

UNIVERSITÁ DI TRENTO

CORSO HCI A.A. 2014-2015

PARTECIPATORY DEVELOPMENT

Gestione delle labels di Github

Bortoli GIANLUCA	159993
Brugnara MARTIN	157791
Dellera ANDREA	158365
Hoxha FATBARDHA	161003

December 16, 2014

Contents

1	Introduzione	2
2	Interviste	3
2.1	Traccia	3
2.2	Analisi dei risultati	3
3	Benchmarking	6
3.1	Contribuire ad un progetto	6
4	Design library: NeoVim	8
4.1	Buoni esempi	8
4.2	Cattivi esempi	10
5	Prototipi	12
5.1	Prototipo a bassa fedeltá	12
5.2	Prototipo ad alta fedeltá	13

1 Introduzione

Il primo obiettivo di questo progetto consiste nell'analizzare é quali siano le principali barriere nella partecipazione ai progetti open source, che oggi sono diventati sempre piú numerosi e di una certa rilevanza (addirittura a livello mondiale in alcuni casi).

Il nostro interesse é ricaduto proprio in queste problematiche che affliggono questo tipo di progetti, dal momento che sono state ravvisate anche in prima persona durante il corso dei nostri studi.

Questo approfondimento deriva in parte anche dalla nostra propensione ed interesse per l'utilizzo di software open source durante la nostra carriera universitaria e lavorativa.

Lo scopo finale é quello di trovare e formulare una possibile soluzione al problema della gestione delle etichette su Github (la piattaforma principe per lo sviluppo open source), dal momento che essa é attualmente molto confusionaria e poco ben gestita, soprattutto in progetti di una certa complessità e grandezza.

2 Interviste

Il primo passo per l'individuazione delle principali problematiche legate all'ambito della partecipazione ai progetti open source é stata fatta tramite delle interviste. Questo tipo di ricerca si presta molto alla valutazione di barriere di questo tipo, dal momento che l'intervistato non si sente limitato nell'esprimersi (come potrebbe risultare da un questionario a risposta multipla), ma al contempo non ha la percezione di dilungarsi troppo (come invece potrebbe accadere se viene presentato un questionario a domanda aperta) che potrebbe indurlo a non scrivere tutto ciò che pensa.

Con il tramite dell'intervista "faccia a faccia" questi inconvenienti vengono meno: la persona si sente piú libera di discutere con l'intervistatore che pone le domande e non blocca l'intervistato mentre sta rispondendo, non interrompendo cosí il flusso delle idee (che é la parte essenziale e di reale interesse dell'intera intervista).

2.1 Traccia

La traccia ¹ delle domande da porre all'intervistato ci é stata fornita direttamente dalla dottoressa Bordin, poichè tale argomento non era ancora stato trattato a lezione nel momento in cui abbiamo svolto questa parte del progetto.

Essa prevedeva delle domande mirate principalmente a capire:

- se e quali software open source vengono utilizzati
- se l'intervistato partecipi/abbia partecipato attivamente a tali progetti
- quali siano le motivazioni che lo hanno portato a farlo oppure no
- cosa significhi *partecipare* ad un progetto open source

2.2 Analisi dei risultati

Il campione su cui é stata effettuata l'intervista non é molto eterogeneo (come é possibile vedere dalla Figura 1, dal momento che é stato piú immediato intervistare

¹http://disi.unitn.it/~deangeli/homepage/lib/exe/fetch.php?media=teaching:hci:hci2014_2015:intervista_pd_motivazioni.pdf

delle persone all'interno del nostro corso di laurea piuttosto che di altri atenei o dei lavoratori.

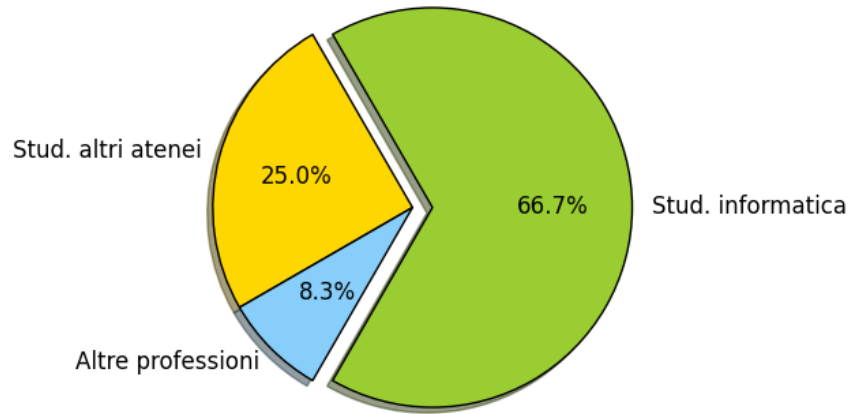


Figure 1: Distribuzione della professione delle persone intervistate.

Abbiamo potuto notare come gli unici due intervistati che hanno contribuito attivamente a progetti open source provenissero esclusivamente da un ambito informatico. Inoltre, all'interno del gruppo stesso ben pochi lo hanno fatto, come é possibile notare dalla Figura 2.

Ciò ci permette di creare un profilo di utenza specifica (una *personas*) in grado di prendere parte a questa tipologia di progetti.

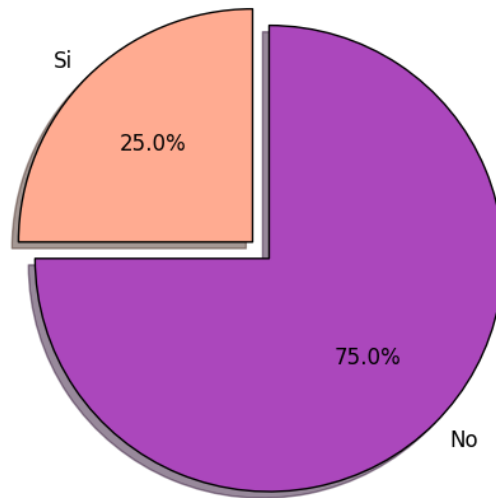


Figure 2: Persone che hanno attivamente partecipato a progetti open source tra gli studenti di informatica.

Da ciò si evince che l'ambito e la possibilità di parteciparvi è molto ristretto e ciò ci permette di delineare una *personas* con delle caratteristiche ben definite, quali:

- consistente background informatico
- interesse verso il campo specifico del progetto considerato
- tempo e impegno da dedicarvi

In aggiunta abbiamo notato come lo *scenario* accademico incentivi e contribuisca ad utilizzare specifici software open source e, di conseguenza, aumenti la probabilità che uno studente si interessi e ne possa far parte.

3 Benchmarking

Durante questa fase abbiamo scelto di concentrarci su tre progetti open source: NeoVim, CyanogenMod e OpenOffice.

Dopo averli presentati e descritti durante un workshop, abbiamo analizzato i pro ed i contro di ciascuno, da cui é emerso che molti utenti utilizzano spesso questi software sia in ambito personale che lavorativo.

Quelli da noi considerati sono gratuiti ², di facile utilizzo e mantenuti costantemente aggiornati dalla comunità di sviluppo. In più le funzionalità offerte sono molto simili a quelle dei rispettivi software equivalenti, se non addirittura le medesime.

Al contrario, essi non vengono largamente utilizzati a causa della scarsa conoscenza del prodotto, dal momento che vengono scarsamente pubblicizzati soprattutto a livello delle istituzioni pubbliche, e per la loro usabilità sulla quale spesso non viene fatto uno studio molto approfondito.

3.1 Contribuire ad un progetto

Dalle interviste é emerso che il significato dato al termine *contribuire* é quello di mettere a disposizione le proprie competenze per il miglioramento di un software e/o la risoluzione di eventuali problemi ad esso legati.

Le motivazioni che spingono a contribuire sono molte: dalla semplice soddisfazione per aver risolto un bug, all'arricchimento personale e l'esperienza acquisita che ne consegue, passando per la necessità di dover implementare una funzionalità che risolva un problema che si ha nel proprio lavoro.

Al contrario, altrettanti sono i motivi che spingono le persone a non contribuire, tra i quali la mancanza di tempo e di competenze e soprattutto la qualità e disponibilità di informazioni riguardanti un progetto specifico. Quest'ultimo a nostro avviso risulta essere il più rilevante, dal momento che un utente (con magari delle capacità per farlo) si trova a non poter contribuire a causa della mancanza di linee guida su come si possa farlo. Ciò potrebbe sembrare banale a primo impatto, ma

²Non bisogna confondere il concetto di open source con quello di freeware: che un software sia open source non implica necessariamente che sia anche distribuito gratuitamente (ad esempio RedHat)

é una costante in tutti i progetti da noi presi in considerazione.

4 Design library: NeoVim

Nella nostra *design library* ci siamo concentrati sul progetto di NeoVim per poter individuare sia degli esempi di cattiva e che di buona progettazione da cui prendere spunto.

4.1 Buoni esempi

Facciamo ora una lista degli esempi più significativi di un design pensato e studiato appositamente per far sì che l'utente interessato a partecipare riesca a farlo nel modo migliore possibile.

Abbiamo notato come sia il sito di NeoVim che la gestione della codebase su Github siano state progettate secondo il paradigma dell'*user driven design*, il quale mira massimizzare l'usabilità del prodotto.

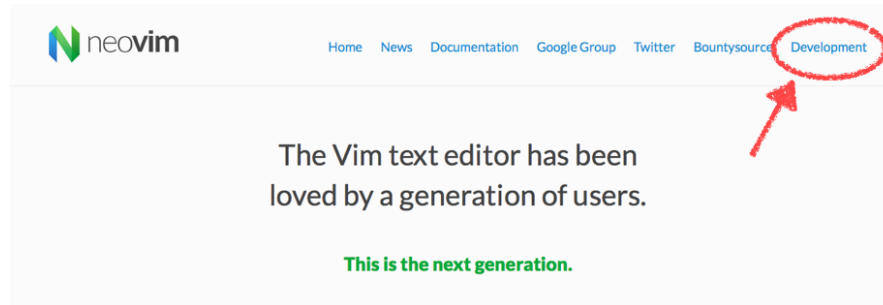



Figure 3: Homepage di NeoVim.

Dalla Figura 3 si può notare come il gruppo di sviluppo del progetto preso in considerazione indichi chiaramente nella pagina principale una sezione per chi volesse contribuire. Già da questo possiamo capire come si punti sullo sviluppo di terze persone: per questo motivo il link al codice sorgente del software su Github viene messo nel menu principale, in modo che sia di facile reperibilità.

What needs to be done

Ready 9

- Entry-level issues 
- Code reviews
- Port OS layer to libuv
- [Merge patches from upstream Vim](#)

Check this [refactoring catalog](#) if you want some inspiration on what to work on.

Figure 4: Cosa c'è da fare.

Un altro dei problemi riscontrati durante la fase di benchmarking é quello di non ritenere le proprie capacità sufficienti per dare il proprio contributo. Ciò é stato parzialmente risolto (Figura 4) introducendo dei *livelli di difficoltà* per le issues/bug presenti nel software.

É stata introdotta un'etichetta per i problemi di più facile risoluzione (entry level), cosa che non avevamo notato in qualsiasi altro progetto da noi preso in considerazione.

```
/*
 * It's possible that autocommands change curbuf to the one being deleted.
 * This might cause curbuf to be deleted unexpectedly. But in some cases
 * it's OK to delete the curbuf, because a new one is obtained anyway.
 * Therefore only return if curbuf changed to the deleted buffer.
 */
if (buf == curbuf && !is_curbuf)
    return;
diff_buf_delete(buf);          /* Can't use 'diff' for unloaded buffer. */
/* Remove any ownsyntax, unless exiting. */
if (firstwin != NULL && curwin->w_buffer == buf)
    reset_synblock(curwin);

/* No folds in an empty buffer. */
FOR_ALL_TAB_WINDOWS(tp, win) {
    if (win->w_buffer == buf) {
        clearFolding(win);
    }
}

ml_close(buf, TRUE);           /* close and delete the memline/memfile */
buf->b_ml.ml_line_count = 0;    /* no lines in buffer */
if ((flags & BFA_KEEP_UNDO) == 0) {
    u_blockfree(buf);          /* free the memory allocated for undo */
    u_clearall(buf);           /* reset all undo information */
}
syntax_clear(&buf->b_s);        /* reset syntax info */
buf->b_flags &= ~BF_READERR;    /* a read error is no longer relevant */
}
```

Figure 5: Homepage di NeoVim.

Infine, viene posta molta cura nella documentazione e nei commenti all'interno del codice. Ciò aiuta chi interviene in quella porzione di sorgente per poter capire meglio e senza eccessiva difficoltà cosa faccia una certa porzione di codice. Inoltre, alla nascita di NeoVim sono state decise delle linee guida ben precise su come si dovesse indentare/organizzare il codice e soprattutto su come commentarlo, in modo da non creare disomogeneità all'interno del progetto (il che creerebbe non pochi problemi di comprensione).

4.2 Cattivi esempi

Al contrario, abbiamo individuato delle parti che risultavano di difficile comprensione e che potevano creare dei problemi con chi volesse interagire.

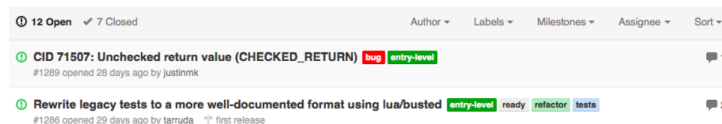


Figure 6: Etichette assegnate alle issue su Github.

Dalla Figura 6 già ad un primo impatto si può notare come le etichette assegnate alle issue ancora da risolvere siano poco esplicative e, a volte, ridondanti. Questo crea molta confusione nel momento in cui lo sviluppatore che vuole dare il proprio contributo controlla cosa ci sia da fare e/o correggere, trovandosi davanti a una miriade di labels con i colori più disparati e visualizzate in modo disorganizzato e con poca coerenza.

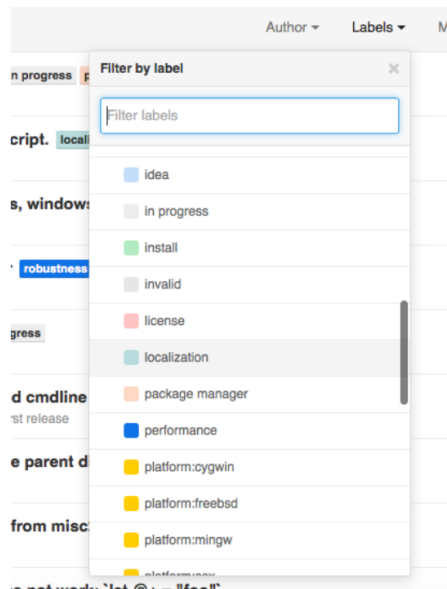


Figure 7: Menu per la selezione delle labels.

Quello della coerenza e dei colori diventa ancora di più un problema nel momento in cui si vuole filtrare tutte le issue per labels.

A colori molto simili, talvolta identici tra loro, sono associate delle problematiche e delle aree di lavoro molto differenti tra loro. Come si può vedere dalla Figura 7, le voci *licence* e *package manager* hanno una colorazione quasi indistinguibile, nonostante riguardino due ambiti molto diversi.

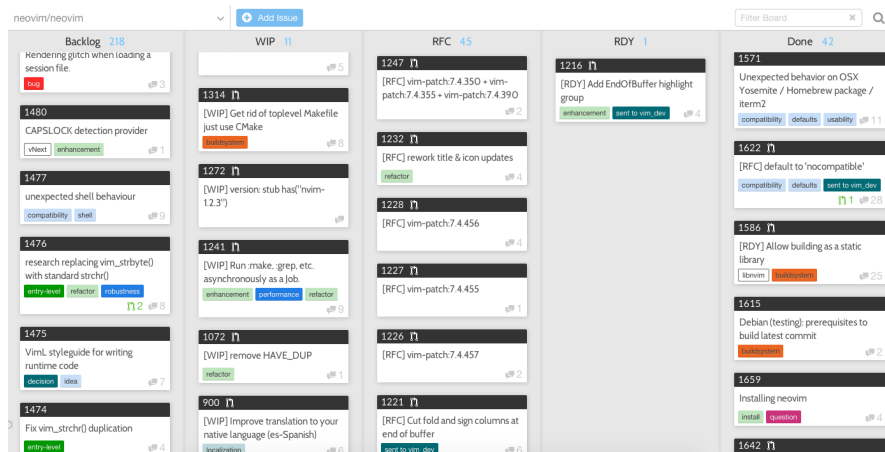


Figure 8: Homepage di NeoVim.

Il problema si estende anche a *Waffle*³, un tool che permette di gestire in modo pi comodo ed immediato le issues di Github in real time. Grazie a questo software direttamente collegato a Github non c' la necessit di controllare manualmente lo stato del lavoro, bens il flusso del lavoro viene direttamente organizzato in base ai commit e alle issue a cui fanno riferimento.

5 Prototipi

5.1 Prototipo a bassa fedelt

Per migliorare il problema delle label, analizzato precedentemente, abbiamo realizzato un prototipo a bassa fedelt tramite *balsamiq*. Le label che fanno riferimento a problemi della stessa area vengono raggruppate in una singola macro-label, espandibile con un click. L'idea originale era di implementare il design pattern file-cartella poi, dopo averci riflettuto, abbiamo deciso di fermarci ad un primo livello. Ci significa che se una label appartiene ad una macro-label non pu avere delle sotto-label. Questo, per quanto intuitivamente possa sembrare una limitazione, aiuta a mantenere la semplicit e la praticit delle label.

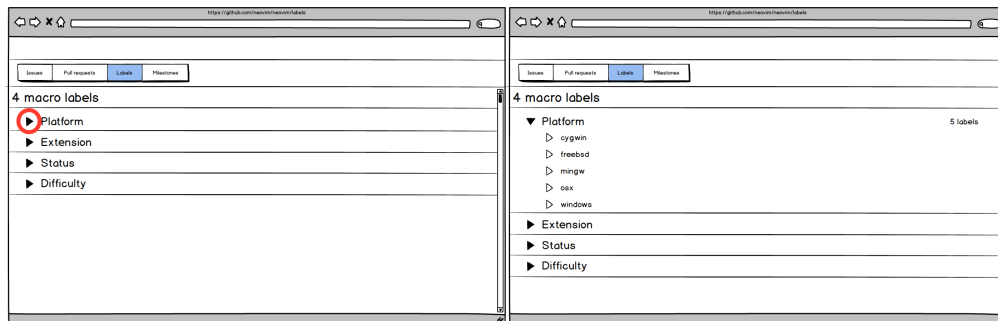


Figure 9: Espansione di una macroarea.

³www.waffle.io

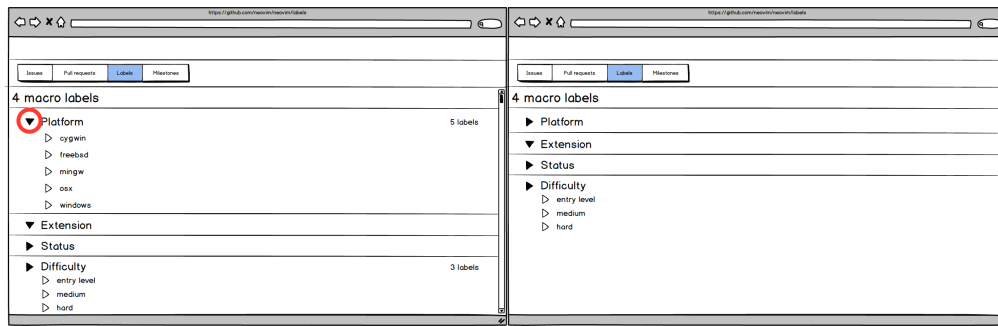


Figure 10: Struttura gerarchica delle etichette.

5.2 Prototipo ad alta fedeltá