il tipo di software (quelle appena descritte es. open source vs software proprietario) e come questo venga distribuito. In generale la distribuzione è indipendente dalla libertà offerta dalla licenza.

Il *Software commerciale* è quello che viene venduto in qualsiasi modo. Il software commerciale per eccellenza è quello prodotto da Microsoft[8] che utilizza una licenza di tipo proprietario. Tuttavia, ci sono anche software open source che hanno lo stesso tipo di distribuzione come ad esempio EMACS scritto da Richard Matthew Stallman[9] (il fondatore della Free Software Foundation) e venduto dalla Free Software Foundation stessa.

Altro metodo di distribuzione è lo *Shareware*. Con tale termine si indica la distribuzione di quei programmi che possono essere utilizzati in prova per un determinato periodo di tempo, ma che necessitano del pagamento di una quota di registrazione per essere utilizzati oltre il tempo stabilito. Questo tipo di distribuzione viene usata sia per programmi di tipo proprietario, sia per quelli liberi, attraverso la richiesta di una donazione agli autori o al progetto.

[8] http://www.microsoft.com
[9] http://www.stallman.org/

A differenza della distribuzione di tipo shareware, i software distribuiti come *Adware* possono essere utilizzati senza richieste di pagamento, ma in cambio mostrano delle inserzioni pubblicitarie durante il funzionamento. Alcuni tipi di programmi sfruttano questa distribuzione per installare piccoli moduli che oltre a scaricare la pubblicità in linea con le abitudini degli utenti, inviano alla software house informazioni personali degli stessi. Questa distribuzione viene usata per software proprietario, anche perché la disponibilità dei codici sorgenti renderebbe facile l'individuazione di queste minacce alla privacy.

Infine, abbiamo il *Software gratuito* che, come indica la parola stessa, viene ceduto gratuitamente. Solitamente i programmi open source rientrano in questa categoria ed è proprio questa situazione che genera le incomprensioni tra licenza e tipo di distribuzione.

1.4. IL SOFTWARE LIBERO E L'OPEN SOURCE

Il software libero nasce nel mondo universitario americano negli anni '60 - '70 dove erano le stesse università che provvedevano a scrivere il proprio sistema operativo ed i programmi necessari,

dando la possibilità a tutti di modificarli a proprio piacimento. Successivamente, negli anni ‘80, le università hanno iniziato ad usare sistemi operativi proprietari, con codice aperto ma distribuiti con licenze di non divulgazione del codice, fatta esclusione per altri licenziatari. Questo limitava la collaborazione tra le università. In questo nuovo scenario l’AT&T aveva sviluppato un sistema Unix per le proprie centraline telefoniche che non poteva essere commercializzato a causa dell’allora legge americana. Decise, quindi, di regalarlo all’università di Berkeley (California), dando così inizio alla Berkeley Software Distribution[10] (BSD). In quello stesso periodo Richard Stallman decise di scrivere EMACS, un editor di testi molto versatile, rendendolo liberamente distribuibile. Ma i due episodi che portarono Stallman verso il “pensiero” del software libero, così come lo conosciamo adesso, furono l’impossibilità di modificare il codice sorgente di una stampante HP che non facesse inceppare la carta e l’uscita di produzione dei computer preferiti dai ricercatori con cui Stallman lavorava: il PDP-10 con il sistema operativo a “codice aperto”. I nuovi computer erano commercializzati senza il codice sorgente dei programmi. Questo portò allo scioglimento della comunità in cui lavorava e l'inizio del regno dei software proprietari. Per questo motivo nel

[10] http://opensource.org/licenses/bsd-license.php

1984, Stallman diede vita al progetto GNU[11] per la creazione di un sistema operativo interamente libero. GNU, acronimo ricorsivo di "GNU's Not Unix" (GNU non è Unix), è un sistema operativo basato su Unix, ma alla data attuale non è ancora stato sviluppato completamente. Qualche anno dopo la nascita di GNU, Linus Torvalds decise di scrivere un sistema operativo libero sulla base di Minix ed utilizzando numerosi programmi del progetto GNU, al fine di ottenere un sistema completo. Da qui la nascita di Linux, un kernel che ora possiede un alto grado di stabilità. A differenza di come lo conosciamo ora, a quel tempo era composto da vari pezzi difficili da utilizzare e da mettere insieme da chi non fosse avvezzo alla programmazione. Quindi, per ovviare a questi problemi, nacquero le "distribuzioni" Linux come le conosciamo oggi, ossia kernel, interfacce e programmi già configurati parzialmente che si adattano con i vari sistemi.

Dopo vari anni di sviluppo, il software libero era tecnicamente utilizzabile, ma ad esso era stato associato il concetto di non pagamento ostacolando la diffusione in ambito aziendale. Proprio per ovviare a tale problematica venne scritta nel 1998 la Open Source Definition, nella quale si definì questo tipo di software in modo mirato alle aziende.

[11] http://www.gnu.org

1.5. REPOSITORY e CVS

Ciò che ha contribuito alla crescita della comunità del software libero è stata anche la diffusione di internet. Internet ha permesso la collaborazione di più persone su progetti diversi anche fisicamente poste in parti opposte del globo. Questa crescita esponenziale dei progetti ha posto, quindi, le basi del problema su come reperire e sviluppare il codice a livello centrale, facendo in modo che gli sviluppatori non scrivessero parti uguali di codice per lo stesso progetto. Nascono così i repository ed i cvs.

Il CVS[12], acronimo di Concurrent Versions System (sistema di versioni concorrente), è un sistema di controllo delle versioni e rappresenta una struttura server che accumula i vari file di un progetto e tiene traccia della storia delle modifiche effettuate al codice. Gli sviluppatori pubblicano sul CVS le varie modifiche di un progetto, e quest'ultimo tenta di fondere le modifiche

[12] http://www.nongnu.org/cvs/

concorrenti. Può accadere, tuttavia, che questo sistema fallisca (ad esempio in caso di modifica concorrente della stessa riga di codice) in questo caso il CVS blocca la scrittura e avvisa il client del problema. Se, invece, le modifiche vanno a buon fine, il CVS le registra ed incrementa automaticamente le versioni, registrando tutte le operazioni nel file di log. Le altre operazioni che si possono fare sul CVS sono quelle di confronto delle varie versioni di un progetto, si può richiedere la storia completa dei cambiamenti o avere la fotografia di un progetto in una determinata data o da un certo numero di revisione. Si possono effettuare anche gli aggiornamenti dei file del proprio progetto con quelli residenti sul server CVS al fine di aggiornare le proprie copie con le nuove versioni presenti, in modo da evitare il "download continuo" di tutto il codice.

Il CVS originariamente venne sviluppato per gestire un singolo progetto, successivamente venne modificato al fine di amministrare più progetti diversi. Il "contenitore" di un CVS, che include i vari file di progetti differenti, viene denominato "repository".

Esempi rilevanti di CVS sono Freshmeat[13], Rubyforge[14], Sourceforge[15]. Questi siti forniscono molti servizi, tra i quali lo

[13] http://Freshmeat.net
[14] http://Rubyforge.org
[15] http://Sourceforge.net

spazio web, il server CVS, forum, mailing list, strumenti per la gestione dei bug, delle patch etc… Questi CVS, pur offrendo quasi gli stessi servizi differiscono principalmente per il tipo di software che gestiscono e per le informazioni aggiuntive che si riescono a reperire. Un esempio semplificato di queste differenze può essere quello tra Rubyforge e SourceKibitzer: il primo è dedicato ai progetti open source Ruby, mentre il secondo ai progetti open source Java. Inoltre Rubyforge fornisce poche informazioni sulla qualità dei propri progetti presentando solo delle semplici statistiche sui bugs o sulle release, mentre SourceKibitzer mostra molte più informazioni sul codice presente dei vari progetti, anche con l'ausilio di report visuali.

Il lavoro di tesi verterà proprio sullo studio e l'analisi dei progetti presenti all'interno di uno di questi repository: Sourceforge.net. che verrà ampiamente illustrato nel seguito.

Capitolo Secondo

2.1. PREMESSA

Negli ultimi anni vi è stata una crescita esponenziale dei progetti open source e, come appena detto, anche grazie alla nascita dei CVS. Provare a decifrare il motivo di questo trend non è affatto semplice, ma si possono elencare alcuni motivi esclusivamente tecnici che potrebbero tentare di giustificare questa evoluzione, senza scendere in considerazioni di tipo etico che sono comunque alla base della nascita dell'Open Source.

2.2. SOFTWARE LIBERO. IL PROBLEMA DELLA SCELTA

Uno dei motivi principali della scelta di un prodotto open source è legato al costo. Solitamente i prodotti open source sono distribuiti

gratuitamente e questo spinge ovviamente l'utente "home" a preferirli rispetto a quelli commerciali.

Un secondo motivo può essere la possibilità di avere e quindi poter modificare il codice sorgente. Questo porta inevitabilmente ad un continuo miglioramento del prodotto durante tutto il ciclo di vita del software, anche se è bene precisare che non tutti i progetti che nascono vengono poi seguiti e migliorati.

Un terzo motivo è l'architettura decentralizzata delle comunità di sviluppo, che rende il progresso del software più modulare e meglio gestibile durante le fasi di "debugging".

Infine il quarto motivo, che può essere definito quello più importante è la qualità dei prodotti Open Source. Infatti, la possibilità di accedere al codice e di centralizzare la distribuzione delle versioni porta al miglioramento continuo del prodotto. Per comprendere meglio questo punto possiamo fare un semplice esempio: una grossa software house, grande per quanto possa essere, non potrà mai utilizzare, provare o testare e quindi migliorare il proprio prodotto, rispetto a quanto può essere fatto da una immensa comunità globale che dà il suo contributo quotidiano su ogni singolo pezzo di codice.

Differente valutazione deve essere fatta per il mercato "business": quello aziendale. Il software libero, fino a poco tempo fa, non

veniva scelto da questa categoria di operatori (bisogna però prendere atto che anche il mercato business sta invertendo il trend). Per spiegare questa differenza bisogna introdurre criteri diversi da quelli descritti per il mercato “home”. Una azienda solitamente, dovendo utilizzare un software per produrre “ricchezza”, si trova di fronte al problema della scelta del software open, dovuto in primis, alla grande quantità di prodotti simili, a volte non più sostenuti dalla comunità che li ha creati.

A differenza di un prodotto commerciale, un prodotto open non assicura il supporto e la comunità che lo ha creato potrebbe anche abbandonare lo sviluppo qualora non dovesse incontrare “sostenitori” nel mercato globale. Quindi, è evidente che un’azienda si ritroverebbe con un software privo di qualsiasi sviluppo e miglioramento.

Un’azienda, nello scegliere un prodotto, deve assicurarsi anche che il software sia longevo e, quindi, che la comunità di sviluppo sia disponibile a correggere eventuali bugs ed a rilasciare le release che migliorino e adattino il programma all’evoluzione dell’azienda stessa.

Altra valutazione da fare riguarda la volatilità dei progetti open source. Alcune comunità preferiscono sviluppare e rilasciare molti aggiornamenti ma di piccola entità, ma questo porta ad una

scomodità dell'utente finale che si ritrova a doversi scontrare spesso con piccoli cambiamenti, mentre, a livello aziendale, sarebbe preferibile avere progetti stabili e che non vengano stravolti assiduamente.

Infine, un ultimo criterio da tenere in considerazione è quello sulla qualità del codice. Infatti, i progetti appena nati (tecnicamente prealpha) presentano una qualità del codice molto bassa.

Detto questo, è chiaro che, prima di adottare un software open, un'azienda deve valutare tutti questi parametri al fine di evitare i problemi discussi. E' bene precisare però che moltissimi progetti open, privi di queste problematiche, hanno un livello di qualità sia in termini di stabilità che di utilizzabilità maggiori dei "rivali" commerciali.

In questo scenario, quindi, si introduce la problematica su come valutare sperimentalmente i software open source.

Come visto in precedenza, le aziende, per scegliere un prodotto software, libero, devono avvalersi del know-how dei propri dipendenti del ramo ICT, per fare scelte oculate ed evitare di incappare in uno dei problemi detti in precedenza. Questo perché non si hanno dei metodi standardizzati per compiere queste scelte; infatti in quest'ultimo periodo si stanno facendo passi avanti in

questa direzione, gettando le basi sui metodi di valutazione del software libero.

2.3. LA VALUTAZIONE DEI PRODOTTI OPEN SOURCE

Con la proliferazione dei prodotti open source e la necessità di doverne scegliere uno piuttosto che un altro, ci si è trovati davanti al problema di doverli valutare al fine di dare un giudizio e poterli così classificare.

I principali metodi di valutazione già esistenti sono: **OSMM, OPEN BRR, QSOS** e l'appena nato **OPEN BQR**. Ci sono anche altri metodi, ma che non vengono esposti in quanto sono di minore diffusione.

2.4. IL METODO OSMM

Il metodo OSMM[16] (Open Source Maturity Model), sviluppato da Capgemini, utilizza sette passi per cercare di determinare quale prodotto risponda meglio alle esigenze di un eventuale cliente. Per

[16] http://www.seriouslyopen.org/

Capgemini uno dei fattori più importanti è quello della maturità del prodotto, infatti quanto più un progetto è maturo più è stabile e solido. Altra considerazione importante di OSMM è che non permette di paragonare i prodotti commerciali con quelli open, è bene però precisare che questa non è una limitazione del metodo, ma una ferma convinzione del suo autore. Tornando alla descrizione del metodo, inizialmente dicevamo che è composto da sette passi, i quali sono:

1. Ricerca dei prodotti;
2. Calcolo del punteggio dello specifico prodotto, tramite Product Indicator;
3. Calcolo del punteggio dello specifico prodotto, tramite Application Indicator;
4. Intervista con il cliente volta all'attribuzione dei pesi;
5. Applicazione dei pesi da parte del cliente insieme a Capgemini;
6. Determinazione del punteggio finale e selezione del prodotto;
7. Valutazione.

Nei passi due e tre vi è il calcolo dei punteggi tramite il "Product Indicator" e "Application Indicator". Questi due indicatori rappresentano la parte "obiettiva" della valutazione e sono le unità di misura del prodotto in quanto formati a loro volta da altri indicatori ai quali viene applicato un punteggio da uno a cinque.

Le caratteristiche del Product Indicator sono:

- **Product**
  - Age
  - Licensing
  - Human Hierarchies
  - Selling Points
  - Developer Comunity
- **Integration**
  - Modularity
  - Collaboration with other products
  - Standards
- **Use**
  - Support
  - Ease of deployment
- **Acceptance**
  - User Community
  - Market penetration

Mentre quelle dell'Application Indicator sono:

- Usability
- Interfacing
- Performance
- Reliability
- Security

- Proven Tecnology
- Vendor Indipendence
- Platform Indipendence
- Support
- Reporting
- Administration
- Advice
- Training
- Staffing
- Implememtation

Come si può facilmente notare, una valutazione passa tramite ventisette parametri diversi ai quali va applicato un punteggio maggiore in base alla maturità del prodotto. Inoltre, OSMM lavora in modo stretto al cliente che dovrebbe utilizzare il prodotto "vincente". Quindi con OSMM il cliente diventa parte attiva del processo di valutazione. Nella scelta si possono compiere due valutazioni differenti, una è quella di scegliere il prodotto che ha totalizzato il punteggio maggiore, l'altra è quella di scartare i prodotti che hanno totalizzato il punteggio minimo rispetto alle caratteristiche alle quali si era attribuito il peso maggiore. Il metodo ha il grande merito di essere veloce, ma rischia di essere poco obiettivo.

2.5. OPEN BRR

OPEN BRR[17] acronimo di Business Reading Rating Model, venne sviluppato da SpikeSource e Intel Corporation ed è oggi uno dei metodi più diffusi di valutazione del software Open Source. E' caratterizzato da quattro fasi differenti:

- Quick Assessment;
- Target Usage Assesment;
- Data Collection & Processing;
- Data Translation.

Nella prima fase i prodotti vengono scremati decidendo quali prendere in considerazione e quali no, in questa fase non vengono fatte valutazioni, ma vengono eliminati solo quei software che non rispondono a determinati requisiti quali ad es. la licenza o il supporto ad una lingua desiderata, la presenza di documentazione etc… Da qui verranno identificati quei componenti che saranno oggetto di misura.

[17] http://www.openbrr.org

Nella seconda fase si compiono dei giudizi sulle caratteristiche del prodotto, e vengono introdotte dodici categorie per valutarlo:

- funzionalità
- usabilità
- qualità
- sicurezza
- performance
- scalabilità
- architettura
- supporto
- documentazione
- adozione
- comunità
- professionalità

Per ogni categoria viene rilasciato un punteggio da uno a cinque.

Al termine della seconda fase si otterranno i punteggi di ogni caratteristica e verranno processati i dati ottenuti assegnando un peso ad ogni categoria compreso tra uno e dodici al crescere di importanza (terza fase).

L'ultima fase (la quarta) infine provvede al compimento della scelta tramite i valori ottenuti e definendo in modo univoco il Business Readiness Rating Score.

2.6. QSOS

Il metodo QSOS[18], acronimo di Qualification and Selection of Software Open Source, è distribuito con licenza GPL ed è giunto alla versione 1.6.

Questo metodo utilizza quattro fasi per valutare un prodotto:

- Definition
- Evaluation
- Qualification
- Selection

E' bene precisare che le quattro fasi del metodo sono interdipendenti, inoltre il metodo è iterativo, nel senso che una volta terminato il processo di valutazione, questo può riprendere nuovamente dalla prima fase per approfondire aspetti che non siano stati trattati in precedenza.

Durante la prima fase viene effettuata una descrizione del prodotto per poterlo catalogare. La catalogazione passa attraverso tre fattori: la famiglia del software; il tipo di licenza che viene identificata da

[18] http://www.qsos.org

tre qualità, Ownership, Virality e Inheritance, (valutando la licenza si prende in considerazione quanto libero sia un prodotto); e la comunità che viene suddivisa in:

- sviluppatore isolato
- gruppo di sviluppatori
- organizzazione di sviluppatori
- entità legale
- entità commerciale

Valutare la comunità ha l'obiettivo di capire quanto un prodotto sarà sviluppato, aggiornato, migliorato o corretto.

Nella seconda fase si passa alla valutazione che si basa su due punti cardine: la compilazione de "La carta d'identità del software" e de "Il foglio di valutazione". La carta d'identità del software riporta le caratteristiche basilari del prodotto come il nome, l'autore, il tipo di servizi offerti, le funzionalità etc… Mentre redigendo il foglio di valutazione si riporterà una vera e propria valutazione delle caratteristiche usando per ognuna di esse un punteggio da zero a due se la caratteristica è rispettivamente: assente, sviluppata in parte o del tutto presente. Inoltre vengono assegnati altri punteggi sempre utilizzando la stessa metodologia in base alle caratteristiche di:

- durabilità intrinseca

- industrializzazione
- integrazione
- adattabilità tecnica
- strategia

Durante la terza fase vengono creati dei filtri alla carta d'identità ed al foglio di valutazione. Il primo filtro serve per eliminare quei prodotti che non potrebbero essere utilizzati in quel contesto (es. software linux in ambito windows). Mentre con il filtro al foglio di valutazione si pesano le funzionalità del prodotto in base alle esigenze dell'utente finale, separandole in funzionalità richieste, opzionali e non richieste. Inoltre ci sono altri due filtri che analizzano i rischi per l'utente ed i rischi per i providers.

Nell'Ultima fase (selezione) viene identificato il software in base alle richieste dell'utente, elaborando due diverse scelte che possono essere: rigida e flessibile. Utilizzando la scelta rigida (quella più restrittiva) si scremano i software incompatibili con i filtri creati durante la terza fase. E' bene precisare che, utilizzando questa scelta, si può correre il rischio di non ottenere nessun risultato valido se i software comparati risultassero incompatibili con almeno uno dei due filtri. Mentre la scelta flessibile utilizza dei punteggi da assegnare tra la differenza delle caratteristiche che il software

presenta e le richieste fatte dall'utente, così facendo non si rischia di incappare nel problema descritto con la scelta rigida.

2.7. OPEN BQR

Il metodo OPEN BQR[19] acronimo di (Open Business Quality Rating) è stato sviluppato da Davide Taibi come progetto di tesi presso l'Università degli Studi dell'Insubria - sede di Como, e rappresenta una fusione tra le differenti qualità del metodo Open BRR e QSOS, a cui introduce nuovi fattori di valutazione mai considerati in precedenza.

Questo metodo si sviluppa di tre fasi:

- Quick Assessment Filter
- Data Collection & Proccesing
- Data Translation

Durante la prima fase viene identificato un insieme di componenti da misurare, così come si fa con l'OPEN BRR e vengono considerati i seguenti parametri:

[19] http://www.taibi.it/openbqr

- selezione di indicatori basati sul target di utilizzo;
- analisi delle qualità interne;
- analisi delle qualità esterne;
- disponibilità di supporto nel tempo;
- verifica dei requisiti fondamentali.

Queste componenti produrranno una tabella con gli indicatori necessari alla valutazione e verrà assegnato il relativo peso, ottenendo così una tabella con gli indicatori ed il relativo peso.

Nella seconda fase si passa alla scrematura della tabella eliminando i pesi uguali a zero e quelli prossimi allo zero e normalizzando quelli restanti in modo da ottenere un punteggio finale per ogni prodotto quale somma dei punteggi ottenuti per ogni area. Il punteggio così ottenuto sarà utile solo al fine di ottenere un rapido sguardo d'insieme, la valutazione complessiva andrà effettuata tramite le griglie ottenute in precedenza.

L'ultima fase serve esclusivamente alla visualizzazione dei risultati ottenuti nelle due fasi precedenti e, se si comparano pochi prodotti, la visualizzazione può essere fatta utilizzando dei grafici che aiutano nella scelta.

Capitolo Terzo

3.1. PREMESSA

Nel capitolo precedente, sono stati illustrati i principali metodi di valutazione dei software esistenti e la loro implementazione, ma, confrontandoli, si è notato che sono ancora immaturi e non prendono in considerazione alcuni aspetti fondamentali specifici del software stesso, come le:

- Qualità interne
- Qualità esterne
- Disponibilità di supporto nel tempo
- Costo necessario per i moduli proprietari

Per cercare di colmare questa mancanza è nato OPEN BQR, proprio come unione dei metodi QSOS, OPEN BRR e OSMM.

Detto questo, quindi, è bene tenere in considerazione quando si valuta un progetto open source sia delle considerazioni fatte da OPEN BQR sia dei requisiti che nemmeno questo metodo considera.

3.2. CARATTERISTICHE ANALIZZATE

Partendo dalla valutazione precedente, il lavoro di tesi si è posto quindi l'obiettivo di provare a valutare la qualità di tutti i prodotti open source presenti all'interno di uno dei maggiori repository on line: SourceForge.net. I metodi attuali cercano di fornire degli indicatori utili per valutare un prodotto nel modo più oggettivo possibile. Alcuni fattori però non sono semplicemente ritrovabili dall'utente perché nascosti oppure perché ottenibili solo attraverso un lungo processo di ricerca all'interno dei portali dei singoli progetti; inoltre i metodi esistenti possono essere eseguiti solo manualmente. Scopo di questo lavoro, quindi, e'stata la ricerca, l'individuazione e l'analisi di quei parametri necessari per una valutazione dei prodotti di SourceForge che potessero successivamente essere inseriti in un metodo di estrazione automatica delle informazioni atto alla semplificazione e all'automazione del processo di valutazione da parte dell'utente.

Per effettuare la valutazione, quindi, prendendo atto dell'alto numero dei progetti (**135834** alla data in cui è stata fatta l'analisi ottobre 2007), non è stato scelto un metodo di valutazione esistente, ma si è deciso di partire dagli aspetti fondamentali osservati da OPEN BQR e di ampliarli con alcuni fattori che non erano stati mai considerati in precedenza. Quindi l'analisi finale del lavoro verterà sullo studio di questi parametri:

Valutazione delle qualità esterne dei prodotti:

- Bugs
- Patches
- Documentazione

Attività della comunità:

- Sviluppatori
- Release
- Donazioni
- Nascita di nuovi progetti
- Download

Supporto a breve termine:

- Tempo di assegnazione dei bugs

– Tempo di risoluzione dei bugs

L'analisi verrà effettuata su tutti i dati di SourceForge nel suo complesso. Nonché verrà effettuato uno studio prendendo in esame un periodo di sei mesi che va dal 01 aprile 2007 al 29 settembre 2007, da noi valutato come relativamente rappresentativo, ma sufficientemente lungo, al fine di comprendere come evolvono i progetti nel tempo e come lavora la comunità in uno specifico arco temporale. Analizzare i progetti in un determinato periodo permette anche di poter scrivere quelle procedure che permetteranno di automatizzare i processi di valutazione in periodi stabiliti, lasciando come "variabili" i dati relativi alle date di ricerca. Ora cercheremo di rappresentare nel dettaglio i parametri che svilupperemo.

3.3. VALUTAZIONE DI ALCUNE QUALITA' ESTERNE

Come detto in precedenza, la prima parte dell'analisi riguarderà alcune delle principali qualità esterne[20] (si dicono esterne perché sono quelle visibili all'utente a differenza di quelle interne che

[20] http://www.issco.unige.ch/projects/ewg96/node14.html#SECTION00311000000000000000

riguardano il codice sorgente) dei software open source di SourceForge andando a studiare:

- I bugs
- Le patches
- La documentazione

Il bug (letteralmente baco) rappresenta il difetto di un programma, un errore nella scrittura del codice che causa un funzionamento errato o inaspettato ed a volte tale da determinare il blocco totale dell'applicativo. Scoprire e quindi correggere un bug è molto importante soprattutto in ambito sicurezza e, infatti in questi ultimi anni si è sempre sentito più parlare di attacchi informatici sfruttando proprio le falle lasciate aperte dai bugs. Il termine bug si utilizza in campo informatico anche per segnalare un inconveniente hardware, si pensi al "Pentium FDIV bug[21]" un difetto delle prime cpu Pentium Intel che fallivano il risultato di alcune divisioni a virgola mobile.

SourceForge mette a disposizione uno strumento molto potente che crea un contatto diretto tra sviluppatori e utenti: il "bug tracking". Il bug tracking è uno strumento che tiene traccia dei bugs scoperti sotto forma di record assegnandogli, per ognuno di essi, alcune

[21] http://support.intel.com/support/processors/pentium/fdiv/wp/

informazioni molto importanti e che mantiene la storia delle modifiche che vengono effettuate. Il record del bug tracking è composto da:

- Identificativo numerico
- Utente che ha segnalato il bug
- Data della segnalazione
- Data ultima modifica
- Utente che ha fatto la modifica
- Numero di commenti
- Numero di file allegati alla segnalazione
- Categoria della sezione che riguarda il bug
- Gruppo della comunità che ha fatto la segnalazione
- Sviluppatore a cui viene assegnato il bug per la risoluzione
- Priorità
- Stato
- Risoluzione (accettazione del bug da parte della comunità di sviluppo)
- Privato (un bug privato è visibile solo al gruppo di sviluppo ed a chi lo ha segnalato)
- Sommario del problema

Con questo sistema si riesce a tenere traccia di tutta la storia della difettosità di un prodotto open source. Detto questo, è necessario

considerare che uno degli elementi più importanti è quello che segnala la priorità. La priorità deve essere assegnata dall'utente che scova il problema con un numero che va da uno a nove in ordine crescente di criticità e, se non viene specificato, SourceForge assegna come valore di default il cinque. Uno sviluppatore del team del progetto ha la possibilità di cambiare il livello di criticità, cambiamento che viene ovviamente registrato. Altro comportamento di default di SourceForge è quello di non assegnare a nessun sviluppatore i bugs che vengono scoperti, lasciando quindi il compito alla comunità di controllare se sono stati scoperti dei bugs, analizzarli e successivamente passarli ad uno sviluppatore del team. Utilizzando quindi queste informazioni si possono effettuare molte analisi statistiche sui prodotti cercando la difettosità dei progetti e, nello specifico, quanti bugs sono scoperti, quanti, dopo essere stati scoperti, vengono assegnati ad uno sviluppatore, quanti rimangono non assegnati a nessuno suddividendoli per livello di stabilità dei progetti e, per ogni grado di stabilità, dividendoli ancora per livello di criticità.

Successivamente all'analisi dei bugs, viene effettuata l'analisi delle patches. Patch (letteralmente pezza) è un termine inglese che indica, come "*hot fix*" (usato normalmente come sinonimo), un file creato per risolvere uno specifico errore di programmazione (il bug) che impedisce il corretto funzionamento del programma. Questi files

vengono rilasciati inizialmente dagli stessi sviluppatori, nell'attesa di pubblicare una nuova versione del software privo del problema corretto. SourceForge gestisce le patches allo stesso modo dei bugs, fornendo quindi un tracker costituito da un record con le stesse informazioni viste in precedenza. Le patches, come è facile intuire, possono essere solo rilasciate dalla comunità di sviluppo, per cui l'analisi riguarderà ovviamente solo il numero delle patches rilasciate suddivise per livello di stabilità dei progetti e, per ogni grado di stabilità, divise ancora per livello di criticità.

L'altro fattore che riguarda la valutazione delle qualità esterne, che andremo ad analizzare, sarà quello relativo alla documentazione presente, cercando di quantificare i progetti che hanno della documentazione e suddividendoli ovviamente per livello di stabilità dei prodotti.

Questo ci permetterà di controllare la difettosità dei progetti open source di SourceForge, verificando, per esempio, se effettivamente su progetti appena nati (prealpha) vengono riscontrati molti bugs e di conseguenza vengono rilasciate altrettante patches, così come ci si aspetta. Controlleremo inoltre se è più facile riscontrare bugs di criticità più bassa (che dovrebbero quindi rappresentare solo anomalie dei progetti) oppure se sono maggiori i problemi legati alla sicurezza, quindi con criticità più alta. Con queste analisi

potremo studiare mediamente come le comunità si suddividono il lavoro cercando di risolvere i bugs, assegnandoli per la risoluzione la maggior parte per i prodotti stabili e maturi, oppure se preferiscono “lavorare maggiormente” sui progetti appena nati (prealpha, alpha e beta). Stessa valutazione potrà essere fatta controllando come vengono rilasciate le patches. Infine, studiando la documentazione, si avrà la possibilità di controllare se le comunità sono vicine alle necessità degli utenti, rilasciando quindi da subito manuali di installazione, di funzionamento dei programmi oppure se sono maggiormente focalizzati sulla realizzazione del progetto e tralasciano il fattore “documenti”, senza quindi smentire i sostenitori del software commerciale che snobbano i prodotti open source in quanto privi di documentazione.

3.4. ATTIVITA’ DELLA COMUNITA’

Dopo aver analizzato le qualità esterne, si passerà all’analisi dell’attività delle comunità che ruotano intorno ai progetti di SourceForge, nello specifico in questa fase del lavoro andremo a valutare:

– Gli sviluppatori

- Le release
- Le donazioni
- La nascita di nuovi progetti
- I download

Nell'analizzare gli sviluppatori, cercheremo di capire quanti sono e come sono suddivisi all'interno delle comunità di SourceForge; inoltre proveremo a controllare come si suddividono il carico di lavoro in base al livello di stabilità dei progetti. Si cercheranno infine le varie figure per comprendere quali sono e come sono suddivisi fra le comunità e come le comunità "investono" per le varie figure professionali. Questo perché ci si può aspettare un maggior numero di sviluppatori a livello assoluto, ma non si conosce come sono distribuite le altre figure e se SourceForge deficita di alcune "professioni".

Le release invece sono particolari "versioni" di un progetto software e rappresentano la normale evoluzione del programma. Quando viene rilasciata una nuova release vuol dire che il software ha avuto un miglioramento, un aggiornamento, una modifica etc... La convenzione di numerazione delle diverse release di un prodotto software prevede l'identificazione di un numero di versione, di un numero di revisione e di un numero di *release*. Ad esempio, il

kernel linux viene distribuito come un archivio compresso, linux-2.4.31.tar.bz2, in cui il primo numero si riferisce alla versione, il secondo indica la revisione ed il terzo la release. La nuova release di un software, come detto prima, comporta piccoli cambiamenti, mentre una nuova versione implica un cambiamento radicale della struttura del software. Seguendo l'evoluzione delle release quindi, potremo conoscere la sua storia. Per questo nello studio che andremo a fare rintracceremo quante release sono state rilasciate per cercare di capire quanta attività viene effettuata dalle comunità.

Per verificare sempre l'attività delle comunità tenteremo anche di controllare come vengono distribuite le donazioni, quali tipi di progetti vengono maggiormente sostenuti e quali utenti della comunità investono maggiori risorse economiche nei progetti, provando a cercare se ci sono aziende che sostengono i prodotti open source.

Altra analisi sarà effettuata sull'attività della comunità e riguarda la nascita di nuovi progetti messi a confronto con l'intero archivio di SourceForge. Questo ci permetterà di capire come le comunità investono nei nuovi prodotti.

Infine, cercheremo di analizzare i download dei software open source per comprendere come gli utenti si avvicinano ai prodotti open source e se "preferiscono" usare e quindi scaricare

maggiormente prodotti nuovi (prealpha, alpha e beta) oppure affidarsi a prodotti sicuri (stabili, o maturi).

Osservare l'attività della comunità ci permette di controllare la disponibilità di supporto nel tempo. Infatti, se un progetto è sostenuto da una buona comunità che investe in una rosa ampia di figure professionali, rilascia nuove release e impegna contributi economici anche da parte di più aziende, si avrà una maggiore probabilità che un determinato progetto possa evolvere ed essere competitivo, e magari utilizzato da un grosso numero di utenti. Differentemente da quanto può essere fatto da una piccola comunità di pochi sviluppatori che potrebbe dar vita ad un nuovo progetto e successivamente abbandonarlo perché non riuscirebbe a portare a termine lo sviluppo per mancanza di fondi, di figure professionali oppure semplicemente perché non trova riscontro verso gli utenti.

3.5. SUPPORTO A BREVE TERMINE

L'ultima parte di analisi riguarderà il supporto a breve termine, che verrà esaminato controllando due aspetti fondamentali:

- Tempo di assegnazione dei bugs
- Tempo di risoluzione dei bugs

Studiando il tempo di assegnazione dei bugs si cercherà di comprendere quanto la comunità è veloce nel rispondere e cercare di correggere i problemi “quotidiani”. Questo perché, se si controllano velocemente i bugs e si assegnano ugualmente velocemente ad uno sviluppatore che possa risolverli, ciò implicherà una forte propensione da parte del team a supportare il progetto nel “breve termine” e di conseguenza si avrà la prospettiva che lo stesso progetto possa essere supportato anche nel futuro. Differentemente, se si assegnano i bugs lentamente o nel peggiore dei casi si lasciano senza assegnazione, (ricordiamo che il comportamento di default di SourceForge è quello di non assegnare a nessuno un bugs scoperto) questo può essere interpretato come una scarsa attività del team che gestisce il prodotto e quindi ci sarà la propensione a classificare il progetto di futuro abbandono. Analoghe considerazioni valgono per l’analisi del tempo di risoluzione dei bugs. Se una comunità è veloce nel risolverli, questo può rappresentare un team che si dedica assiduamente a quel progetto e non rappresenta un lavoro che viene fatto solo come “hobby”.

Capitolo Quarto

4.1. PREMESSA

Nel capitolo precedente è stato presentato l'obiettivo che ci siamo prefissati in questo lavoro di tesi, analizzando nello specifico tutte le caratteristiche che saranno prese in considerazione. Mentre nel capitolo quarto sarà studiato SourceForge, le sue caratteristiche ed il suo contenuto. Inoltre, verrà mostrato il reperimento dei dati che verranno utilizzati durante lo studio, gli eventuali problemi riscontrati e le scelte adottate.

4.2. SOURCEFORGE.NET

SourceForge.net è un CVS che ospita e gestisce tramite il proprio sito web i progetti open source, offrendo loro spazio web, risorse per lo sviluppo del codice, risorse per la collaborazione fra le

comunità, la potenza di calcolo per gestire grossi download etc... Inoltre mette a disposizione degli utenti, sedici utili informazioni su ogni progetto esistente:

- La comunità che ha creato il progetto
- Gli amministratori del progetto
- Il numero di sviluppatori
- Il livello di stabilità
- Destinatari del progetto
- Licenza
- Sistema operativo supportato
- Linguaggio di programmazione
- Tipo di progetto
- Lingue supportate
- Tipo di interfaccia utente
- Nome del progetto
- Data di creazione
- Percentuale di attività sul progetto
- Statistica sull'attività del progetto
- Una lista di RSS feeds ai quali iscriversi per conoscere le novità

E’ giusto precisare che il livello di stabilità dei prodotti non viene calcolato algoritmicamente da SourceForge, ma è l’amministratore del progetto che decide quando cambiarlo seguendo rispettivamente questo ordine: prealpha, alpha, beta, stabile e maturo.

Oltre a queste informazioni, nella home page di ogni progetto (che viene creato come nome a dominio di terzo livello es. project_name.sourceforge.net) vengono anche mostrate alcune informazioni sulla qualità del prodotto:

- Numero di bugs aperti/totali
- Richieste di supporto aperte/totali
- Patches aperte/totali
- Richieste di Feature aperte/totali
- Forum Aperti totali/numero di post
- Numero di mailing list
- Sottoversioni
- Link al codice sorgente

Queste informazioni sono riportate solo numericamente e non viene fatta nessuna analisi automatizzata. SourceForge inoltre, classifica sia i bugs che le patches in nove livelli di criticità, assegnandogli un numero da uno a nove rispettivamente dal meno importante al più importante. Questa assegnazione deve essere fatta da chi scopre e

pubblica il bug/patch, altrimenti il comportamento di default di SourceForge è quello di assegnargli il valore intermedio: cinque.

Il CVS (Concurrent Versions System, sistema di versioni concorrente) di SourceForge, che si occupa della gestione dei progetti, nasce anch'esso come progetto open source dalla società VA_Software[22], purtroppo a causa dei cambiamenti della politica aziendale, non viene più rilasciato il codice sorgente giunto ormai prima della chiusura e della trasformazione in software closed source alla versione 2.61.

4.3. IL REPERIMENTO DEI DATI

Per effettuare l'analisi inizialmente si è scelto di usare i dati del progetto FLOSSmole[23] (collaborative collection and analysis of free/libre/open source project data).

Dall'home-page del blog del progetto (Figura 1) si scopre che è possibile ottenere i dati grezzi di SourceForge necessari per avviare l'analisi, e nello specifico:

- Dati grezzi dei progetti open source

[22] http://web.sourceforge.com/index.php
[23] http://ossmole.sourceforge.net/

– Report sui dati dei progetti open source

– Integrazione dei dati provenienti da altri gruppi di ricerca

– Alcuni strumenti per gestire i dati

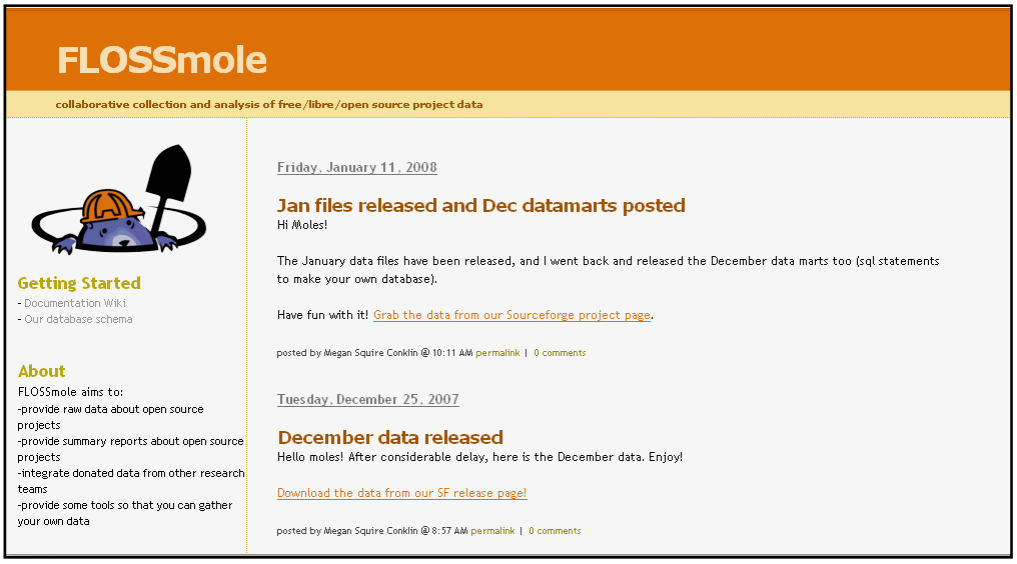


Figura 1 *Home Page di FLOSSMole*

Un'altra valutazione che inizialmente aveva spinto all'utilizzo dei dati di FLOSSmole è stata la possibilità di averne una copia "in locale", permettendo una manipolazione molto più accurata dell'analisi. Infatti, nel reperire i dati si viene rediretti verso la pagina che permette di scegliere se scaricare i file *.SQL che permettono di creare una copia locale del database. Oppure si può

scegliere di scaricare i file *.TXT dei dati che sono le stesse informazioni, ma salvati su file di testo che permettono la manipolazione con i classici fogli di calcolo (es. Excel), o in alternativa si può usare il "query tools" per effettuare direttamente on-line le interrogazioni. E' bene precisare che i file, reperibili nell'aria download del progetto FLOSSmole, vengono aggiornati periodicamente ogni due mesi (così come si legge dal wiki[24]), creando ogni volta una nuova copia del file e non facendo l'upgrade dei nuovi dati. Questo non permette di creare dei tools automatici che possano effettuare una analisi continua dei nuovi dati in arrivo da SourceForge; inoltre abbiamo notato che ogni nuovo upgrade dei file riporta anche alcune modifiche alla struttura del database che viene creato, aggiungendo l'eventuale problema di non poter utilizzare l'analisi fatta su copie di tempi diversi del database. Inoltre (si legge sempre dal wiki), che FLOSSmole non scarica direttamente i dati, ma lancia lo "spider" che scansiona i progetti di SourceForge e raccoglie i dati direttamente dalle pagine html salvandoli nel proprio database.

Nel lavoro di tesi si è scelto di usare i file *.SQL per creare una copia locale del DB di Sourceforge ed avere la possibilità di effettuare un'analisi più accurata possibile.

[24] http://ossmole.wiki.sourceforge.net/datadescrSF

Dopo aver creato il database locale dei dati grezzi di SourceForge, si è passati all'analisi.

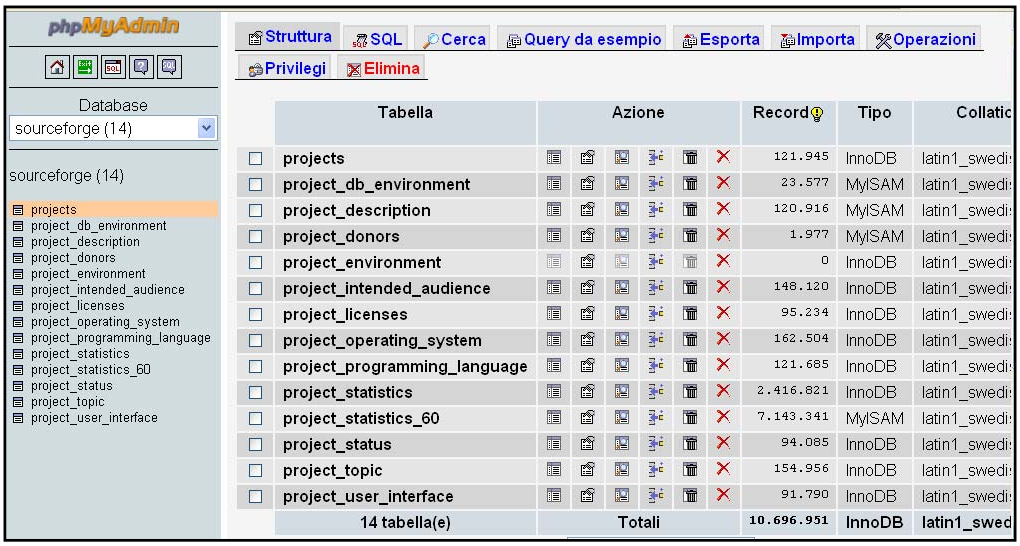


Figura 2 *Database di SourceForge dai dati di FLOSSmole*

Come si nota dalla Figura 2, i dati di FLOSSmole ricostruiscono il database di SourceForge, con quattordici tabelle e ogni tabella rappresenta una caratteristica dei progetti.

Come detto in precedenza, la prima analisi da effettuare è quella della valutazione della qualità andando a cercare la presenza di bugs e suddividendoli per priorità. Nella ricerca di questi dati ci si è accorti però che il database creato da FLOSSmole non ha a disposizione i dati "puri" del CVS, ma solo una raccolta statistica. Infatti i bugs non vengono suddivisi in priorità, ma aggregati fra di

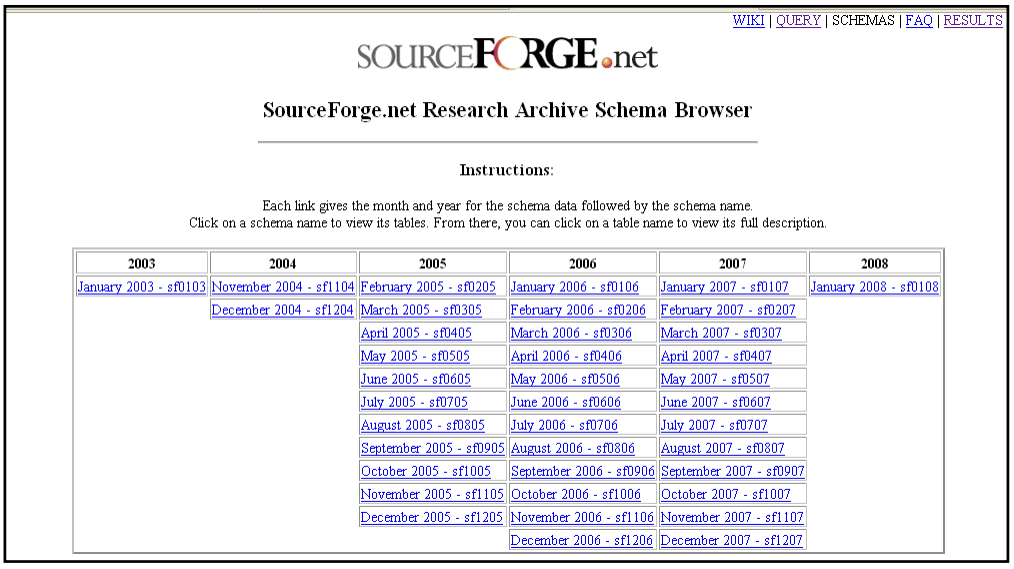
loro e, così facendo, si può solo conoscere quanti bugs ha un determinato progetto, ma non si potranno avere notizie sulla criticità. Un altro fattore critico che presentano questi dati è sulle date, in quanto le uniche informazioni che si hanno di questo tipo sono relative alla creazione del dump della collezione, e questo non permette di effettuare tutta la parte di analisi su quell'arco temporale che ci eravamo prefissati, né di calcolare il tempo necessario alla risoluzione dei bugs. Questi stessi problemi sono stati riscontrati anche per le patches e le release, inoltre sono completamente assenti le informazioni sulla documentazione. Discorso a parte deve essere fatto per gli sviluppatori in quanto dal wiki si legge che può accadere che in dumps diversi non ci siano le stesse informazioni per la comunità. Questo accade se uno sviluppatore inizialmente faceva parte del team di un progetto, ma successivamente lo abbandona, in quanto questo farà sì che nel primo dump dei dati sia presente quello sviluppatore, mentre nel secondo sarà assente, senza registrare i cambiamenti. A causa di tali problematiche riscontrate, si è deciso quindi di abbandonare l'uso di questi dati e provare ad utilizzare quelli presenti in un altro database che ha le stesse finalità di FLOSSmole, ma raccoglie i dati in maniera diversa.

4.4. UNIVERSITY OF NOTRE DAME. L'ACCESSO AL NUOVO SET DI DATI

L'università di Notre Dame[25] sta sviluppando un progetto sulla comprensione dello sviluppo del software open source, sfruttando i dati di SourceForge. Questi dati, previa registrazione, sono messi a disposizione della comunità scientifica, fornendo un wiki ed una descrizione di come vengono reperiti.

A differenza di quanto è avvenuto con il set di FLOSSmole, leggendo la pagina introduttiva dei nuovi dati[26] si nota che questa volta avremo a disposizione l'intero database di SourceForge e non una copia di elementi estrapolati. Mensilmente la VA_Software, riversa la copia del proprio database in quello dell'Università di Notre Dame, creando così un dump relativo a tutta la storia di SourceForge fino al mese in questione. Questa operazione crea però un database differente per ogni mese introducendo la problematica di dover accedere a database fisici differenti se si vogliono analizzare i dati relativi a nuovi periodi (vedi Figura 3).

[25] http://www.nd.edu/
[26] http://www.nd.edu/%7Eoss/Data/data.html



WIKI | QUERY | SCHEMAS | FAQ | RESULTS

SOURCEFORGE.net

SourceForge.net Research Archive Schema Browser

Instructions:

Each link gives the month and year for the schema data followed by the schema name.
Click on a schema name to view its tables. From there, you can click on a table name to view its full description.

2003	2004	2005	2006	2007	2008
January 2003 - sf0103	November 2004 - sf1104	February 2005 - sf0205	January 2006 - sf0106	January 2007 - sf0107	January 2008 - sf0108
	December 2004 - sf1204	March 2005 - sf0305	February 2006 - sf0206	February 2007 - sf0207	
		April 2005 - sf0405	March 2006 - sf0306	March 2007 - sf0307	
		May 2005 - sf0505	April 2006 - sf0406	April 2007 - sf0407	
		June 2005 - sf0605	May 2006 - sf0506	May 2007 - sf0507	
		July 2005 - sf0705	June 2006 - sf0606	June 2007 - sf0607	
		August 2005 - sf0805	July 2006 - sf0706	July 2007 - sf0707	
		September 2005 - sf0905	August 2006 - sf0806	August 2007 - sf0807	
		October 2005 - sf1005	September 2006 - sf0906	September 2007 - sf0907	
		November 2005 - sf1105	October 2006 - sf1006	October 2007 - sf1007	
		December 2005 - sf1205	November 2006 - sf1106	November 2007 - sf1107	
			December 2006 - sf1206	December 2007 - sf1207	

Figura 3 *Schema dei database di SourceForge*

E' facile intuire che questa scelta rende le analisi statiche relative solo a periodi prefissati a priori e non permette la realizzazione di procedure automatiche. Creando invece un unico database che seguisse anche le modifiche alla struttura e consentendo l'accesso diretto al db, sarebbe possibile effettuare analisi più pertinenti. E' bene precisare che, prima della creazione dei dump all'interno del database dell'Università di Notre Dame, vengono preventivamente eliminate alcune tabelle, relazioni e dati per ovvie ragioni di sicurezza e privacy (si pensi ad esempio alla possibilità di avere la tabella con i dati delle password degli utenti). Diversamente da FLOSSmole, l'accesso ai dati può essere effettuato esclusivamente

tramite il query tools messo a disposizione, infatti il wiki[27] precisa "*Unfortunately, our contract with SourceForge does not allow us to give out the actual data sets. We are working on providing Web Service access, which should ameliorate most of the difficulties associated with the web form*". Purtroppo, il query tools non è altro che un form html (Figura 4) che cattura la stringa della query per interrogare il database e la lancia verso uno script cgi. E' facile intuire che anche questo metodo non permette di creare procedure automatizzate che possano interrogare il database in modo continuo, rendendo quindi necessaria l'esecuzione manuale di ogni singola query per l'estrazione dei dati da analizzare.

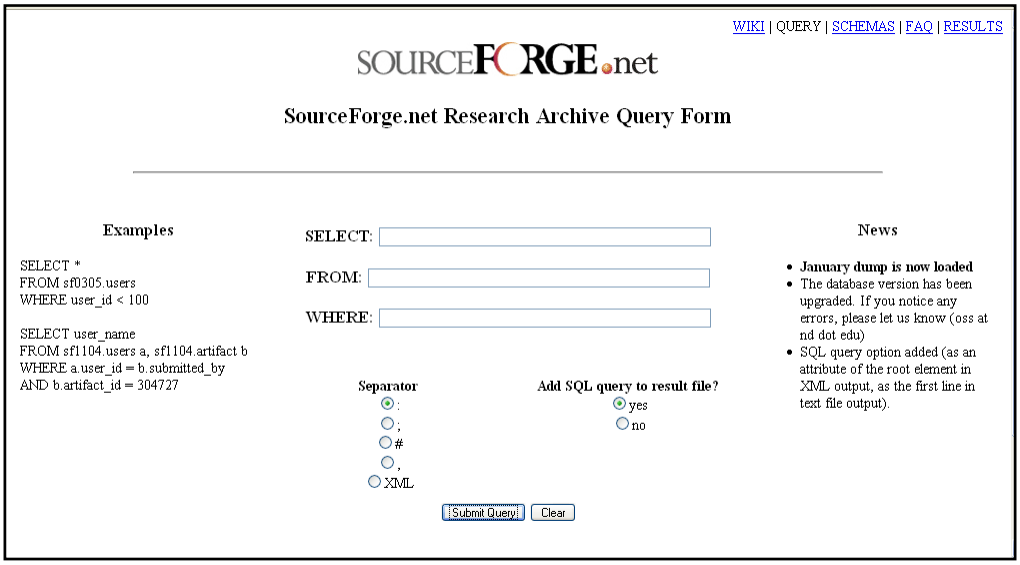


Figura 4 *Query form di Notre Dame*

[27] https://zerlot.cse.nd.edu/mywiki/index.php?title=FAQ

Oltre a quanto detto, l'università di Notre Dame mette a disposizione lo schema ER del DB di SourceForge (Figura 5), precisando che è una versione vecchia, in quanto negli ultimi due anni la struttura del database ha subito molte modifiche, ma senza indicare con precisione quali siano state le modifiche. Questi cambiamenti fanno sì che il lavoro di analisi effettuato per un periodo determinato non può essere utilizzato su dump differenti perché presentano una struttura eterogenea.

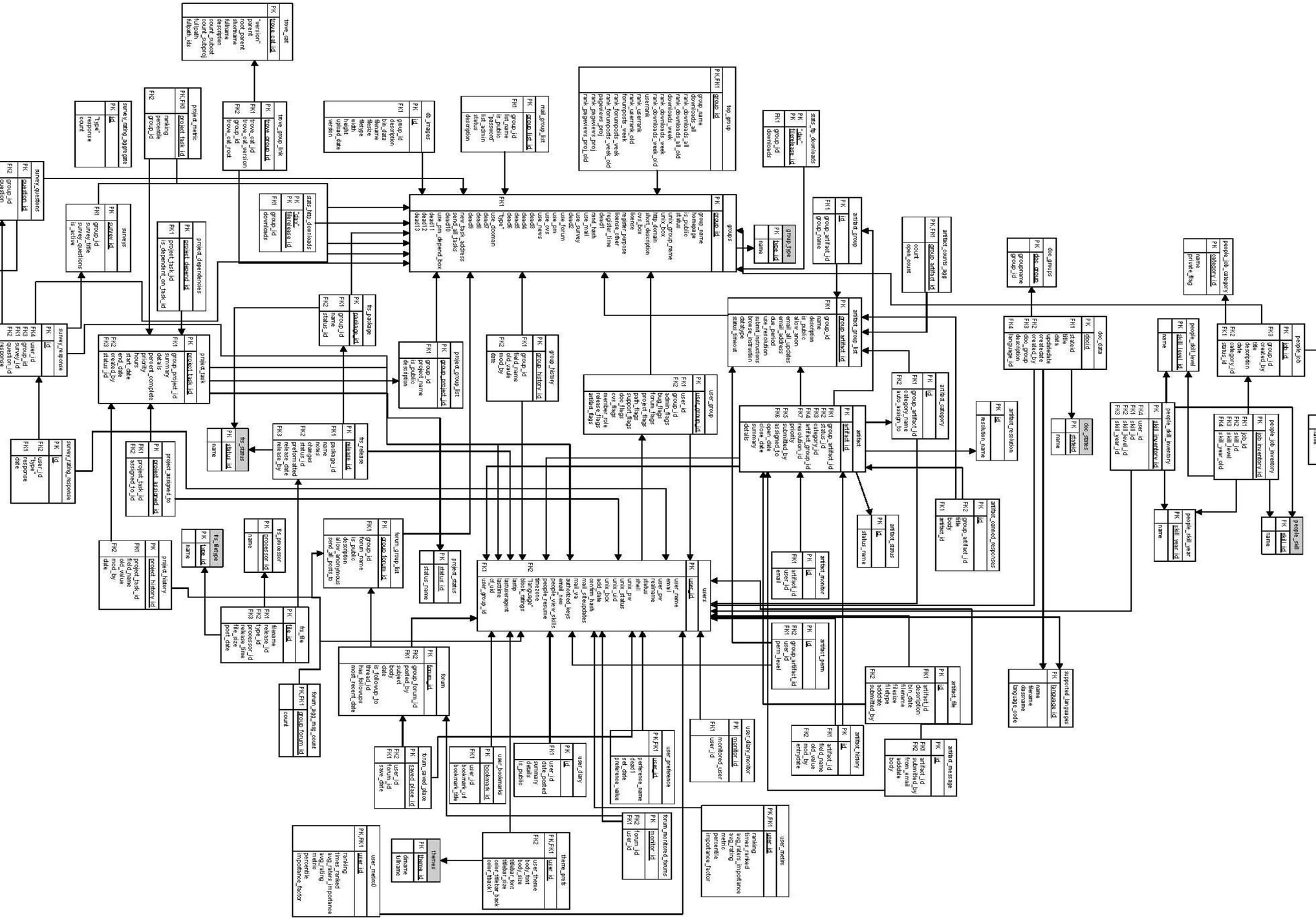


Figura 5 *Schema ER del DB di SourceForge*

Analizzando la struttura dello schema ER e paragonandola con lo schema del database di ottobre 2007 (si è scelto di usare questo in quanto era l'ultimo dump caricato quando l'analisi ha avuto inizio), si nota che lo schema ER è composto da sessantanove (69) tabelle, mentre lo schema del database di ottobre 2007 da settantotto (78); risulta quindi scontata l'incongruenza. Successivamente, sono state confrontate le singole tabelle fra di loro ed è emerso che nello schema ER ce n'erano alcune non presenti nel database da analizzare, ed allo stesso modo il database di ottobre 2007 pur presentando numericamente molte più tabelle di quelle dello schema ER, ne ha altre non presenti nello schema. Infine l'altra incongruenza riscontrata è quella presente nelle tabelle comuni allo schema ed al database di ottobre 2007, in quanto, confrontandole singolarmente, si è visto che presentano una struttura interna differente e quindi una collezione dei dati diversa.

Preso atto di questo, si è deciso di utilizzare lo schema ER come guida per comprendere grosso modo le relazioni che ci potessero essere fra le varie tabelle ed, essendoci l'ulteriore mancanza di documentazione su quali dati potessero essere contenuti all'interno, si è prima proceduto ad individuare le tuple necessarie per l'analisi e successivamente si è passati alla fase di studio vera e propria.

Capitolo Quinto

5.1 PREMESSA

Nei capitoli precedenti abbiamo introdotto il mondo dell'open source, dei metodi attualmente utilizzati per valutare questi software, abbiamo esposto gli obiettivi che ci siamo prefissati e cosa vorremmo ottenere, per passare successivamente allo studio vero e proprio tramite i dati di FLOSSMole prima e di quelli di Notre Dame dopo. Ora invece mostreremo l'analisi vera e propria tramite l'utilizzo dei dati ottenuti. E' opportuno precisare che alla data di ottobre 2007 (momento in cui ha avuto inizio lo studio), SourceForge gestiva 135.834 progetti.

5.2 L'ANALISI DEI BUGS

L'analisi dei bugs, che rappresenta una delle qualità esterne di un software, ci permetterà di capire quanto i prodotti open source sono "difettosi" e come questi difetti sono distribuiti tra i vari prodotti.

5.2.1 Studio dei Bugs Scoperti

I primi dati raccolti sono stati quelli riguardanti tutti i bugs scoperti relativamente a tutto il database di SourceForge, differenziandoli per livello di criticità (priorità) crescente e per distribuzione tra i vari livelli di stabilità dei progetti.

A titolo esemplificativo si riporta il testo della query eseguita sul database di Notre Dame per reperire questi dati:

SELECT *artifact.priority, count(*) AS Nr.Bugs , trove_group_link.trove_cat_id*

FROM *sf1007.artifact_group_list, sf1007.artifact, sf1007.trove_group_link*

WHERE *(artifact.open_date != 0) and artifact_group_list.name = 'Bugs' and (artifact_group_list.group_artifact_id = artifact.group_artifact_id) and (artifact_group_list.group_id = trove_group_link.group_id) and trove_group_link.trove_cat_id between 8 and 12 group by artifact.priority, trove_group_link.trove_cat_id order by artifact.priority, trove_group_link.trove_cat_id*

STABILITA'	PRIORITY 1	PRIORITY 2	PRIORITY 3	PRIORITY 4	PRIORITY 5	PRIORITY 6	PRIORITY 7	PRIORITY 8	PRIORITY 9	PRIORITY 10
PREALPHA	2058	884	1856	729	47146	1239	2743	1464	2726	42
ALPHA	3689	2907	3458	1330	85588	2092	4445	2235	3920	
BETA	7365	3254	7192	2807	188902	5153	9758	4768	9454	
STABILE	13872	5076	10799	4479	297895	7592	14573	6977	12812	
MATURO	2033	2585	2036	938	50137	1174	2078	1123	1567	

Tabella 1 *Bugs totali scoperti*

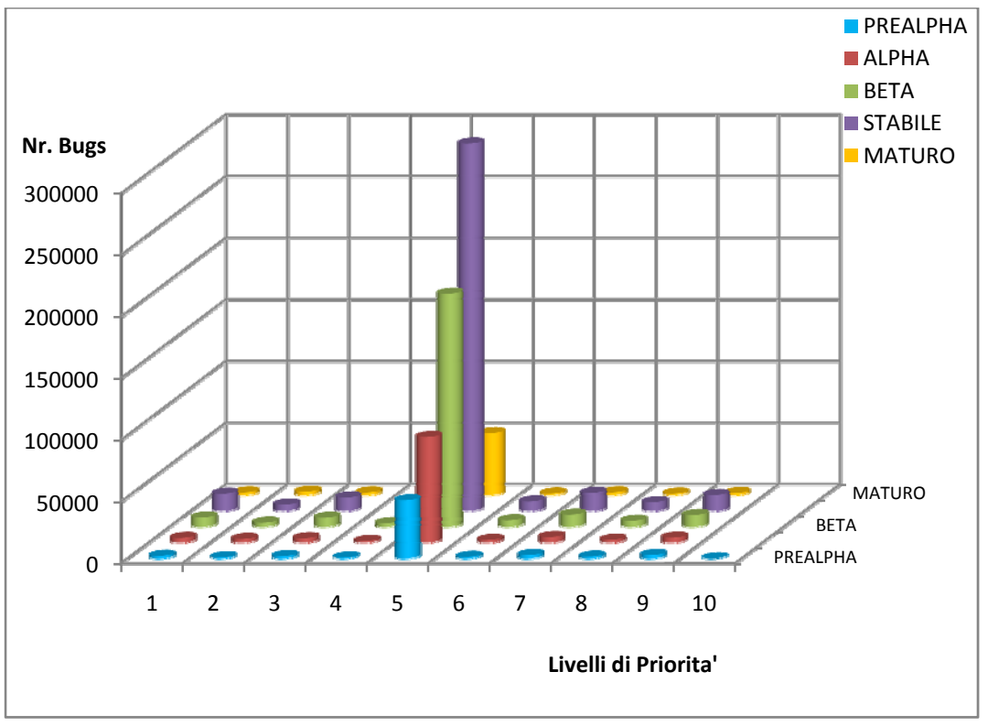


Figura 6 *Bugs totali scoperti*

Questa prima analisi ci mostra già una incongruenza con i livelli di priorità dei bugs, perché per definizione SourceForge li classifica in nove livelli mentre qui si nota che sono stati rilasciati bugs di livello dieci. Per cercare di capire se questa sia una incongruenza su tutto il database di SourceForge, si sono cercati allora tutti i bugs di livello dieci avendo come risultato un unico progetto “Open Visual Gaming Viewer” che ha quarantadue (42) bugs: stessa quantità ottenuta nella ricerca globale. Ciò si può spiegare solo come un errore da parte della comunità di sviluppo nell’aver assegnato una priorità non contemplata da SourceForge. Ritornando allo studio del grafico, si nota inoltre che la rappresentazione della serie dei bugs di priorità cinque è predominante rispetto agli altri, non permettendo di visualizzare accuratamente le altre serie e di conseguenza l’andamento degli altri livelli di priorità. Analizzando solo i bugs di livello cinque, si nota che i progetti “giovani” (prealpha, alpha e beta) presentano un quantitativo di bugs crescente man mano che si sale di stabilità, mentre ci si sarebbe potuti attendere un comportamento contrario perché un progetto appena nato dovrebbe presentare un numero di problemi maggiori dello stesso giunto ad un livello di stabilità alto (stabile). Infatti i prodotti maturi presentano un numero di difetti (limitatamente alla priorità cinque) significatamente minori, perché questi prodotti sono ormai consolidati. Per cercare di analizzare il resto dei dati, si mostra di seguito lo stesso grafico al quale sono stati eliminati i dati di livello cinque.

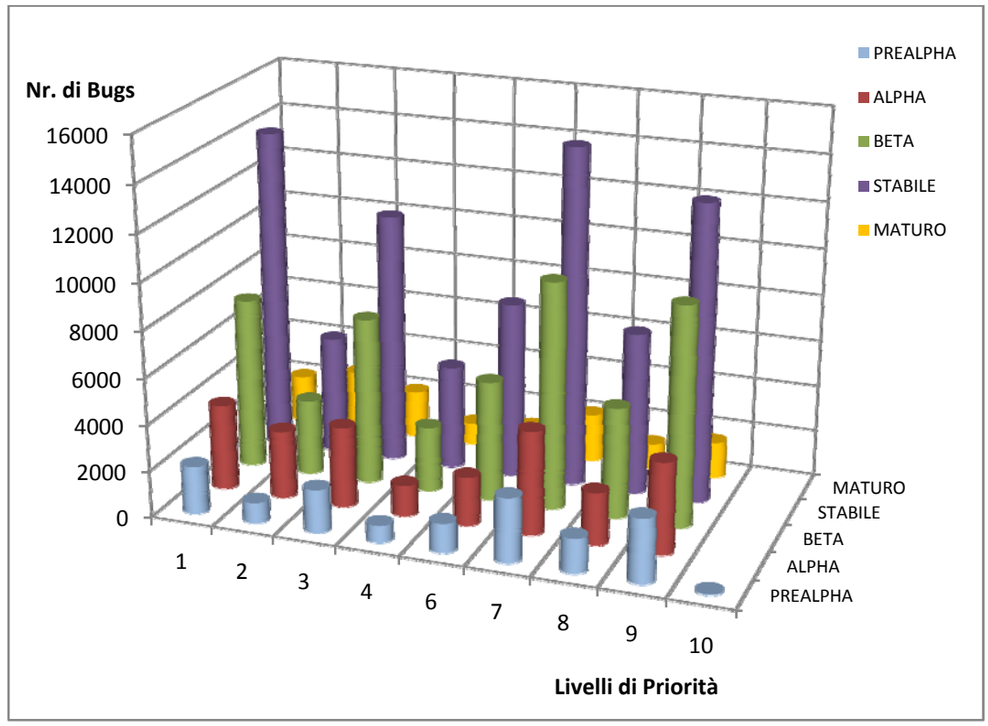


Figura 7 *Bugs totali scoperti senza priorità 5*

Con questa nuova visualizzazione (vedi Figura 7) ed essendo i dati dello stesso ordine di grandezza, si riesce a studiare la distribuzione di tutti i bugs. Anche qui si nota l'andamento crescente del numero dei bugs con l'aumentare della stabilità (come era stato notato con i bugs di criticità cinque) finché non si raggiunge il quarto livello (stabile) per avere successivamente un crollo una volta che i prodotti vengono dichiarati maturi. Questo comportamento si può provare a spiegare supponendo che le comunità creano un nuovo prodotto e lo espongono all'esterno; inizialmente si cominciano a scovare i bugs, ma il team di sviluppo, oltre a risolverli, rilascia nuovi componenti per far crescere il prodotto che inevitabilmente introducono ulteriori problemi. Questo comportamento continua finché non si raggiunge l'ultimo livello di stabilità (maturo) dove non vengono più rilasciati componenti aggiuntivi, ma essendo ormai il prodotto consolidato, si provvede solo a correggere i bugs che vengono scoperti.

5.2.2 Studio Dei Bugs Scoperti Nel Semestre

Come detto in precedenza, ora sarà mostrato lo studio fatto prendendo in esame un periodo di sei mesi (dal 01 aprile al 29 settembre 2007) al fine di comprendere come evolvono i progetti nel tempo e come lavora la comunità in uno specifico arco temporale. Verranno mostrati nove grafici: ognuno di essi rappresenta un livello di criticità e raffigura il numero di bugs scoperti relativamente al solo periodo preso in esame e differenziati per i livelli di stabilità dei progetti.

Anche in questo caso si mostra il testo di una delle query usate per reperire i dati al fine di far notare come vengono inseriti i valori di timestamp per ottenere quanto cercato:

```
SELECT   artifact.priority, count(*) AS Nr.Bugs , trove_group_link.trove_cat_id
FROM     sf1007.artifact_group_list, sf1007.artifact, sf1007.trove_group_link
WHERE    (artifact.open_date between 1175378400 and 1175896800) and
         artifact_group_list.name = 'Bugs' and
         (artifact_group_list.group_artifact_id = artifact.group_artifact_id)
         and (artifact_group_list.group_id = trove_group_link.group_id) and
         trove_group_link.trove_cat_id between 8 and 12 group by
         artifact.priority, trove_group_link.trove_cat_id order by
         artifact.priority, trove_group_link.trove_cat_id
```

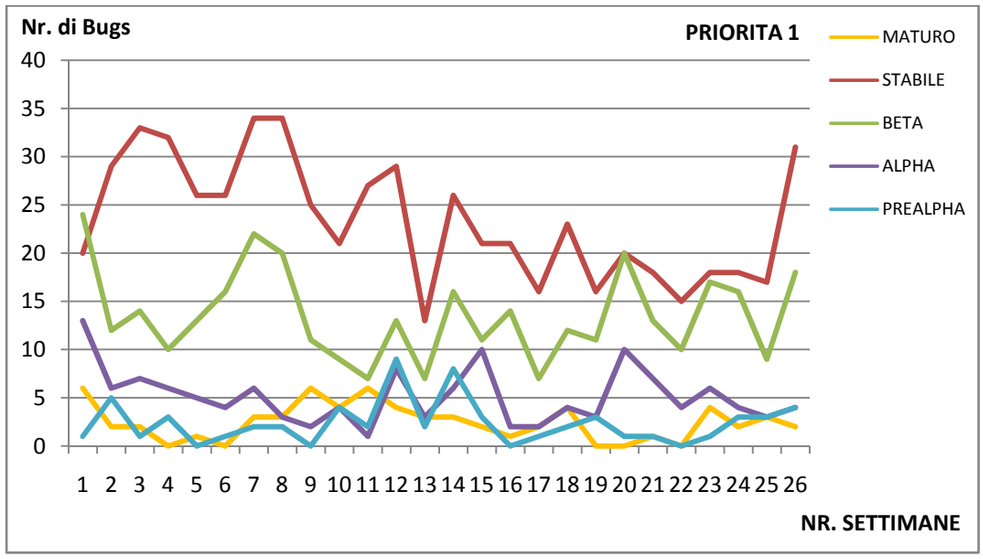


Figura 8 *Numero di bugs scoperti di priorità 1 nel semestre*

Il livello di priorità uno rappresenta la criticità più bassa e dalla Figura 8 si nota che la distribuzione dei bugs rispetto ai livelli di stabilità dei progetti è analoga a quanto è stato illustrato in precedenza, inoltre si può notare che i bugs scoperti per i progetti “maturi” alcune settimane hanno come valore zero e ciò, considerando che la valutazione viene fatta su tutto il database di SourceForge, lascia come unica spiegazione quella fatta in precedenza: dove i progetti dichiarati maturi non subiscono più grossi cambiamenti e di conseguenza non c’è la possibilità di riscontrare piccoli problemi di funzionamento.

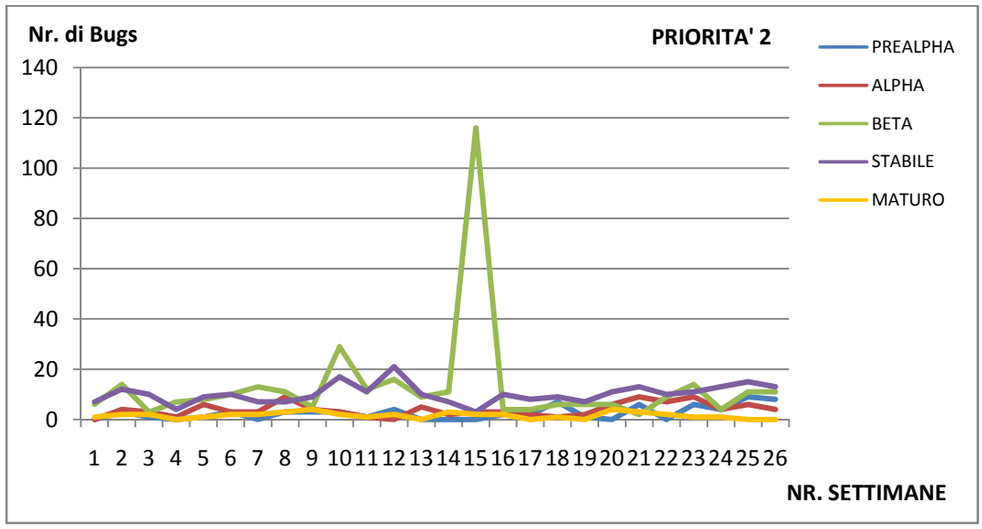


Figura 9 *Numero di bugs scoperti di priorità 2 nel semestre*

La Figura 9 mostra il numero di bugs di priorità due scoperti nel semestre in esame, evidenziando nella quindicesima settimana un picco per i progetti beta. Questo è facilmente spegabile tenendo in considerazione l'andamento della stessa curva nel resto del grafico in quanto è pressochè costante, quindi il picco sarà imputabile ad una casualità su di un progetto. Ad esempio una comunità può avere introdotto un miglioramento ad un suo prodotto ma questo ha inserito un nuovo malfunzionamento che può essere stato segnalato da più utenti che hanno provato nella stessa settimana il software.

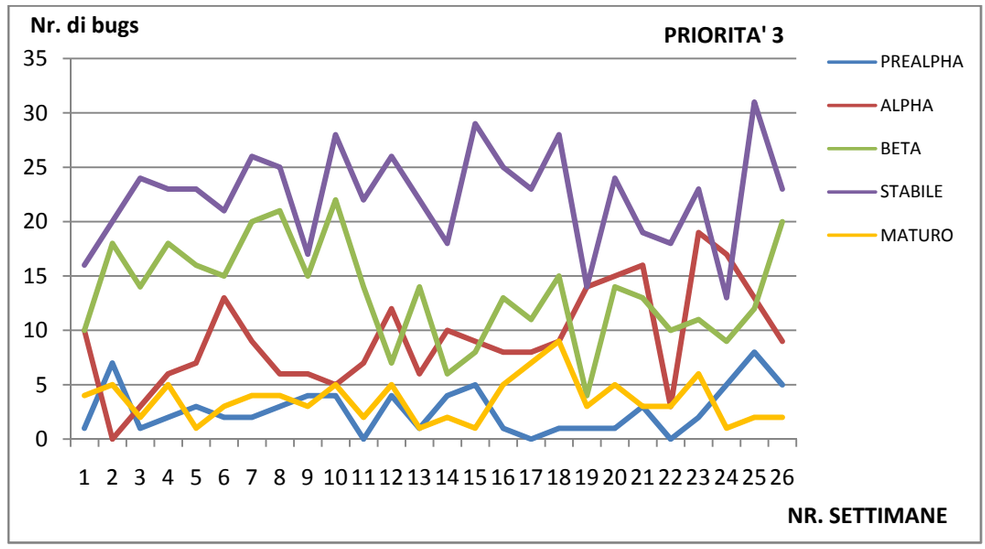


Figura 10 *Numero di bugs scoperti di priorità 3 nel semestre*

La Figura 10, oltre a confermare l'andamento dei bugs scoperti relativamente ai prodotti maturi, mette in evidenza durante le prime dieci settimane la separazione netta tra i bugs dei vari livelli. Nella seconda parte, invece, si nota l'inversione di tendenza per i progetti alpha e beta che non presentano una distinzione netta, ma sono pressochè simili. Ciò non influisce sui risultati ottenuti fino a questo momento.

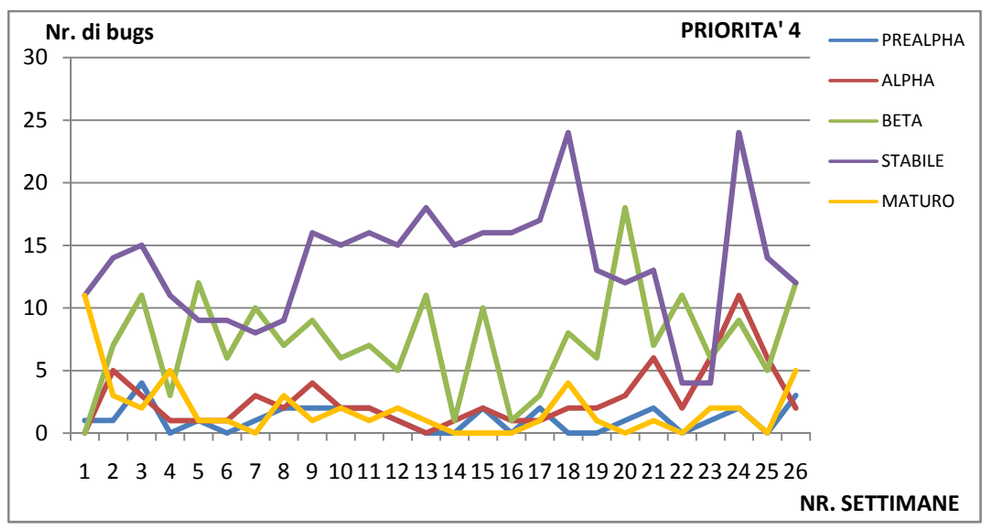


Figura 11 *Numero di bugs scoperti di priorità 4 nel semestre*

Il grafico di Figura 11 mostra un comportamento differente da quanto visto finora, perché i bugs scoperti per i progetti prealpha e alpha non presentano una distinzione netta nel numero di bugs, ma sono bensì sovrapposti con i bugs dei progetti maturi che mantengono lo stesso fattore di grandezza quasi nullo e che quindi possono essere trascurati.

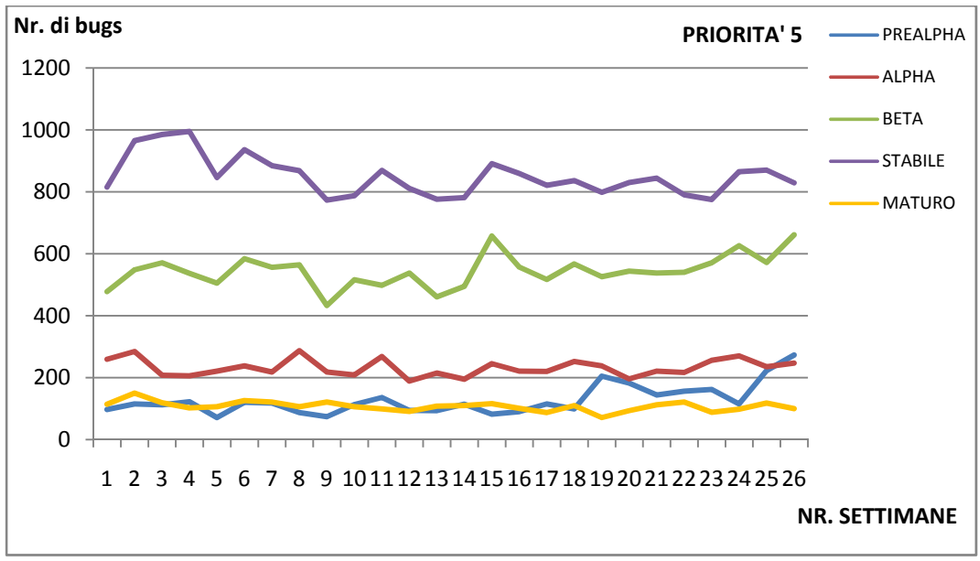


Figura 12 *Numero di bugs scoperti di priorità 5 nel semestre*

La Figura 12, che rappresenta i bugs di priorità cinque, mostra lo stesso andamento visto nel grafico dei bugs totali (si precisa che questo è l'andamento generale della priorità per la grande maggioranza dei bugs), rispecchiando la distribuzione del numero dei bugs per ogni livello di stabilità, quindi a stabilità più basse si hanno un numero di bugs minore, compreso quello dei prodotti maturi che si attesta sempre su un livello numerico più basso. E' da notare che rispetto agli altri grafici, l'ordine di grandezza è nettamente superiore (maggiore di 10) e questo è da imputare al comportamento di default di SourceForge che assegna (se non espressamente specificato) il valore cinque.

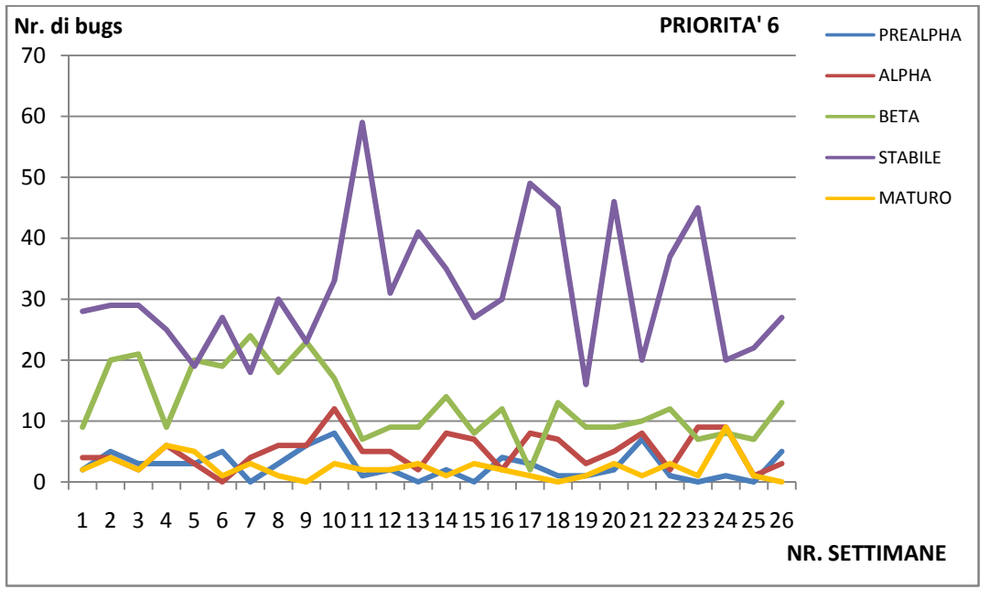


Figura 13 *Numero di bugs scoperti di priorità 6 nel semestre*

Il livello di criticità sei assegnato ai bugs, può già essere considerato tra quelli che indicano già problemi importanti o di sicurezza e la Figura 13 mostra la distribuzione di questi problemi scoperti nel semestre in esame. Qui si nota una piccola differenza con le distribuzioni precedenti in quanto sono maggiori i bugs scoperti per i progetti di livello stabile, con un ordine di grandezza superiore di circa quattro volte. Mentre, i prodotti di livello beta, pur presentando un numero di bugs superiore agli altri, rimangono mediamente vincolati all'interno di un insieme che non supera il fattore dieci, così come i progetti di livello prealpha, alpha e maturo.

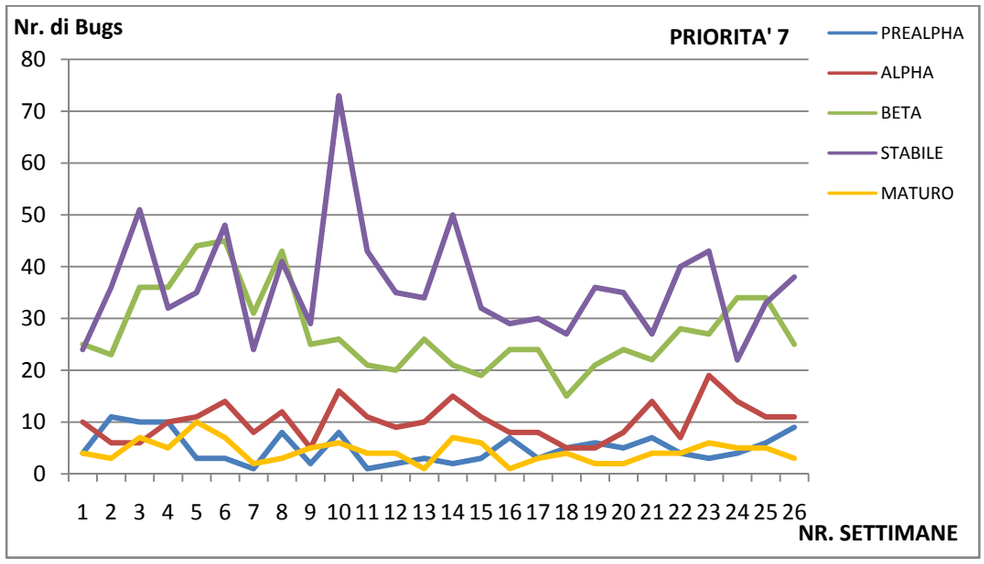


Figura 14 *Numero di bugs scoperti di priorità 7 nel semestre*

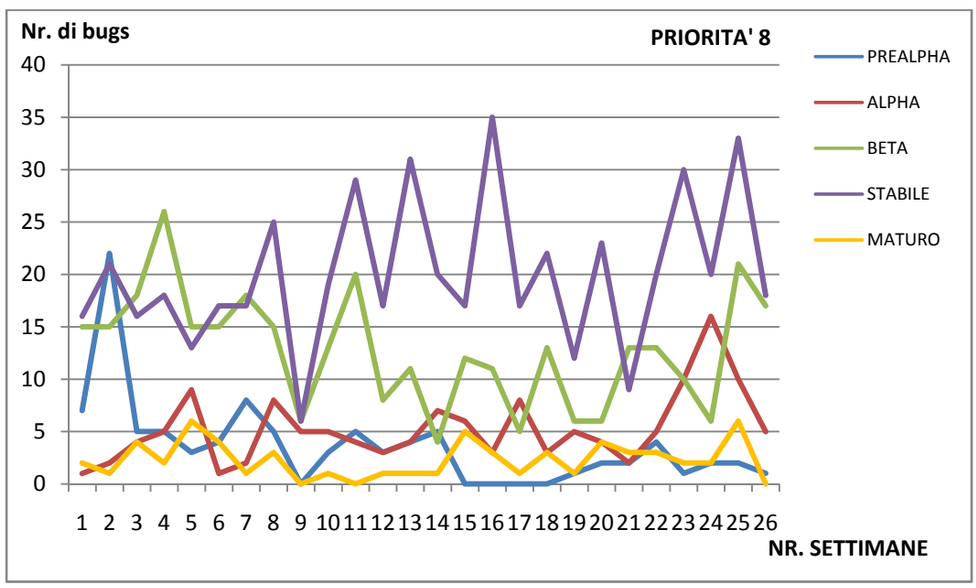


Figura 15 *Numero di bugs scoperti di priorità 8 nel semestre*

Le Figura 14 eFigura 15 mostrano i bugs scoperti relativamente ai livelli di criticità sette ed otto. Questi due grafici sono molto simili perché presentano la ormai consueta distribuzione vista in precedenza in base al livello di stabilità dei prodotti, confermando anche la quasi assenza di problemi sui prodotti maturi. E' probabilmente da considerare un caso l'intersezione dei bugs dei prodotti prealpha con

quelli dei progetti stabili durante la seconda settimana nella Figura 15, in quanto superata quella settimana, l'andamento riprende normalmente.

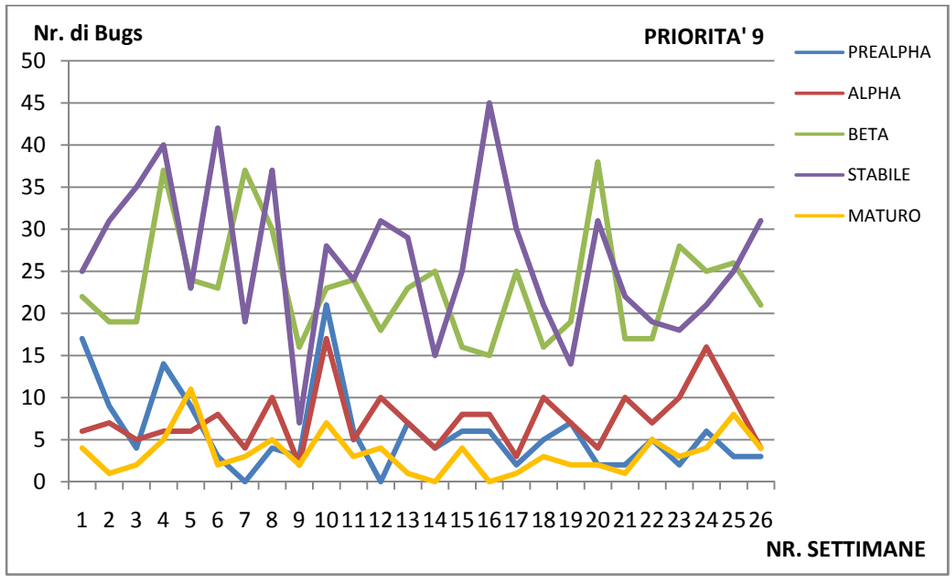


Figura 16 *Numero di bugs scoperti di priorità 9 nel semestre*

Il grafico di Figura 16, l'ultimo di questa serie, mostra i bugs con il più alto livello di criticità, quindi quelli che possono compromettere la stabilità del sistema oppure la sua sicurezza, oltre a non far funzionare il software stesso. I bugs scoperti con questa criticità mostrano un comportamento lievemente differente dal solito, poiché prima di tutto non hanno la solita distribuzione in base al livello di stabilità, inoltre si nota come i bugs dei prodotti stabili non sono superiori come al solito rispetto a quelli di livello beta, al contrario in alcuni punti sono inferiori. Stesso comportamento si riscontra con i bugs dei progetti di livello prealpha e alpha che, pur essendo numericamente minori di quelli dei "fratelli" di livello superiore, tendono ad assumere un andamento non predominante di uno rispetto all'altro, fatta eccezione per l'ultima parte del semestre. Inoltre, si nota sempre che i prodotti dichiarati maturi soffrono anche in questo caso di problemi molto

critici. L'andamento differente notato in questo grafico può essere ricondotto al tipo di problematiche che introducono i bugs di criticità nove. Infatti, come detto in precedenza, rappresentando il massimo grado di problema che può affliggere un software, questi sono facilmente riscontrabili in tutti i gradi di stabilità di un prodotto e, di conseguenza, saranno segnalati tempestivamente. Per contro, un bugs di criticità molto bassa può passare inosservato o non essere del tutto scoperto, quindi non segnalato.

5.2.3 Studio Dei Bugs Non Assegnati

Dopo aver esaminato i bugs scoperti, ora verranno analizzati i difetti scoperti, ma che non sono assegnati ad uno sviluppatore del team per essere risolti, rimanendo quindi aperti senza soluzione.

STABILITA'	PRIORITY 1	PRIORITY 2	PRIORITY 3	PRIORITY 4	PRIORITY 5	PRIORITY 6	PRIORITY 7	PRIORITY 8	PRIORITY 9
PREALPHA	693	305	524	177	23608	253	622	287	595
ALPHA	1455	1770	1108	382	48340	498	1123	521	852
BETA	2886	1210	2299	805	101876	1446	2423	1130	2180
STABILE	5169	1775	3207	1150	154317	1801	3423	1432	2652
MATURO	1180	1932	698	227	26083	265	416	190	261

Tabella 2 *Totale dei bugs non assegnati agli sviluppatori*

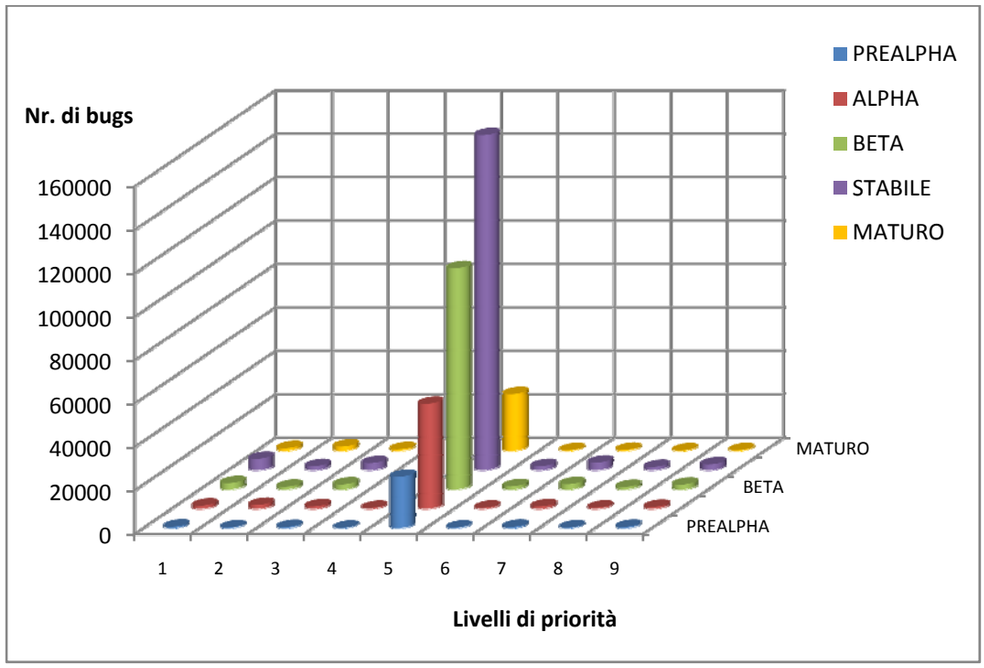


Figura 17 *Totale dei bugs non assegnati agli sviluppatori*

Come si nota dalla Figura 17, anche in questo caso, i bugs di livello cinque, che non sono stati assegnati, sono predominanti rispetto agli altri e questo non permette di effettuare uno studio approfondito dei dati raccolti. Però, permette nuovamente di notare l'andamento registrato dall'analisi precedente relativo alla distribuzione dei bugs rispetto ai livelli di stabilità dei progetti, dove per stabilità più basse si hanno meno bugs non assegnati a nessun sviluppatore rispetto a quanto si ha per stabilità più alte. Anche in questo caso, si nota che i bugs non assegnati di livello maturo sono numericamente minori, ma di quest'ultima considerazione si proverà a dare una spiegazione pertinente dopo aver esaminato il grafico successivo che mostrerà gli stessi dati, eliminando preventivamente i bugs di priorità cinque.

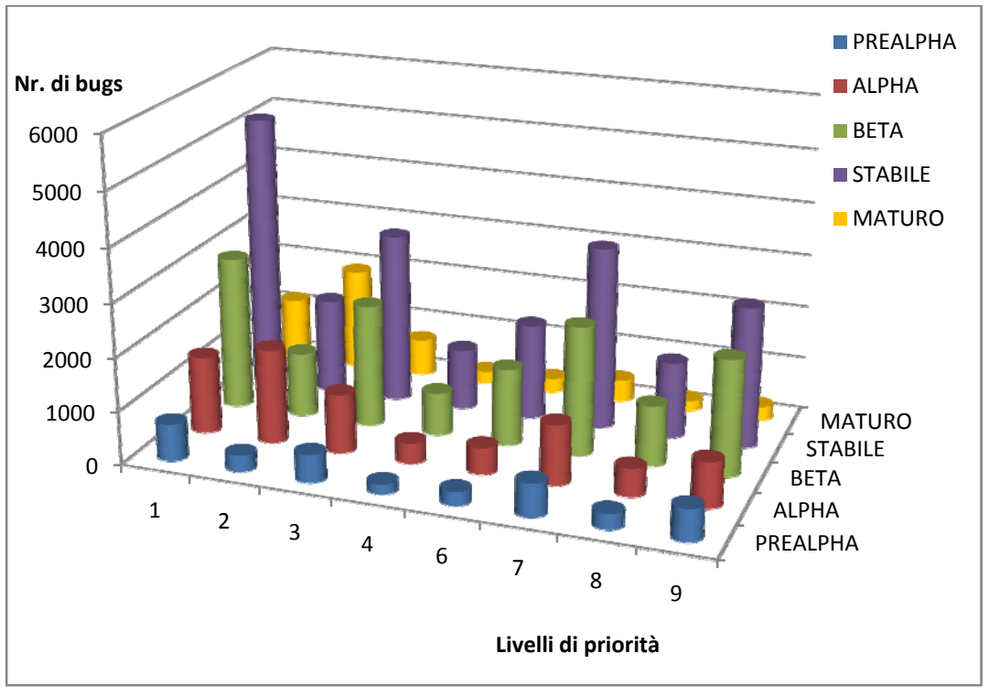


Figura 18 *Totale dei bugs non assegnati agli sviluppatori esclusa la priorità 5*

La Figura 18, come detto in precedenza, mostra la distribuzione dei bugs non assegnati agli sviluppatori divisi per livelli di criticità e per stabilità dei prodotti. La prima considerazione da fare è quella per i progetti maturi, dove si nota che sono numericamente inferiori rispetto agli altri. Ciò si può spiegare ricordando quanto notato dalla Figura 7, dove i bugs totali scoperti relativi ai progetti maturi sono minori degli altri, quindi essendo minori quelli scoperti, saranno minori anche quelli che non verranno assegnati. E' bene precisare però che ci si aspetta per i bugs dei progetti maturi un alto numero di assegnazione agli sviluppatori per poterli risolvere, perché un progetto di questo livello dovrebbe garantire stabilità e sicurezza. Continuando lo studio, si evidenza anche la stessa distribuzione vista nell'analisi precedente, dove a stabilità più basse si riscontrano un numero di bugs non assegnati minori. Inoltre, in questo caso vi è la stranezza che riguarda i livelli di priorità dispari (1,3,7,9), dove si notano un numero di bugs

più alto rispetto ai livelli pari (2,4,6,8) e questo è un comportamento difficilmente comprensibile.

5.2.4 Studio Dei Bugs Non Assegnati Nel Semestre

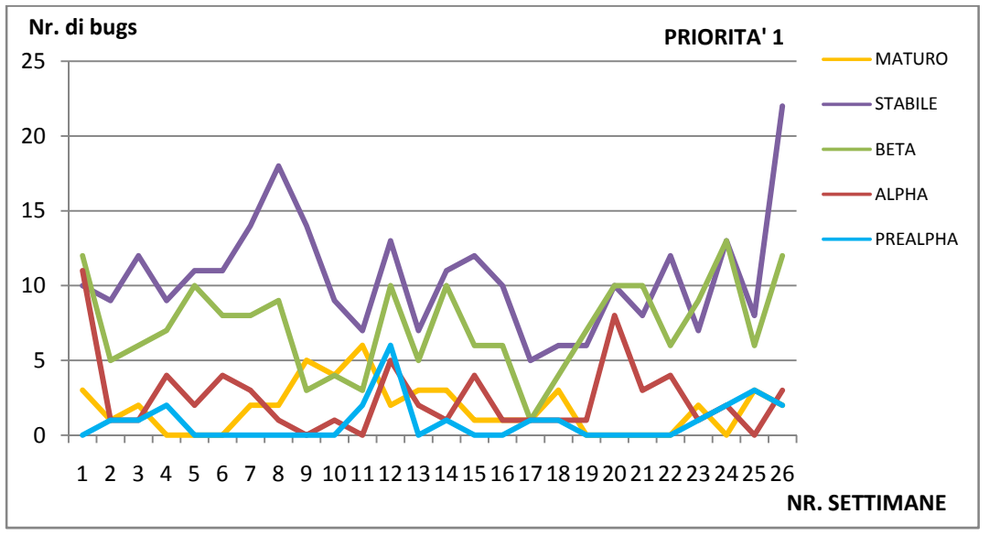


Figura 19 *Bugs priorità 1 non assegnati agli sviluppatori nel semestre*

La Figura 19 mostra i bugs di livello uno non assegnati a nessun sviluppatore. Si nota che i livelli di stabilità tendono a distribuirsi secondo il consueto ordine (dal prealpha allo stabile), ma non in modo netto, come è stato più evidente nell'analisi precedente dei bugs scoperti, bensì in questo caso le linee tendono ad intrecciarsi maggiormente, rilevando una diversa improduttività o indifferenza da parte degli sviluppatori verso i bugs di questo livello.

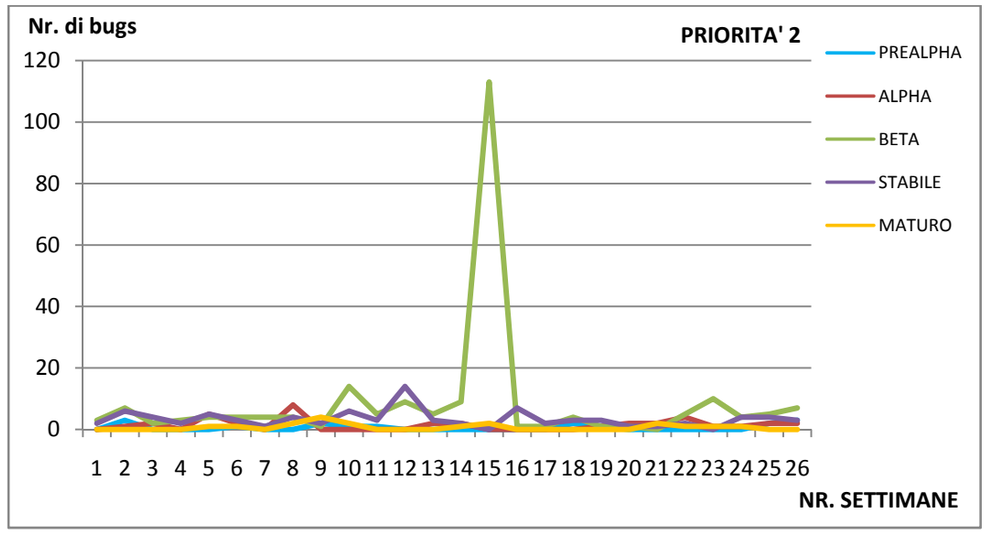


Figura 20 *Bugs priorità 2 non assegnati agli sviluppatori nel semestre*

Nel grafico di Figura 20 si ritrova quanto visto nel grafico di Figura 9 con un andamento dei bugs relativamente costante ma con un picco nella quindicesima settimana. Quindi è facile comprendere che i bugs scoperti in quella settimana sono rimasti non assegnati a nessun sviluppatore e quindi in attesa di soluzione.

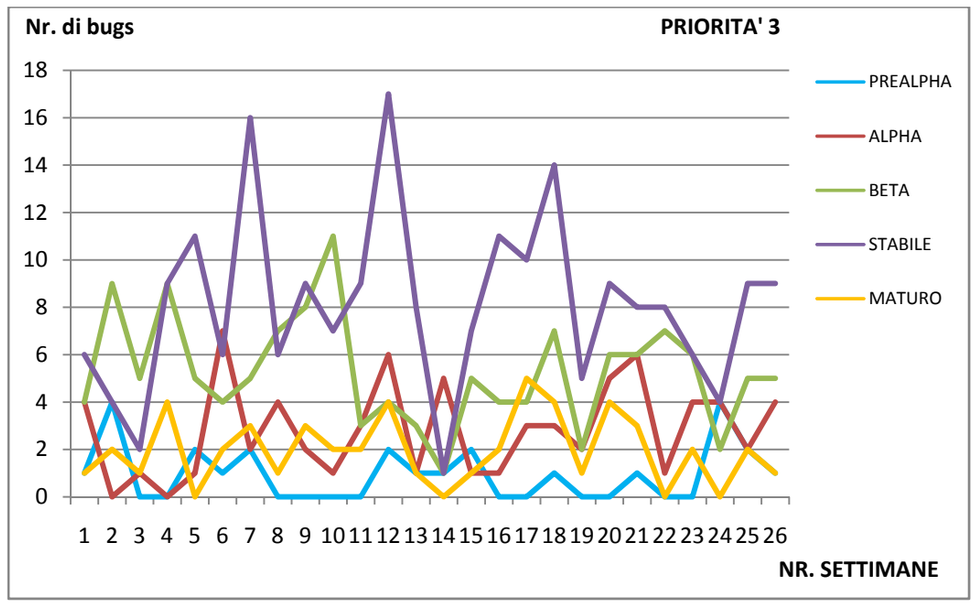


Figura 21 *Bugs priorità 3 non assegnati agli sviluppatori nel semestre*

La Figura 21 mostra una predominanza dei bugs non assegnati solo per i prodotti di livello stabile, mentre gli altri livelli si attestano su un fattore di grandezza non supeiore a dieci. Perciò, considerando che l'analisi tratta l'intero database di SourceForge per un periodo relativamente lungo di sei mesi, il numero di bugs non assegnati sono numericamente pochi quindi si può considerare l'attività delle comunità abbastanza alta.

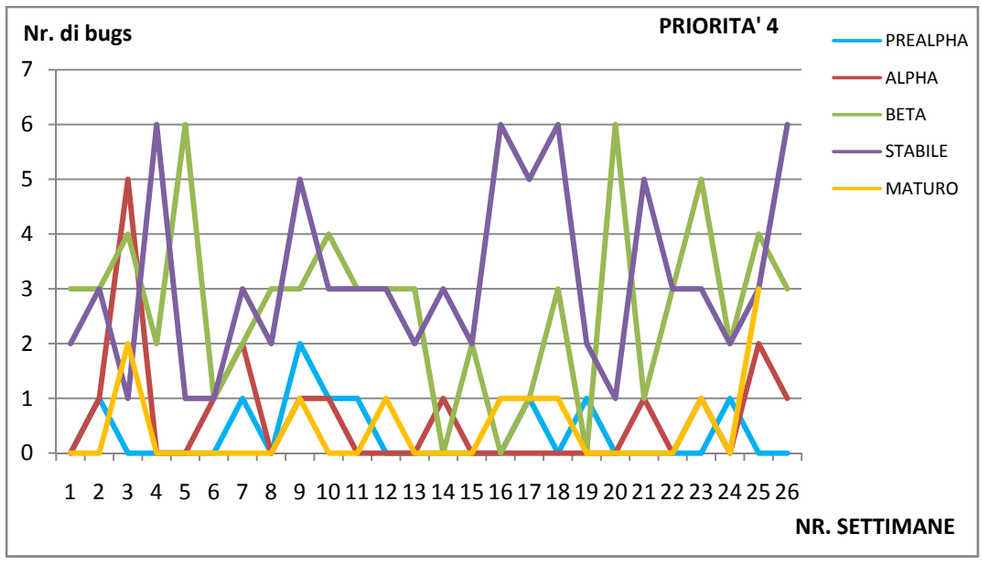


Figura 22 *Bugs priorità 4 non assegnati agli sviluppatori nel semestre*

I bugs di priorità quattro, rappresentati nel grafico di Figura 22, sono di facile interpretazione in quanto il livello più alto raggiunto dalle due serie dei progetti beta e stabile toccano il punto sei, mentre le serie che rappresentano i prodotti prealpha, alpha e maturo al massimo hanno un bug non assegnato e quindi possono essere considerati quasi uguali a zero.

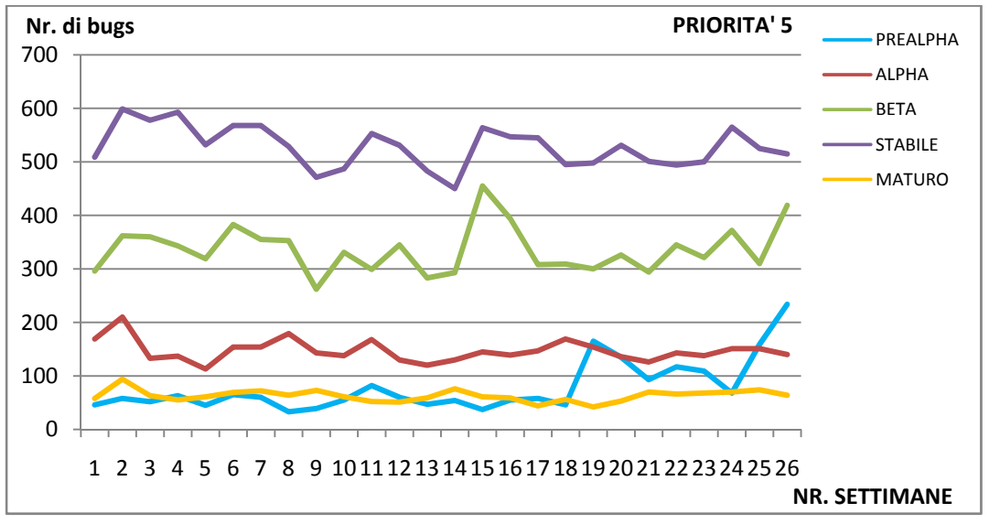


Figura 23 *Bugs priorità 5 non assegnati agli sviluppatori nel semestre*

A differenza di quanto visto fin ora, la Figura 23, relativa ai bugs di priorità cinque, mostra un andamento diverso dai grafici dei bugs non assegnati, ma simili a quelli dei bugs scoperti. Si noti prima di tutto l'ordine di grandezza che è circa cento volte più grande, inoltre è netta la differenza tra i livelli di stabilità di prodotti, mentre quelli di livello maturo continuano a mantenersi sul livello più basso. Anche questo grafico denota come il comportamento di default di SourceForge influisce molto nella scelta di assegnazione. Ciò può essere interpretato considerando che l'utente che scopre il bug nella maggior parte dei casi non fa parte del team di sviluppatori e normalmente non è in grado di capire che livello di criticità assegnare al problema, perciò si fida delle impostazioni automatiche che lasciano il bug con priorità cinque, facendo crescere enormemente solo questo grado di priorità. Ma questo espone i prodotti ad un controllo superficiale dei bugs scoperti perché gli sviluppatori, non potendo discriminare "a vista d'occhio" i problemi critici, potrebbero preferire continuare lo sviluppo del progetto piuttosto che controllare periodicamente la scoperta di problemi critici.

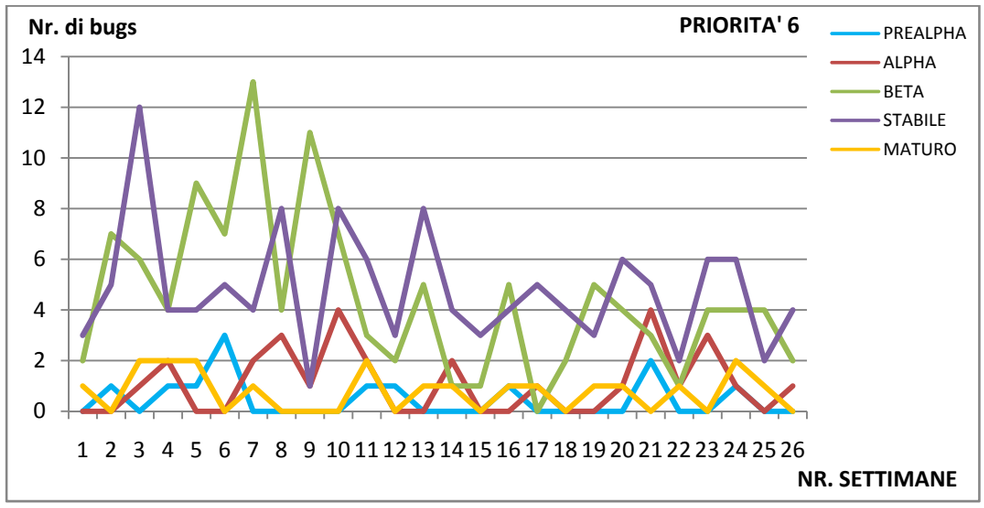


Figura 24 *Bugs priorità 6 non assegnati agli sviluppatori nel semestre*

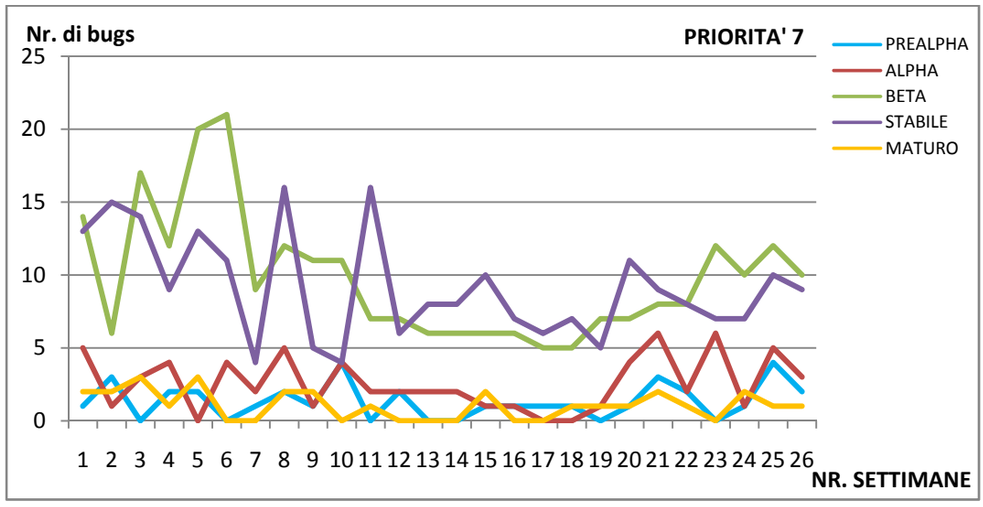


Figura 25 *Bugs priorità 7 non assegnati agli sviluppatori nel semestre*

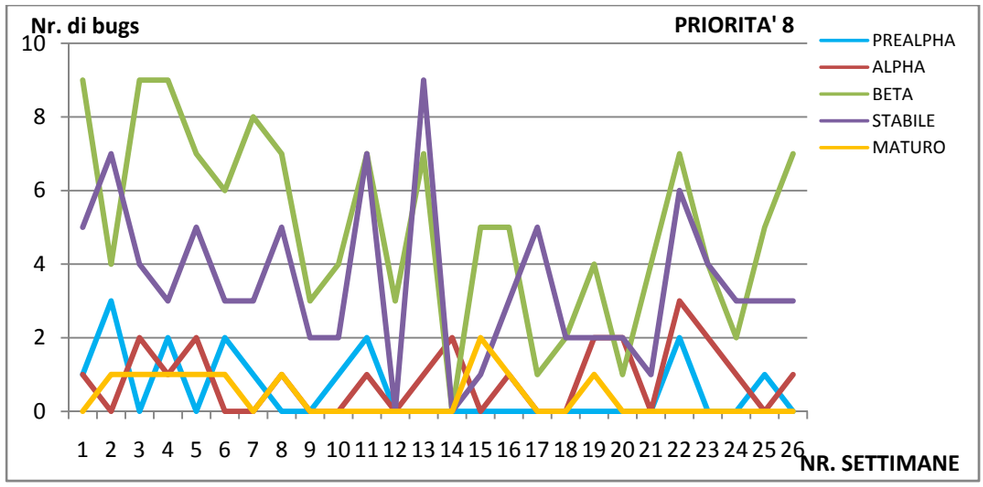


Figura 26 *Bugs priorità 8 non assegnati agli sviluppatori nel semestre*

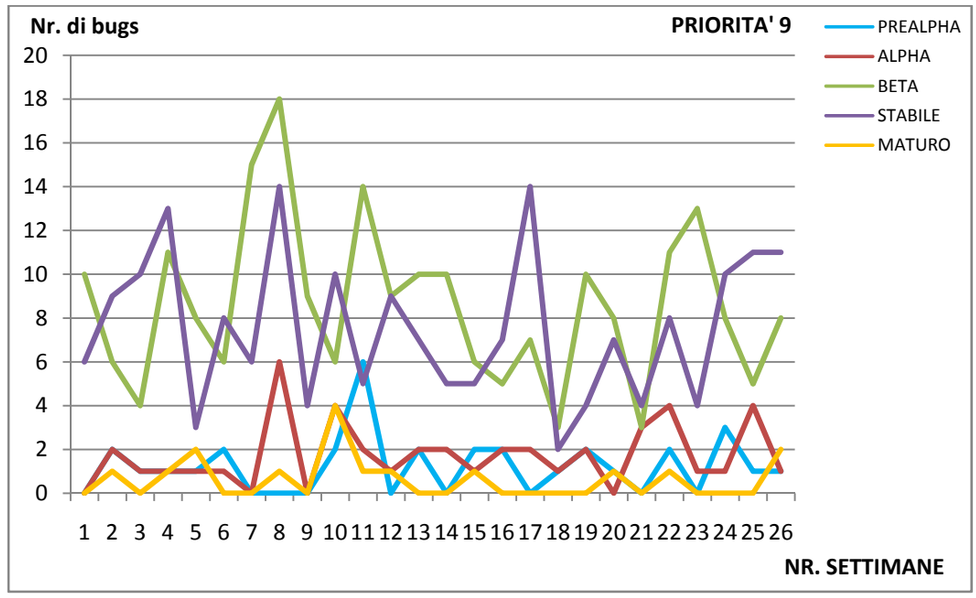


Figura 27 *Bugs priorità 9 non assegnati agli sviluppatori nel semestre*

I grafici di Figura 24,Figura 25,Figura 26 eFigura 27 mostrano rispettivamente i bugs di priorità sei, sette, otto e nove scoperti e non assegnati agli sviluppatori. Sono analizzati insieme perché presentano lo stesso andamento e si differenziano solo per la differenza minima del numero dei bugs. Come si può notare, i difetti riguardanti i prodotti beta e stabile, sono numericamente maggiori di quelli dei progetti prealpha, alpha e maturi, anche se questi ultimi nel peggiore dei casi non superano il valore cinque, poco rilevante considerata la mole dei progetti di SourceForge. Inoltre, si nota un'altra diversità con l'analisi fatta dei bugs scoperti, perché le serie che rappresentano i progetti beta e stabile, sono sovrapposti e non si distaccano fra di loro denotando nessuna prevalenza da parte delle comunità a preferire un tipo di progetto rispetto ad un altro. Queste due serie sono anche numericamente maggiori rispetto alle altre tre, ma anche questo può essere attribuito ad una casualità statistica, perché come visto in precedenza i bugs scoperti (in valore assoluto) di questi livelli sono maggiori e di conseguenza è più probabile che anche quelli non assegnati siano superiori.

5.2.5 Studio Dei Bugs Assegnati Agli Sviluppatori

Dopo aver analizzato i bugs scoperti, nonché, tra questi, quali non sono stati assegnati agli sviluppatori per poter essere risolti, studiamo ora i bugs che sono stati assegnati ai team di sviluppo e che quindi a differenza degli altri possono essere corretti. L'approccio usato è uguale a quello utilizzato negli altri casi, e nello specifico analizzeremo prima la totalità dei difetti che sono stati riscontrati ed affidati ad uno sviluppatore e, successivamente, l'andamento dei bugs nel semestre preso in considerazione.

STABILITA'	PRIORITY 1	PRIORITY 2	PRIORITY 3	PRIORITY 4	PRIORITY 5	PRIORITY 6	PRIORITY 7	PRIORITY 8	PRIORITY 9
PREALPHA	399	284	553	274	2283	386	781	399	664
ALPHA	591	496	859	457	3597	657	1225	668	992
BETA	1093	873	1577	899	6579	1263	2278	1289	1912
STABILE	1583	1239	2093	1128	8200	1636	2899	1645	2388
MATURO	249	226	310	185	1068	239	382	236	304

Tabella 3 *Totale dei bugs assegnati agli sviluppatori*

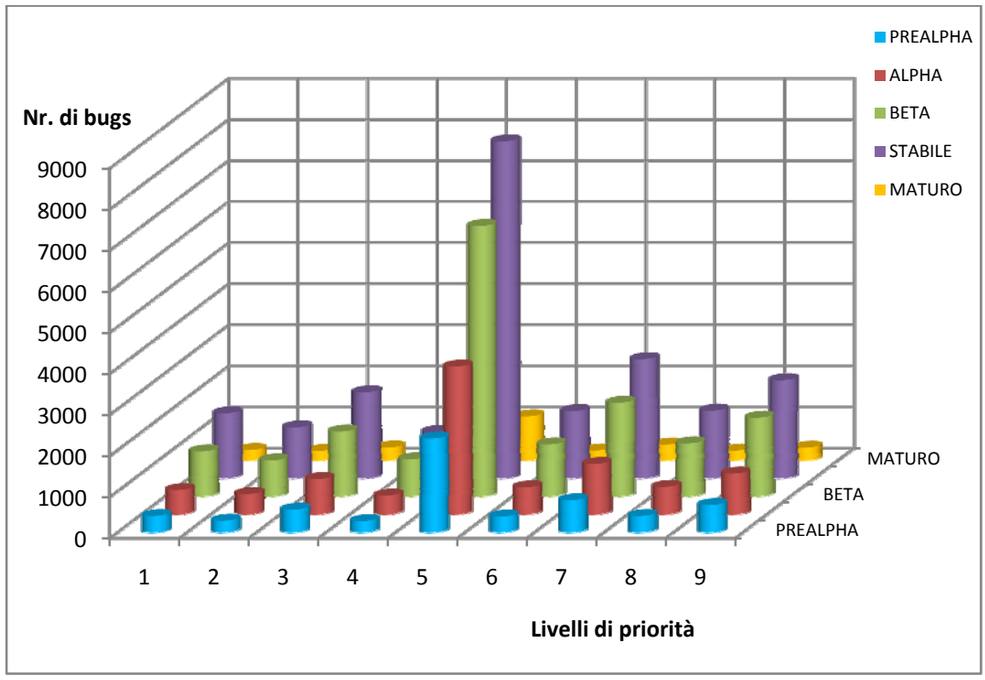


Figura 28 *Totale dei bugs assegnati agli sviluppatori*

Il grafico di Figura 28 mostra il totale dei bugs assegnati agli sviluppatori e si nota immediatamente, anche in questo caso, la predominanza dei difetti di criticità cinque con un fattore di grandezza di circa 10, anche se in percentuale minore rispetto alle analisi viste in precedenza. Viene inoltre rispecchiata la tendenza che vuole assegnazioni maggiori man mano che si sale di stabilità nei progetti e un livello più basso per i prodotti maturi. Per effettuare un'analisi più accurata, anche questa volta si ripropone lo stesso grafico eliminando la serie che identifica la priorità cinque.

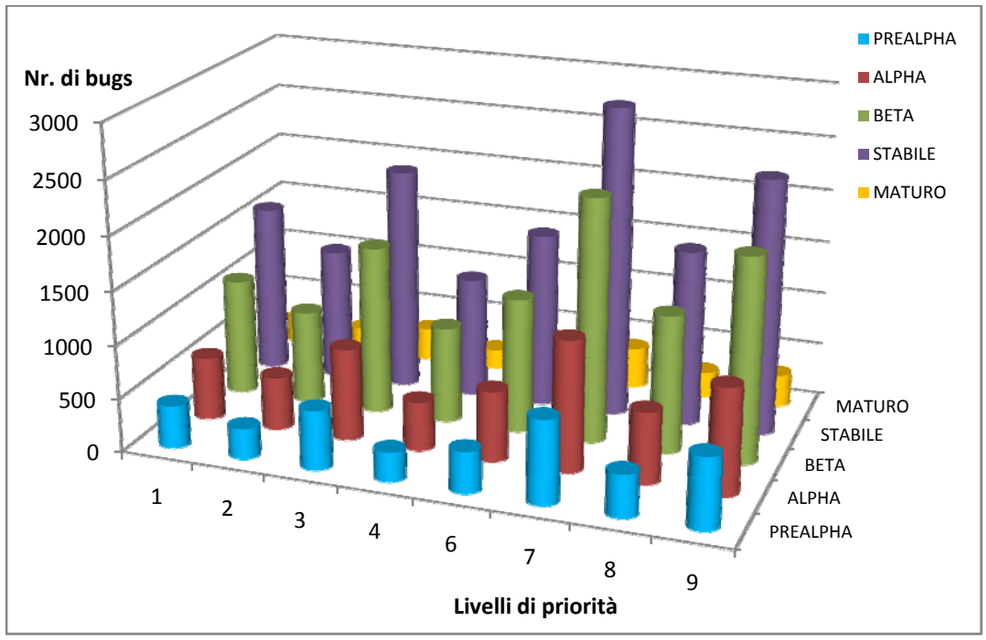


Figura 29 *Totale dei bugs assegnati agli sviluppatori senza la priorità 5*

Dalla Figura 29 si nota come gli sviluppatori tendono ad assegnare maggiormente i bugs relativi alle criticità più alte. Questa tendenza è di facile intuizione perché le criticità più alte rappresentano i problemi di sicurezza nonché quelli che potrebbero rendere instabile il sistema operativo ospitante. Non è comunque casuale la minoranza di problemi di livello nove rispetto a quelli di livello sette, perché i primi rappresentando problemi molto gravi si cerca di evitarli direttamente dalla progettazione del prodotto. E' strano notare che i bugs assegnati per la risoluzione dei progetti maturi siano minori di quelli rimasti senza assegnamento e questa tendenza è confermata anche per tutti gli altri livelli di stabilità. Questo può indicare una scarsa attività da parte delle comunità di sviluppo, l'incapacità dei team di focalizzare e quindi risolvere i problemi che si presentano oppure l'impossibilità di riuscire a gestire proficuamente la grossa mole di problemi che gli utenti scovano e propongono ai team. Quest'ultima opzione, che potrebbe sembrare un falla di questa gestione dei bugs e dei progetti open source in generale, invece rappresenta un punto di forza, di superiorità dei prodotti open rispetto ai prodotti commerciali perché,

essendoci una condivisione immediata anche dei problemi riscontrati, si arriva inevitabilmente ad uno sviluppo fornito anche da persone esterne al team che ha creato il progetto. Proviamo a chiarire meglio questo punto con un esempio: Supponiamo che un determinato progetto venga scelto da una azienda “X” per il suo lavoro quotidiano. Nell’utilizzo da parte degli utenti si scopre un bugs che viene immediatamente segnalato. A questo punto può accadere che il difetto non venga corretto velocemente, quinidi l’azienda “X” che ha adottato il software si trova di fronte a tre scelte: la prima è quella di non fare nulla ed aspettare che il team risolva il problema; la seconda è quella di comunicare il difetto agli utenti che utilizzano il software per informarli dell’anomalia e permettergli di non incappare nell’inconveninte oppure la terza è quella di utilizzare il “Know-how” dei propri tecnici cercando di risolvere il difetto e condividere il risultato anche con gli sviluppatori che hanno creato il programma. Questa terza opzione porta al miglioramento del prodotto ed alla risoluzione dei problemi molto più velocemente di quanto possa essere fatto con un progetto che utilizza software closed source.

5.2.6 Studio Dei Bugs Assegnati Agli Sviluppatori Nel Semestre

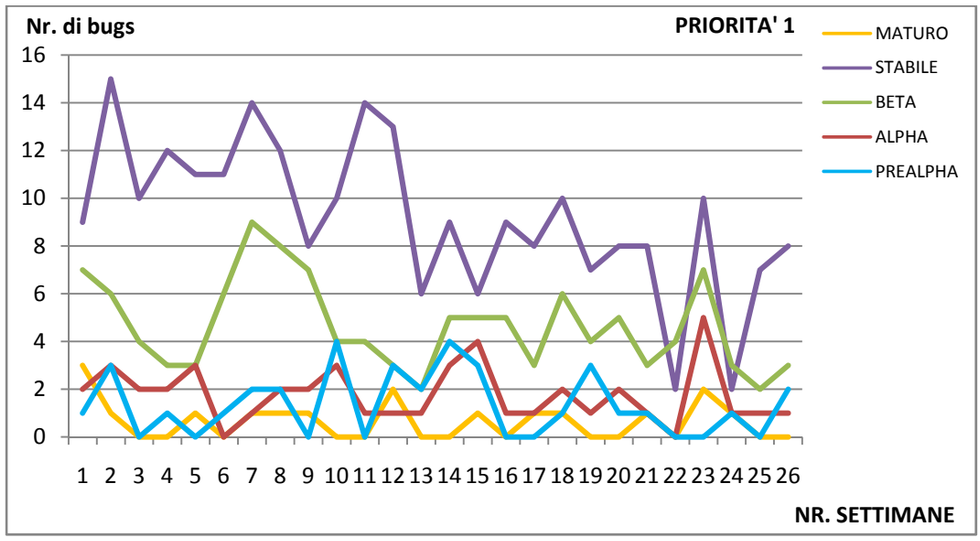


Figura 30 *Bugs di priorità 1 assegnati nel semestre*

La Figura 30 mostra l'andamento dei bugs assegnati per la risoluzione nel semestre preso in esame e, da uno studio iniziale, si nota la preferenza ad assegnare e quindi risolvere maggiormente i bugs dei progetti stabili. Di minore entità sono invece le assegnazioni fatte per i progetti appena nati e per quelli che hanno ormai raggiunto lo stato maturo. Da notare anche la quantità di bugs assegnati per i prodotti diversi da stabili che non superano mai i dieci bugs in un arco temporale relativamente lungo.

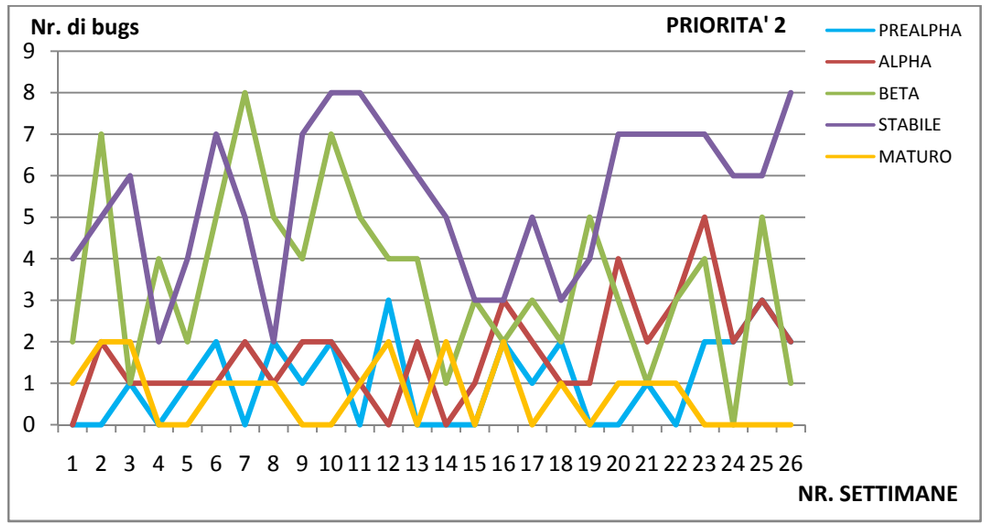


Figura 31 *Bugs di priorità 2 assegnati nel semestre*

La Figura 31, a differenza di quella precedente, mostra un tendenza simile sia per i bugs dei prodotti stabili che per quelli beta, andando quasi a mescolarsi nella parte centrale del periodo preso in esame. In questo caso le serie non superano mai le dieci unità, mostrando per questa priorità una bassa assegnazione.

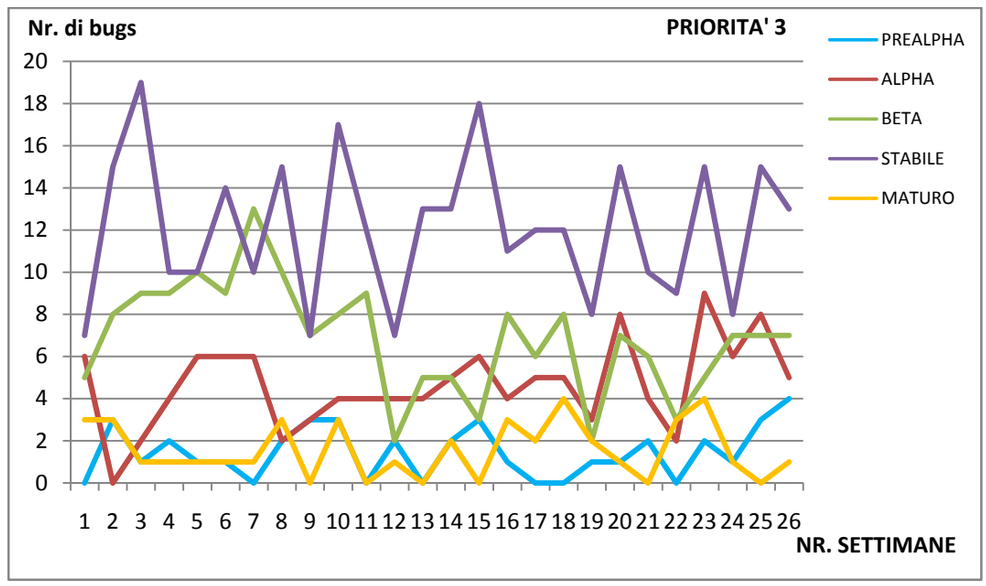


Figura 32 *Bugs di priorità 3 assegnati nel semestre*

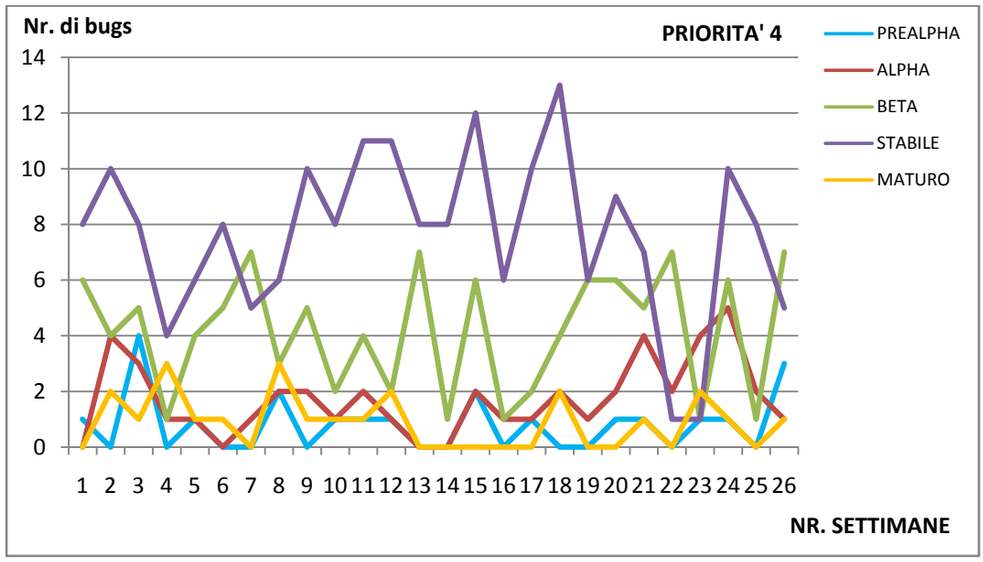


Figura 33 *Bugs di priorità 4 assegnati nel semestre*

I grafici di Figura 32 eFigura 33 sono mostrati insieme perché presentano le stesse caratteristiche e lo stesso fattore di grandezza. In questo caso i prodotti stabili sono predominanti sugli altri anche perché sono numericamente maggiori i bugs scoperti di questa categoria. I bugs assegnati dei proggeti prealpha, alpha e maturo possono essere considerati come un'unica serie, senza notare andamenti inaspettati, considerando sempre la bassa quantità di bugs che rappresentano in un periodo lungo.

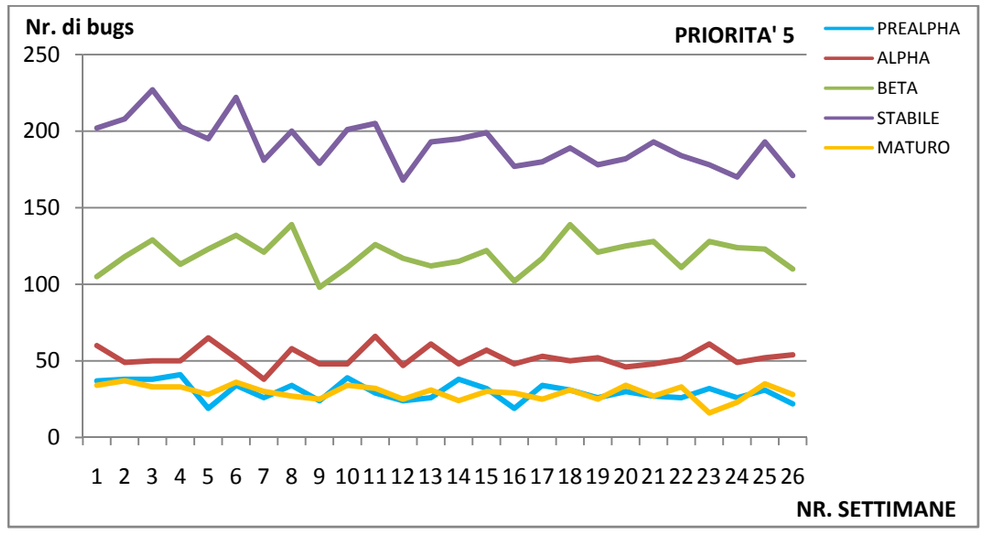


Figura 34 *Bugs di priorità 5 assegnati nel semestre*

Discorso a parte deve essere fatto per il grafico di Figura 34 che rappresenta i bugs assegnati per la risoluzione di priorità cinque. Questa priorità, come al solito, ha il massimo valore numerico di difetti coinvolti (ricordiamo che di default i bugs vengono assegnati con questa priorità). Si nota immediatamente la separazione netta dei bugs relativi alla stabilità dei progetti, così come è stato mostrato in precedenza, dove ogni serie ha un fattore di grandezza preciso, fatta eccezione per i prodotti prealpha e maturo che condividono la stessa area del grafico e che sono staccati di poco dai progetti alpha. Inoltre, la tendenza dei bugs assegnati dei prodotti beta tende a rimanere stabile a differenza di quella dei progetti stabili che, verso la fine del periodo in esame, tende a descrescere, anche se ciò può essere un caso non imputabile a qualcosa di specifico. E' strano comunque notare come i bugs assegnati dei prodotti maturi non superino mai quelli dei progetti prealpha, quando invece ci si sarebbe aspettato l'opposto in quanto, una volta che un progetto viene dichiarato maturo, ci si dovrebbe aspettare da parte del team di sviluppo un controllo continuo dei problemi di questi prodotti per lasciare il software in uno stato di "forte" stabilità.

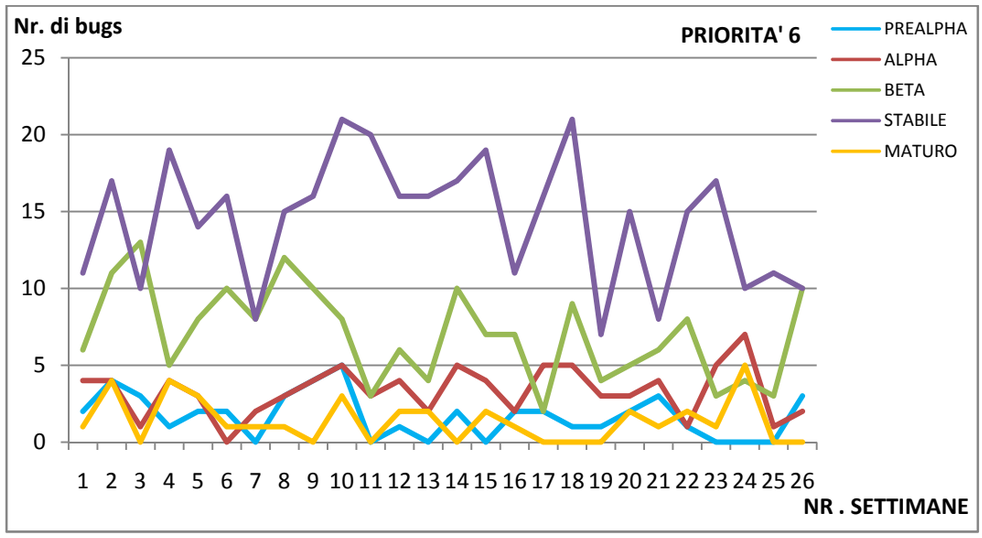


Figura 35 *Bugs di priorità 6 assegnati nel semestre*

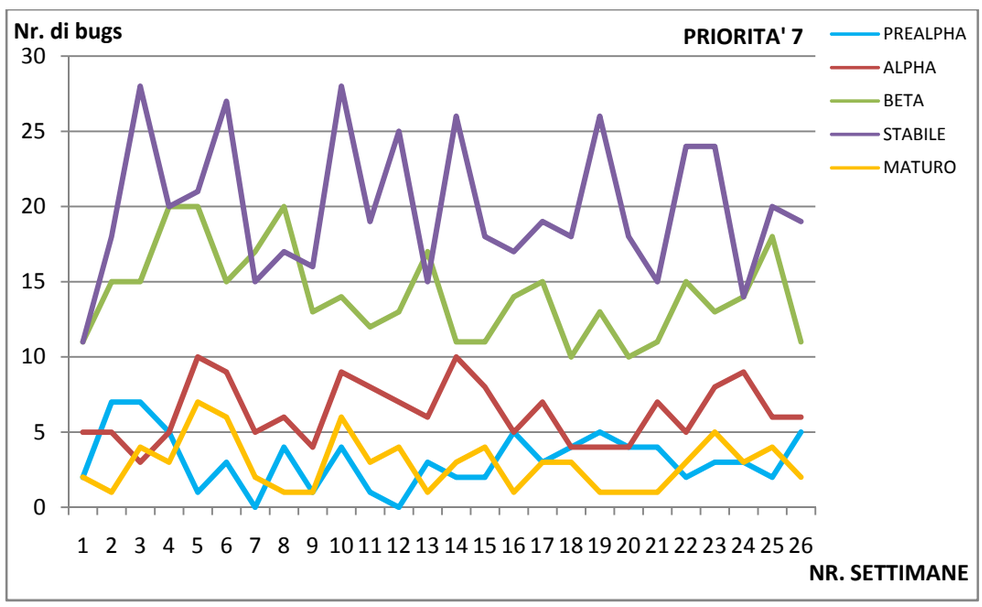


Figura 36 *Bugs di priorità 7 assegnati nel semestre*

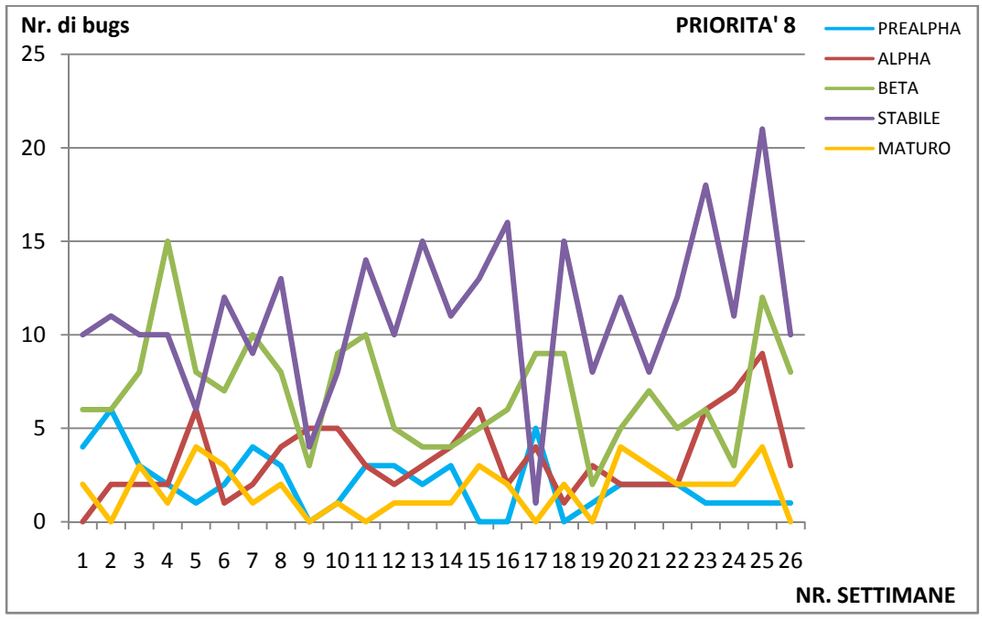


Figura 37 *Bugs di priorità 8 assegnati nel semestre*

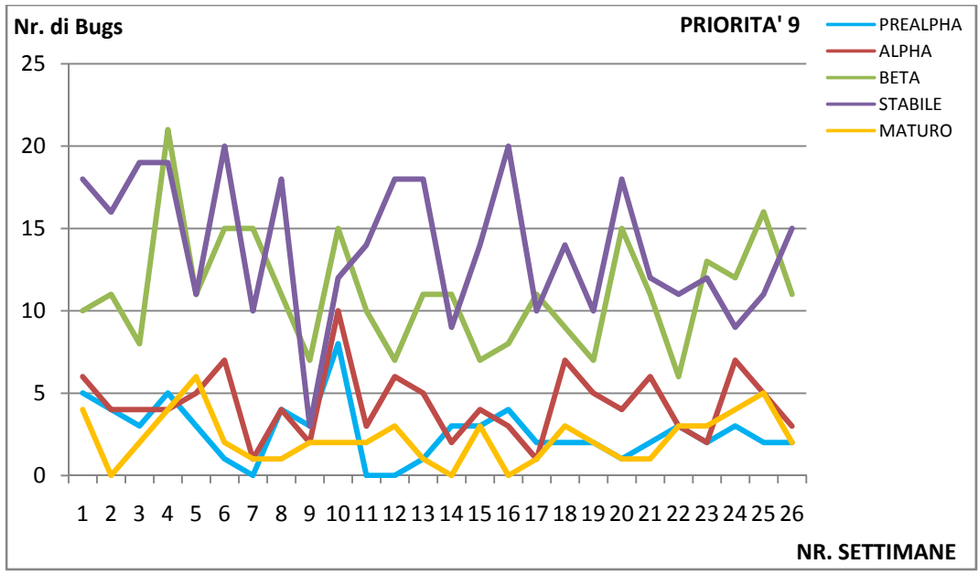


Figura 38 *Bugs di priorità 9 assegnati nel semestre*

Anche in questo caso, si è deciso di analizzare unitamente i grafici di Figura 35,Figura 36,Figura 37 eFigura 38 rispettivamente dei bugs assegnati di priorità sei, sette otto e nove, perché non presentano grosse differenze fra di loro. Innanzitutto gli ordini di grandezza sono simili e, nel migliore dei casi, non superano i venti bugs assegnati. Uguale è anche la differenza tra i progetti prealpha, alpa e maturo

disponendosi nella parte bassa del grafico, a differenza di quelli dei progetti beta e stabile che si proiettano su valori di poco superiori. Ma, abbiamo visto in precedenza che questo è da imputare esclusivamente ad una questione statistica (numero maggiore di bugs relativi a questi prodotti). L'unica novità che si rileva è quella della Figura 38 dei bugs di priorità nove, dove tra la settima e la dodicesima settimana si ha una flessione dei bugs assegnati dei prodotti più stabili e la relativa crescita dei bugs assegnati per i progetti "giovani", stranezza che va risolvendosi con il passare del tempo riprendendo l'andamento abituale.

5.3 L'ANALISI DELLE PATCHES

L'analisi delle patch (pezze), mostra la capacità delle comunità di sviluppo a risolvere i problemi trovati e segnalati con i bugs. Si procederà con lo stesso sistema visto finora, analizzando prima la totalità delle patch rilasciate per risolvere i problemi dei progetti di tutto SourceForge, e successivamente si analizzeranno quelle relative al semestre 01 aprile 2007 - 29 settembre 2007.

5.3.1 Studio delle Patches

STABILITA'	PRIORITY 1	PRIORITY 2	PRIORITY 3	PRIORITY 4	PRIORITY 5	PRIORITY 6	PRIORITY 7	PRIORITY 8	PRIORITY 9
PREALPHA	64	21	52	14	6364	21	67	17	91
ALPHA	190	136	161	54	11387	88	136	72	135
BETA	376	112	226	169	24035	206	310	157	325
STABILE	983	320	516	210	43920	370	632	253	474
MATURO	177	144	336	274	14859	197	250	93	125

Tabella 4 *Totale delle patches rilasciate*

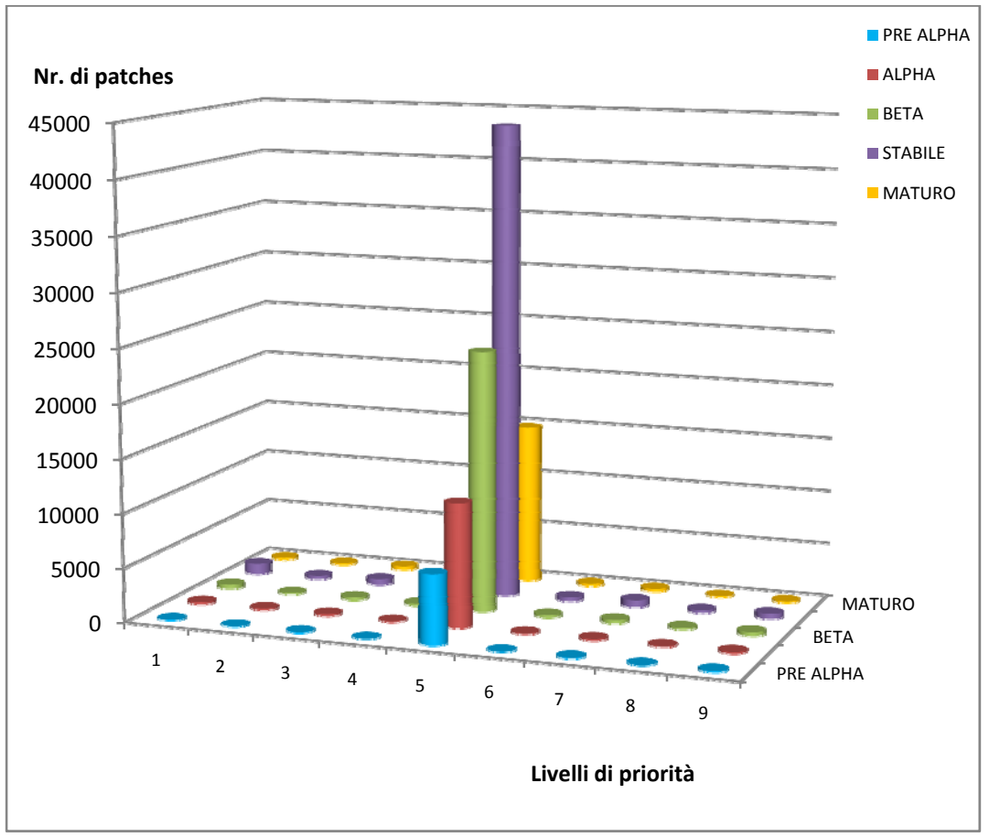


Figura 39 *Totale delle patches rilasciate*

Il grafico di Figura 39 mostra la totalità delle patches che sono state rilasciate per tutti i progetti di SourceForge. Si nota, come negli altri casi la quasi unicità di patches di priorità cinque rispetto agli altri livelli di criticità, inoltre è confermata la distribuzione delle quantità rispetto ai livelli di stabilità dei progetti: distribuzione crescente per prodotti che vanno dal prealpha fino a quelli stabili, ma minori per i prodotti maturi. Al fine di analizzare più approfonditamente il grafico, anche questa volta lo si ripropone omettendo la serie centrale del livello di criticità cinque.

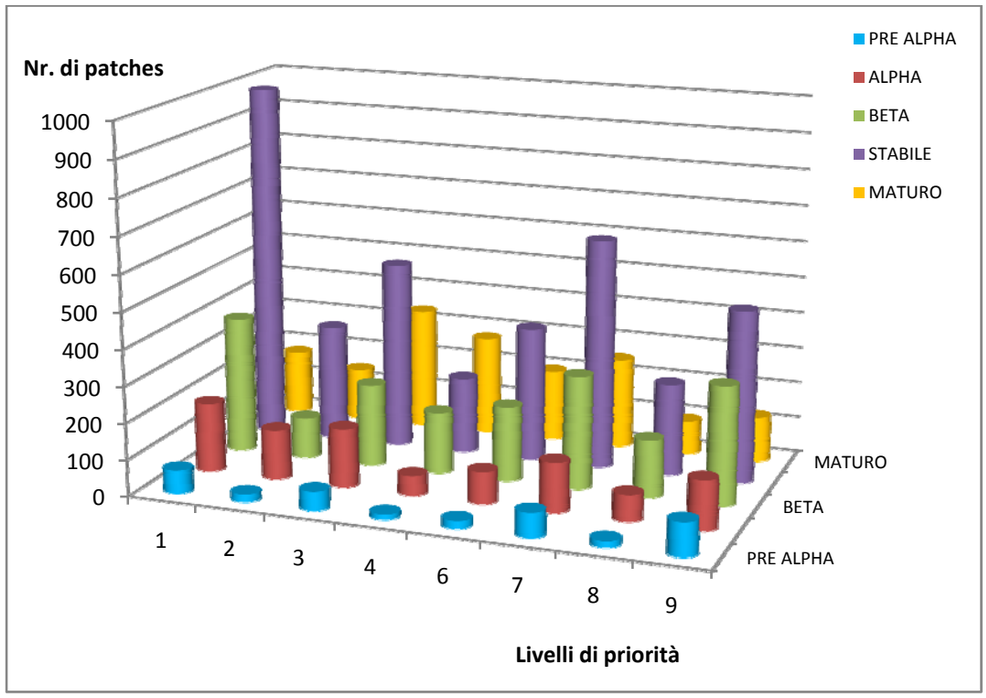


Figura 40 *Totale delle patches rilasciate senza la priorità 5*

La Figura 40, come si può notare, risulta estremamente più leggibile omettendo la serie di dati che mostrano le patches di priorità cinque, e questo ci permette di notare una distribuzione diversa da quelle viste fino a questo momento. Si noti che le patches rilasciate per i prodotti maturi sono numericamente maggiori di tutti gli altri, fatta eccezione per quelli di livello stabile, mentre sono minime quelle per i prodotti giovani: prealpha e alpha. Questo ci mostra che gli sviluppatori effettivamente tendono a risolvere prima i problemi sui progetti con una stabilità maggiore (comportamento che ci aspettavamo) e successivamente si “dedicano” a sviluppare soluzioni per i software appena nati. Da notare inoltre il basso numero di patches sviluppate per i progetti prealpha, il che può solo indicare la propensione da parte dei team nel cercare di far evolvere quanto più velocemente possibile i propri prodotti cercando di raggiungere prima stabilità maggiori senza rilasciare tante patches per progetti ancora immaturi. Se questa considerazione non fosse vera, avremmo dovuto trovare un

quantitativo di patches quasi omogeneo tra i vari livelli di prodotti e questo non è avvenuto. Inoltre, è da notare la quanitità enorme, rispetto al resto del grafico, di patches di priorità uno rilasciate per i prodotti di livello stabile; questo rappresenta quel grande numero di piccoli difetti che si possono riscontrare nei software, non critici, ma che possono infastidire l'utilizzatore del programma, si pensi ad esempio ad un pulsante che non funziona. Questi difetti sono di semplice risoluzione e vengono quindi prontamente eliminati. E' da precisare che questa stessa situazione l'avevamo notata nell'analisi dei bugs scoperti ma non assegnati (vedi Figura 18). Si può quindi dedurre che molte volte gli sviluppatori preferiscono risolvere direttamente il problema, senza curarsi dell'aspetto formale del bug tracking di SourceForge non preocupandosi di dover segnalare il difetto ad uno sviluppatore specifico. Se questa deduzione fosse verà però potrebbe mettere in cattiva luce i progettisti del programma nel caso di una valutazione formale di quel progetto.

5.3.2 Studio delle Patches Nel Semestre

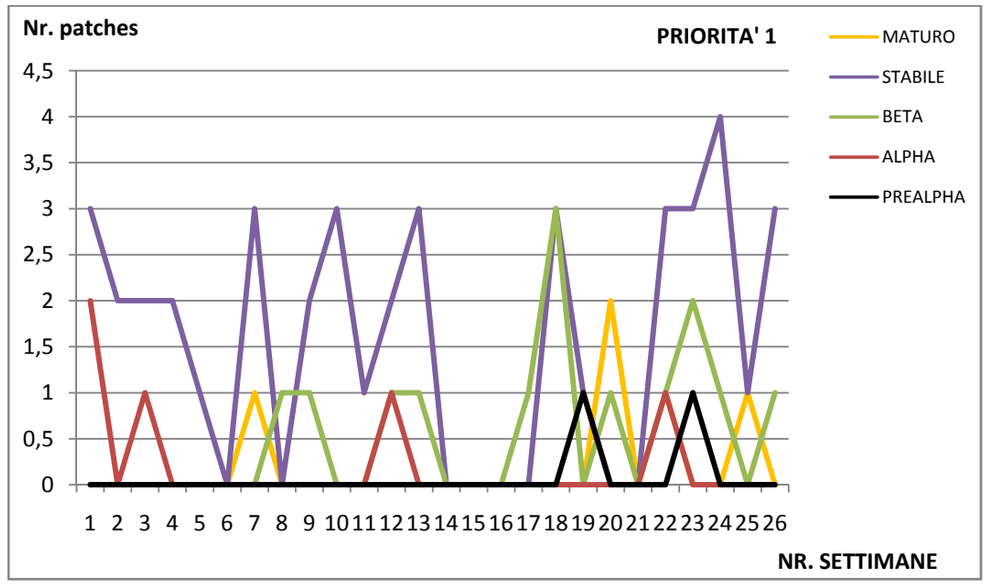


Figura 41 *Patches rilasciate nel semestre di priorità 1*

La Figura 41 delle patches rilasciate di priorità uno mostra una predominanza di patches rilasciate per i progetti stabili, confermando quindi la tendenza di questo comportamento e coincidendo con quanto studiato nell'analisi totale delle patches vista in precedenza. E' bene comunque precisare che, nell'arco temporale preso in esame (sei mesi), il valore assoluto delle patches non supera quota quattro, quindi è bene dire che la frequenza di questo tipo di correzione non è comunque alta. Non si esprimono commenti sugli altri livelli, perché sono numericamente irrilevanti.

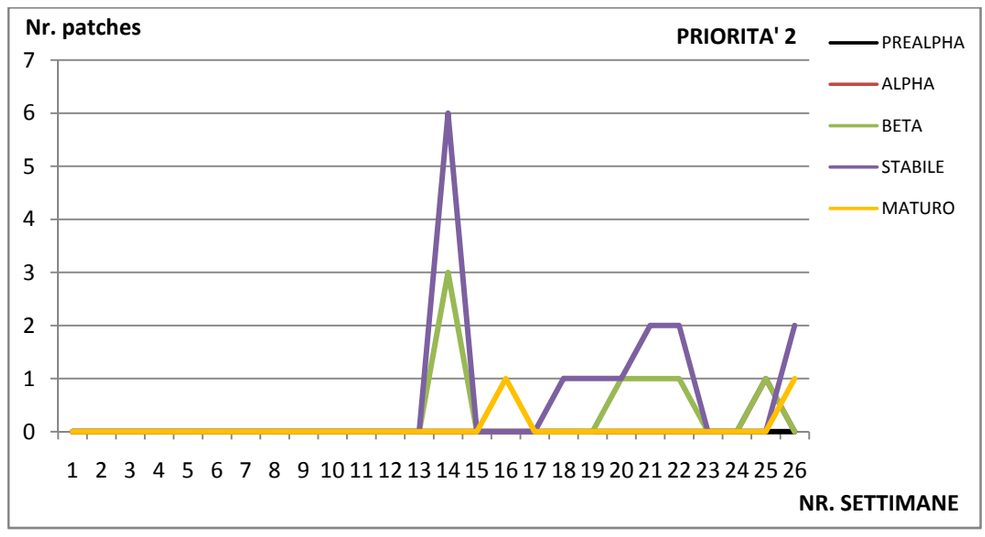


Figura 42 *Patches rilasciate nel semestre di priorità 2*

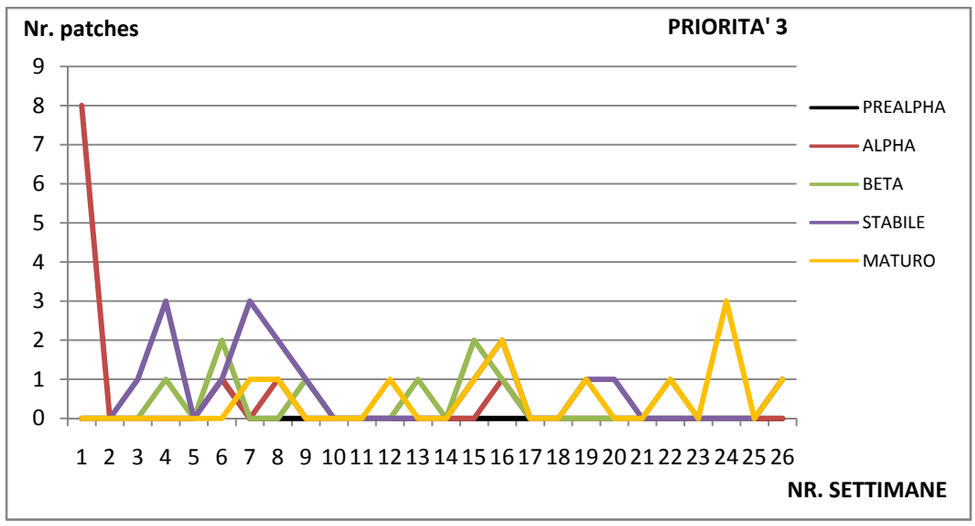


Figura 43 *Patches rilasciate nel semestre di priorità 3*

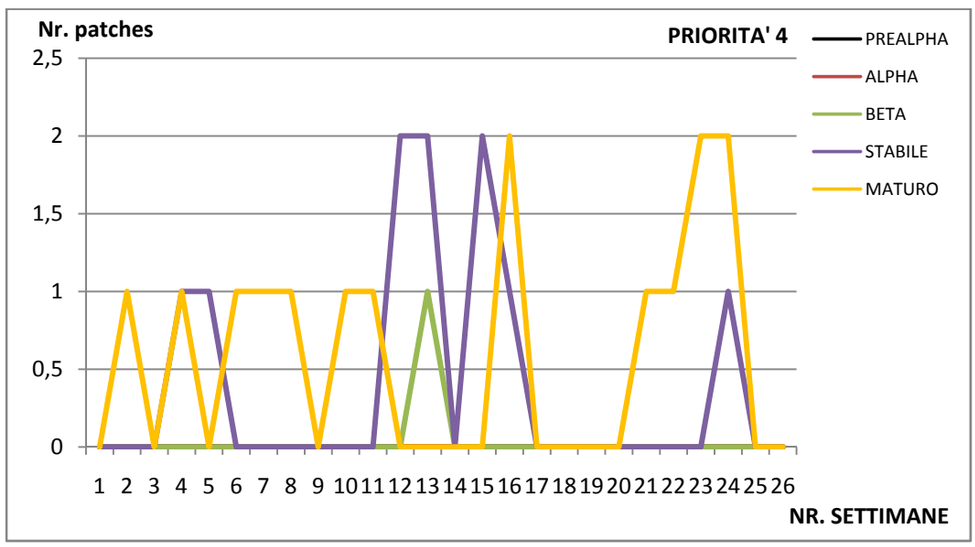


Figura 44 *Patches rilasciate nel semestre di priorità 4*

Anche in questo caso, si è deciso di mostrare congiuntamente i grafici di Figura 42,Figura 43 eFigura 44 relativi alle patches rilasciate rispettivamente per i livelli di priorità due, tre e quattro, in quanto mostrano un andamento quasi simile fatta eccezione per il grafico di priorità due per il quale non vengono completamente rilasciate patches per ben tre mesi consecutivi. E' da notare comunque la predominanza

di soluzioni per i software dichiarati maturi, confermando quindi la tendenza a risolvere maggiormente i problemi di questi prodotti.

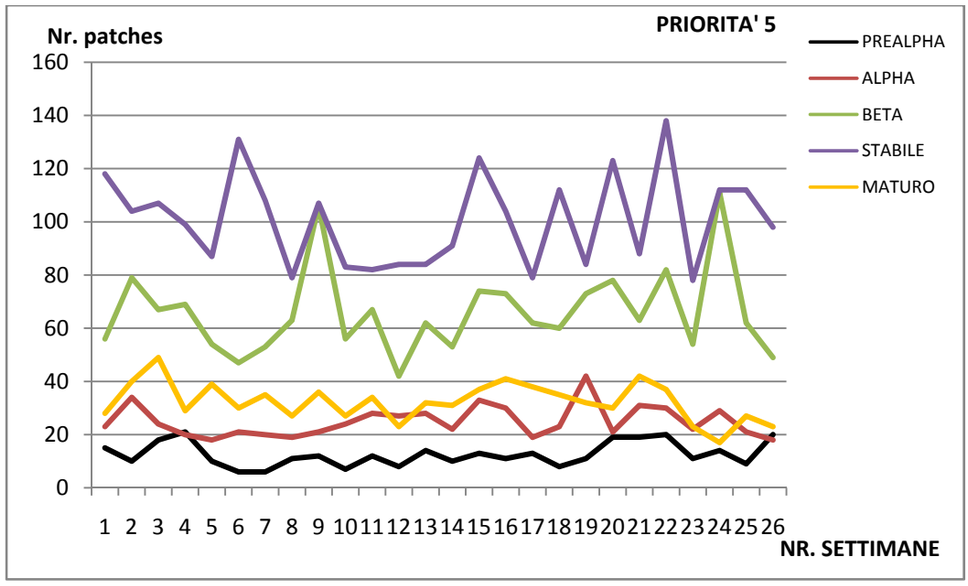


Figura 45 *Patches rilasciate nel semestre di priorità 5*

Il grafico di Figura 45 relativo alle patches rilasciate di priorità 5 per il semestre preso in esame mostra innanzitutto la differenza nel fattore di grandezza tra le varie serie, fissando come maggiori prodotti che usufruiscono di patches quelli di livello stabile e beta, ma, a differenza di quanto visto con i bugs, le soluzioni rilasciate per i prodotti maturi sono maggiori di quelli giovani, confermando l'analisi fatta sul totale delle patches di SourceForge. Ovviamente, questi ultimi sono numericamente minori in valore assoluto, perché anche i bugs rappresentano una minoranza di problemi per questi progetti.

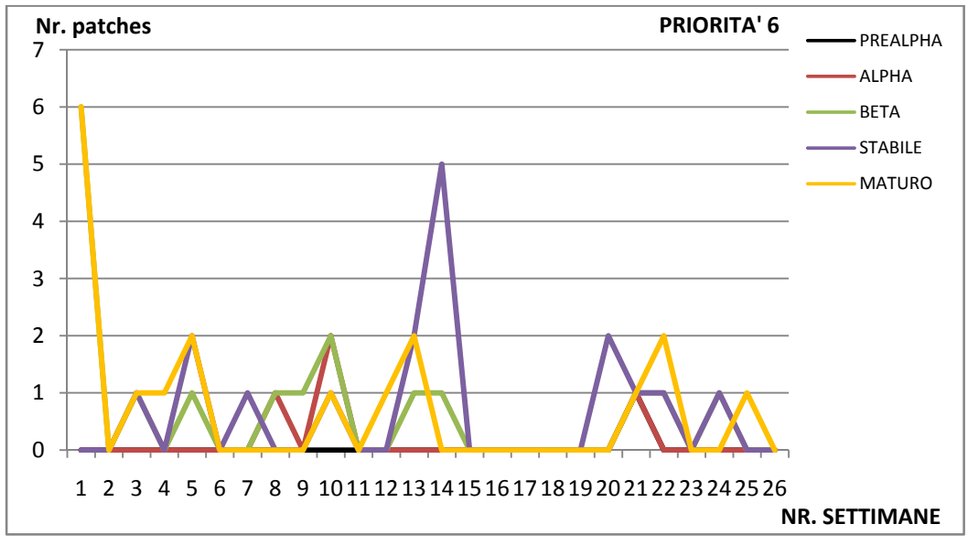


Figura 46 *Patches rilasciate nel semestre di priorità 6*

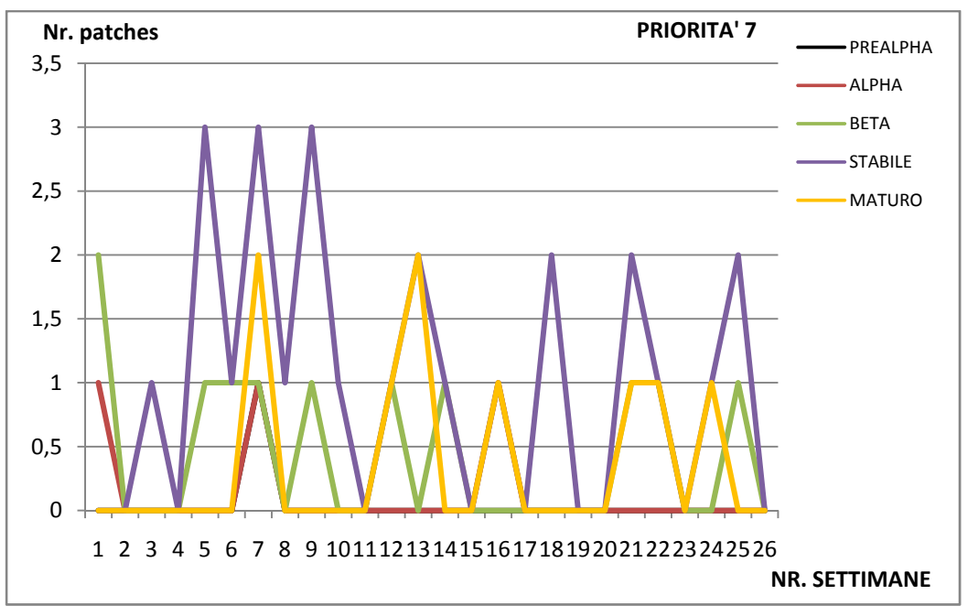


Figura 47 *Patches rilasciate nel semestre di priorità 7*

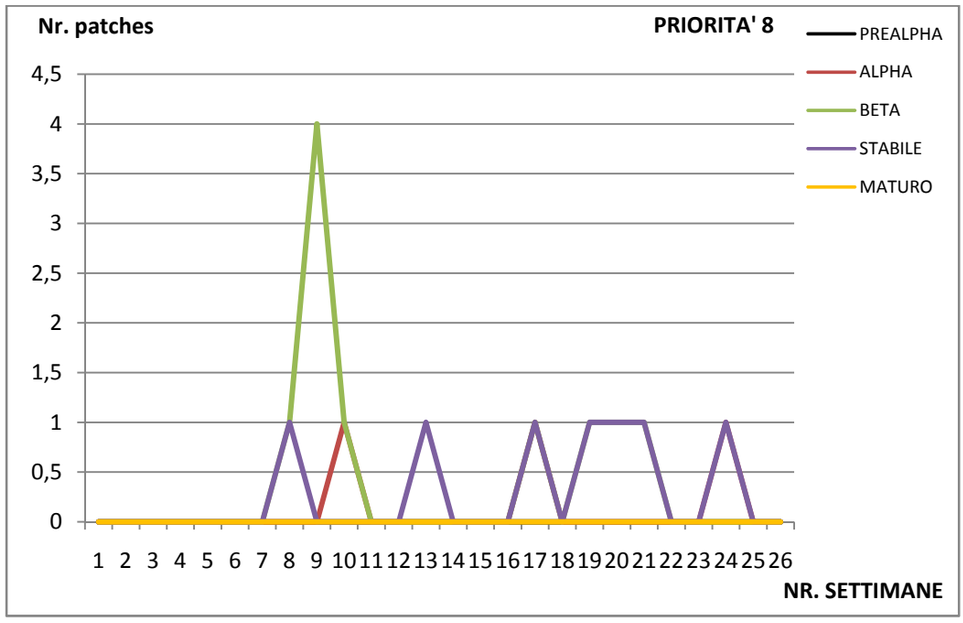


Figura 48 *Patches rilasciate nel semestre di priorità 8*

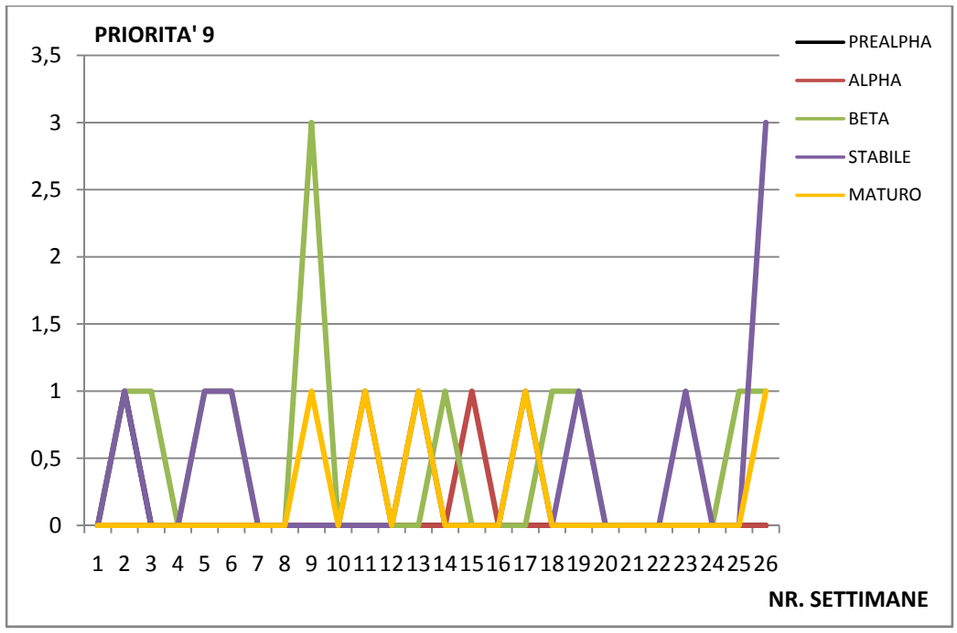


Figura 49 *Patches rilasciate nel semestre di priorità 9*

A causa del basso numero di patches rilasciate nel periodo preso in esame, anche i grafici di Figura 46,Figura 47,Figura 48 eFigura 49, rispettivamente delle patches rilasciate per le criticità sei, sette, otto e nove, sono mostrati congiuntamente. Da questi grafici non è possibile

fare delle analisi attendibili, perché ci dicono semplicemente che durante questo periodo periodicamente vengono rilasciate alcune patch per i vari prodotti, l'unica notazione degna di rilievo è quella sui prodotti appena nati (prealpha e alpha), per i quali non viene rilasciata nessuna patch. Questo comportamento può essere rappresentato pensando che le priorità più alte riguardano i problemi di sicurezza, quindi per i progetti "giovani" le comunità preferiscono non risolvere i problemi con le patches, ma utilizzando il sistema delle release. Quindi, si può supporre che le release dei progetti giovani siano relativamente maggiori.

5.4 L'Analisi Della Documentazione

La documentazione è stata analizzata seguendo lo schema ormai consueto di studiare in primo luogo tutto il materiale presente in SourceForge e successivamente cercando i dati relativi al semestre fissato come finestra per questo lavoro di tesi.

5.4.1 Studio della Documentazione

STABILITA'	NR. DOCUMENTI
PREALPHA	4246
ALPHA	4975
BETA	7692
STABILE	7804
MATURO	776

Tabella 5 *Documenti presenti in SourceForge divisi per stabilità di progetto*

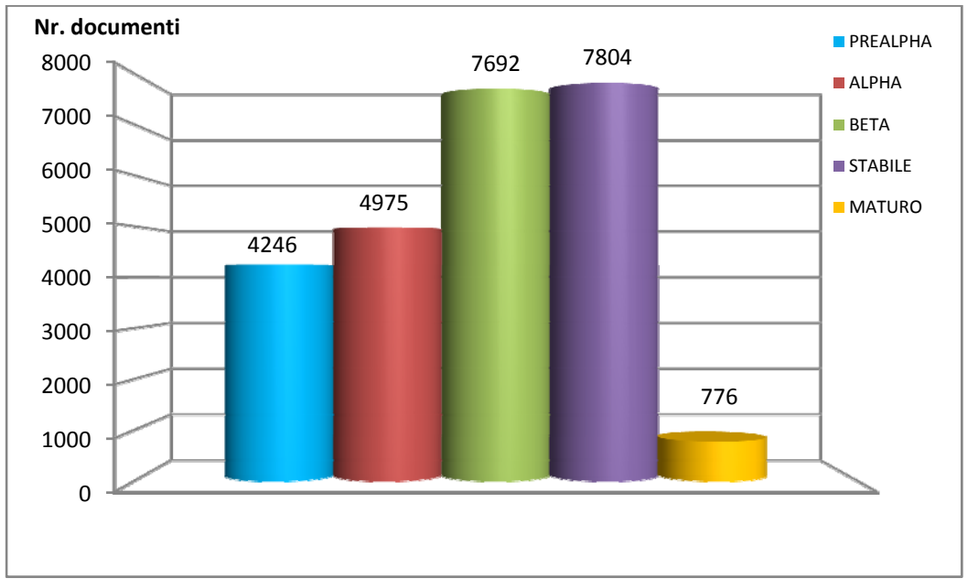


Figura 50 *Totale documenti inseriti in SourceForge*

La Figura 50 dei documenti inseriti dagli sviluppatori all'interno di SourceForge mostra ovviamente anch'essa la tendenza a rilasciare maggiore documentazione per progetti che non siano giovani, anche se ci si poteva aspettare un numero relativamente superiore di informazioni per progetti prealpha, in modo da spingere gli utenti a testare il nuovo software proposto. E' bene precisare che il grafico mostra la fotografia della documentazione presente per i progetti dichiarati con quel grado di stabilità in quel determinato momento. Questo significa che provando a rieseguire la query su dump diversi, avremmo come risultato un ovvio incremento sul numero assoluto di documenti, ma non si avrà una distribuzione molto differente da quella mostrata in figura perché la documentazione è collegata al progetto quindi, nel momento in cui il prodotto cambia il grado di stabilità, anche la documentazione risulterà relativa alla nuova stabilità raggiunta. Questa affermazione però non è congruente con la quantità di documenti presenti nel database relativi ai progetti maturi, perché ci saremmo dovuti aspettare una serie più alta delle altre, in quanto il processo produttivo dei software porta inevitabilmente tutti i prodotti

verso la maturità. Per spiegare questa incongruenza è stato necessario analizzare manualmente una grossa mole di questi progetti, e da qui è emerso che i team di sviluppo utilizzano SourceForge come meccanismo per produrre e sviluppare il software in modo centralizzato, introducendo poca documentazione e per la maggior parte relativa solo agli sviluppatori. Sfruttano inoltre la potenza di calcolo di SourceForge come mirror per il download del prodotto, ma pubblicizzano il programma mettendo a disposizione del pubblico la documentazione necessaria, maggiormente tramite il proprio sito web. Questo spiega anche l'eclatante risultato relativo ai pochi documenti in valore assoluto inseriti, confrontati con il numero di progetti gestiti da SourceForge (circa 130.000 al momento in cui si è effettuato lo studio) rappresentando appena il 10%.

5.4.2 Studio della documentazione nel semestre

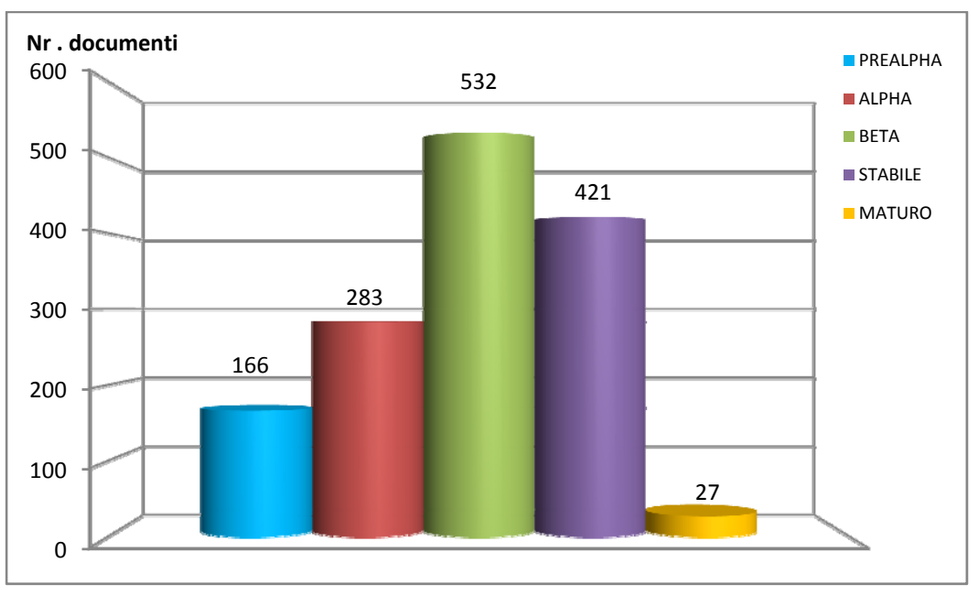


Figura 51 *Documenti inseriti nel semestre in esame*

Dopo aver analizzato i dati ottenuti dallo studio effettuato sulla documentazione di SourceForge, sono stati raccolti i dati della documentazione relativamente al consueto semestre studiato, ma questa volta, a causa del basso numero di dati raccolti, si è deciso di non mostrarli differenziandoli per ogni settimana, ma raccogliendoli in un unico grafico (vedi Figura 51). Il grafico mostra la stessa tendenza di quanto visto in precedenza e non introduce fattori che possano smentire l'analisi appena formulata.

5.5 L'ANALISI DEGLI SVILUPPATORI

Gli sviluppatori rientrano in quella fase di studio che rappresenta l'analisi della comunità e come questa interagisce tra le varie figure al fine di produrre un software di successo.

5.5.1 Studio degli Sviluppatori

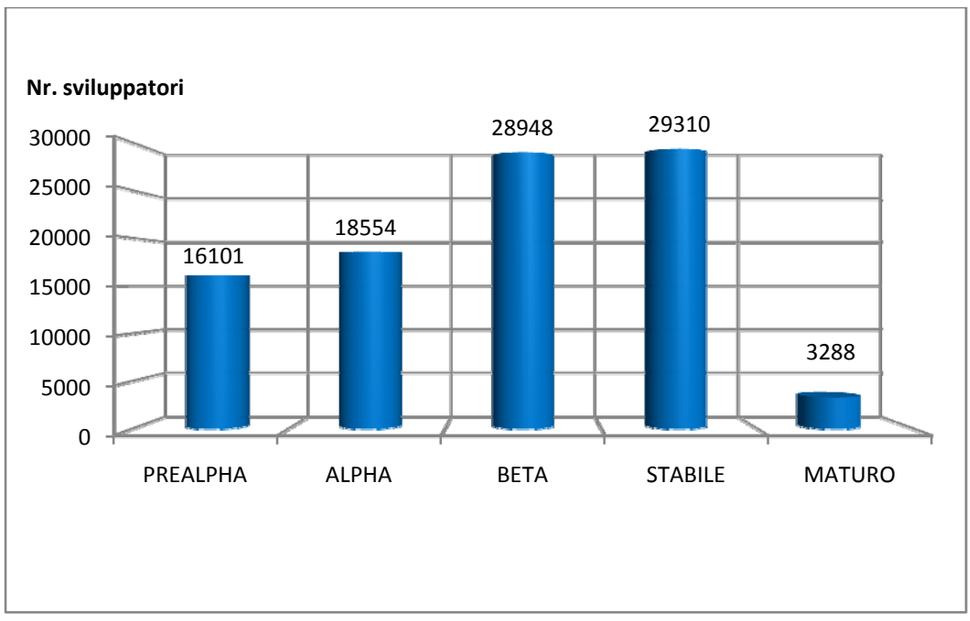


Figura 52 *Sviluppatori divisi per stabilità dei progetti*

La Figura 52 mostra come è distribuita la sola figura professionale degli sviluppatori (development) all'interno di SourceForge e quanti di loro si dedicano per lo sviluppo di uno specifico prodotto. Si noti sempre come la distribuzione risulti maggiormente spostata verso progetti beta e stabili, a differenza dei pochi professionisti che si dedicano ai progetti maturi e quindi solo alla correzione dei problemi di questi programmi.

In seguito sono state cercate tutte le figure professionali che ruotano attorno a SourceForge e come sono distribuiti fra di loro, vedi Tabella 6.

RUOLO MEMBRI	PREALPHA	ALPHA	BETA	STABILE	MATURO
Developer	16101	18554	28948	29310	3288
Project Manager	5274	5522	8044	7463	784
Unix Admin	216	198	322	282	20
Doc Writer	261	290	482	523	72
Tester	502	583	924	839	99
Support Manager	93	87	171	202	21
Graphic/Other Designer	591	408	375	298	52
Translator (I18N/L10N)	214	302	817	1190	139
Editorial/Content Writer	113	85	126	135	8
Packager (.rpm, .deb etc)	84	155	441	521	65
Analysis / Design	176	119	150	101	16
Advisor/Mentor/Consultant	258	294	480	462	61
Distributor/Promoter	35	74	83	89	9
Content Management	56	66	101	123	14
Requirements Engineering	35	28	60	59	9
Web Designer	478	468	623	594	72
Porter (Cross Platform Devel.)	73	119	180	247	49
Anonimus	21953	23924	31383	26352	2605
All Hands Person (Gestori)	1670	1603	2269	2045	233
Other	433	399	661	677	97
User Interface (UI) Designer	62	37	56	47	3
Support Technician	28	31	70	105	12
DBA (Database Administrator)	29	18	19	12	4

Tabella 6 *Ruoli di tutti i membri di SourceForge e distribuzione*

Come si può notare dalla tabella, le figure professionali di SourceForge si suddividono in ventitre compiti differenti, andando

così a coprire tutti i vari rami che possono servire per sviluppare appieno i prodotti. E’ da precisare che non ci si aspettava la presenza di alcune professioni come quella del promoter o del redattore di documenti, pensando che fossero compiti lasciati anonimi ed effettuati dallo stesso personale che si è occupato dello sviluppo del codice. Ciò dimostra la settorialità dei membri di sviluppo di un software.

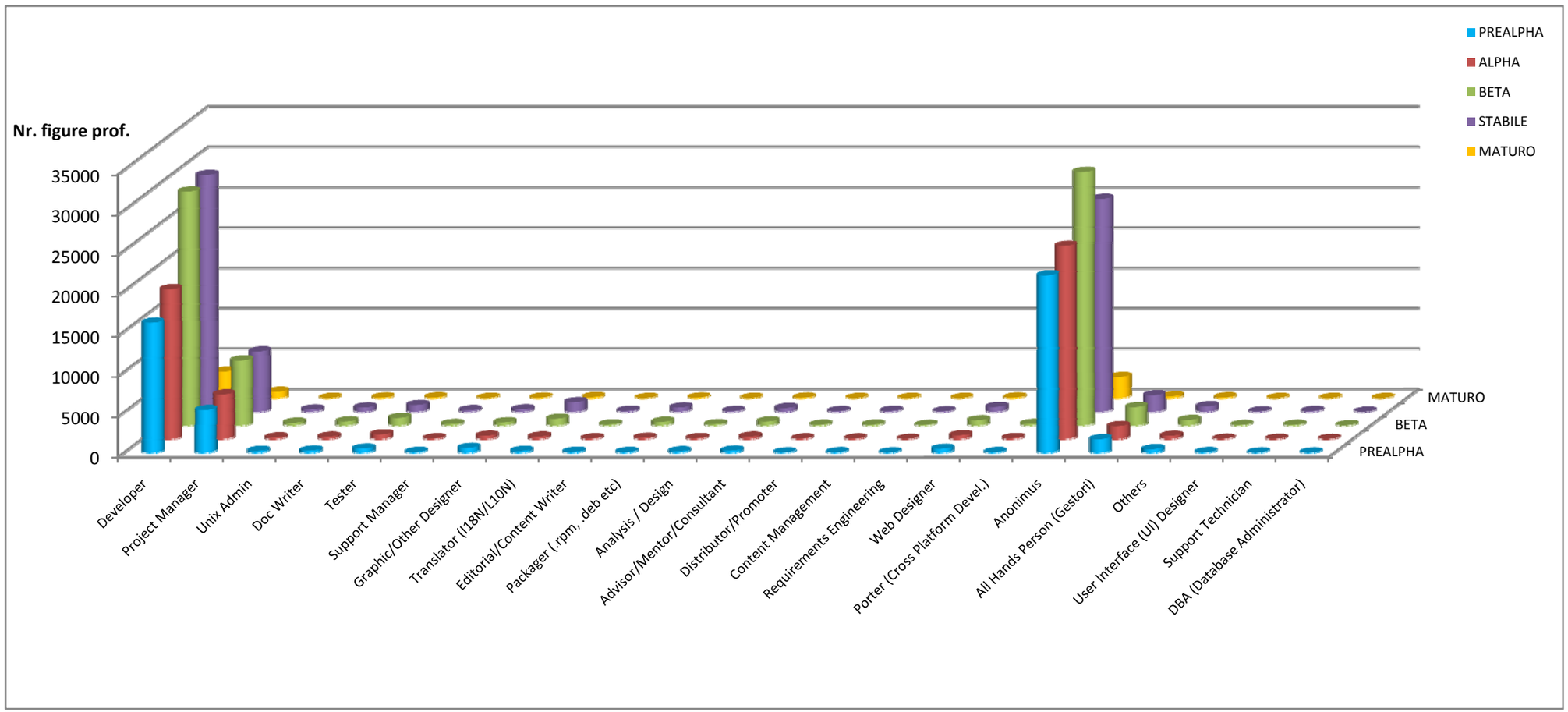


Figura 53 *Distribuzione di tutte le figure di professionali di SourceForge*

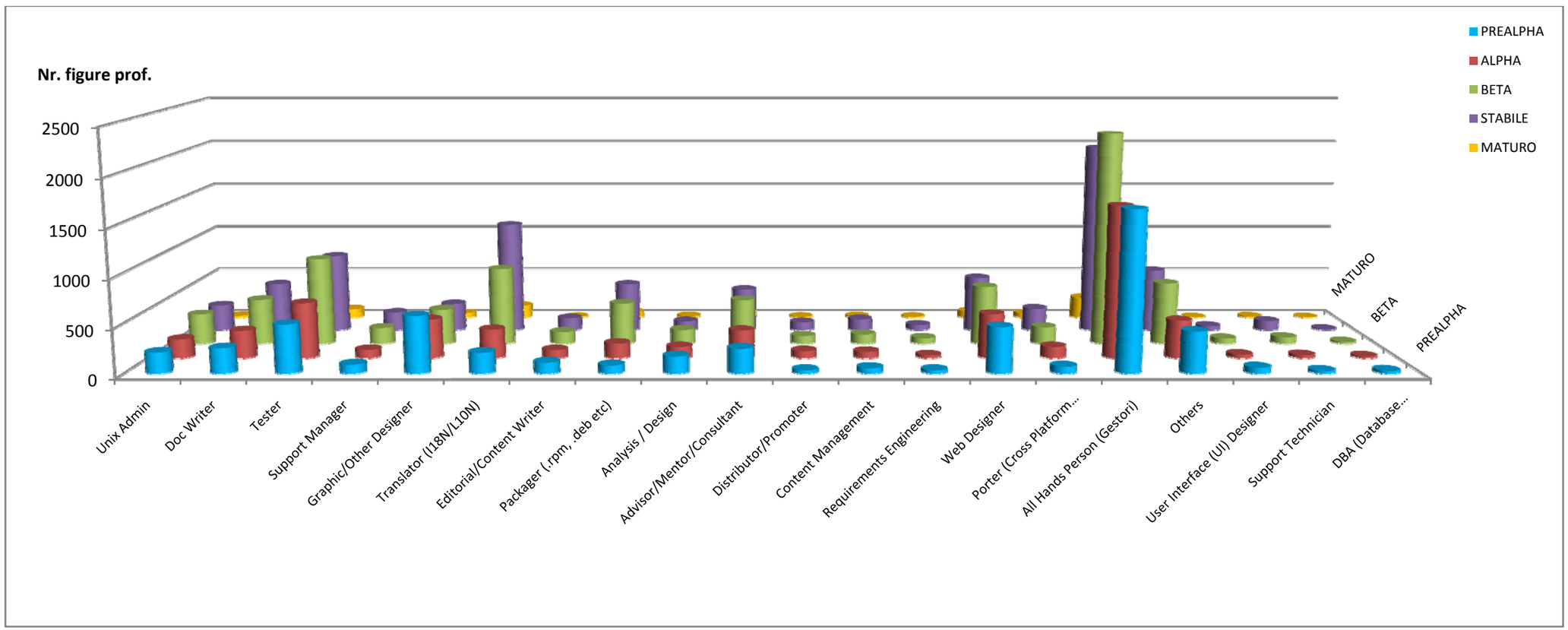


Figura 54 *Distribuzione di tutte le figure di professionali di SourceForge eliminando quelle predominanti*

La Figura 53 mostra come sono distribuite le varie figure professionali e si nota la predominanza di tre soggetti distinti (sviluppatori, amministratori dei progetti e gli anonimi). Questo risultato è ampiamente confermato dalle aspettative: infatti la maggior parte del lavoro viene impiegato per lo sviluppo e successivamente entrano in gioco le altre figure che si collocano ad un livello più alto nella gerarchia di progettazione. Si pensi ad esempio ai designer, che inizieranno il loro lavoro solo dopo che tutti gli sviluppatori avranno finito il loro compito.

Per cercare di comprendere meglio anche la distribuzione delle altre figure è stata inserita la Figura 54, che mostra gli stessi dati del grafico precedente, ma eliminando le tre figure predominanti, così da avere una distribuzione più leggibile. Come era facile prevedere, la maggior parte del personale è quello impiegato per funzioni tecniche, mentre le figure che si occupano di pubblicizzare il software sono numericamente inferiori. Si noti anche la poca differenza esistente tra le varie figure professionali suddivise per la stabilità dei progetti di cui si occupano, differenza che seppur minima deve esistere in quanto, come abbiamo visto dalle analisi precedenti, man mano che un progetto cresce di stabilità aumentano il suo grado di complessità e i difetti che vanno ovviamente risolti.

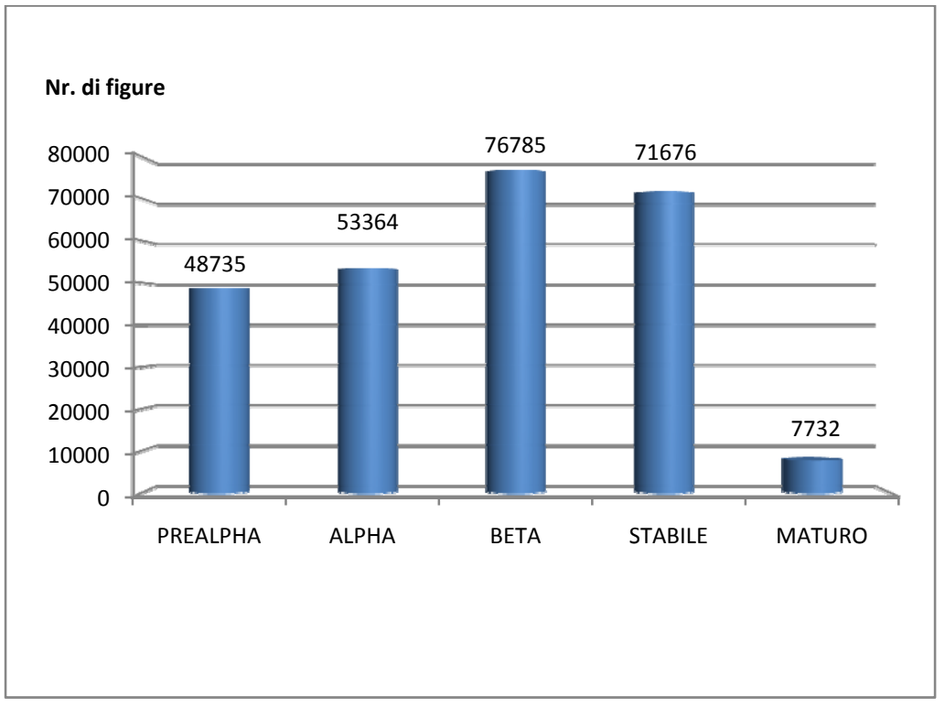


Figura 55 *Totale di tutte le figure suddivise per i livelli di stabilità dei prodotti*

A titolo puramente dimostrativo viene presentato il grafico di Figura 55 che visualizza il numero di tutte le figure professionali che lavorano sui progetti di SourceForge divisi per i livelli di stabilità dei prodotti. Si noti nella somma degli sviluppatori la predominanza per i progetti beta, quando invece ci si aspettava il solito trend in salita con il culmine per i prodotti di tipo stabile.

5.6 L'ANALISI DELLE RELEASE

Le release rappresentano le piccole evoluzioni dei software e, studiandole, andremo a testare sempre quel ramo che analizza l'attività delle comunità (quantità di lavoro) che ruotano intorno ai progetti open source.

5.6.1 Studio delle Release

STABILITA'	NR. RELEASE
PREALPHA	47749
ALPHA	83311
BETA	161741
STABILE	186716
MATURO	20160

Tabella 7 *Totale delle release rilasciate*

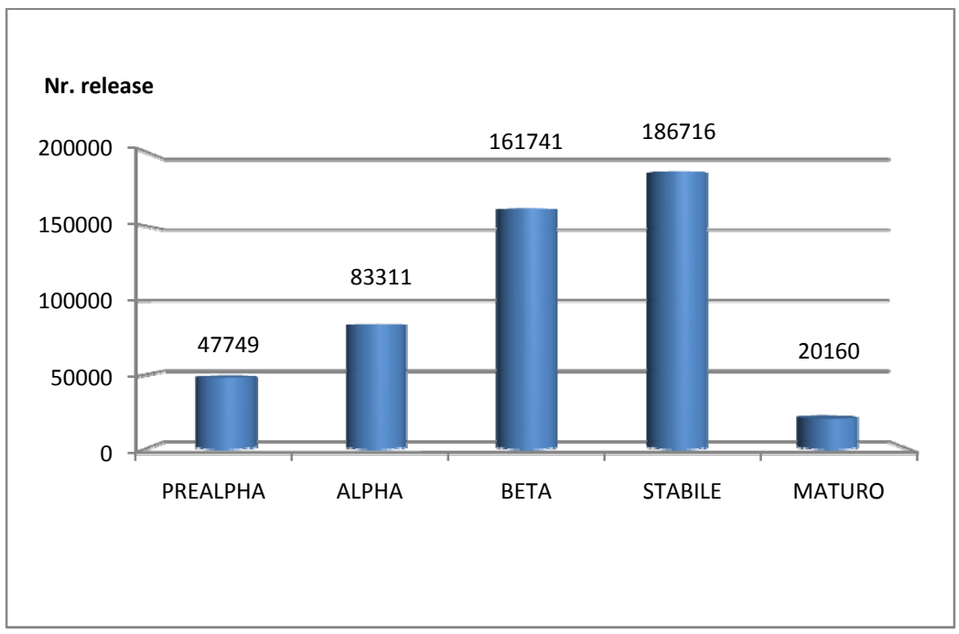


Figura 56 *Totale delle release rilasciate*

La Figura 56 mostra la quantità di release rilasciate e come consueto divise per la stabilità dei prodotti, notando immediatamente il basso numero per i progetti maturi. Ciò, allo stato attuale, ci lascia lo spazio per una considerazione sui software che raggiungono questo stato, perché ci si aspettava prima del compimento di questo lavoro di tesi una “frenesia” molto maggiore su questi programmi in quanto c’era la convinzione che le comunità di sviluppo mantenessero molto attivi

questi prodotti. Invece, si è visto che si lavora tanto finché i prodotti raggiungono lo stato stabile, successivamente, appena dichiarati maturi, vengono “abbandonati”, lasciati in quello stato senza inserire nuove funzionalità o sviluppare altri componenti che possano compromettere la “maturità” dei progetti. Questo comportamento è giustificato se si pensa che un utente che utilizza un determinato programma dichiarato maturo, preferisce affidarsi ad un software ampiamente consolidato e sicuro in termini di sicurezza e stabilità del sistema, abbandonando eventuali componenti aggiuntivi che potrebbero minare le sicurezze ottenute in precedenza.

5.6.2 Studio delle Release Semestrali

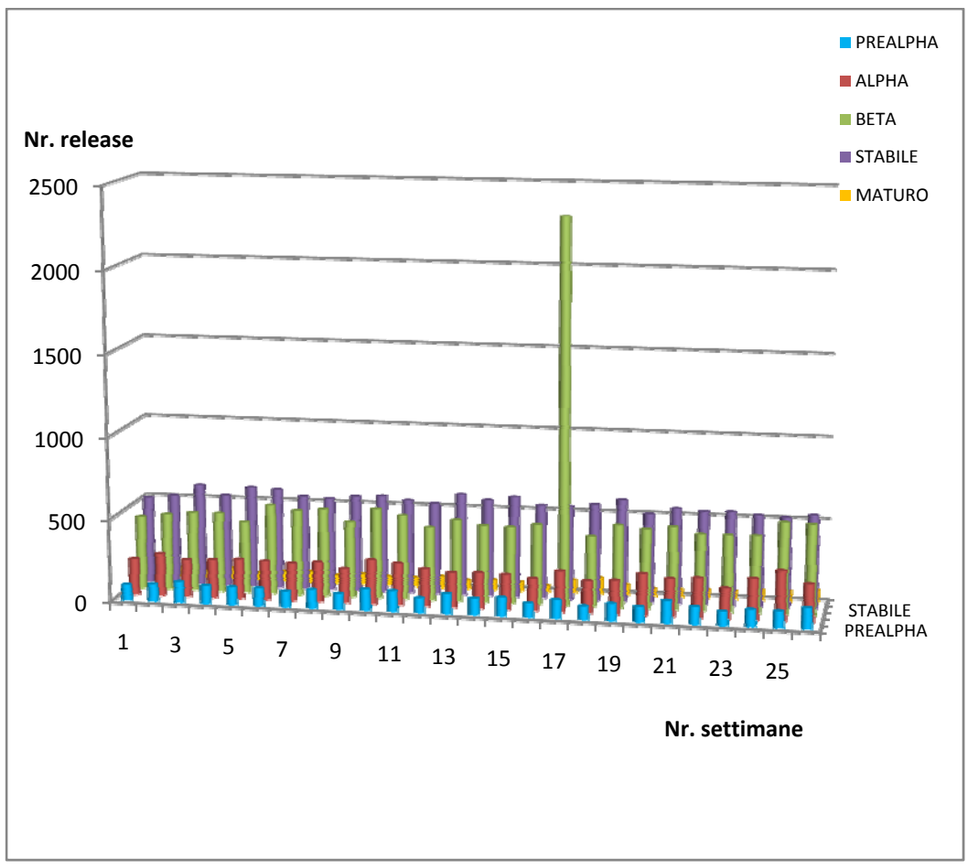


Figura 57 *Release rilasciate nel semestre*

Anche in questo caso, lo studio ha riguardato l'analisi nel tempo (sei mesi) delle release rilasciate, raggruppando i risultati di ogni settimana. Risultati che confermano quanto analizzato nel totale, anche se in questo caso risalta il picco delle release rilasciate nella diciassettesima settimana per i progetti beta, comportamento che avevamo già notato per i bugs dei prodotti beta nella quindicesima settimana, quindi è logico pensare che queste release siano state rese disponibili per correggere quei difetti riscontrati.

5.7 L'ANALISI DELLE DONAZIONI

Anche le donazioni rientrano nello studio delle attività delle comunità, ed è semplice intuire che più una comunità ha a disposizione contributi economici (quindi donazioni) più è spinta a lavorare concretamente su di un prodotto.

5.7.1 Studio delle Donazioni

STABILITA'	NR. PROGETTI
PREALPHA	2406
ALPHA	2764
BETA	4708
STABILE	4946
MATURO	386

Tabella 8 *Progetti che hanno avuto una donazione*

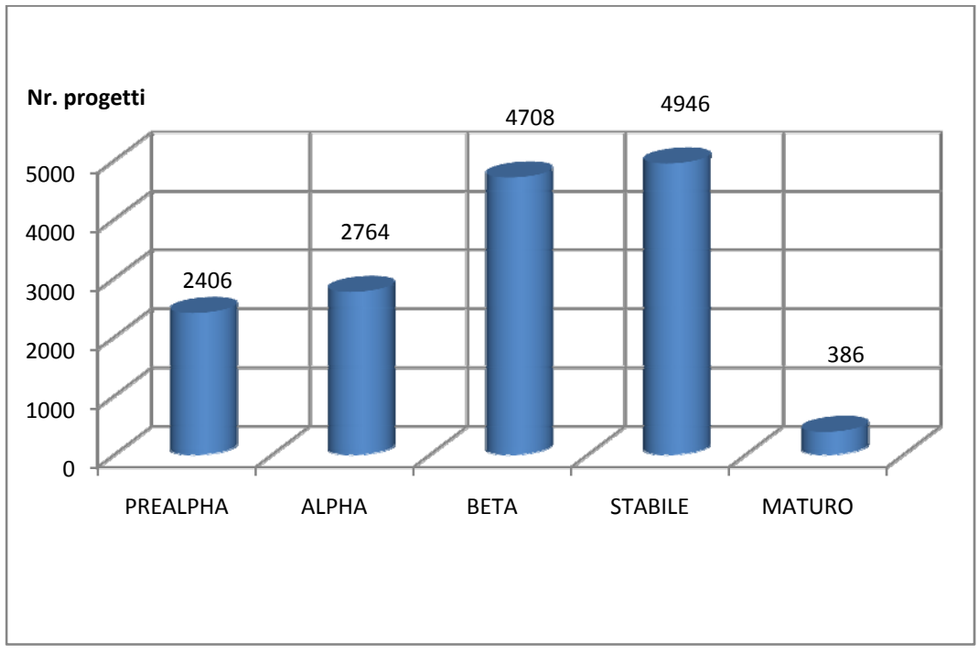


Figura 58 *Progetti che hanno avuto una donazione*

Nello studio delle donazioni sarebbe stato interessante avere notizia su eventuali aziende che sostengono determinati progetti e con quali entità aziende e utenti supportano i progetti a cui sono interessati. Purtroppo i dati messi a disposizione da Notre Dame non riguardavano queste informazioni, quindi è stato possibile analizzare solo quanti progetti hanno avuto delle donazioni e quanti degli utenti (figure professionali) visti in precedenza hanno sostenuto un progetto. Sono assenti anche le relazioni che legano la donazione di un utente verso un progetto specifico. Il grafico di Figura 58 quindi mostra la distribuzione dei progetti che hanno ricevuto una donazione, confermando il supporto per i progetti beta e stabile a discapito dei prodotti dichiarati ormai maturi. Completamente dimezzate anche le donazioni per i software "giovani", per i quali ci si aspettava un sostegno maggiore.

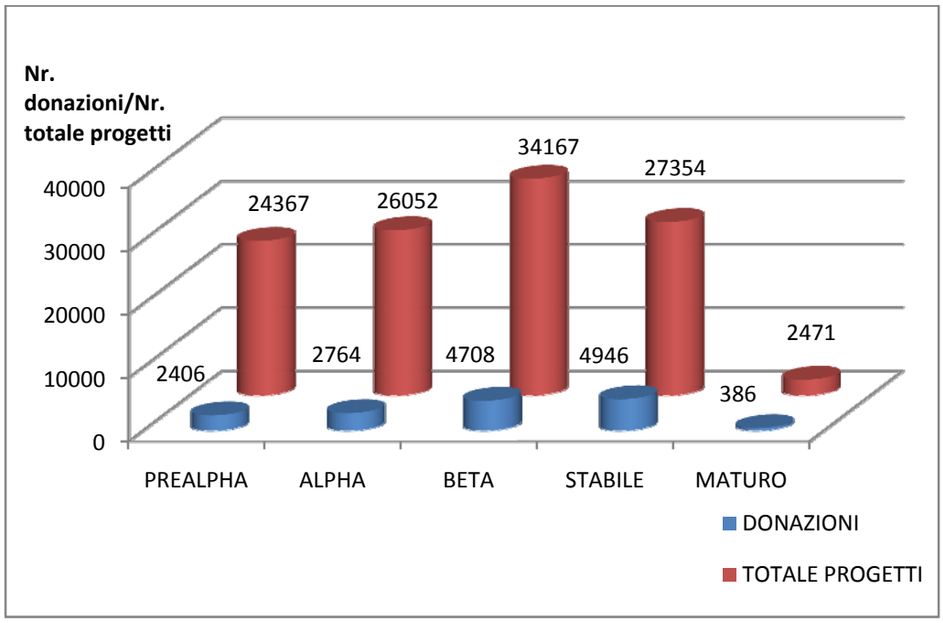


Figura 59 *Confronto tra le donazioni ed il totale dei progetti*

A titolo dimostrativo, il grafico di Figura 59 mostra il confronto tra i progetti che hanno avuto una donazione ed il totale dei software presenti in SourceForge; come si può notare solo circa il 10% dei prodotti viene sostenuto, precisando che ci si poteva aspettare questo risultato per i prodotti maturi, essendo le donazioni a carattere volontario e i progetti maturi ormai consolidati. Ma ciò non dovrebbe valere per i progetti appena nati in quanto, se venisse trovato un software potenzialmente utile, gli sviluppatori andrebbero sostenuti per incentivarli nel miglioramento del programma.

RUOLO DEI MEMBRI	NR. DI DONATORI
Developer	5032
Project Manager	3023
Unix Admin	82
Doc Writer	40
Tester	87
Support Manager	27
Graphic/Other Designer	47
Translator (I18N/L10N)	102
Editorial/Content Writer	14

Packager (.rpm, .deb etc)	61
Analysis / Design	19
Advisor/Mentor/Consultant	71
Distributor/Promoter	15
Content Management	20
Requirements Engineering	5
Web Designer	92
Porter (Cross Platform Devel.)	38
Anonimus	12548
All Hands Person (Gestori)	715
Others	119
User Interface (UI) Designer	10
Support Technician	15
DBA (Database Administrator)	5

Tabella 9 *Ruoli di utenti che hanno fatto una donazione*

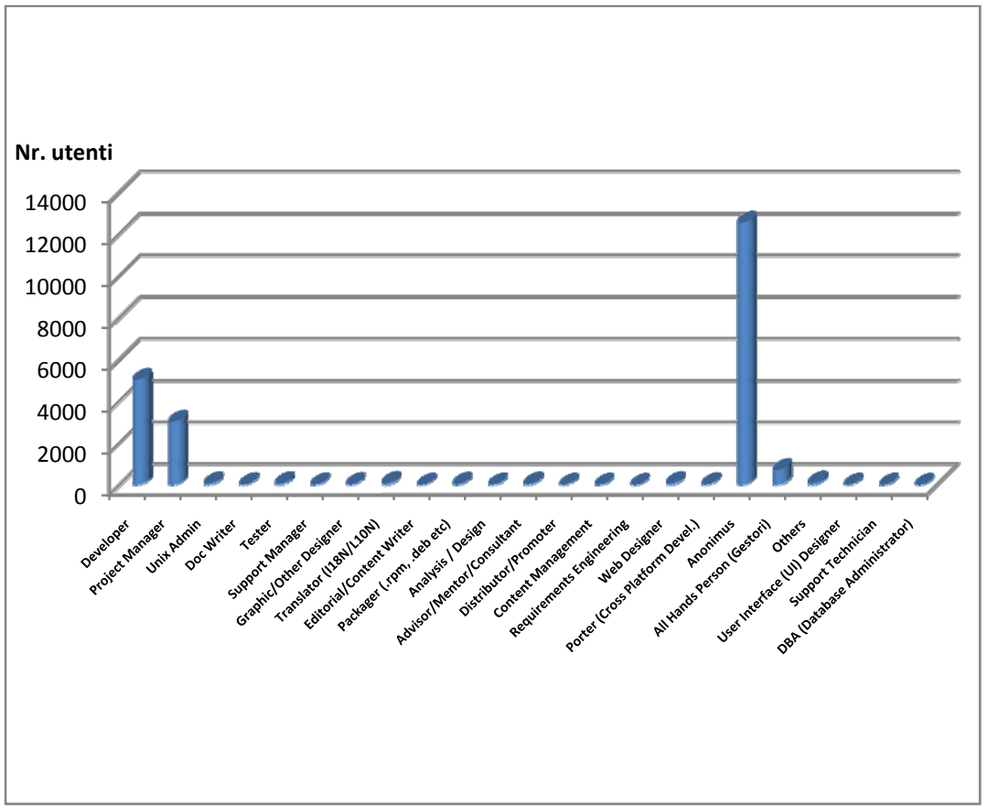


Figura 60 *Ruoli degli utenti che hanno fatto una donazione*

La Figura 60 mostra quali ruoli degli utenti hanno effettuato una donazione, dimostrando che i progetti vengono maggiormente sostenuti dagli utenti (anonymus), ma anche gli sviluppatori ed i project manager investono economicamente nelle proprie creazioni. Per cercare di comprendere come anche le altre figure investano nei progetti si ripropone lo stesso grafico eliminando le componenti predominanti.

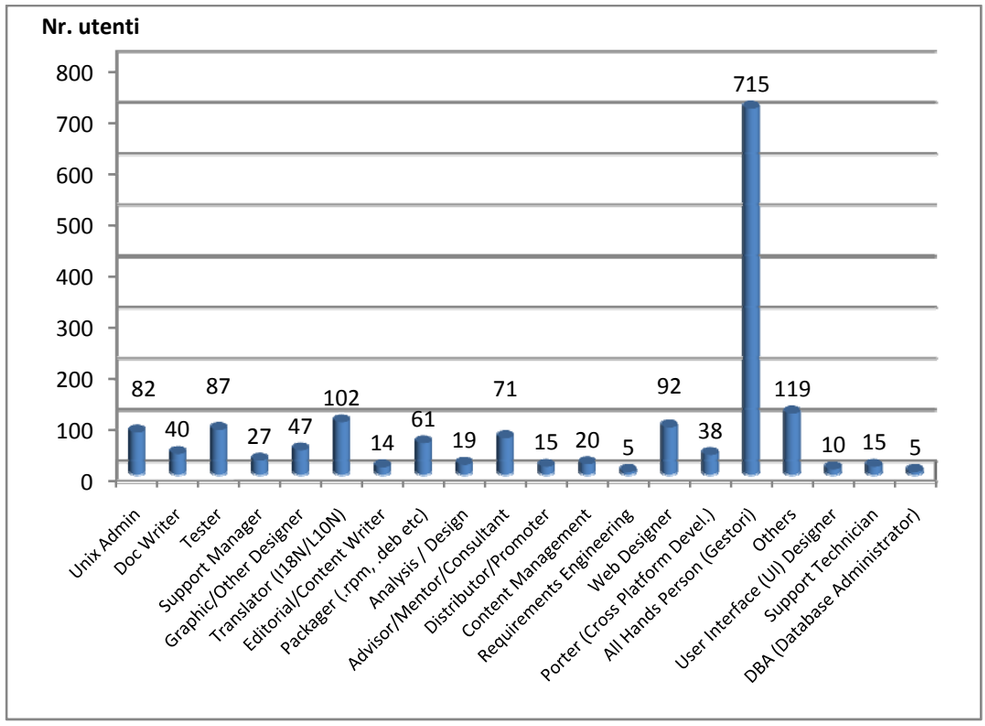


Figura 61 *Ruoli di utenti che hanno effettuato una donazione senza componenti predominanti*

La Figura 61 mostra il numero di utenti di un determinato ruolo che ha sostenuto un progetto. Come si vede, fatta eccezione per la serie dei gestori (che sono comunque numericamente maggiori), gli altri non hanno una fattore di grandezza elevato quindi si dimostra che i progetti open source, pur essendo economicamente poco sostenuti si avvalgono comunque di competenze variegate che lavorano per migliorare i propri prodotti. Viene mostrato ora il grafico di Figura 62 che fa notare

la differenza tra i ruoli degli utenti che hanno effettuato una donazione ed il totale degli stessi.

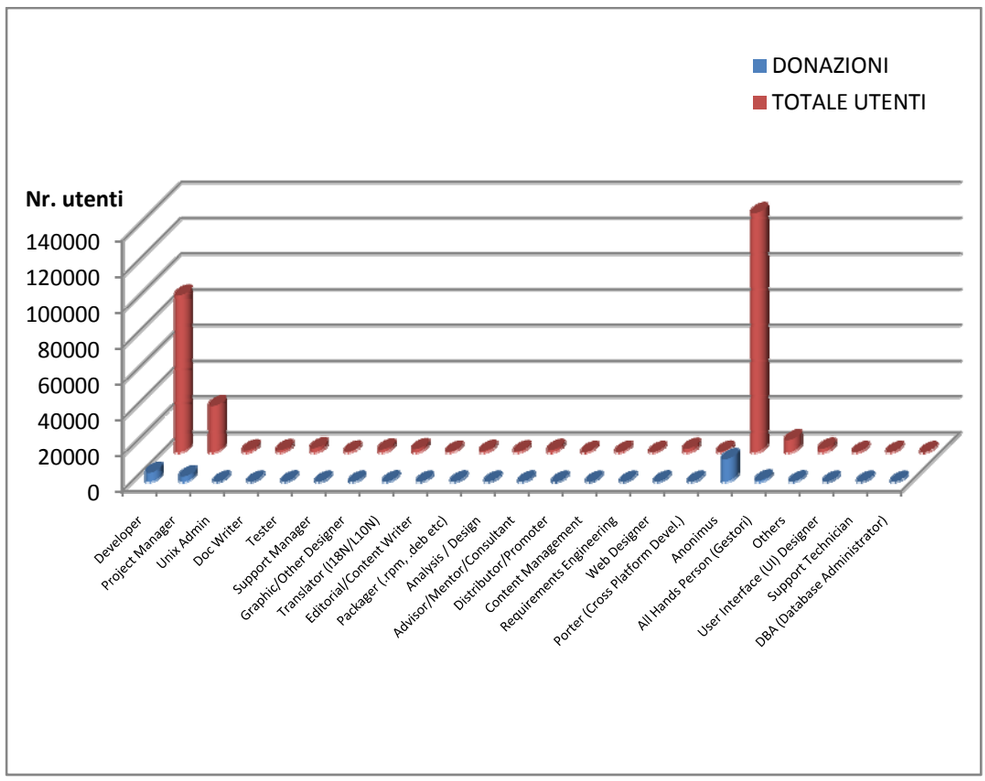


Figura 62 *Confronto fra i ruoli degli utenti che hanno donato ed il totale*

Come è evidente i ruoli che sono numericamente predominanti non lasciano interpretare correttamente il grafico, quindi lo stesso viene riproposto (Figura 63) eliminando queste serie. E' eloquente la differenza tra la totalità degli utenti con chi sostiene il progetto, differenza che non lascia spazio a commenti per la comprensione del fenomeno. Infine, viene mostrata la Tabella 10 che indica la differenza dei valori in percentuale dei donatori, confrontati sempre fra i vari ruoli.

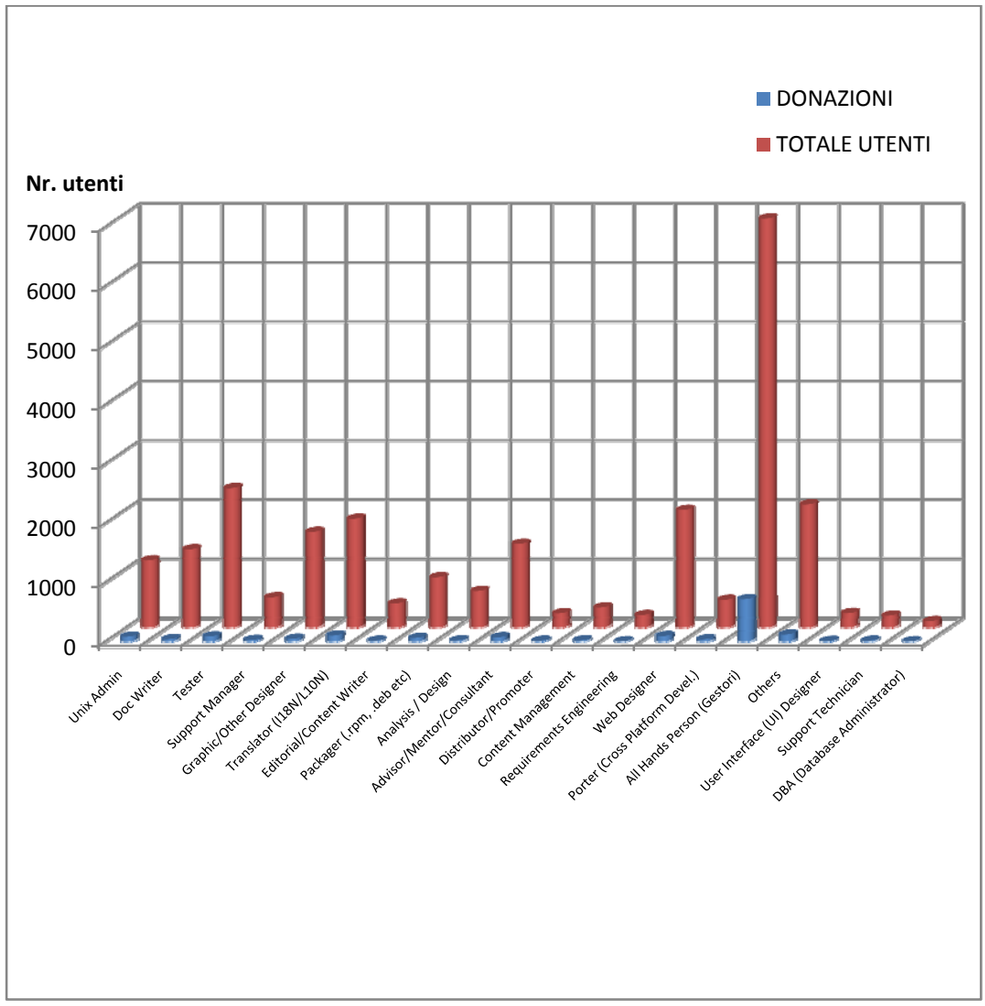


Figura 63 *Confronto tra ruoli degli utenti che hanno donato e totale senza componenti predominanti*

RUOLO DEI MEMBRI	NR. DI DONATORI	TOTALE PROGETTI	DONATORI IN %
Developer	5032	87803	5,73
Project Manager	3023	25976	11,64
Unix Admin	82	1133	7,24
Doc Writer	40	1318	3,03
Tester	87	2352	3,70
Support Manager	27	508	5,31
Graphic/Other Designer	47	1614	2,91
Translator (I18N/L10N)	102	1835	5,56
Editorial/Content Writer	14	405	3,46
Packager (.rpm, .deb etc)	61	848	7,19
Analysis / Design	19	615	3,09

Advisor/Mentor/Consultant	71	1412	5,03
Distributor/Promoter	15	240	6,25
Content Management	20	338	5,92
Requirements Engineering	5	209	2,39
Web Designer	92	1984	4,64
Porter (Cross Platform Devel.)	38	466	8,15
Anonimus	12548	133840	9,38
All Hands Person (Gestori)	715	6903	10,36
Others	119	2077	5,73
User Interface (UI) Designer	10	242	4,13
Support Technician	15	200	7,50
DBA (Database Administrator)	5	107	4,67

Tabella 10 *Differenze in percentuale tra i donatori dei vari ruoli*

5.8 L'ANALISI DEI NUOVI PROGETTI

La creazione di nuovi progetti rappresenta la parte più evidente dell'attività svolta dalle varie comunità e rappresenta quindi un ottimo fattore per valutare i team di sviluppo ed il loro lavoro.

5.8.1 Studio Dei Nuovi Progetti

Alla data di ottobre 2007, SourceForge gestiva 135.834 progetti distinti. Per comprendere quindi il grado di attività delle comunità, sono stati analizzati i progetti nati nello stesso semestre studiato nelle precedenti analisi (vedi Figura 64).

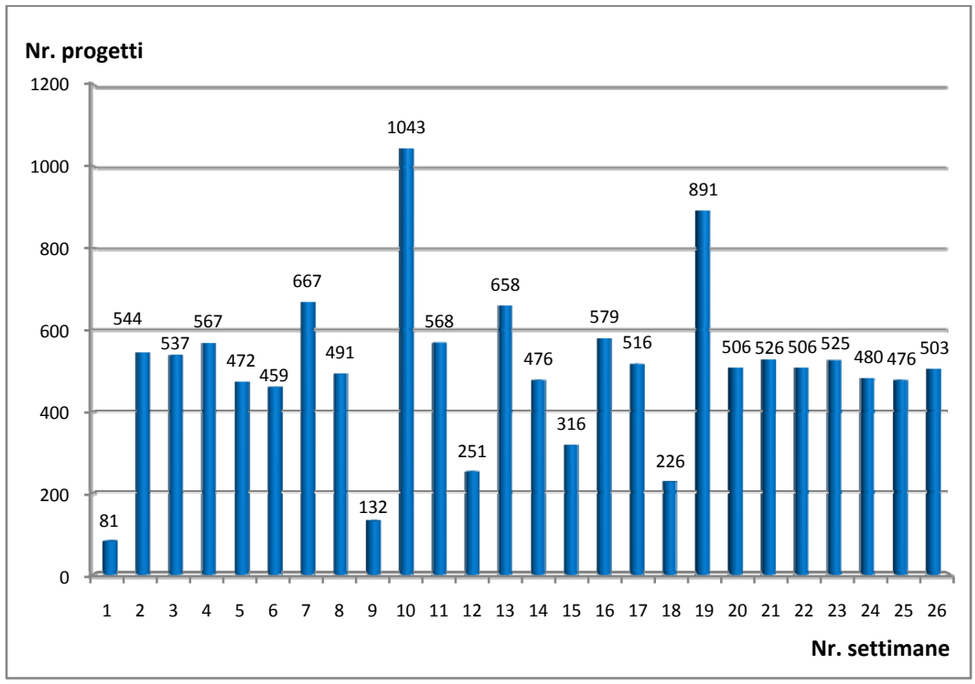


Figura 64 *Progetti nati nel semestre*

Come si nota dalla figura, il trend delle nascite di nuovi progetti è all'incirca di 500 prodotti a settimana, confermando un alto livello di produttività. E' interessante osservare che, tenendo conto del totale dei progetti (circa 135.000) e della somma dei software nati nel periodo in esame (circa 13.000), si nota che ogni sei mesi nasce circa il 10% del totale dei software, quindi con questo trend ogni dieci anni SourceForge tende a raddoppiare.

Sarebbe stato utile poter confrontare questi dati con quelli dei progetti che "muoiono" o che vengono dichiarati chiusi con lo scopo di comprendere perché un prodotto non viene più utilizzato. Ciò potrebbe dipendere da una mera logica commerciale (così come fa una tra le più grandi software house al mondo), ma questa considerazione per il caso dei software open source di SourceForge non potrebbe valere perché i progetti sono tutti distribuiti gratuitamente. Oppure potrebbe dipendere dall'inutilizzo del software, ma in questo caso sarebbe stato vantaggioso capire quando un prodotto non viene più usato. Purtroppo

SourceForge non prevede questo tipo di clausola e quindi non si potrà conoscere la fine evolutiva dei software open source e non si potranno compiere questi tipi di analisi.

5.9 L'ANALISI DEI DOWNLOAD

L'ultimo fattore che analizziamo è quello dei download, che pur rientrando tra lo studio dell'attività delle comunità, non rispecchia il lavoro degli sviluppatori, ma analizza come gli utenti si avvicinano ai prodotti open source e quali tipi di prodotti preferiscono.

5.9.1 Studio Dei Download

STABILITA'	NR. DOWNLOAD
PREALPHA	377010644
ALPHA	300998647
BETA	1196021883
STABILE	1874022287
MATURO	207252402

Tabella 11 *Totale dei download*

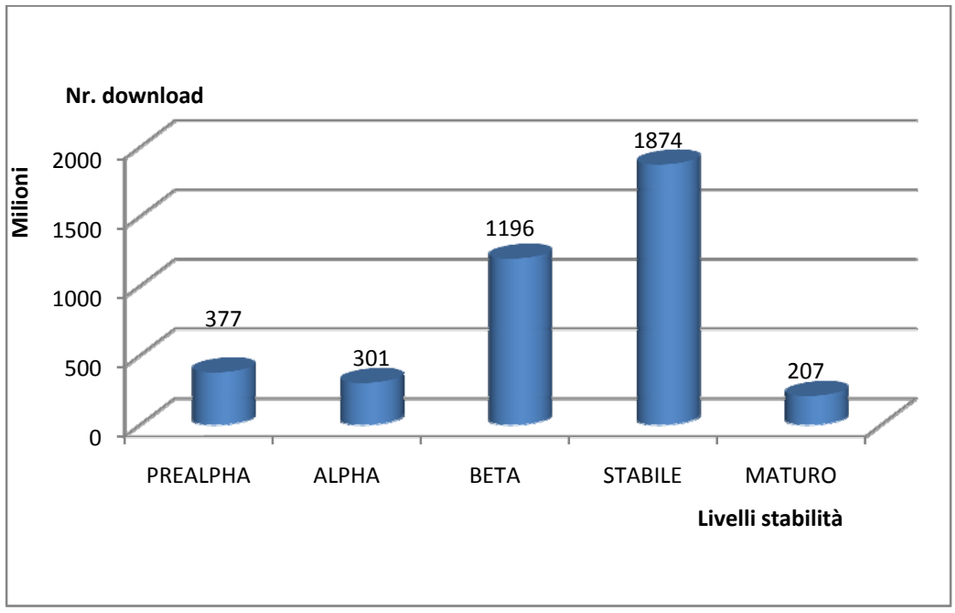


Figura 65 *Totale download*

Il grafico di Figura 65, mostra i download dei progetti eseguiti da SourceForge (i valori sono espressi in milioni). Si noti come i prodotti più scaricati sono quelli stabili, non ci si aspettava però che il livello più basso fosse rappresentato dai progetti maturi, che hanno un numero di download inferiore anche ai software alpha. L'unico modo che può giustificare questo trend è quello di pensare che i progetti dichiarati maturi non subiscono più quel processo di miglioramento che invece è alla base dei progetti stabili, si rivedano le analisi fatte precedentemente (patch, release e bugs), ciò fa si che, pur essendo il software stabile e sicuro, non "trasmette" quella sensazione di novità, data dalle continue innovazioni, archiviandolo come software "vecchio". Questa affermazione infatti trova conferma proprio dai dati trovati, perché si è visto che i prodotti stabili, pur avendo il maggior numero di bugs, hanno anche il maggior numero di patch e release, componenti che introducono continuamente migliorie e innovazioni, rendendo quindi il software "sempre nuovo". Interessante è anche il dato dei download dei prodotti prealpha, leggermente maggiori degli alpha, questo a nostro avviso è un'altra conferma della teoria appena

espressa, perché scaricando maggiormente un prodotto nuovo gli utenti cercano in questo ciò che non hanno ancora trovato in uno dei tanti software disponibili, prodotto che viene però presto abbandonato perché essendo ancora allo stato “embrionale” presenta ancora molti problemi e parti non funzionanti, quindi la flessione dei progetti alpha. La tendenza viene poi invertita con i prodotti beta per raggiungere l’apice con quelli stabili.

5.10 L’ANALISI DEL TEMPO MEDIO DI SOLUZIONE DEI BUGS

Lo studio del tempo medio di soluzione dei bugs rappresenta un’attività necessaria per comprendere se esiste un supporto a breve termine dei prodotti open source e se, di conseguenza, potrà essere garantito anche uno sviluppo futuro del software.

5.11 Studio del tempo medio di soluzione dei bugs

STABILITA	PRIORITA’									
	1	2	3	4	5	6	7	8	9	10
PREALPHA	138,86	149,01	214,22	149,07	116,52	97,71	91,14	51,11	78,65	
ALPHA	118,71	129,48	108,66	85,07	82,73	69,21	74,21	62,78	56,06	1,60
BETA	119,56	115,11	119,57	125,15	93,38	90,23	74,68	60,74	49,82	
STABILE	107,57	125,10	116,96	130,42	89,29	91,46	78,98	65,09	59,92	

MATURO	150,32	149,63	152,60	211,11	100,55	224,07	133,30	128,19	79,91	

Tabella 12 *Tempo medio di risoluzione dei bugs espresso in giorni*

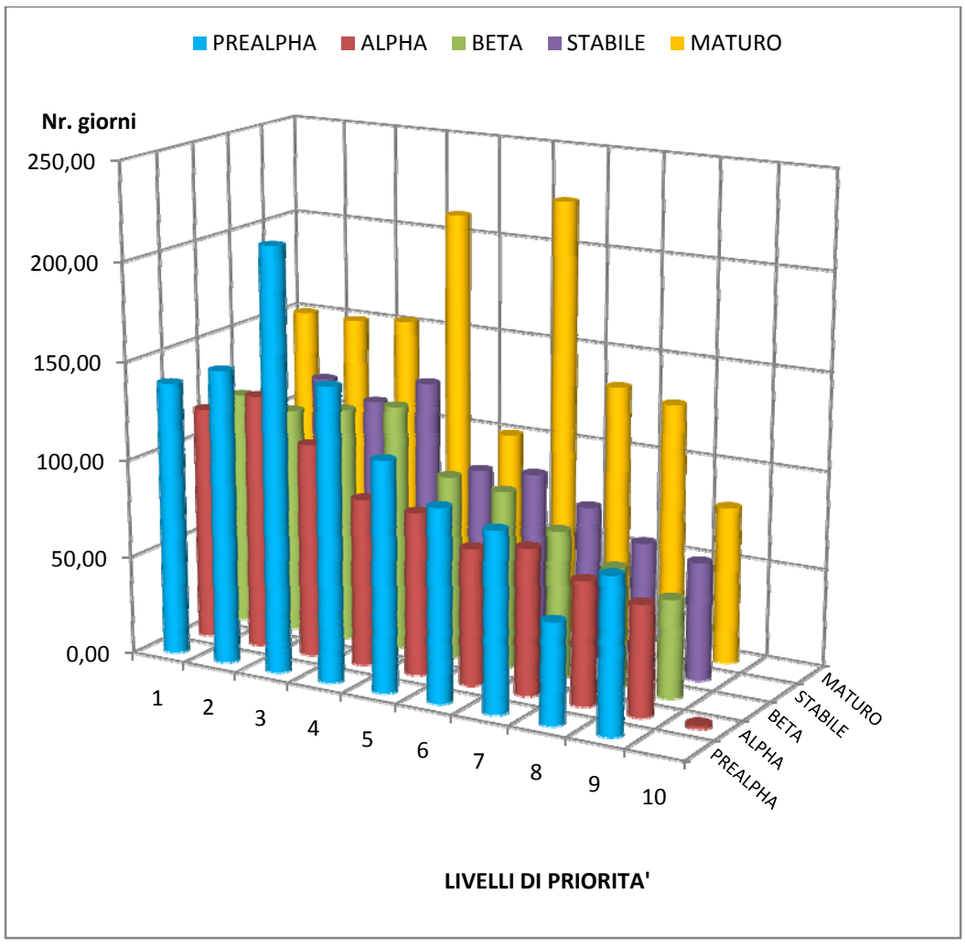


Figura 66 *Tempo medio di risoluzione dei bugs espresso in giorni*

La Figura 66 mostra i tempi medi di risoluzione dei bugs rappresentati in giorni, precisando in primo luogo che ci si aspettava valori leggermente inferiori, perché questi dati mostrano un supporto nel breve termine abbastanza alto, ma se consideriamo che alcuni bugs possono non essere stati chiusi (quindi non risolti), i tempi medi tendono necessariamente a crescere e di conseguenza a falsare

leggermente i risultati. Ciò non implica che questa analisi non sia esatta perché, focalizzando l'attenzione sui tempi di risoluzione dei bugs di livello cinque, che sono quelli con il fattore di grandezza più alto e quindi quelli con probabilità maggiore ad essere errati, presentano tempi proporzionati con gli altri, di conseguenza possono essere considerati comunque molto attendibili. Detto ciò però si nota come la distribuzione dei tempi per i livelli di criticità e di stabilità dei prodotti non rispecchia quella vista nelle altre analisi, bensì ha valori maggiori per criticità più basse e livelli di progetti più giovani, a differenza dei valori più bassi inerenti ai livelli di criticità più alte e prodotti con stabilità maggiore. Infatti, le comunità tendono a risolvere più velocemente i problemi relativi alla sicurezza ed alla stabilità dei sistemi (livelli di criticità alte) riferiti a software stabili. Questo perché, essendo i programmi più utilizzati e più sostenuti anche a livello economico (così come abbiamo visto), devono avere un buon supporto nel breve termine. Invece, si tende a lasciare in secondo piano i problemi dei progetti giovani. Inoltre, durante tutto questo studio è emerso che non appena i progetti vengono dichiarati maturi, subiscono una sorta di abbandono, infatti, come si può notare, anche i tempi di risoluzione dei bugs sono mediamente quelli più alti rispetto agli altri.

5.12 Studio Del Tempo Medio di Soluzione Dei Bugs Nel Semestre

Anche in questo caso, si è proceduto come di consueto all'analisi del tempo medio di risoluzione dei bugs avendo come riferimento il solito semestre e raggruppando i dati in settimane.

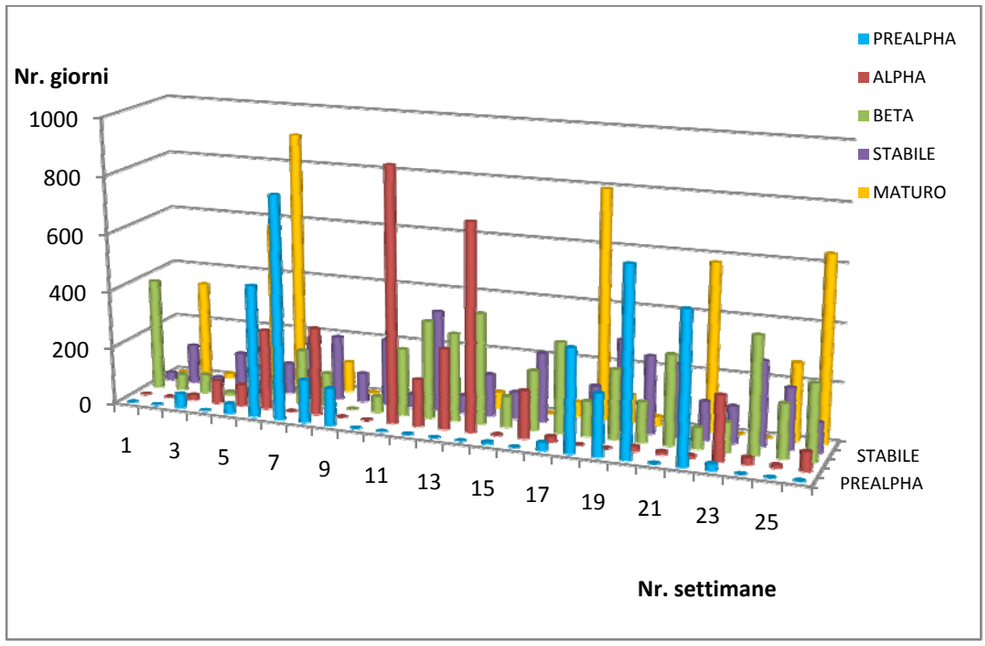


Figura 67 *Tempi medi di risoluzione dei bugs di priorità 1*

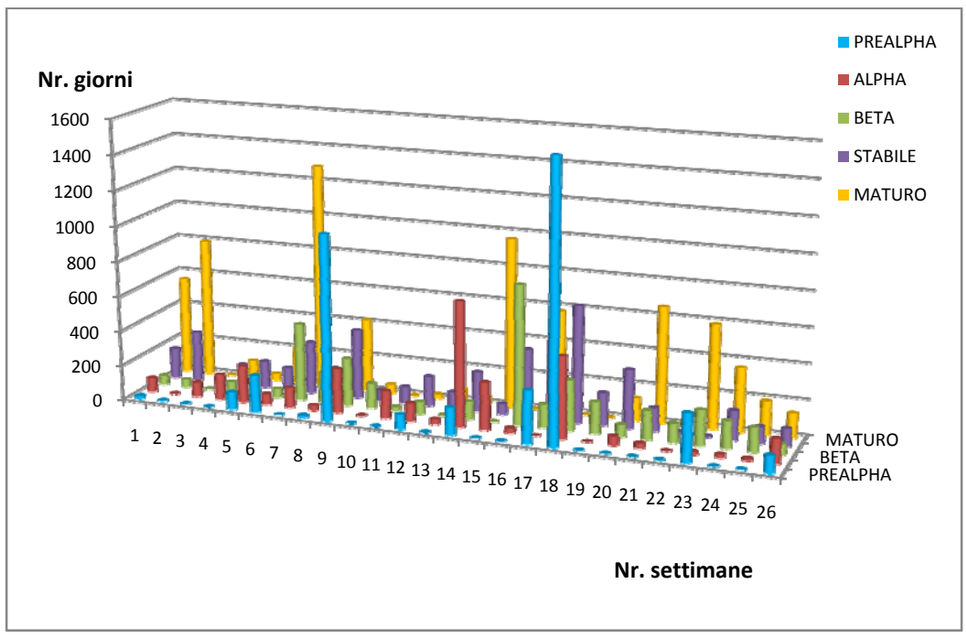


Figura 68 *Tempi medi di risoluzione dei bugs di priorità 2*

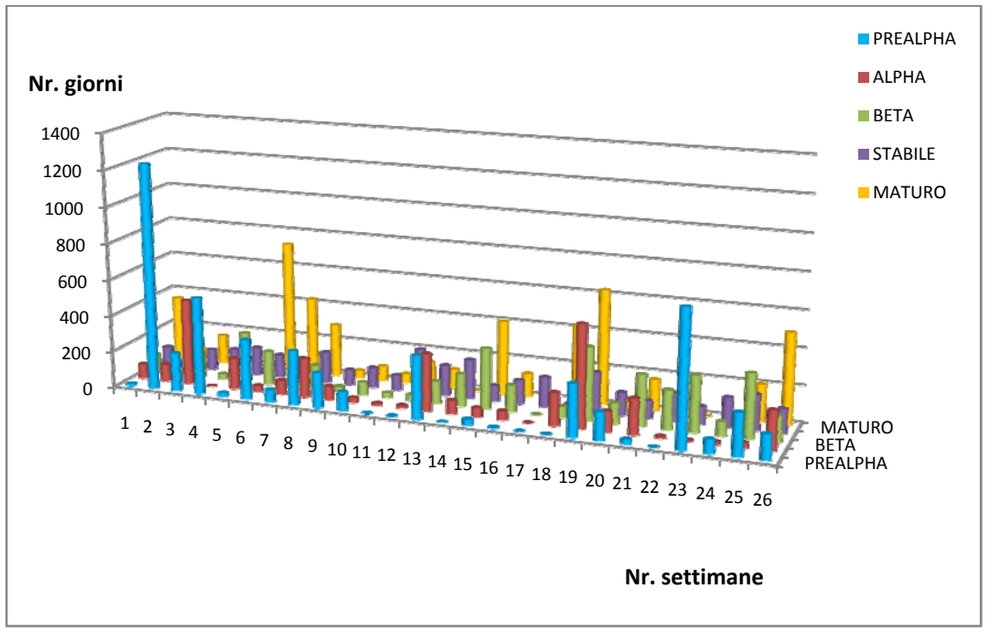


Figura 69 *Tempi medi di risoluzione dei bugs di priorità 3*

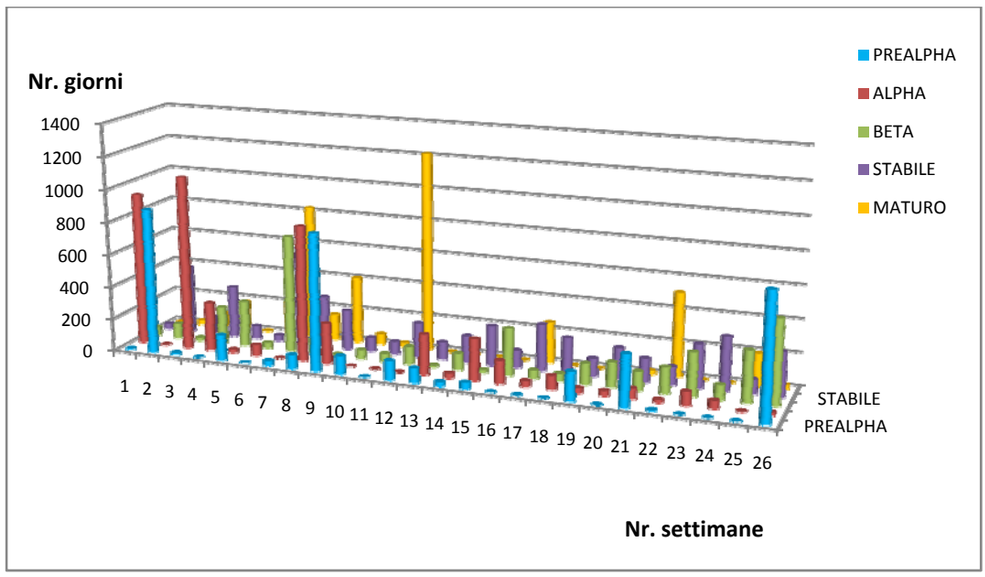


Figura 70 *Tempi medi di risoluzione dei bugs di priorità 4*

I grafici di Figura 67,Figura 68,Figura 69 eFigura 70 mostrano quanto lavoro hanno compiuto le comunità nei sei mesi considerati. Come si può notare, i dati non presentano una omogeneità su tutto il periodo, bensì mostrano molti punti a valore zero (che verranno illustrati in seguito). Risaltano invece gli alti valori di alcuni punti che

rappresentano la risoluzione di bugs aperti molto tempo prima, a conferma del fatto che alcuni bugs possono anche non essere mai stati risolti. Questi grafici però confermano l'analisi fatta sul totale dei tempi di soluzione, in quanto i dati incongruenti sono riscontrati maggiormente sui progetti giovani e non su quelli maturi confermando la tendenza nel tentativo di risolvere prima i difetti dei progetti più stabili.

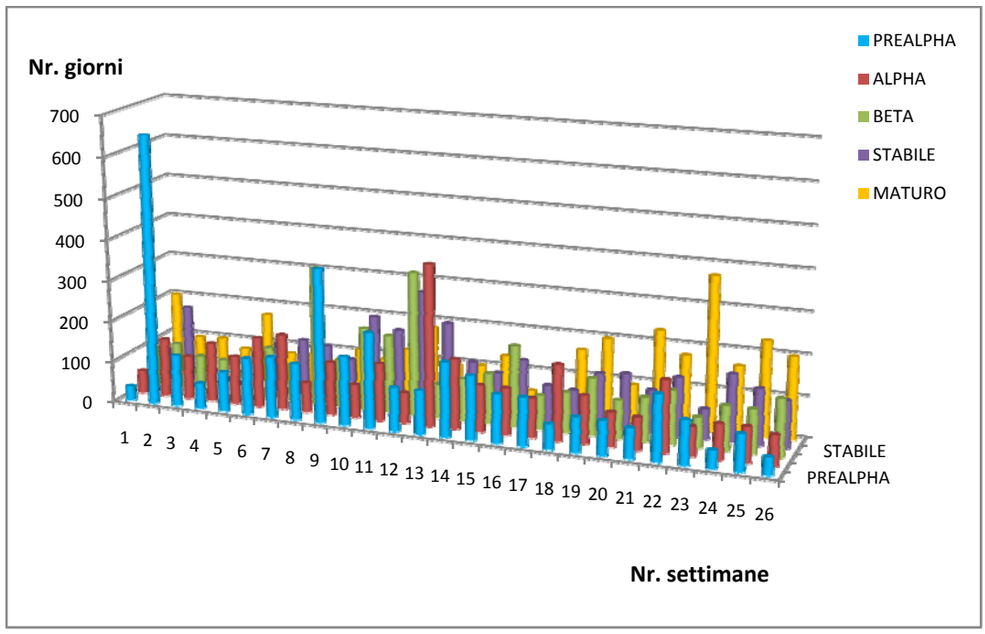


Figura 71 *Tempi medi di risoluzione dei bugs di priorità 5*

Come consueto la Figura 71, relativa ai bugs di priorità cinque, è quella che riporta i dati più completi, infatti mancano valori uguali a zero. Si noti però come la distribuzione sia leggermente differente rispetto a quella dei dati sul totale, presentando valori quasi simili tra le varie stabilità dei prodotti, mentre è confermato il tempo maggiore per le risoluzioni dei problemi di livello maturo.

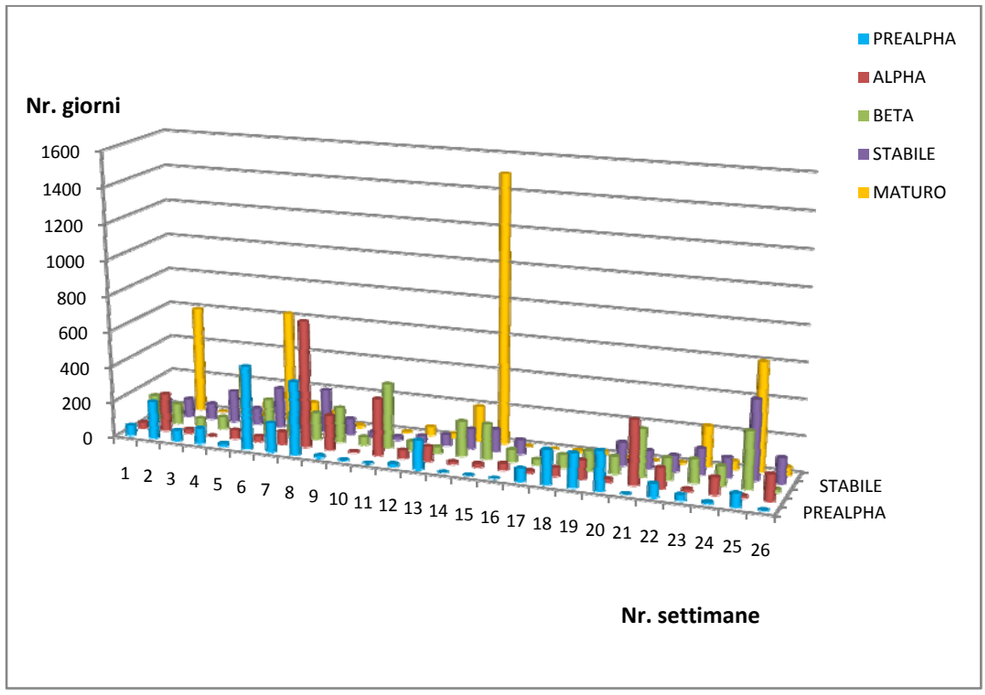


Figura 72 *Tempi medi di risoluzione dei bugs di priorità 6*

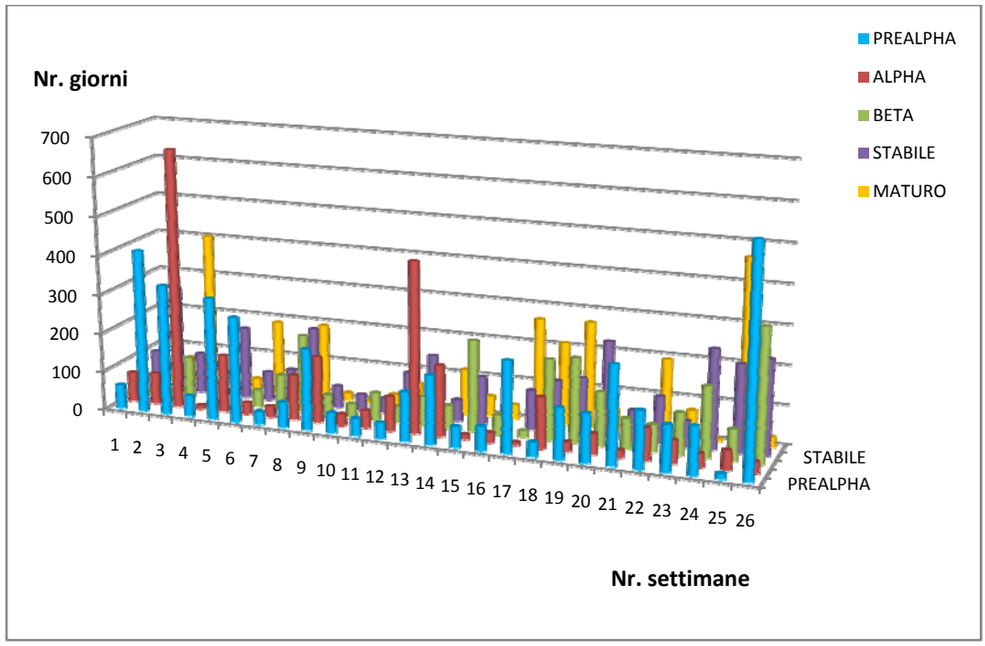


Figura 73 *Tempi medi di risoluzione dei bugs di priorità 7*

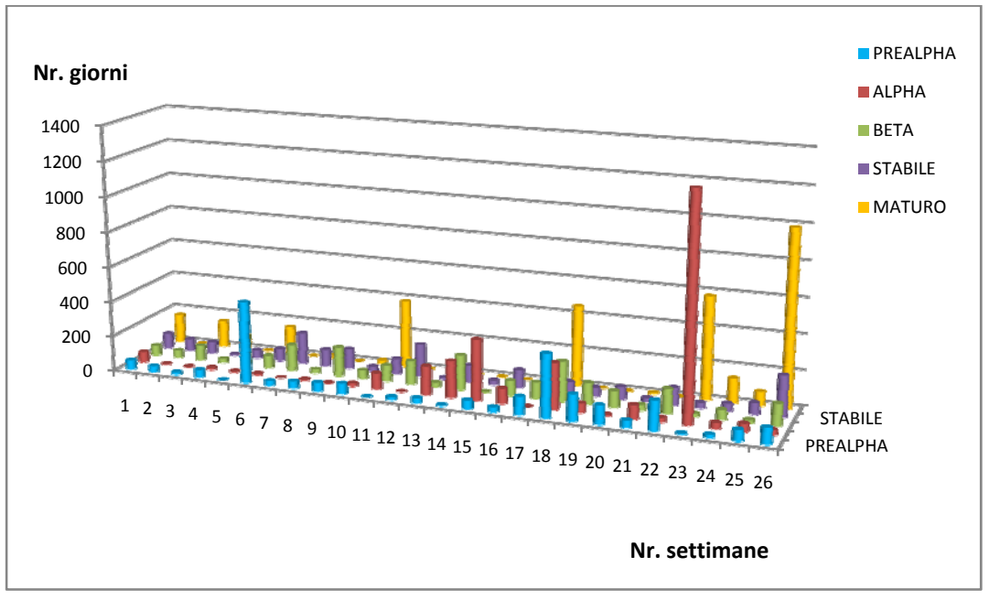


Figura 74 *Tempi medi di risoluzione dei bugs di priorità 8*

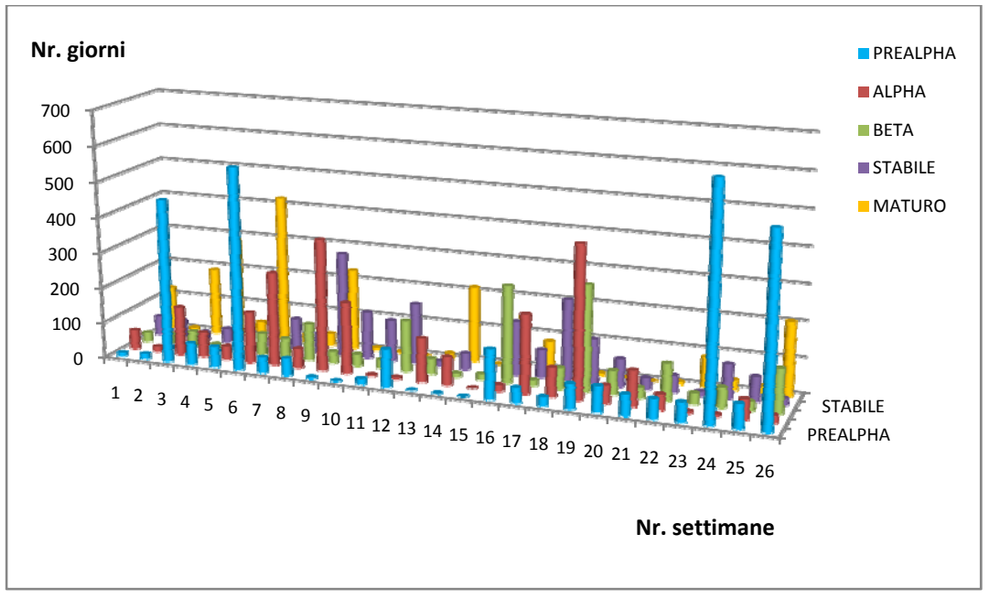


Figura 75 *Tempi medi di risoluzione dei bugs di priorità 9*

Questi ultimi grafici, che mostrano i tempi di risoluzione dei bugs per le criticità più alte, possiedono anche loro molti dati uguali a zero. Questi risultati sono stati impostati a zero perché è emerso durante l'analisi che molti avevano valori negativi, ciò significava che alcuni bugs fossero stati risolti prima che venissero scoperti, perciò, essendo

l'incongruenza troppo evidente, sono stati settati con valore nullo. Questo però non ha inficiato lo studio perché, come si può notare, man mano che si sale di criticità diminuisce il tempo che le comunità impiegano nel risolvere i problemi dei progetti più stabili. Mentre i valori più alti sono visibili per i progetti più giovani.

Queste incongruenze hanno inizialmente fatto pensare ad un uso non standard dei valori di timestamp da parte di SourceForge, anche perché il wiki[28] di Notre Dame spiegava che i valori erano espressi in secondi e non in millisecondi come di consueto, quindi poteva essere probabile qualche altra trasformazione non comunicata. Per scongiurare questa evenienza si è proceduto con qualche semplice test di verifica partendo dai dati in chiaro del sito di SourceForge. La verifica è stata effettuata in questo modo: sono stati estratti direttamente dal bug tracking del sito gli identificativi di alcuni bugs di progetti diversi insieme alle date "in chiaro" di scoperta; dopo sono stati estrapolati i valori di timestamp degli stessi bugs dal database di Notre Dame e sono stati convertiti in date utilizzando un semplice metodo creato ad hoc con il linguaggio JAVA che si riporta:

```
public Date getData(long timestamp){
Date test = new Date(timestamp*1000);
return test; }
```

Questo test ha evidenziato che i valori di timestamp controllati erano uguali a quelli ottenuti direttamente dal sito, quindi ciò ha evidenziato alcuni errori nei valori di timestamp presenti all'interno del database di Notre Dame.

[28] https://zerlot.cse.nd.edu/mywiki/index.php?title=FAQ#Date_Format

5.13 L'ANALISI DEL TEMPO MEDIO DI ASSEGNAZIONE DEI BUGS

L'altro metodo che viene utilizzato per analizzare il supporto a breve termine è quello di controllare il tempo medio di assegnazione dei bugs agli sviluppatori, purtroppo SourceForge non prevede questo tipo di informazione, come non ha il dato del momento in cui un bugs viene assegnato. Considerato ciò, per effettuare questa analisi è stato necessario usare alcune informazioni del bug tracking. Abbiamo notato che alcune comunità (ma purtroppo non tutte) quando assegnano il bugs ad uno sviluppatore del team per la risoluzione, annotano questo cambiamento di stato registrando nel bug tracking la clausola "assigned_to", notato questo quindi le query di ricerca di questo studio sono state modellate al fine di comprendere anche questo vincolo. Ciò ha portato a risultati a nostro avviso abbastanza veritieri anche perché, oltre ai dati dei tempi medi, sono stati estrapolate anche il numero dei bugs coinvolti nell'analisi (vedi Tabella 13) e, come si può notare, il numero dei bugs non si discosta tanto da quello del totale.

5.14 Studio Del Tempo Medio Di Assegnazione Dei Bugs

	PRIORITA' 1		PRIORITA' 2		PRIORITA' 3		PRIORITA' 4		PRIORITA' 5	
STABILITA'	AVG	BUG	AVG	BUG	AVG	BUG	AVG	BUG	AVG	BUG
PREALPHA	62,85	1048	58,15	322	82,74	784	51,37	313	57,01	16199
ALPHA	39,03	1384	70,61	709	49,24	1506	63,35	588	49,79	25627
BETA	43,56	3756	70,36	1330	63,38	3269	58,97	1232	45,94	61600
STABILE	43,16	7613	59,04	2281	61,80	5124	65,87	2081	52,13	101414
MATURO	64,70	726	122,77	532	100,86	944	116,41	550	69,85	18321

	PRIORITA' 6		PRIORITA' 7		PRIORITA' 8		PRIORITA' 9	
STABILITA'	AVG	BUG	AVG	BUG	AVG	BUG	AVG	BUG

PREALPHA	44,12	535	39,58	1170	22,90	593	28,95	1030
ALPHA	37,74	1024	50,15	1896	44,31	1000	31,07	1876
BETA	55,62	2515	46,34	4386	30,85	2117	26,99	4287
STABILE	60,87	4012	53,73	7802	48,54	3660	42,33	6313
MATURO	131,85	705	118,82	1258	90,62	789	62,50	1080

Tabella 13 *Tempo medio di assegnazione dei bugs*

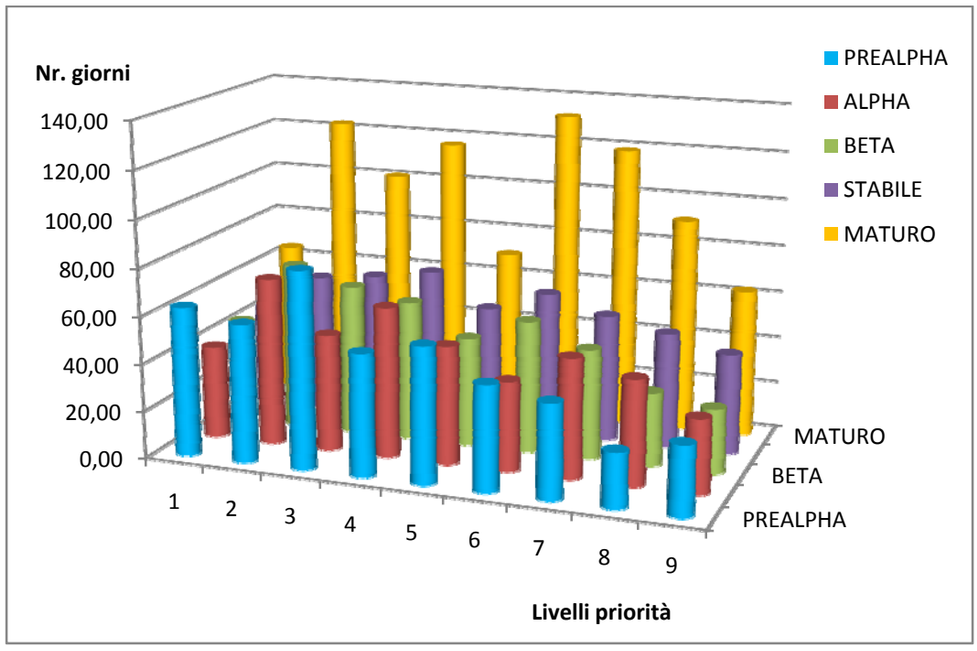


Figura 76 *Tempo medio di assegnazione dei bugs*

Il grafico di Figura 76 mostra i tempi medi di assegnazione dei bugs differenziandoli per livelli di criticità e per stabilità dei prodotti. Si nota inizialmente l'alto numero di giorni per assegnare i bugs per i prodotti maturi, andamento simile a quanto visto con lo studio dei tempi di risoluzione, ciò conferma ulteriormente la spiegazione data in precedenza in quanto questi prodotti, essendo estremamente solidi e sicuri presentano problemi che possono essere risolti in un secondo momento. A differenza dei prodotti maturi, gli altri mostrano una distribuzione abbastanza omogenea tendendo però a diminuire nel tempo di assegnazione man mano che si procede verso criticità più alte, così come era stato notato nello studio dei tempi di risoluzione.

5.15 Studio Del Tempo Medio Di Assegnazione Dei Bugs Nel Semestre

Anche in questo caso si è proceduto allo studio dei tempi di assegnazione nel semestre preso come riferimento, ma a causa del metodo utilizzato per reperire i dati, purtroppo non si sono ottenuti risultati adatti per effettuare analisi significative sui bugs di priorità diverse dal cinque. Infatti, i dati ottenuti maggiormente sono valori negativi, rispecchiando la stessa incongruenza spiegata prima (bugs assegnati prima che venissero scoperti) oppure valori riferiti a numero di bugs troppo piccoli per poter essere considerati validi, per questo si omettono anche i relativi grafici. Viene però riportato il grafico riguardante i bugs di priorità cinque, il quale ha valori che permettono di effettuare una valida analisi.

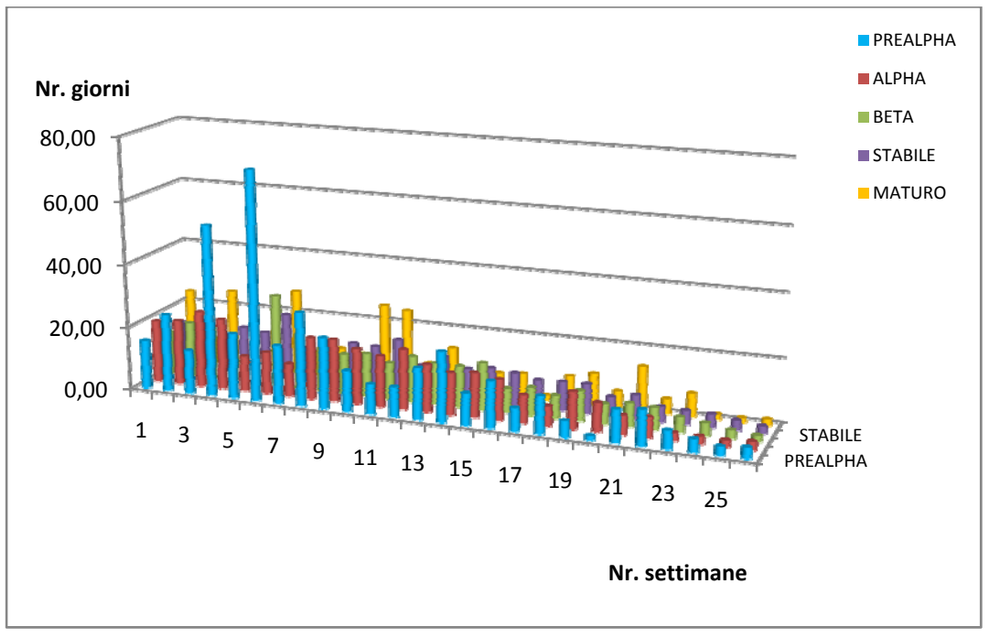


Figura 77 *Tempi medi di assegnazione bugs priorità 5*

Come si può notare dal grafico, i tempi di assegnazione nel semestre rispecchiano quelli visti nello studio di tutto il db di SourceForge, confermando quindi la bontà del metodo utilizzato. Inoltre, si ha l'ulteriore conferma sulla preferenza da parte delle comunità di sviluppo a risolvere con più rapidità i problemi dei progetti stabili, quindi risaltano i valori più alti dei progetti giovani. Non ci si aspettava però questa tendenza nella diminuzione del tempo di assegnazione verso la fine del semestre, diminuzione che tende a "livellare" quasi tutti i punti e per il quale non si è in grado di fornire una giustificazione se non quella di affidarla ad un caso fortuito.

Capitolo Sesto

6.1. CONCLUSIONI

Il lavoro proposto è partito con l'idea di individuare quella serie di dati, normalmente di difficile individuazione da parte degli utenti, sia per motivi tecnici in quanto nascosti, sia per motivi di incapacità stessa dell'utente, che potessero essere utilizzati per poter effettuare alcune valutazioni sui software open source del repository di Sourceforge. Le procedure scritte per reperire i dati (le query), inoltre dovevano servire per automatizzare processi automatici che potessero analizzare grosse moli di dati per effettuare valutazioni pertinenti. Purtroppo a causa dei problemi di accesso ai dati effettuabile solo tramite form manuale e per via dei database differenti a cui ci si deve collegare non è stato possibile creare i tool robotizzati che ci potessero dare quella automatizzazione distribuita e suddivisa per l'intera vita di SourceForge.

Questi inconvenienti, se da una parte hanno rallentato il lavoro, dall'altra ci hanno permesso di comprendere su quale strada continuare lo sviluppo e su cosa sarebbe meglio orientarsi per migliorare i prodotti e renderli così appetibili al grande pubblico che ancora non si è avvicinato al mondo dell'open source.

6.2. SVILUPPI FUTURI

Questo lavoro di tesi ha permesso di analizzare varie caratteristiche dei software open source, ma ovviamente non tutte. Ciò mette le basi per un primo sviluppo che potrebbe essere fatto: quello di continuare l'analisi appena terminata andando a studiare la popolarità e l'attività dei prodotti, cercando la soddisfazione degli utenti rispetto ai software presenti nel CVS. Ciò potrebbe essere fatto analizzando i forum, le mailing list, le richieste di supporto.

Inoltre i problemi riscontrati ci hanno permesso di capire anche quali caratteristiche migliorare, come per esempio la centralizzazione dei dati in un unico database e l'accesso diretto allo stesso.

Altro sviluppo potrebbe essere quello di sensibilizzare le comunità open source che scrivono i sistemi di tracking, al fine di evitare i problemi visti con quello di SourceForge, si ricordi infatti la cattiva gestione di assegnamento alla priorità intermedia.

Bibliografia

1. Elena Grandi, *Introduzione al mondo del Software Libero e dell'Open Source*
2. Richard M. Stallman, *Free Software, Free Society: Selected Essays of Richard M. Stallman*
3. Lawrence Lessig, *Cultura libera. Un equilibrio fra anarchia e controllo, contro l'estremismo della proprietà intellettuale*
4. Di Bona, Ockman, Stone, *Open Sources. Voci dalla rivoluzione Open Source*
5. R. Borruso, *La tutela giuridica del software. Diritto d'autore e brevettabilità*
6. D.Taibi, L.Lavezza, S.Morasca. *OPENBQR: A Framework for the assessment of the q.s.s. In proceding OSS2007*
7. Jeff Tian, *Software Quality Engineering – Testing, Quality Assurance and Quantifiable Improvement : John Wiley & Sons, Inc. 2005*
8. Luigi Buglione, *Misurare il software. Quantità, qualità, standards e miglioramento di processo nell'Information Technology*
9. Karl Fogel, *Open Source Development With Cvs*
10. Fuggetta, A., *Open source software, an evaluation, The Journal of Systemsand Software*
11. Perry Donham, *Ten Rules for Evaluating Open Source Software*
12. Stamelos I., Angelis L., Oikonomou A., Bleris G., *Code quality analysis in open source software development, Systems J,*
13. Cignoni Giovanni, De Risi Piero, *Il test e la qualità del software : Un modello per lo sviluppo, il controllo di qualità e l'attività di test del software*
14. Casambenti Walter, *Progetto Qualisoft*
15. Capiluppi A., Lago P., Morisio M., *Characteristics of Open Source Prjects*
16. Weiss D., *Quantitative Analysis of Open Source Projects on SourceForge*

Sitografia

1. http://www.notiziariogiuridico.it/software1.html
2. http://www.pluto.it/files/ildp/doc-it/intro-swlibero/
3. http://www.fsf.org
4. http://www.fsf.org/licensing/essays/free-sw.html
5. http://opensource.org/docs/osd
6. http://opensource.org
7. http://www.gnu.org/copyleft/copyleft.html
8. http://www.gnu.org/copyleft/gpl.html
9. http://www.microsoft.com
10. http://www.stallman.org/
11. http://opensource.org/licenses/bsd-license.php
12. http://www.gnu.org
13. http://www.nongnu.org/cvs/
14. http://Freshmeat.net
15. http://Rubyforge.org
16. http://ossmole.sourceforge.net/
17. http://sourceforge.net/
18. https://zerlot.cse.nd.edu/
19. http://zerlot.cse.nd.edu/mywiki/
20. http://www.seriouslyopen.org/
21. http://www.openbrr.org
22. http://www.qsos.org
23. http://www.taibi.it/openbqr
24. http://ossmole.sourceforge.net/
25. https://zerlot.cse.nd.edu/mywiki/index.php?title=FAQ#Date_Format
26. http://support.intel.com/support/processors/pentium/fdiv/wp/

27. http://www.issco.unige.ch/projects/ewg96/node14.html#SECTION00311000000000000000
28. http://www.dicom.uninsubria.it/qualipso/doku.php?id=literature_reviews
29. http://www.dicom.uninsubria.it/qualipso/doku.php?id=characteristics_os_projects
30. http://www.dicom.uninsubria.it/qualipso/doku.php?id=floss_repositories
31. http://www.dicom.uninsubria.it/qualipso/doku.php?id=quantitative_analysis_of_open_source_projects_on_sourceforge

Appendice A

Elenco delle query usate per reperire i dati totali all'interno del database di Notre Dame

1. Totale dei bugs scoperti divisi per livello di criticità e per stabilità dei progetti:

SELECT *artifact.priority, count(*) AS Nr.Bugs , trove_group_link.trove_cat_id*

FROM *sf1007.artifact_group_list, sf1007.artifact, sf1007.trove_group_link*

WHERE *(artifact.open_date != 0) and artifact_group_list.name = 'Bugs' and (artifact_group_list.group_artifact_id = artifact.group_artifact_id) and (artifact_group_list.group_id = trove_group_link.group_id) and trove_group_link.trove_cat_id between 8 and 12 group by artifact.priority, trove_group_link.trove_cat_id order by artifact.priority, trove_group_link.trove_cat_id*

2. Totale dei bugs scoperti ed assegnati ad uno sviluppatore divisi per livello di criticità e per stabilità dei progetti:

SELECT *count(distinct artifact.assigned_to), priority, trove_group_link.trove_cat_id*

FROM *sf1007.artifact, sf1007.artifact_group_list, sf1007.trove_group_link*

WHERE *artifact.assigned_to != 100 and artifact_group_list.name = 'Bugs' and (artifact_group_list.group_artifact_id = artifact.group_artifact_id) and artifact.open_date != 0 and (artifact_group_list.group_id = trove_group_link.group_id) and trove_group_link.trove_cat_id between 8 and 12 group by priority, trove_group_link.trove_cat_id order by priority, trove_group_link.trove_cat_id*

3. Totale dei bugs scoperti e non assegnati ad uno sviluppatore divisi per livello di criticità e per stabilità dei progetti:

SELECT *count(*), trove_group_link.trove_cat_id, priority*

FROM *sf1007.artifact, sf1007.artifact_group_list, sf1007.trove_group_link*

WHERE *assigned_to = 100 and open_date != 0 and artifact_group_list.name = 'Bugs' and (artifact_group_list.group_artifact_id = artifact.group_artifact_id) and (artifact_group_list.group_id = trove_group_link.group_id) and trove_group_link.trove_cat_id between 8 and 12 group by priority, trove_group_link.trove_cat_id*

order by trove_group_link.trove_cat_id, priority

4. Tempo medio di assegnazione dei bugs diviso per livello di criticità e per stabilità dei progetti:

SELECT *artifact.priority, avg((artifact_history.entrydate - artifact.open_date)/60/60/24), count(*), trove_group_link.trove_cat_id*

FROM *sf1007.artifact_history, sf1007.artifact, sf1007.artifact_group_list, sf1007.trove_group_link*

WHERE *(artifact_group_list.group_artifact_id = artifact.group_artifact_id) and (artifact.artifact_id = artifact_history.artifact_id) and artifact.open_date != 0 and artifact_group_list.name = 'Bugs' and artifact_history.field_name = 'assigned_to' and artifact.assigned_to != 100 and (artifact_group_list.group_id = trove_group_link.group_id) and trove_group_link.trove_cat_id between 8 and 12 group by artifact.priority, trove_group_link.trove_cat_id order by artifact.priority, trove_group_link.trove_cat_id*

5. Tempo medio di risoluzione dei bugs diviso per livello di criticità e per stabilità dei progetti:

SELECT *priority, avg((close_date-open_date) /60/60/24), trove_group_link.trove_cat_id*

FROM *sf1007.artifact, sf1007.artifact_group_list, sf1007.trove_group_link*

WHERE *artifact_group_list.name = 'Bugs' and (artifact_group_list.group_artifact_id = artifact.group_artifact_id) and (artifact_group_list.group_id = trove_group_link.group_id) and trove_group_link.trove_cat_id between 8 and 12 and close_date != 0 and open_date != 0 group by priority, trove_group_link.trove_cat_id order by priority, trove_group_link.trove_cat_id*

6. Totale delle patches rilasciate divise per livello di criticità e per stabilità dei progetti:

SELECT *artifact.priority, count(*) , trove_group_link.trove_cat_id*

FROM *sf1007.artifact_group_list, sf1007.artifact, sf1007.trove_group_link*

WHERE *artifact_group_list.name = 'Patches' and (artifact_group_list.group_artifact_id = artifact.group_artifact_id) and (artifact_group_list.group_id = trove_group_link.group_id) and trove_group_link.trove_cat_id between 8 and 12 group by artifact.priority, trove_group_link.trove_cat_id order by artifact.priority, trove_group_link.trove_cat_id*

7. Totale delle release rilasciate divise per stabilità dei progetti:

SELECT *count(release_id) as NrRelease, trove_group_link.trove_cat_id*

FROM *sf1007.frs_package, sf1007.groups, sf1007.frs_release, sf1007.trove_group_link*

WHERE *(groups.group_id = frs_package.group_id) and (frs_package.package_id = frs_release.package_id) and (frs_package.group_id = trove_group_link.group_id) and trove_group_link.trove_cat_id between 8 and 12 group by trove_group_link.trove_cat_id order by trove_group_link.trove_cat_id, NrRelease*

8. Nr. di documenti presenti all'interno del CVS divisi per stabilità dei progetti:

SELECT *count (*), trove_group_link.trove_cat_id*

FROM *sf1007.doc_groups, sf1007.doc_data, sf1007.trove_group_link*

WHERE *(doc_groups.group_id = trove_group_link.group_id) and trove_group_link.trove_cat_id between 8 and 12 and (doc_data.doc_group = doc_groups.doc_group) group by trove_group_link.trove_cat_id order by trove_group_link.trove_cat_id*

9. Nr. di progetti che hanno avuto una donazione confrontati con quelli che non ne hanno avuta divisi per stabilità:

SELECT *count(*) , trove_group_link.trove_cat_id, groups.donate_optin*

FROM *sf1007.trove_group_link, sf1007.groups*

WHERE *(groups.group_id = trove_group_link.group_id) and trove_group_link.trove_cat_id between 8 and 12 and groups.donate_optin between 0 and 1 group by trove_group_link.trove_cat_id, groups.donate_optin order by trove_group_link.trove_cat_id*

10. Nr. di utenti che hanno fatto una donazione confrontati con quelli che non ne hanno fatto divisi per ruolo ricoperto:

SELECT *count(*), user_group.member_role, users.donate_optin*

FROM *sf1007.users, sf1007.user_group*

WHERE *user_group.member_role between 1 and 105 and (user_group.user_id = users.user_id) and users.donate_optin between 0 and 1 group by user_group.member_role, users.donate_optin order by user_group.member_role*

11. Nr. di download divisi per stabilità dei prodotti:

SELECT *sum (stats_groupid_alltime_agg.downloads), trove_group_link.trove_cat_id*

FROM *sf1007.stats_groupid_alltime_agg, sf1007.trove_group_link, sf1007.groups*

WHERE *(stats_groupid_alltime_agg.group_id= groups.group_id) and (groups.group_id = trove_group_link.group_id) and trove_group_link.trove_cat_id between 8 and 12 group by trove_group_link.trove_cat_id order by trove_group_link.trove_cat_id*

12. Nr. di progetti presenti all'interno del CVS:

SELECT *count (*)*

FROM *sf1007.groups*

13. Nr. di tutti i ruoli delle comunità di SourceForge raggruppati per stabilità dei progetti e per ruoli:

SELECT *count(*), trove_group_link.trove_cat_id, user_group.member_role*

FROM *sf1007.trove_group_link, sf1007.user_group*

WHERE *user_group.member_role between 1 and 105 and (user_group.group_id = trove_group_link.group_id) and trove_group_link.trove_cat_id between 8 and 12 group by trove_group_link.trove_cat_id, user_group.member_role order by trove_group_link.trove_cat_id, user_group.member_role*

14. Somma del solo ruolo "developer" raggruppato per stabilità dei progetti:

SELECT *count(*), trove_group_link.trove_cat_id*

FROM *sf1007.trove_group_link, sf1007.user_group*

WHERE *user_group.member_role = 1 and (user_group.group_id = trove_group_link.group_id) and trove_group_link.trove_cat_id between 8 and 12 group by trove_group_link.trove_cat_id order by trove_group_link.trove_cat_id*

15. Somma di tutti i ruoli delle comunità di SourceForge raggruppati solo per stabilità dei progetti:

SELECT *count(*), trove_group_link.trove_cat_id*

FROM *sf1007.trove_group_link, sf1007.user_group*

WHERE *user_group.member_role between 1 and 105 and (user_group.group_id = trove_group_link.group_id) and trove_group_link.trove_cat_id between 8 and 12 group by trove_group_link.trove_cat_id order by trove_group_link.trove_cat_id*

16. Totale degli utenti divisi per ruolo:

SELECT *count(*), user_group.member_role*

FROM *sf1007.users, sf1007.user_group*

WHERE *user_group.member_role between 1 and 105 and (user_group.user_id = users.user_id) group by user_group.member_role order by user_group.member_role*

17. Decodifica gli id dei ruoli degli sviluppatori:

SELECT *

FROM *sf1007.people_job_category order by category_id*

18. Decodifica gli id in valori testuali:

SELECT *

FROM *sf1007.trove_cat order by trove_cat_id*

19. Progetto che ha i bugs di criticità 10:

SELECT *count(*), groups.group_name*

FROM *sf1007.artifact, sf1007.artifact_group_list, sf1007.groups*

WHERE *artifact_group_list.name = 'Bugs' and (artifact_group_list.group_artifact_id = artifact.group_artifact_id) and (artifact_group_list.group_id = groups.group_id) and priority = 10 group by groups.group_name*

Elenco delle query usate per reperire i dati dello studio semestrale. Per ogni query nella clausola "WHERE"è necessario sostituire i valori di timestamp presenti con tutti quelli riportati in appendice B per ottenere i dati completi.

20. Bugs scoperti nella settimana in esame divisi per livello di criticità e per stabilità dei progetti:

SELECT *artifact.priority, count(*) , trove_group_link.trove_cat_id*

FROM *sf1007.artifact_group_list, sf1007.artifact, sf1007.trove_group_link*

WHERE *(artifact.open_date between 1175378400 and 1175896800) and artifact_group_list.name = 'Bugs' and (artifact_group_list.group_artifact_id = artifact.group_artifact_id) and (artifact_group_list.group_id = trove_group_link.group_id) and trove_group_link.trove_cat_id between 8 and 12 group by artifact.priority, trove_group_link.trove_cat_id order by artifact.priority, trove_group_link.trove_cat_id*

21. Bugs scoperti nella settimana in esame ed assegnati ad uno sviluppatore divisi per livello di criticità e per stabilità dei progetti:

SELECT *count(distinct artifact.assigned_to), priority, trove_group_link.trove_cat_id*

FROM *sf1007.artifact, sf1007.artifact_group_list, sf1007.trove_group_link*

WHERE *artifact.assigned_to != 100 and artifact_group_list.name = 'Bugs' and (artifact_group_list.group_artifact_id = artifact.group_artifact_id) and artifact.open_date between 1175378400 and 1175896800 and (artifact_group_list.group_id = trove_group_link.group_id) and trove_group_link.trove_cat_id between 8 and 12 group by priority, trove_group_link.trove_cat_id order by priority, trove_group_link.trove_cat_id*

22. Bugs scoperti nella settimana in esame e non assegnati ad uno sviluppatore divisi per livello di criticità e per stabilità dei progetti:

SELECT *count(*), trove_group_link.trove_cat_id, priority*

FROM *sf1007.artifact, sf1007.artifact_group_list, sf1007.trove_group_link*

WHERE *assigned_to = 100 and open_date between 1175378400 and 1175896800 and artifact_group_list.name = 'Bugs' and (artifact_group_list.group_artifact_id = artifact.group_artifact_id) and (artifact_group_list.group_id = trove_group_link.group_id) and trove_group_link.trove_cat_id between 8 and 12 group by priority, trove_group_link.trove_cat_id order by trove_group_link.trove_cat_id, priority*

23. Tempo medio di assegnazione dei bugs diviso per livello di criticità e per stabilità dei progetti, nella settimana in esame:

SELECT *artifact.priority, avg((artifact_history.entrydate - artifact.open_date)/60/60/24), count(*),*

trove_group_link.trove_cat_id

FROM *sf1007.artifact_history, sf1007.artifact, sf1007.artifact_group_list, sf1007.trove_group_link*

WHERE *(artifact_group_list.group_artifact_id = artifact.group_artifact_id) and (artifact.artifact_id = artifact_history.artifact_id) and artifact.open_date between 1175378400 and 1175896800 and artifact_group_list.name = 'Bugs' and artifact_history.field_name = 'assigned_to' and artifact.assigned_to != 100 and (artifact_group_list.group_id = trove_group_link.group_id) and trove_group_link.trove_cat_id between 8 and 12 group by artifact.priority, trove_group_link.trove_cat_id order by artifact.priority, trove_group_link.trove_cat_id*

24. Tempo medio di risoluzione dei bugs diviso per livello di criticità e per stabilità dei progetti, nella settimana in esame:

SELECT *priority, avg((close_date-open_date) /60/60/24), trove_group_link.trove_cat_id*

FROM *sf1007.artifact, sf1007.artifact_group_list, sf1007.trove_group_link, sf1007.groups*

WHERE *artifact_group_list.name = 'Bugs' and (artifact_group_list.group_artifact_id = artifact.group_artifact_id) and (artifact_group_list.group_id = trove_group_link.group_id) and trove_group_link.trove_cat_id between 8 and 12 and close_date between 1175378400 and 1175896800 and open_date != 0 group by priority, trove_group_link.trove_cat_id order by priority, trove_group_link.trove_cat_id*

25. Nascita di nuovi progetti, nella settimana in esame:

SELECT *count (*)*

FROM *sf1007.groups*

WHERE *register_time between 1175378400 and 1175896800*

26. Patches rilasciate nella settimana in esame divise per livello di criticità e per stabilità dei progetti:

SELECT *artifact.priority, count(*) , trove_group_link.trove_cat_id*

FROM *sf1007.artifact_group_list, sf1007.artifact, sf1007.trove_group_link*

WHERE *(artifact.open_date between 1175378400 and 1175896800) and artifact_group_list.name = 'Patches' and (artifact_group_list.group_artifact_id = artifact.group_artifact_id) and (artifact_group_list.group_id = trove_group_link.group_id) and trove_group_link.trove_cat_id between 8 and 12 group by*

artifact.priority, trove_group_link.trove_cat_id order by artifact.priority, trove_group_link.trove_cat_id

27. Release rilasciate nella settimana in esame divise per stabilità dei progetti:

SELECT *count(release_id) as NrRelease, trove_group_link.trove_cat_id*

FROM *sf1007.frs_package, sf1007.groups, sf1007.frs_release, sf1007.trove_group_link*

WHERE *(groups.group_id = frs_package.group_id) and (frs_package.package_id = frs_release.package_id) and (frs_package.group_id = trove_group_link.group_id) and trove_group_link.trove_cat_id between 8 and 12 and frs_release.release_date between 1175378400 and 1175896800 group by trove_group_link.trove_cat_id order by trove_group_link.trove_cat_id, NrRelease*

Appendice B

Valori di timestamp usati per lo studio dei dati semestrali per il periodo 01 aprile 2007 – 30 settembre 2007.

Nr. Settimane	Timestamp Iniziale	Timestamp finale
Settimana 1	1175378400000	1175896800000
Settimana 2	1175983200000	1176501600000
Settimana 3	1176588000000	1177106400000
Settimana 4	1177192800000	1177711200000
Settimana 5	1177797600000	1178316000000
Settimana 6	1178402400000	1178920800000
Settimana 7	1179007200000	1179525600000
Settimana 8	1179612000000	1180130400000
Settimana 9	1180216800000	1180735200000
Settimana 10	1180821600000	1181340000000
Settimana 11	1181426400000	1181944800000
Settimana 12	1182031200000	1182549600000
Settimana 13	1182636000000	1183154400000
Settimana 14	1183240800000	1183759200000
Settimana 15	1183845600000	1184364000000
Settimana 16	1184450400000	1184968800000
Settimana 17	1185055200000	1185573600000
Settimana 18	1185660000000	1186178400000
Settimana 19	1186264800000	1186783200000
Settimana 20	1186869600000	1187388000000
Settimana 21	1187474400000	1187992800000
Settimana 22	1188079200000	1188597600000
Settimana 23	1188684000000	1189202400000
Settimana 24	1189288800000	1189807200000
Settimana 25	1189893600000	1190412000000
Settimana 26	1190498400000	1191016800000

Appendice C

Struttura delle tabelle del database di Notre Dame usate nelle query.

Table "sf1007.groups" Contiene i dati relativi ai progetti.

Column	**Type**	**Modifiers**
group_id	integer	not null default nextval(('groups_pk_seq'::text)::regclass)
group_name	character varying(40)	
homepage	character varying(128)	
is_public	integer	not null default 0
status	character(1)	not null default 'A'::bpchar
unix_group_name	character varying(30)	not null default ''::character varying
http_domain	character varying(80)	
short_description	character varying(255)	
license	character varying(16)	
register_time	integer	not null default 0
use_mail	integer	not null default 1
use_forum	integer	not null default 1
use_pm	integer	not null default 1
use_cvs	integer	not null default 1
use_news	integer	not null default 1
preferred_support_type	integer	not null default 1
preferred_support_resource	text	not null default ''::text
type	integer	not null default 1
use_docman	integer	not null default 1
not_open_source	integer	not null default 0
send_all_tasks	integer	not null default 0
use_pm_depend_box	integer	not null default 1
potm	integer	
donation_request	text	
donate_optin	integer	default 0
big_mirror	integer	
project_submitter	integer	
row_modtime	integer	
use_screenshots	integer	not null default 1
use_svn	integer	not null default 0
disable_mostactive	integer	
additional_trove_cats	integer	not null default 0
use_wiki	integer	not null default 0

Table "sf1007.users" Contiene i dati relativi agli utenti

Column	Type	Modifiers
user_id	integer	not null default nextval(('users_pk_seq'::text)::regclass)
user_name	text	not null default ''::text
realname	character varying(32)	not null default ''::character varying
status	character(1)	not null default 'A'::bpchar
unix_uid	integer	default 0
add_date	integer	not null default 0
people_resume	text	not null default ''::text
timezone	character varying(64)	default 'GMT'::character varying
language	integer	not null default 275
stay_anon	integer	default 0
donation_request	text	
donate_optin	integer	default 0
last_sitestatus_view	integer	default 0
row_modtime	integer	

Table "sf1007.user_group" Mette in relazione gli utenti con i progetti.

Column	Type	Modifiers
user_group_id	integer	not null default nextval(('user_group_pk_seq'::text)::regclass)
user_id	integer	not null default 0
group_id	integer	not null default 0
admin_flags	character(16)	not null default ''::bpchar
forum_flags	integer	not null default 0
project_flags	integer	not null default 2
doc_flags	integer	not null default 0
member_role	integer	not null default 100
release_flags	integer	not null default 0
artifact_flags	integer	
added_by	integer	not null default 100
grantcvs	integer	not null default 1
grantshell	integer	not null default 1
row_modtime	integer	
news_flags	integer	not null default 0
screenshot_flags	integer	not null default 0
grantsvn	integer	not null default 1

Table "sf1007.artifact" Contiene i dati delle caratteristiche sottoposte a tracking

Column	Type	Modifiers
artifact_id	integer	not null default nextval(('"artifact_artifact_id_seq"'::text)::regclass)
group_artifact_id	integer	not null
status_id	integer	not null default 1
category_id	integer	not null default 100
artifact_group_id	integer	not null default 0
resolution_id	integer	not null default 100
priority	integer	not null default 5
submitted_by	integer	not null default 100
assigned_to	integer	not null default 100

open_date	integer	not null default 0
close_date	integer	not null default 0
summary	text	not null
details	text	not null
closed_by	integer	default 100
is_private	integer	default 0

Table "sf1007.artifact_group_list" Mette in relazione la tabella del tracking con quella dei progetti

Column	**Type**	**Modifiers**
group_artifact_id	integer	not null default nextval(('"artifact_grou_group_artifac_seq"'::text)::regclass)
group_id	integer	not null
name	text	
description	text	
is_public	integer	not null default 0
allow_anon	integer	not null default 0
email_all_updates	integer	not null default 0
due_period	integer	not null default 2592000
use_resolution	integer	not null default 0
submit_instructions	text	
browse_instructions	text	
datatype	integer	not null default 0
status_timeout	integer	
due_period_initial	integer	not null default 0
due_period_update	integer	not null default 0
reopen_on_comment	integer	not null default 0

Table "sf1007.artifact_history" Contiene la storia dei cambiamenti di ogni componente sottoposto a tracking

Column	**Type**	**Modifiers**
id	integer	not null default nextval(('"artifact_history_id_seq"'::text)::regclass)
artifact_id	integer	not null default 0
field_name	text	not null default ''::text
old_value	text	not null default ''::text
mod_by	integer	not null default 0
entrydate	integer	not null default 0

Table "sf1007.trove_group_link" Identifica le proprietà esterne di un progetto

Column	**Type**	**Modifiers**
trove_group_id	integer	not null default nextval(('trove_group_link_pk_seq'::text)::regclass)
trove_cat_id	integer	not null default 0
trove_cat_version	integer	not null default 0
group_id	integer	not null default 0
trove_cat_root	integer	not null default 0
entity_type	integer	not null default 3

Table "sf1007.trove_cat" Decodifica gli id della tabella trove_group_link in valori testuali

Column	Type	Modifiers
trove_cat_id	integer	not null default nextval(('trove_cat_pk_seq'::text)::regclass)
version	integer	not null default 0
parent	integer	not null default 0
root_parent	integer	not null default 0
shortname	character varying(80)	
fullname	character varying(80)	
description	character varying(255)	
fullpath	text	not null default ''::text
fullpath_ids	text	
parent_only	integer	not null default 0
people_skill	integer	not null default 0

Table "sf1007.frs_release" Contiene i dati relativi alle release

Column	Type	Modifiers
release_id	integer	not null default nextval(('frs_release_pk_seq'::text)::regclass)
package_id	integer	not null default 0
name	text	
notes	text	
changes	text	
status_id	integer	not null default 0
preformatted	integer	not null default 0
release_date	integer	not null default 0
released_by	integer	not null default 0

Table "sf1007.frs_package" Mette in relazione le release con i progetti

Column	Type	Modifiers
package_id	integer	not null default nextval(('frs_package_pk_seq'::text)::regclass)
group_id	integer	not null default 0
name	text	
status_id	integer	not null default 0

Table "sf1007.doc_data" Contiene i dati relativi alla documentazione

Column	Type	Modifiers
docid	integer	not null default nextval(('doc_data_pk_seq'::text)::regclass)
stateid	integer	not null default 0
title	character varying(255)	not null default ''::character varying
data	text	not null default ''::text
updatedate	integer	not null default 0
createdate	integer	not null default 0
created_by	integer	not null default 0
doc_group	integer	not null default 0
description	text	
language_id	integer	not null default 275

Table "sf1007.doc_groups" Mette in relazione i documenti con i progetti

Column	Type	Modifiers
doc_group	integer	not null default nextval(('doc_groups_pk_seq'::text)::regclass)
groupname	character varying(255)	not null default ''::character varying
group_id	integer	not null default 0

Table "sf1007.people_job_category" Contiene la decodifica dei ruoli degli sviluppatori

Column	Type	Modifiers
category_id	integer	not null default nextval(('people_job_category_pk_seq'::text)::regclass)
name	text	
private_flag	integer	not null default 0