UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Quantificazione Automatica del Contributo degli Sviluppatori in Progetti Open Source

RELATORE

Prof. **Filomena Ferrucci**

Dott.ssa **Giulia Sellitto**

Università degli studi di Salerno

CANDIDATO

Mattia Sapere

Matricola: 0512106799

Sommario

La collaborazione senza alcun dubbio risulta essere l'elemento chiave che ha permesso ad ingegneri e programmatori di cimentarsi nello sviluppo di progetti.
L'open source è uno dei concetti fondamentali che ha rivoluzionato il mondo software permettendone una decisiva evoluzione. Il grande utilizzo e l'integrazione di servizi di repository online promuove l'archiviazione e la gestione del codice sorgente.
Suddette repository offrono molti dati che possono essere recuperati ed utilizzati per monitorare il processo di sviluppo del software.
Attraverso l'analisi del codice sorgente e l'utilizzo delle API diviene possibile accedere ad una grande quantità di informazioni utili per monitorare il processo di sviluppo software.
In questo contesto saremo in grado di stabilire l'effettivo contributo versato dagli sviluppatori che partecipano ad un determinato progetto software open source.
A tal proposito è stato progettato ed implementato un Plug-in, denominato RepositoryAnalyzer, che analizza i dati di GitHub tramite l'utilizzo delle Check Runs API al fine di calcolare delle metriche che determinano il contributo versato dagli sviluppatori. È necessario specificare che è stata effettuata un'analisi qualitativa e quantitativa del codice sorgente.
A tal fine sono state elaborate e fornite diverse tipologie di metriche mediante le quali è possibile individuare non solo la produttività di ogni singolo sviluppatore, ma anche la qualità dell'intervento apportato.
Analizziamo il contributo inerente gli ultimi 30 commit, testando il Plug-in su differenti progetti al fine di dimostrare l'effettiva precisione derivante dall'output ottenuto.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Contesto

La crescita ed il veloce progresso dell'open source ha cambiato il modo in cui il software viene sviluppato. Lo sviluppo software è un'attività cooperativa che riunisce un determinato numero di persone, ognuno con conoscenze specifiche, al fine del raggiungimento di un obiettivo comune.
In questo contesto i progetti sono dati dall'unione di differenti contributi che possono derivare sia da persone appartenenti al progetto sia da persone esterne. Da come è possibile dedurre, un concetto fondamentale è quello di collaborazione. Per collaborazione non si intende soltanto la scrittura di codice sorgente, ma anche la stesura di documentazione, la proposta di nuove idee e funzionalità, la risoluzione di bug.
Nella prima sezione del documento, infatti, si procede analizzando l'open source nella sue caratteristiche principali. In seguito vengono riportati differenti esempi di software open source di uso comune in modo da facilitarne la comprensione. Ovviamente ne vengono individuati in dettaglio i punti di forza e di debolezza mediante l'utilizzo di grafici che permettono di osservare in modo intuitivo dati reali rappresentanti statistiche inerenti l'open source.

1.2 Problema e Motivazioni

Esistono varie problematiche inerenti lo sviluppo di progetti software, ma nello specifico viene affrontata quella relativa all'identificazione dell'abbandono e allo stabilimento effettivo del contributo versato dagli sviluppatori. I fattori che inducono i contributor ad abbandonare un progetto sono molteplici quali ad esempio mancanza di tempo, mancanza di interesse, conflitti interni. A tal proposito vengono individuate, mediante l'ausilio di grafici, le motivazioni ricorrenti. Risulta necessario soffermarsi sulle conseguenze che la mutata realtà operativa del software comporta e le soluzioni per evitare che ciò accada.
Uno strumento utile è sicuramente l'individuazione di metriche mediante le quali è possibile quantificare il contributo apportato da ogni sviluppatore e al tempo stesso un eventuale abbandono.

1.3 Obiettivo Fissato

Esistono numerose piattaforme che permettono di offrire servizi di hosting come GitLab, BitBucket e AWS CodeCommit, ma nello specifico abbiamo deciso di considerare GitHub, fondata nel 2008 con il nome di Logical Awesome, la quale attualmente conta una community costituita da oltre 73 milioni di sviluppatori [1].
In questo contesto è stato costruito un Plug-in per Google Chrome, denominato "Repository Analyzer" che esegue una scansione qualitativa e quantitativa della repository GitHub interessata. Nello specifico, dopo aver ottenuto l'URL della repository in questione, vengono analizzati i dati relativi ai contributi come ad esempio il numero di righe di codice inserite, modificate o eliminate da ogni contribuente. Repository Analyzer non ha un dataset fisso e prestabilito, ma è in grado di analizzare una qualsiasi repository GitHub pubblica.
È stato scelto di testarlo su differenti progetti software open source considerando un oracolo completo ed affidabile ed evitando di incorrere in problemi di oracolo incompleto, oracolo parziale ed oracolo artificiale.
L'obiettivo è quello di indagare sul contributo che gli sviluppatori forniscono quando partecipano ad un progetto software open source. Inoltre, basandoci sulle metriche offerte da RepositoryAnalyzer e sugli ultimi 30 commit, siamo in grado di riuscire a comprendere se un progetto sta per essere abbandonato o meno e stabilire l'effettivo lavoro svolto da ogni sviluppatore.

1.4 Risultati Ottenuti

L'effettiva realizzazione di RepositoryAnalyzer rappresenta un grande risultato: mediante tale Plug-in, installabile facilmente da Chrome Web Store, è divenuto possibile quantificare il contributo degli sviluppatori per riuscire a identificarne come conseguenza l'abbandono.
Al capitolo 3 e seguenti si procede ad una dettagliata analisi del funzionamento di RepositoryAnalyzer la cui architettura può essere suddivisa in tre moduli fondamentali descritti in dettaglio che sono: API GitHub, Fetch Data e Quantity and Quality Analyzer. Mediante la visione di grafici è possibile osservare il Plug-in in esecuzione consentendo a tutti di comprenderne moduli e caratteristiche. Successivamente effettuando un'attenta validazione si è riscontrata una precisione elevata delle metriche fornite in output da RepositoryAnalyzer.

CAPITOLO 2

Background

In questo capitolo è illustrato lo stato dell'arte e i lavori correlati trattati nel contesto dei sistemi software open source. Inizialmente viene analizzata la struttura dei sistemi in questione, successivamente si discute dei principali problemi che determinano l'abbandono dei progetti open source da parte degli sviluppatori.
Infine sono esaminate e discusse in dettaglio le principali tecniche di quantificazione del contributo da parte degli sviluppatori in un progetto GitHub per riuscire a determinare l'effettivo lavoro svolto da ogni singolo membro.

2.1 Sviluppo Open Source

2.1.1 Cosa si Intende per Open Source

Nell'ottobre del 1984 Richard Stallman, membro della comunità hacker e del laboratorio di intelligenza artificiale del MIT, fondò la Free Software Foundation(FSF), un movimento socio-tecnologico che rivoluzionò il mondo del software ispirando la creazione di molti programmi che utilizziamo quotidianamente come WordPress, sistema operativo Android, browser Firefox e moltissimi altri.
Da qui nacque il termine *open source* che si riferisce a materiale che le persone possono modificare e condividere essendo pubblicamente accessibile. Nello specifico, con questo termine, si indica un software rilasciato con una licenza che permette agli sviluppatori di accedere liberamente al codice sorgente. Questo permette una certa collaborazione tra sviluppatori che possono interagire offrendosi a vicenda supporto, risorse e materiale. La

GPL (GNU Public License) è la licenza “free software” più comunemente utilizzata, basata su leggi internazionali consente al codice sorgente di un determinato software di essere liberamente utilizzato, modificato e ridistribuito. Essa fornisce i seguenti criteri:

- Tutti possono scaricare ed eseguire il codice.
- Tutti possono modificare il codice.
- Tutti possono ridistribuire versioni gratuite del software.
- Tutti possono distribuire versioni modificate del software.

Attualmente vigono e sono utilizzate due differenti interpretazioni: "free software" e "open source".
Il termine "free software" nasce dal progetto GNU e lascia gli sviluppatori liberi di copiare, cambiare, aggiornare e modificare il codice sorgente.
Il termine "open source" è stato coniato successivamente e secondo Stallman [2] si tratta di un tentativo di esprimere lo stesso concetto, ma con un approccio differente.
Infatti, come sostiene Stallman [2], il significato specifico di open source software è: "puoi vedere il codice sorgente." Questo è un criterio più debole del free software; include free software, ma anche programmi semi-free e alcuni programmi proprietari.
In contrapposizione al termine precedente, con closed source, si indica un software che è proprietario ovvero non distribuito al pubblico, infatti essendo crittografato, soltanto gli autori originali hanno il permesso di vedere e modificare il codice sorgente.
Affinché possa essere utilizzato, un software proprietario dovrà essere acquistato; questo comporterà una maggiore sicurezza per quanto riguarda assistenza e supporto tecnico immediato.

2.1.2 Esempi di Software Open Source

Tra i principali esempi di software open source riportiamo:

- **Mozilla Firefox** è un browser Internet personalizzabile e un software open source gratuito.

Figura 2.1: Esempio del browser Mozilla Firefox dal sito designrush.com

- **VLC Media Player** è un lettore multimediale che permette la riproduzione di file audio e video in diversi formati e su vari dispositivi.

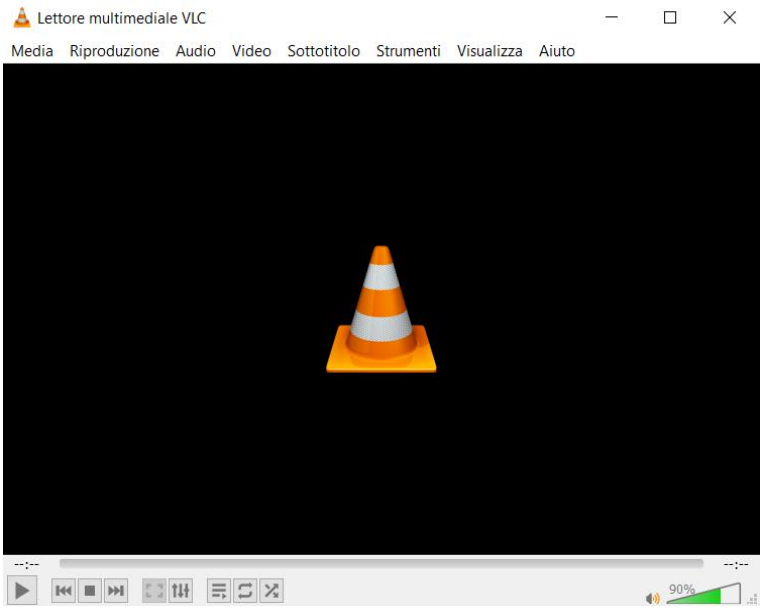

Figura 2.2: Esempio del lettore multimediale VLC Media Player dal sito designrush.com

- **Linux** è un sistema operativo costituito da diverse distribuzioni altamente personalizzabile.

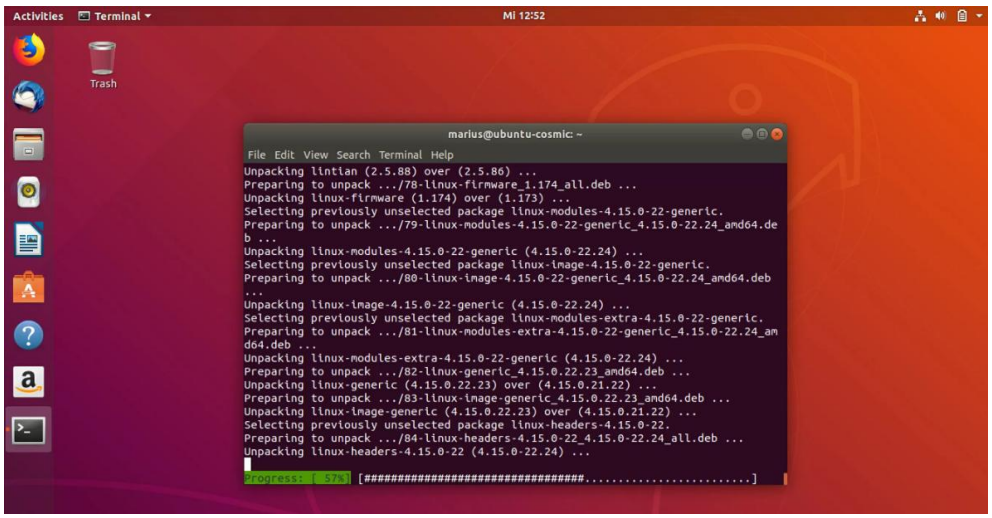

Figura 2.3: Esempio del sistema operativo Linux dal sito designrush.com

2.1.3 Punti di Forza e di Debolezza

Negli ultimi anni l'interesse per l'open source è aumentato notevolmente per tre motivi principali: il successo di Linux e Apache, il timore per il monopolio di Microsoft ed infine la richiesta sempre maggiore di applicazioni software efficaci e affidabili.
Nel corso degli anni gli esperti ICT hanno argomentato una serie di ragioni a favore e contro l'adozione di soluzioni open source.

Punti di Forza

- **Costi minori:** l'adozione di OSS porta ad un risparmio iniziale in termini di risorse, licenze, formazione, migrazione, installazione e gestione.

- **Indipendenza dai fornitori:** consistente nel poter affidare il supporto, l'evoluzione, le personalizzazioni ecc. di un prodotto open source a un'azienda selezionata con gara non essendoci alcun vincolo diretto ed esclusivo tra software adottato e uno specifico fornitore (software house). Si diminuisce così sensibilmente il rischio e il costo di fenomeni di lock-in verso uno o più fornitori.

- **Presenza delle Community:** queste forniscono aiuto e supporto nei confronti di sviluppatori.

- **Flessibilità e Riusabilità:** è possibile realizzare versioni specifiche e personalizzate di un qualsiasi software. Ciò è dovuto alla libera modificabilità del codice sorgente che permette l'adattamento del software a seconda delle proprie esigenze e necessità
- **Interoperabilità:** data la disponibilità del codice sorgente e per la natura stessa della sua programmazione, la creazione di interfacce e applicativi risulta più complessa e costosa per software commerciali rispetto a quelli di tipo open source.

Punti di Debolezza

- **Compatibilità con standard commerciali:** la grande diffusione di alcuni software proprietari ha portato al loro affermarsi come standard de facto sul mercato.

  Nel corso del tempo i software open source hanno migliorato la loro compatibilità dei propri formati con quelli proprietari; per questo motivo questo problema sembra divenire sempre più debole nel corso degli anni.
- **Garanzia e supporto diretto:** per la natura delle licenze OS la garanzia per danni e supporto tecnico non sono garantite.

Un altro concetto fondamentale è quello di **sicurezza** che potrebbe rappresentare un'arma a doppio taglio.
Da una parte avere a disposizione il codice sorgente dei programmi permette di avere una maggiore sicurezza in quanto è possibile effettuare con una maggiore agevolazione controlli su eventuali velnerabilità ad attacchi esterni, dall'altra essendo il codice sorgente pubblicamente accessibile, utenti malevoli potrebbero studiare il software e individuare punti di debolezza da attaccare.
Sono molti i casi di vulnerabilità di componenti e librerie open source che hanno aperto le porte ad attacchi cyber alle aziende. L'ultimo avvenuto nel dicembre del 2021 ha riguardato una vulnerabilità di Log4j (una utility di logging di Apache) che permetteva agli attaccanti d'iniettare codice per sottrarre dati, installare malware o altro.
È inoltre necessario tener presente che le vulnerabilità in questo tipo di software sono generalmente rese pubbliche rapidamente. Infatti se non vengono adottate misure appropriate ed efficaci i dati potrebbero essere in pericolo.
Per questo motivo c'è bisogno di una continua e costante manutenzione del software che prevede la correzione di eventuali bug o/e all'estensione o modifica di alcune caratteristiche.

2.1.4 GitHub e Open Source

GitHub è un servizio web e cloud-based che aiuta gli sviluppatori ad archiviare e gestire il codice permettendo di tener traccia delle modifiche, semplificandone il controllo e la gestione. Ciò garantisce agli sviluppatori un "luogo sicuro" per ospitare progetti software.
A tale scopo GitHub ha concretizzato il progetto "Arctic Code Vault", che ha garantito la salvaguardia del codice open source in un deposito artico per i prossimi 1000 anni.
Essendo per metà un social network e per l'altra un repository, GitHub offre piani di hosting sia a pagamento per progetti privati sia gratuiti per progetti software open source.
Considerando i 73 milioni e più di sviluppatori totali e tenendo conto del codice e delle comunità costruite nell'ultimo anno su GitHub, sono circa 17 milioni i nuovi utenti e 61 milioni le nuove repository create.

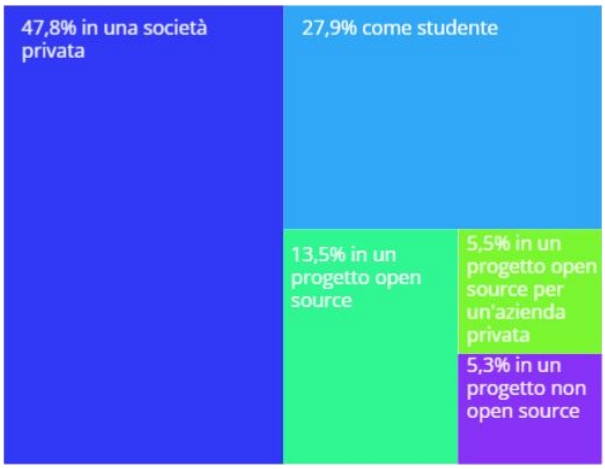

Figura 2.4: Distribuzione dei contributi dati dagli sviluppatori a progetti software, dal sito octoverse.github.com

Il grafico in figura 2.4 mostra che circa il 13.5% degli sviluppatori iscritti a GitHub lavorano su progetti software open source [3].
Inoltre è più probabile che i team con un'elevata fiducia reciproca e una sana cultura collaborativa abbiano una maggiore produttività. Sono molti i fattori che permettono un aumento della produttività, della fiducia e della collaborazione. Tra questi bisogna considerare il numero di repository che hanno o meno il file README, il quale fornisce informazioni generali circa la natura del sistema software in questione.
Alla base della maggior parte dei software open source c'è proprio il README che permette agli sviluppatori di essere il 55% più produttivi.

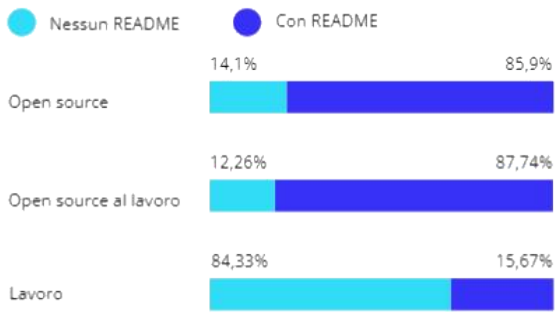

Figura 2.5: File README dal sito octoverse.github.com

In conclusione lo sviluppo open source su GitHub è in continua crescita rappresentando il futuro della programmazione.

2.2 Partecipazione nei Progetti Open Source

2.2.1 Abbandono dei Progetti Open Source

Il codice open source è diventato una fonte primaria per lo sviluppo applicativo, non solo per creare programmi con licenza gratuita, ma per comporre gli stack software di quasi tutti i servizi e i prodotti commerciali.
La creazione, lo sviluppo ed il mantenimento di un progetto software open source dipendono dalla forza e dalla salute della comunità che lo compone. L'evoluzione dei suddetti progetti dipenderà da molti fattori.
Potrebbe ad esempio accadere che un progetto non venga consegnato in tempo, che sfori il budget economico o che non raggiunga i risultati previsti. L'eventuale fallimento di un progetto può essere misurato in termini:

- Motivazionali.
- Economici e temporali.
- Personali ed organizzativi.
- Di distruzione del valore e della reputazione di impresa.

Innanzitutto una comunicazione inefficace è la ragione principale per cui molti progetti non raggiungono gli obiettivi prestabiliti o, in altre parole, falliscono. Le cause alla base di una cattiva comunicazione sono varie: barriere linguistiche, canali di comunicazione inadeguati o aspetti inerenti l'ambiente lavorativo. Bisognerà, quindi, definire un piano efficace che permetta di colmare tutti gli aspetti relativi alla comunicazione.
Un altro aspetto fondamentale è la pianificazione che, in accordo con l'ente PMI(Project Management Institute), definisce cinque aree di processo. Il ciclo inizia con la definizione del progetto, si estende con un livello di pianificazione, continua con l'esecuzione delle attività effettuando costanti monitoraggi e controlli fino alla conclusione con la consegna dell'output finale al cliente. L'analisi delle motivazioni di insuccesso pone al primo posto l'indeterminatezza, ambiguità o genericità degli obiettivi progettuali ovvero, in altre parole, la scarsa qualità della formulazione.
L'utilizzo di software inadeguati e poco flessibili o eventuali modifiche nella portata del progetto potrebbero causarne l'abbandono da parte di uno o più sviluppatori.
Consideriamo, successivamente, il numero di sviluppatori che partecipano ad un determinato progetto, il quale ovviamente risulterà direttamente proporzionale alla grandezza e alla

complessità di quest'ultimo. Se il numero di sviluppatori risulta essere limitato ed inferiore ad una soglia stabilita, allora l'effort impiegato da ogni membro del team sarà superiore a quello dovuto. Questo comporterebbe il verificarsi di eventi dannosi che potrebbero minacciare l'intera continuazione del progetto in questione.

Sussistono molti altri elementi che potrebbero compromettere lo sviluppo del software, [4] tra cui:

- Mancanza di tempo.
- Mancanza di interesse.
- Il progetto si basa su tecnologie obsolete.
- Conflitti tra sviluppatori.

È possibile che nuovi sviluppatori assumano la manutenzione del progetto permettendone la sopravvivenza.
A tal proposito molti studi si concentrano sull' individuazione dei potenziali problemi che portano gli sviluppatori ad abbandonare progetti open source.
Alcune ricerche trattano e discutono il cosiddetto "truck factor" che identifica il grado in cui un progetto dipende da un ristretto gruppo di persone.
Ad esempio Avelino [5] ha calcolato il truck factor per i progetti GitHub e ha scoperto che la sopravvivenza di quasi due terzi di essi dipende da uno o due sviluppatori.
La piattaforma GitHub fornisce grafici che mostrano la sequenza temporale dei contributi forniti da uno sviluppatore ad un progetto.
Osservando le linee temporali in figura2.6 è possibile osservare la frequenza di commit ovvero la modifica e la scrittura di codice pubblicata da uno sviluppatore. Dalla figura2.6, si nota la cronologia di un contribuente molto frequente.

Figura 2.6: Grafico della cronologia di un contributor molto frequente

Attraverso l'ispezione di diversi grafici si è notato che il ritmo dei commit varia. Ciò è

dovuto al fatto che diversi programmatori riducono la frequenza del contributo o addirittura scompaiono per un certo periodo di tempo.
Nell' esempio che viene mostrato in figura2.7, invece, si nota una cronologia di contributi poco frequenti.

Figura 2.7: Grafico della cronologia di contributi poco frequenti

Manutenzione dei Progetti Open Source

Il software dopo il suo rilascio, soprattutto se destinato a lunga vita, necessita di alcune fasi di manutenzione. Per manutenzione si intende sia la correzione di eventuali bug, sia l'estensione o modifica di alcune componenti. Il suo costo risulta essere superiore rispetto a quello della produzione; questo a sottolineare l'importanza assunta dagli sviluppatori in eventuali modifiche e sviluppi futuri. Gli sviluppatori non possono permettersi il lusso di lanciare un prodotto e abbandonarlo a se stesso, devono stare sempre in allerta per correggerlo e migliorarlo al fine di rimanere competitivi e rilevanti.
Esistono tre tipi di manutenzione:

- **correttiva:** che permette la correzione di bug e l'eliminazione di difetti, i quali possono compromettere la funzionalità del prodotto
- **effettiva:** che permette l'adattamento a nuovi ambienti
- **evolutiva:** che permette di effettuare miglioramenti o fornire nuove funzionalità.

2.2.2 Stabilire l'Abbandono di un Progetto Open Source

Per capire se un progetto è stato abbandonato bisogna considerare metriche che indagano sia sul fattore temporale, sia sull'effettivo lavoro svolto da ogni contributor appartenente ad un determinato progetto software.
GitHub fornisce molte metriche che permettono di stabilirne l'effettivo abbandono. Più nello

specifico nella sezione "Insights" ci sono molti grafici che mostrano l'effettivo periodo attivo degli sviluppatori ovvero l'intervallo di tempo in cui viene scritto e prodotto codice sorgente. Infatti analizzando diversi progetti e osservando i corrispondenti grafici si riesce a stabilire quale di questi potrebbe o non essere abbandonato. Nella figura 2.8 si nota che la frequenza di commit è ristretta ad un determinato periodo della durata di circa 1 mese in cui i contribuenti hanno effettuato commit. Supponendo che ogni periodo a cui fa riferimento il grafico in questione ha la durata di una settimana, è possibile notare un'elevata partecipazione che va ulteriormente a crescere nel secondo e diminuendo di poco in volta.

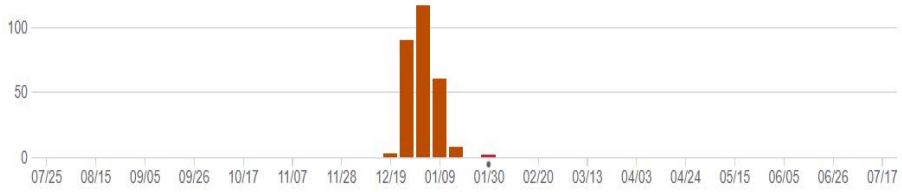

Figura 2.8: Grafico di un progetto ritenuto chiuso dal sito GitHub.com

Questo grafico potrebbe essere quello di un progetto ultimato e chiuso a differenza della figura 2.9. Innanzitutto il periodo ricopre un arco temporale molto elevato di circa un anno. Da notare la frequenza dei commit poco regolare seguita da lunghi periodi di inattività. Questo grafico, a differenza del precedente potrebbe essere quello di un progetto non ultimato,forse destinato a sviluppi futuri o addirittura abbandonato.

Figura 2.9: Grafico di un progetto ritenuto non ultimato, destinato a sviluppi futuri o abbandonato dal sito GitHub.com

Inoltre i progetti su GitHub possono essere marcati come "Closed". Il Project Close Out è la fase che sancisce ufficialmente la conclusione di un progetto.
In secondo momento è possibile considerare le issue, che vengono usate su GitHub per tenere traccia di bug, segnalare che una determinata componente potrebbe essere migliorata o in generale per effettuare altre richieste di progetto.
La presenza di issue aperte da un certo tempo potrebbe essere un campanello d'allarme.
Altri fattori determinanti potrebbero essere:

- Community open source poco attive.
- Mancanza di commit da parte dei contributor, causato ad esempio dall'utilizzo di una nuova tecnologia.
- Utilizzo di una tecnologia obsoleta.

2.2.3 Conseguenze dell'Abbandono

Risulta necessario soffermarsi sulle conseguenze che la mutata realtà operativa del software comporta. L'abbandono, infatti, potrebbe implicare il sorgere di innumerevoli problematiche lesive innanzitutto per gli utenti che ne fanno utilizzo.
In alcuni casi, un programma open source potrebbe utilizzare diverse componenti aggiuntive. Il problema si presenta quando queste ultime smettono di esistere o non vengono aggiornate compromettendo il corretto funzionamento del software.
Di conseguenza i fruitori, anche se inconsapevolmente, sono esposti a rischi relativi alla propria sicurezza. La riservatezza e l'integrità delle informazioni caricate sui suddetti software sono pericolosamente compromesse. Oltre alla mancata tutela dei dati il servizio fornito risulterà inefficiente e anacronistico a causa dei mancati interventi di manutenzione.
In conclusione da come è possibile notare il fenomeno dell'abbandono comporta conseguenze non trascurabili che hanno origine prevalentemente da uno scorretto mantenimento e sviluppo futuro del software.

2.2.4 Soluzioni all'Abbandono

La problematica dell'abbandono non risulta, tuttavia, priva di soluzione.
Ovviare a tale inconveniente significherebbe, innanzitutto, non rendere vano il lavoro degli sviluppatori che ne hanno consentito la nascita. In termini di costi, pianificazione ed energie spese, risulta necessario trovare un espediente affinché la vita dei suddetti software sia più duratura.
Gli sviluppatori prima dell'abbandono, se ritengono che il progetto abbia ancora un valore, possono affidare quest'ultimo ad un nuovo sviluppatore che avrà l'opportunità di portarlo a termine. Affinché quest'idea possa essere realmente realizzata sarebbe necessaria la presenza di un servizio offerto da GitHub per facilitare il passaggio di proprietà del progetto.
Una possibile proposta per la realizzazione del servizio sarebbe quella di scansionare regolarmente le repository considerando le ultime date dei commit; nel caso in cui non si verifichi un commit per un determinato periodo di tempo, il gestore addetto alla manutenzione delle

repository, dopo aver ricevuto un avviso, potrà valutare di inserire il progetto all'interno di una repository speciale designata per progetti abbandonati. Questa, in definitiva, potrebbe rappresentare un'efficace soluzione per quanto meno ridurre la percentuale di progetti abbandonati.

2.3 Quantificazione del Contributo per lo Sviluppo Software Open Source

I progetti software, soprattutto quelli di grandi dimensioni, sono portati a termine da team di lavoro in cui ogni membro ricopre un ruolo fondamentale.
Lavorare in team significa agire e collaborare insieme ad altre persone per uno scopo condiviso e per il perseguimento di obiettivi comuni.
Ad ogni membro del team saranno affidate delle mansioni che dovranno essere eseguite per il raggiungimento di tale obiettivo.
Risulta, in molti casi, essere di fondamentale importanza identificare il contributo effettivo di ogni singolo sviluppatore che partecipa al progetto in questione.
Infatti, come spiegato nelle sezioni precedenti, monitorare l'avanzamento del progetto è cruciale per evitarne il fallimento. Analizzare il contributo che il singolo sviluppatore porta al progetto può permettere di prevedere se e quando questi sarà in procinto di abbandonarlo, e intervenire tempestivamente per evitare ripercussioni sul progetto.
Affinché questo possa essere realizzato bisognerà analizzare le righe di codice modificate ed inserite in modo da riuscire a stabilire se lo sviluppatore sta contribuendo positivamente o meno al corretto sviluppo del progetto. Per quantificare il lavoro svolto bisognerà indagare ed analizzare determinati dati come:

- Numero di commit.
- Numero di linee di codice aggiunte.
- Numero di linee di codice eliminate.
- Numero di issue apert
- Numero di issue risolte.

Questi possono essere considerati fattori determinanti che a volte potrebbero trarre in inganno. Difatti, oltre a quest'analisi quantitativa dovrà essere effettuata anche un'analisi

qualitativa. Bisognerà quindi indagare sull'effettivo codice sorgente prodotto da ognuno degli sviluppatori. Molto importante risulta essere l'identificazione delle essential e non-essential change, ovvero delle modifiche essenziali e non, per identificare gli sviluppatori che hanno contribuito maggiormente alla riuscita del progetto.

2.3.1 Rilevazione delle Non-essential Change

I sistemi di repository come GitHub utilizzati sin dagli anni settanta, permettono di conservare e custodire una significativa quantità di codice sorgente consentendo di tener traccia delle diverse versioni degli artefatti prodotti durante la fase di implementazione dello sviluppo software.
Esistono molte tecniche che permettono di acquisire informazioni dalle repository permettendone l'analisi. A tal proposito i tipici sistemi di controllo memorizzano le modifiche come line-based textual-deltas considerando i file di codice sorgente soggetti a commit.
A differenza dei sistemi di controllo, gli approcci basati sul cambiamento hanno come obiettivo quello di operare su rappresentazioni più significative del cambiamento. Un elemento fondamentale, presente soprattutto nei più moderni approcci basati sul cambiamento, è quello di ignorare cambiamenti ritenuti secondari o marginali, come ad esempio modifiche inerenti la formattazione o l'eventuale presenza di spazi bianchi. Questo è un presupposto generale alla base di questa strategia perché, considerando i cambiamenti non essenziali, hanno meno probabilità di produrre modifiche significative; di conseguenza non saranno considerati come essential change.
Bisogna incorporare informazioni sull'essenzialità del codice permettendo agli approcci basati sul cambiamento di selezionare con molta più precisione e con un livello di dettaglio superiore le singole modifiche di basso livello utilizzate per ottenere rappresentazioni di alto livello.
A tal proposito i ricercatori Kawrykow e Robillard [6] hanno sviluppato un software, denominato DIFFCAT, che permette di rilevare la presenza di codice non essenziale in diversi progetti software. Nello specifico sono stati considerati 7 progetti open source di lunga durata per un analisi totale di circa 24.000 cambiamenti.
Nella prima colonna della tabella 2.1 sono indicati i nomi dei sistemi in questione, nella seconda la data del primo commit effettuato, nel terzo la data dell'ultimo commit effettuato per concludere con l'ultima in cui viene mostrato il numero di change set per ciascun sistema.
Nello specifico è stato utilizzato DIFFCAT per determinare le differenze nei change set. Il software è stato impostato in modo da non considerare differenze basate su spazi bianchi,

System	First	Last	Change Set
Ant	6-Dec-2001	17-Jul-2007	3853
Azureus	12-Nov-2003	14-Jul-2004	3103
Hibernate	4-Dec-2003	19-Aug-2005	3922
JDT-Core	17-Jan-2002	15-Jul-2003	4192
JDT-UI	20-Aug-2001	15-May-2002	3081
Spring	1-Feb-2004	6-Feb-2006	3627
Xerces	17-May-2001	8 Nov-2007-2	2681

Tabella 2.1: Caratteristiche dei sistemi considerati

documentazione o commenti.

System	-	£	+	R	K	L	Non Ess.
Ant	113	35	301	6.8	.8	.4	8.0(22.9%)
Azureus	49	95	108	2.6	.0	.1	2.7(2.8%)
Hibernate	63	35	196	2.9	.0	.2	3.1(8.9%)
JDT-Core	47	16	73	1.8	.6	.2	2.6(16.3%)
JDT-UI	72	23	100	1.3	.0	.1	1.4(6.1%)
Spring	43	27	126	3.8	.6	.2	4.6(17.0%)
Xerces	62	15	196	1.0	.0	.1	1.1(7.3%)

Tabella 2.2: Code Churn in Target Systems (in kLOC)

La tabella 2.2 mostra complessivamente l'abbandono del codice per i sistemi in questione. Nello specifico sono indicate il numero di righe di codice cancellate(-), inserite(+) o modificate per ciascun sistema(£). Inoltre sono presenti anche le differenze indotte da rinominazione(R), aggiornamenti di keyword banali(K) ed infine refactoring di variabili locali(L).
Dalla tabella 2.2 possiamo dedurre che sono stati aggiornati tra il 2,8% e il 22,9% delle righe di codice solo per differenze non essenziali.
La tabella 2.3 mostra il numero totale di aggiornamenti del metodo(Total), il numero di tali aggiornamenti indotti da differenze non essenziali(Non-Essential), il numero di differenze indotte da rinominazione(R), il numero di aggiornamenti di keyword banali(K) ed infine il numero di refactoring di variabili locali(L). È possibile notare, ovviamente, che la somma di R,K ed L corrisponde proprio al numero di differenze non essenziali.
In definitiva, secondo lo studio svolto da Kawrykow [6], tra il 2,6% ed il 15,5% di tutti gli

System	Total	Non Ess.	R	K	L
Ant	17792	2759 (15.5%)	2227	531	110
Azureus	8731	229 (2.6%)	227	1	5
Hibernate	15881	1153 (7.3%)	1136	6	52
JDT-Core	8837	673 (7.6%)	542	133	98
JDT-UI	9681	426 (4.4%)	424	0	13
Spring	11047	1715 (15.5%)	1508	216	74
Xerces	8409	256 (3.0%)	250	6	5

Tabella 2.3: Method Updates in Target Systems

aggiornamenti dipendono da modifiche non essenziali.

2.3.2 Rilevazione del Refactoring

Il refactoring è una tecnica strutturata che permette la modifica della struttura interna di porzioni di codice senza modificarne il comportamento esterno. Questa tecnica viene applicata per migliorare alcune caratteristiche del software come leggibilità, manutenibilità e riusabilità permettendo anche una riduzione della complessità attraverso l'utilizzo dei design pattern. I design pattern sono template di soluzioni a problemi comuni che sono state raffinate nel corso del tempo dagli sviluppatori. La presenza di refactoring è motivata, spesso, dalla rilevazione di "code smell"; quest'espressione è utilizzata per indicare caratteristiche del codice sorgente che determinano la presenza di difetti di programmazione. Non si tratta di veri e propri bug, ma debolezze di progettazione che riducono la qualità del software.
L'azione del refactoring ha come obiettivo quello di eliminare il problema attraverso un insieme di punti che dovranno essere il più semplice possibile.

Molti IDE forniscono funzioni di refactoring tra cui:

- IntelliJ IDEA.
- Eclipse.
- NetBeans.
- Visual Studio e molti altri.

Esistono molti algoritmi che permettono la rilevazione di refactoring, tuttavia, ricerche recenti hanno messo in dubbio l'accuratezza di questi algoritmi dato che potrebbero rappresentare

una minaccia in termini di affidabilità. A tal proposito, uno studio indipendente [7] ha dimostrato che REF-FINDER, uno degli algoritmi di rilevamento di refactoring più utilizzati, ha una precisione del 35% per un richiamo complessivo del 24%. Studi ancor più recenti hanno dimostrato che tale strumento ha una precisione media complessiva del 27% [8] [9].
Il rilevamento di refactoring errati risulta essere grave perché renderebbe le conclusioni di uno studio empirico errate. Inoltre nel caso in cui ci siano software dipendenti da esso verrebbero applicate operazioni sbagliate con il conseguimento di output non esatto.
Come soluzione ad un refactoring errato, un lavoro recente di Tsantalis [10] ha proposto un nuovo strumento di rilevazione del refactoring dal nome RefactoringMiner. Questo algoritmo prende in input due revisions di un progetto Java ovvero un commit ed il suo parent dalla cronologia dei commit di una determinata repository producendo come output un elenco di operazioni di refactoring, applicate tra le due revisions in questione.
Alla base di RefactoringMiner è presente un algoritmo basato su AST.
Quest'algoritmo permette di ottenere una rappresentazione ad albero della struttura sintattica del codice sorgente che risulta essere astratta nel senso che non viene rappresentato ogni dettaglio presente nella sintassi reale, ma solo quelli strutturali o relativi al contenuto.
RefactoringMiner supporta il rilevamento di 15 tipi diversi di refactoring per 4 differenti tipi di codice ovvero: package, type, method e field.
In conclusione questo strumento ha una precisione molto alta che, secondo lo studio condotto, [10] risulta essere superiore al 95%.
Allo stesso tempo sono presenti anche alcune limitazioni tra cui:

- **Contesto mancante:** RefactoringMiner analizza solo i file aggiunti, eliminati o modificati tra due revisions.
- **Refactoring nidificati:** RefactoringMiner non è in grado di rilevare operazioni di refactoring nidificati.
- **Refactoring non supportati:** RefactoringMiner non è in grado di rilevare tutti i tipi di refactoring. Nello specifico sono 88 i tipi di refactoring supportati considerando la versione 2.3 del software.

CAPITOLO 3

Repository Analyzer

In questo capitolo è illustrata la creazione e l'esecuzione del Plug-in denominato "Repository Analyzer".
Inizialmente vengono analizzati gli obiettivi principali, successivamente viene trattata la tecnica adottata per il recupero dei dati di interesse dalla piattaforma GitHub.
Viene, inoltre, esaminato il funzionamento e la struttura di Repository Analyzer discutendo in dettaglio delle metriche che permettono di quantificare il contributo apportato dagli sviluppatori di un Team che partecipano ad un determinato progetto software open source.
Infine è analizzata l'interfaccia del suddetto Plug-in per comprenderne al meglio il corretto funzionamento.

3.1 Repository Analyzer

3.1.1 Obiettivi

Nel capitolo precedente abbiamo discusso della grande importanza dell'open source approfondendo anche le motivazioni che portano gli sviluppatori, in alcuni casi, ad abbandonare un determinato progetto.
A tal proposito l'obiettivo del presente elaborato finale è la realizzazione di un Plug-in denominato RepositoryAnalyzer che consenta di stabilire l'effettivo lavoro svolto da ogni contributor che partecipa ad un progetto software open source. La principale finalità da perseguire mediante l'utilizzo del Plug-in è l'individuazione della presenza o meno di elementi con i quali si potrà stabilire se un progetto sta per essere abbandonato.

È divenuto necessario cimentarsi nella creazione di uno strumento che consentisse una reale ed effettiva indagine sullo status dell'attività di sviluppo software perché il monitoraggio degli sviluppatori, siano essi produttivi o improduttivi, e la gestione delle attività di sviluppo di un particolare progetto, sono decisamente gli elementi fondamentali dell'intero iter.
La produttività degli sviluppatori risulta essere essenziale per il successo delle organizzazioni che si occupano di sviluppo dei software. Tuttavia alcuni studi recenti [11] [12] hanno misurato la produttività degli sviluppatori basandosi sulle informazioni presenti in repository riuscendo ad individuare una serie di metriche maggiormente specifiche.
Vengono analizzati nello specifico soltanto gli ultimi 30 commit effettuati dagli sviluppatori poiché mediante l'utilizzo delle Check Runs API è possibile accedere a questa quantità di dati. In tale contesto è stato deciso di effettuare non solo un'analisi quantitativa, ma anche qualitativa.
Ai fini della progettazione e per le finalità del sistema verranno introdotte assunzioni e semplificazioni, anche al fine di poter concentrare gli sforzi concettuali ed operativi sulle principali funzionalità del sistema, prescindendo da servizi supplementari che potranno essere eventualmente introdotti in un momento successivo.

3.1.2 Raccolta Dati

La nascita delle API risale agli inizi dell'era dell'informazione collocabile prima della nascita dei moderni personal computer. Agli albori le API erano utilizzate come librerie per i sistemi operativi, soltanto nei primi anni 2000 sono state riconosciute come un'importante tecnologia per l'integrazione remota dei dati.
Le API, acronimo di Application Programming Interface, sono un insieme di definizioni e protocolli che permettono ad applicazioni di interfacciarsi con altre applicazioni per agevolare e facilitare la creazione, programmazione ed integrazione di software applicativi.
L'obiettivo è, dunque, quello di semplificare la possibilità di integrazione tra un'applicazione ed un'altra in modo da evitare ridondanze o duplicazioni di codice. Lo sviluppo delle applicazioni risulta così molto più agevole con un conseguente risparmio di tempo e denaro. Promuovono, inoltre, la collaborazione tra azienda e team perché permettono di integrare nuovi componenti applicativi nell'architettura esistente.
Grazie alle API è possibile collegare con facilità l'infrastruttura mediante lo sviluppo di app cloud native, nonché condividere i dati con i clienti e con altri utenti esterni.

In base al criterio di rilascio, possono essere classificate in:

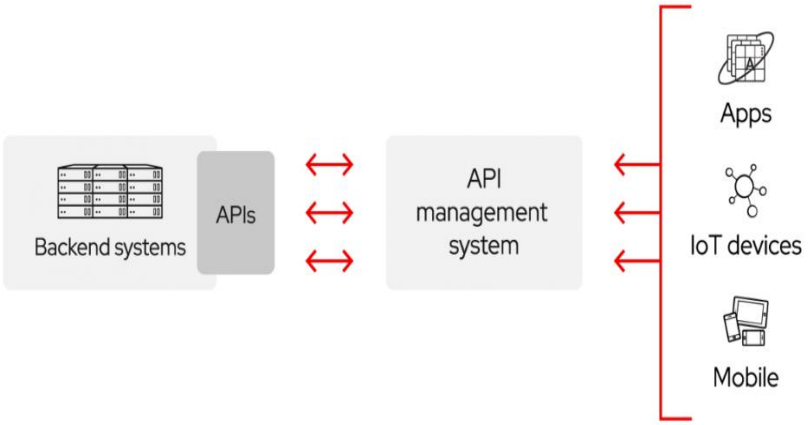

Figura 3.1: Funzionamento delle API dal sito redhat.com

- **API private:** destinate ad un utilizzo ristretto o chiuso. Questo approccio offre alle aziende un controllo totale ed ottimale delle API.
- **API pubbliche:** destinate ad un utilizzo aperto quindi disponibili a chiunque. Questo approccio consente a terze parti di sviluppare app che interagiscono con le API in questione rappresentando quindi una fonte di innovazione.
- **API partner:** destinate all'utilizzo ristretto di partner aziendali.

Le API destinate all'utilizzo con i partner e le API pubbliche possono essere utilizzate da un ampio bacino di utenza ed infatti possono beneficiare dello sforzo creativo di intere comunità introducendo nuove idee ed opinioni a differenza delle API private utilizzate da un ristretto numero di beneficianti.
Per il raggiungimento del nostro obiettivo abbiamo usufruito della Check Runs API che consente la creazione di applicazioni. Quest'ultime eseguono potenti controlli sulle modifiche apportate al codice sorgente della repository considerata.
Inoltre se si è iscritti alla versione Beta di GitHub Advanced Security code scanning è possibile utilizzare API custom con la possibilità di poter usufruire di nuove funzionalità offerte come ad esempio:

- ottenere un'analisi di scansione del codice recente per una repository;
- aggiornare lo stato di avviso di scansione del codice;
- caricare un SARIF file per creare avvisi dell'applicazione GitHub o del workflow GitHub actions;
- ricevere eventi webhook per avvisi di scansione del codice.

3.2 Design

Le tecnologie utilizzate per lo sviluppo del Plug-in sono state JavaScript, HTML e CSS.
JavaScript è un linguaggio di programmazione che consente di implementare funzionalità su pagine web in modo dinamico.
HTML è il linguaggio di markup che utilizziamo per strutturare e dare significato ai nostri contenuti web.
CSS è un linguaggio di regole di stile che utilizziamo per applicare uno stile al nostro contenuto HTML.
La logica generale alla base di RepositoryAnalyzer consente di effettuare richieste HTTP usufruendo delle API messe a disposizione dalla piattaforma GitHub producendo come risposta un file JSON contenente tutte le informazioni necessarie.

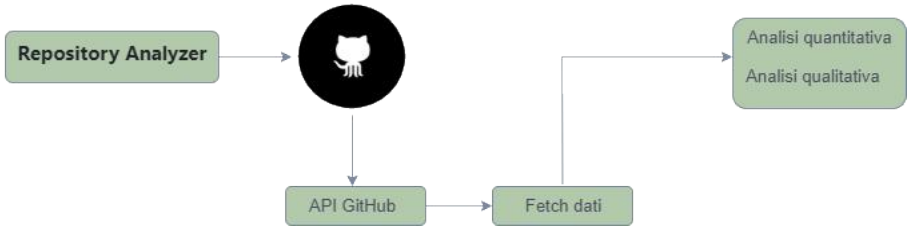

Figura 3.2: Panoramica dell'architettura

La figura 3.2 rappresenta l'architettura del nostro Plug-in.
Quest'ultima è costituita da tre moduli fondamentali che sono: API GitHub, fetch data, quantity and quality analyzer.

API GitHub

Tramite le **API GitHub** è possibile recuperare tutte le informazioni messe a disposizione dalla piattaforma GitHub di una determinata repository. Nello specifico, per comprendere maggiormente il ruolo ed il funzionamento delle API, consideriamo come esempio la repository "DigitalDonation", raggiungibile al seguente link:
https://github.com/CFerrara98/DigitalDonation, creata nell'anno 2021/2022.
L'applicativo è stato realizzato dal team C09 del corso di Ingegneria del Software 2021/22 Resto 1 del dipartimento di Informatica dell'Università degli Studi di Salerno.
DigitalDonation è un'applicazione web open source, progettata nell'ambito di una collaborazione simulata tra il team C09 e l'ente fittizio ENRS (Ente Nazionale Ricerca Sangue), per la gestione del flusso di organizzazione delle sedute di donazione sanguigne e delle

prenotazioni dei Volontari Donatori tesserati presso l'ente.
Per poter accedere a tutte le informazioni della repository in questione, messe a disposizione da GitHub, bisognerà consultare il seguente URL:
https://api.github.com/repos/CFerrara98/DigitalDonation.

Name	**Link or Description**
description	Applicazione web, progettata nell'ambito di una collaborazione simulata tra il team C09 e l'ente fittizio ENRS (Ente Nazionale Ricerca Sangue) per la gestione del flusso di organizzazione delle sedute di donazione sanguigne e delle prenotazioni dei Volontari Donatori tesserati presso l'ente
languages_url	https://api.github.com/repos/CFerrara98/DigitalDonation/languages
contributors_url	https://api.github.com/repos/CFerrara98/DigitalDonation/contributors
commits_url	https://api.github.com/repos/CFerrara98/DigitalDonation/commits/sha
created_at	2021-12-21T07:07:12Z
updated_at	2022-01-17T12:09:22Z
branches_url	https://api.github.com/repos/CFerrara98/DigitalDonation/branches

Tabella 3.1: Informazioni della repository a cui è possibile accedere tramite API GitHub

A tal proposito nella tabella 3.1 sono stati riportati solo alcuni tra i tanti dati cui è possibile accedere. Nello specifico, considerando singolarmente le informazioni della suddetta tabella, abbiamo:

- **description:** descrizione del progetto.
- **languages_url:** link alla risorsa messa a disposizione dalle API GitHub. Accedendo a questa risorsa possiamo conoscere i linguaggi di programmazione e formattazione utilizzati. In aggiunta sono presenti anche le linee di codice totali.
- **contributors_url:** sono presenti informazioni inerenti ogni membro del team come immagine, username e numero di commit effettuati.
- **commits_url:** sono presenti informazioni inerenti gli ultimi 30 commit.
  Accedendo a commits_url sono presenti anche gli sha i quali sono rappresentati da un codice avente la seguente forma: "08ec38ff23b39ad832239cd23978e6671db44d9b". Infatti, copiando ed incollando questo contenuto in coda all'URL in questione è possibile accedere al commit singolo per avere ancor più informazioni circa un determinato

commit. Infatti, nella tabella 3.2 possono essere visualizzati i dati aggiuntivi a cui è possibile accedere aggiungendo in coda all'URL il codice sha da analizzare.

Name	**Description**
commit.author.name	Nome dello sviluppatore.
commit.author.email	Email dello sviluppatore.
commit.author.date	Data e ora in cui lo sviluppatore ha effettuato il commit.
commit.message	Descrizione del messaggio che lo sviluppatore ha inserito all'atto del commit.
stats.total	Numero delle linee di codice totali.
stats.additions	Numero delle linee di codice aggiunte.
stats.deletions	Numero di linee di codice eliminate.
files.filename	Nome del file modificato.
files.patch	Linee di codice sorgente inserite modificate ed eliminate.

Tabella 3.2: Informazioni aggiuntive circa un determinato commit identificato tramite codice sha.

- **created_at e updated_at:** rappresentano rispettivamente la data di creazione e la data di chiusura del progetto.
- **branches_url:** sono presenti informazioni che indicano tutti i branch creati a partire dal branch principale contrassegnato come "main".

Name	**Link or Description**
login	nameAdmin
avatar_url	imageAdmin
repos_url	https://api.github.com/users/CFerrara98/repos

Tabella 3.3: Informazioni sul creatore della repository a cui è possibile accedere tramite API GitHub

Oltre ai dati trattati in precedenza nella tabella 3.3, sono riportati anche informazioni riguardanti l'utente creatore della repository ottenibili tramite API GitHub all'URL https://api.github.com/repos/CFerrara98/DigitalDonation come:

- **login:** che indica il nome.
- **avatar_url:** che indica l'immagine.
- **repos_url:** che indica il link alla repository.

Fetch Data

Un altro modulo fondamentale di Repository Analyzer riguarda il fetch dei dati, ottenibili dalla piattaforma GitHub tramite una serie di richieste HTTP. L'Hypertext Transfer Protocol(HTTP) è il protocollo principale del World Wide Web. È progettato per fornire comunicazione tra un browser ed un server, infatti, viene utilizzato per inviare informazioni in un formato comprensibile sia al client che al server.
Il protocollo HTTP si basa su diversi metodi di richiesta tra cui GET e POST.
Il metodo GET è utilizzato per richiedere dati da una risorsa specifica. Alcune note sono:

- Le richieste GET possono essere memorizzate nella cache.
- Le richieste GET rimangono nella cronologia del browser.
- Le richieste GET possono essere aggiunte ai preferiti.
- Le richieste GET non dovrebbero mai essere utilizzate quando si tratta di dati sensibili.
- Le richieste GET hanno limitazioni di lunghezza.
- Le richieste GET vengono utilizzate solo per richiedere dati.

Il metodo POST viene utilizzato per inviare dati ad un server per creare o aggiornare una risorsa. Alcune note sono:

- Le richieste POST non vengono mai memorizzate nella cache.
- Le richieste POST non rimangono nella cronologia del browser.
- Le richieste POST non hanno restrizioni sulla lunghezza dei dati.

A tal proposito per la realizzazione del Plug-in abbiamo usufruito del metodo GET in modo da poter richiedere dati alla risorsa specifica.
In particolar modo effettuiamo una richiesta HTTP che produrrà come risposta un JSON contenente tutte le informazioni della richiesta necessarie. JSON, acronimo di JavaScript Object Notation, è un formato utilizzato per lo scambio dati tra applicazioni in un'architettura client-server. Il formato JSON è molto utilizzato soprattutto per accedere alle risposte di API remote, come nel nostro caso.
Questo è stato l'approccio utilizzato per le varie operazioni di fetch effettuate sulla repository.

Quantity and Quality Analyzer

Quantity and quality analyzer utilizza le informazioni recuperate da GitHub per ottenere una serie di metriche che quantificano l'attività di ciascuno sviluppatore in relazione ad un determinato progetto.
Analizzando gli ultimi 30 commit siamo in grado di stabilire l'effettivo contributo versato dagli sviluppatori.

3.2.1 Calcolo delle Metriche

La produttività corrisponde al rapporto tra ciò che viene prodotto, ovvero l'output, e le risorse applicate per la produzione, ovvero l'input [13]. Tale concetto può essere utilizzato in molti settori, dall'agricoltura all'economia.
Tuttavia, considerando il settore dell'ingegneria del software, un problema molto complesso consiste nella quantificazione dei risultati prodotti in un progetto software in termini di dimensioni ed effettivo contributo apportato dagli sviluppatori. A tal proposito per la realizzazione di RepositoryAnalyzer abbiamo deciso di considerare sia metriche quantitative che metriche qualitative.
Le **metriche quantitative** indagano sulla quantità di codice inserito da ogni singolo sviluppatore.
Le **metriche qualitative** indagano sulla qualità del codice inserito da ogni singolo sviluppatore riuscendo a stabilire se quest'ultimo ha introdotto documentazione, codice sorgente o whitespace.
Risulta necessario porre in evidenza il fatto che valutare la produttività degli sviluppatori mediante le sole metriche basate su commit è complesso poiché l'analisi del singolo fattore quantitativo non consente di ottenere una panoramica effettiva dell'attività svolta.
Di conseguenza un passaggio fondamentale è quello di analizzare le proprietà del codice al fine di valutarne anche la qualità.

Nella tabella 3.4 vengono mostrate tutte le metriche che permettono di migliorare la comprensione della produttività tramite misurazione.
Consideriamo le seguenti metriche: code lines percentage, documentation percentage e white lines percentage.
Queste permettono di comprendere, considerando gli ultimi 30 commit, se uno sviluppatore ha inserito linee di codice che corrispondono a documentazione, whiteline o codice sorgente.
Per la realizzazione di queste metriche abbiamo elaborato un parser che fornisce l'accesso ai

Name	Description
Total code lines	Questa metrica indica le righe di codice totali che corrispondono all'insieme delle righe di codice aggiunte e rimosse negli ultimi 30 commit.
Removed code lines	Questa metrica indica le righe di codice rimosse in base agli ultimi 30 commit.
Average file changed for commit	Questa metrica indica la media dei file modificati per un singolo commit effettuato da uno sviluppatore.
Number of closed issue	Questa metrica indica il numero di issue chiuse in base all'intero progetto.
Issues fixed percentage	Questa metrica indica la percentuale di issue chiuse e risolte in base all'intero progetto
Code lines percentage	Questa metrica indica la percentuale delle righe di codice sorgente aggiunte in base agli ultimi 30 commit.
Documentation percentage	Questa metrica indica la percentuale delle righe di codice che corrispondono a documentazione in base agli ultimi 30 commit.
White lines percentage	Questa metrica indica la percentuale delle righe di codice che corrispondono a spazi bianchi in base agli ultimi 30 commit.

Tabella 3.4: Metriche del plug-in RepositoryAnalyzer

dati di sola lettura, incrementando differenti variabili per contraddistinguere la presenza di spazi bianchi, documentazione o codice sorgente.
Nello specifico la presenza di spazi bianchi è contrassegnata dalla stringa blank indicata con " " mentre la presenza di documentazione dalla presenza di keyword indicate con "//", "/* */" o "<!– –>". In altre parole il parser effettua un'analisi del codice scomponendolo in "piccoli frammenti" ed incrementando la variabile, precedentemente nominata, solo nel caso in cui viene trovata una corrispondenza.

3.2.2 Deployment

RepositoryAnalyzer è un Plug-in open source per Google Chrome che permette di analizzare una qualsiasi repository GitHub pubblica.
La prima caratteristica fondamentale di Repository Analyzer è quella di non avere un Dataset prestabilito e fisso, infatti è in grado di processare ed analizzare una qualsiasi repository

GitHub pubblica.
La seconda, invece, è quella di essere un Plug-in open source dal momento che qualsiasi sviluppatore potrebbe accedere liberamente al codice sorgente scaricandolo, modificandolo ed eseguendolo.
È stato scelto Google Chrome come Browser per vari motivi tra cui:

- **Semplicità:** in quanto Chrome fornisce un layout grafico ottimale per la navigazione e la ricerca sul web.
- **Velocità:** in quanto Chrome è progettato per stare al passo per qualsiasi esigenza, infatti, si avvia velocemente dal desktop, carica le pagine in pochi secondi ed è agevole nell'esecuzione di applicazioni web.
- **Sicurezza:** in quanto Chrome è progettato per garantire massima sicurezza con protezione da malware e phishing integrata.
- **Personalizzazione:** in quanto esistono moltissime app ed estensioni applicabili a Chrome. Un ultimo punto non meno importante risulta essere la grande facilità e linearità con cui Chrome permette di gestire le estensioni.

Inoltre, essendo un Plug-in open source, è possibile accedere al codice sorgente consultando il seguente link : https://github.com/Zengr098/RepositoryAnalyzer.

Figura 3.3: Interfaccia principale Plug-in

Come da figura 3.3, RepositoryAnalyzer possiede un'interfaccia semplice ed intuitiva consentendo a tutti di comprendere al meglio caratteristiche e funzionalità.
Una volta individuata la repository GitHub pubblica da analizzare basta inserirne l'URL nell'apposito text input e cliccare sul bottone "Submit".
Essendo stata utilizzata per RepositoryAnalyzer la piattaforma GitHub come servizio di hosting mostreremo l'esecuzione relativa a questo caso specifico. Effettuati questi semplici passi saremo redirezionati ad una pagina web ed il restante lavoro sarà processato da RepositoryAnalyzer.

Figura 3.4: Plug-in RepositoryAnalyzer

Nella figura 3.4 sono presenti informazioni generali come descrizione, data di creazione e data di ultimo commit della repository caricata.
La struttura generale della pagina web, come mostrato dalla figura 3.5, è la seguente:

- **Contributors:** in cui sono presenti tutti gli sviluppatori che partecipano al progetto con relative metriche.
- **Metrics:** in cui vengono descritte in dettaglio tutte le metriche utilizzate.
- **About:** in cui sono presenti informazioni generali circa i partecipanti del progetto RepositoryAnalyzer.

L'effettivo funzionamento del Plug-in consente un'acquisizione rapida ed intuitiva di determinati dati. Per identificare singolarmente ogni sviluppatore viene mostrata in foreground nell'interfaccia grafica di RepositoryAnalyzer l'immagine ed il nome dello sviluppatore in questione. Le figure 3.5, 3.6 e 3.7 mostrano rispettivamente l'interfaccia grafica di contributors, metrics e about.
Inoltre è stato deciso di pacchettizzare RepositoryAnalyzer. Accedendo a Chrome Web Store e inserendo RepositoryAnalyzer come keyword nella barra di ricerca è possibile installarlo permettendo a tutti gli utenti interessati di utilizzarlo in pochi e semplici passi.

Figura 3.5: Contributors section in RepositoryAnalyzer

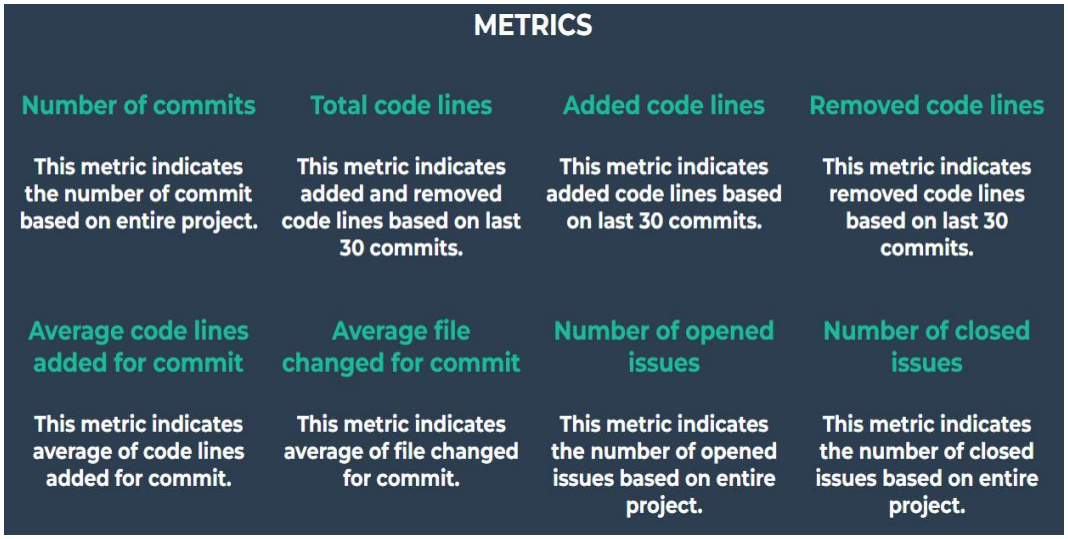

Figura 3.6: Metrics section in RepositoryAnalyzer

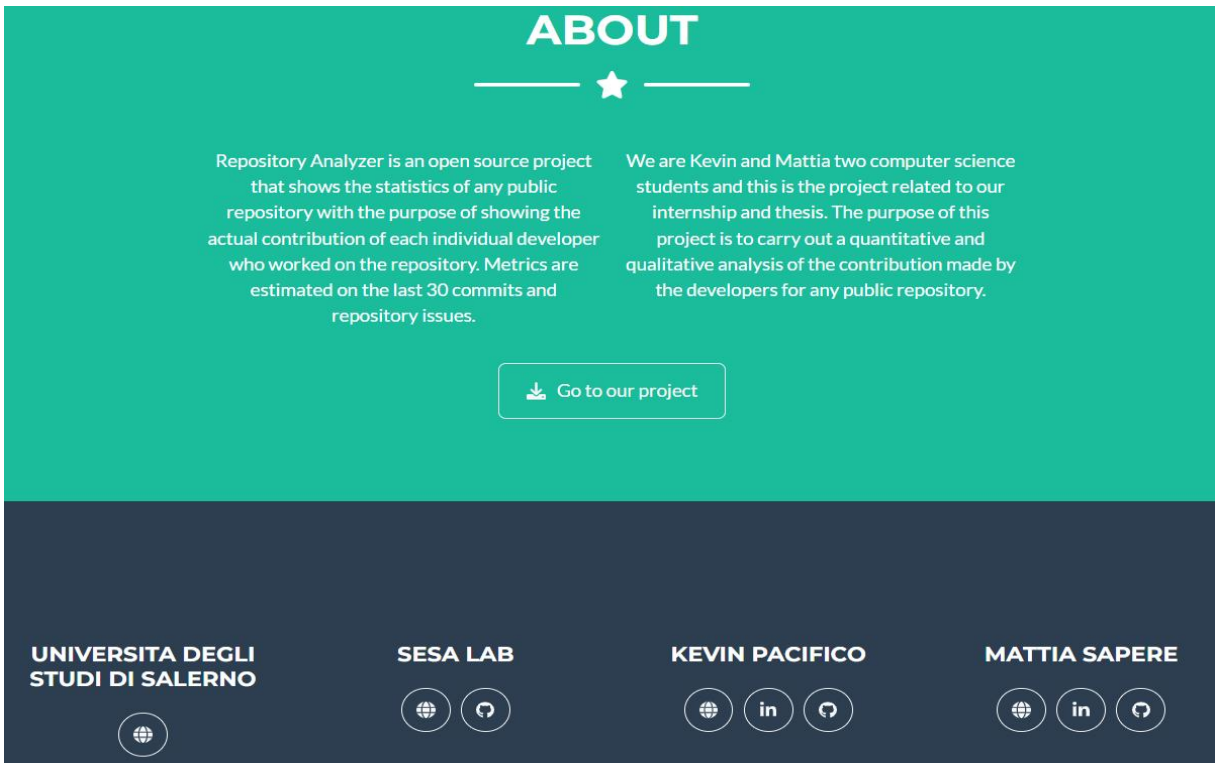

Figura 3.7: About section in RepositoryAnalyzer

CAPITOLO 4

Validazione

In questo capitolo è illustrata la validazione di RepositoryAnalyzer su differenti progetti usufruendo di un oracolo affidabile e completo.

4.1 Validazione

In una community di sviluppo che si sta legando sempre di più al riutilizzo di componenti, le repository online rappresentano un luogo prezioso in cui custodire dati. Ciò permette, nel caso dei progetti open source, di accedere ad una grandissima quantità di informazioni che consentono di quantificare il processo di sviluppo del software.
Per valutare empiricamente RepositoryAnalyzer è stato considerato un oracolo affidabile e completo evitando di insorgere in problemi di:

- Oracolo incompleto.
- Oracolo parziale.
- Oracolo artificiale.

L'oracolo rappresenta l'output atteso, ovvero il risultato che il programma dovrebbe fornire. Esistono differenti tipologie di oracolo:

- **Oracolo euristico:** che fornisce risultati esatti e approssimati per un determinato insieme di dati in input.

- **Oracolo statistico:** che utilizza caratteristiche statistiche.
- **Oracolo consistente:** che confronta i risultati dell'esecuzione di un test con quelli di un altro simile.
- **Oracolo basato sui modelli:** che utilizza modelli simili per generare e verificare il comportamento del sistema.
- **Oracolo umano:** che utilizza il giudizio di un essere umano.

A tal proposito RepositoryAnalyzer è stato testato su diversi progetti software open source. Da premettere, come già accennato nei capitoli precedenti, è che RepositoryAnalyzer quantifica e qualifica le informazioni inerenti gli ultimi 30 commit di un progetto open source.
Come primo è stato deciso deciso di testare RepositoryAnalyzer su se stesso. Nello specifico, per codesto test iniziale, sono stati considerati due differenti tipi di oracolo:

- **Oracolo umano:** effettuando un'analisi personale e manuale dell'intero progetto. Trattandosi di un progetto open source di medie dimensioni è stato possibile analizzare le singole linee di codice inserite ottenendo risultati conformi con quelli aspettati.
- **Oracolo consistente:** confrontando i risultati dell'esecuzione di RepositoryAnalyzer con quelli di un altro sistema simile, ovvero gli insights GitHub. In questo caso i risultati prodotti dai due software sono differenti.

Per tale motivo è stato deciso di approfondire tale tematica al fine di comprendere il motivo di tale divergenza.
Il primo fattore è che RepositoryAnalyzer ha come input l'analisi degli ultimi 30 commit a differenza di GitHub che effettua un'analisi sull'intero numero di commit effettuati.
Il secondo fattore è che GitHub considera il contributo versato dagli sviluppatori inerente unicamente il branch main, escludendo merge commit e bot account a differenza di RepositoryAnalyzer che li include. Dunque tale divergenza è dovuta unicamente da questi due fattori.
Terminata questa prima fase di testing è stato concordato di considerare un progetto open source costituito da:

- Un singolo branch, che corrisponde al branch main.
- Un numero di commit inferiore di 30.
- Totale assenza di merge commit.
- Totale assenza di bot account.

In questo modo, stando alle osservazioni precedenti e considerando un oracolo consistente, l'analisi dei due software, GitHub e Repositorynalyzer, dovrebbe coincidere.

Figura 4.1: Metriche offerte dalla piattaforma GitHub.

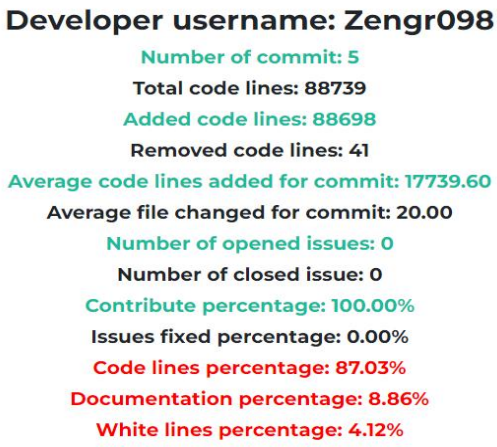

Figura 4.2: Metriche offerte dal Plug-in RepositoryAnalyzer.

Nelle figure 4.1 e 4.2 è possibile osservare che le metriche offerte dai due software coincidono. Ciò permette di concepire l'elevata precisione delle metriche offerte dal Plug-in RepositoryAnalyzer. Quest'ultimo è stato testato su differenti progetti Java, JavaScript producendo risultati in linea rispetto a quelli aspettati.
Inoltre è in grado di analizzare differenti tecnologie e linguaggi di programmazione/formattazione come Java/JavaScript con relativi Framework, C, C++, C#, HTML, CSS.
Sistemi sviluppati in altri linguaggi di programmazione potrebbero non mostrare proporzioni sicure e precise.

CAPITOLO 5

Conclusioni e Sviluppi Futuri

5.1 Conclusioni

Tramite le ricerche effettuate e con il conseguente lavoro svolto è stato raggiunto l'obiettivo principale di quantificare e qualificare il contributo degli sviluppatori che partecipano ad un progetto software open source riuscendo a produrre metriche tramite le quali si potrebbe riuscire a stabilire se un progetto sta per essere abbandonato o meno.
Quanto ultimato durante tutto questo studio può essere riepilogato nei seguenti punti, i quali rappresentano i temi cruciali analizzati:

- Stabilire l'importanza dell'open source e la sua evoluzione nel corso del tempo fornendo esempi dettagliati e descrivendone punti di forza e debolezza.
- Individuare le cause, le conseguenze e le soluzioni dell'abbandono da parte degli sviluppatori ad un progetto software.
- Esaminare le metriche che permettono di stabilire l'abbandono.
- Realizzare il Plug-in denominato RepositoryAnalyzer con riferimenti agli obiettivi da raggiungere e alle strategie attuate per lo sviluppo.
- Validazione di RepositoryAnalyzer su differenti dataset al fine di stabilirne la corretta esecuzione.

I dati sui contributi presenti su GitHub possono offrire informazioni utili circa il processo di sviluppo del software e produrre risultati che potrebbero essere utilizzati per migliorarlo.
Valutare la produttività degli sviluppatori risulta essere essenziale per il successo e la riuscita di un determinato progetto software.
Una conclusione che è possibile effettuare riguardo al lavoro svolto è che un Plug-in potrebbe quantificare e qualificare informazioni senza particolari complessità: "La semplicità è il risultato finale di un lungo e duro lavoro, non il punto di partenza"[14].
La citazione a Frederic William Maitland sopra esposta, si ispira al noto modello di pensiero "Rasoio di Occam" il quale viene anche definito "principio della parsimonia" e suggerisce che, a parità di tutte le altre condizioni, sia sempre da preferire la spiegazione più semplice di un fenomeno o la soluzione più immediata di un problema.
Le soluzioni più semplici hanno anche il vantaggio di poter essere testate più facilmente ed essendo basate su poche assunzioni possono essere agevolmente generalizzate cioè applicate per interpretare un più ampio ventaglio di fenomeni.[14]

5.2 Sviluppi futuri

Tra i possibili sviluppi futuri, oltre al poter introdurre affinamenti per il Plug-in, potrebbero essere sviluppate ulteriori metriche di quantificazione del contributo indagando sull'intera attività di sviluppo di un determinato progetto software open source, considerando il totale dei commit effettuati dagli sviluppatori. Così facendo si potrebbe avere una visione generale e completa dell'intero processo di sviluppo di ogni singolo sviluppatore.
Inoltre, essendo RepositoryAnalyzer un Plug-in nato unicamente per Google Chrome, si potrebbe estendere il funzionamento ad altri browser web.

Ringraziamenti

Ringrazio:

- La mia relatrice, Professoressa Filomena Ferrucci, per la professionalità con la quale mi ha seguito per l'elaborazione della tesi e lo svolgimento dell'attività di tirocinio.
- La mia seconda relatrice, Dottoressa Giulia Sellitto, per la disponibilità e pazienza con la quale mi ha seguito per l'elaborazione della tesi.
- La mia famiglia per essermi sempre stata vicina, aver sempre creduto in me ed aver fatto di tutto affinché io coltivassi le mie passioni.
- La mia ragazza per la pazienza e l'amore con cui ha saputo starmi accanto in questo percorso.
- Tutti i miei amici e le persone a me vicine, che mi hanno sempre sostenuto, rincuorato ed aiutato nei periodi bui.

Bibliografia

[1] T. F. Bissyandé, D. Lo, L. Jiang, L. Réveillere, J. Klein, and Y. Le Traon, "Got issues? who cares about it? a large scale investigation of issue trackers from github," in *2013 IEEE 24th international symposium on software reliability engineering (ISSRE)*. IEEE, 2013, pp. 188–197. (Citato a pagina 2)

[2] A. Fuggetta, "Open source software—-an evaluation," *Journal of Systems and software*, vol. 66, no. 1, pp. 77–90, 2003. (Citato a pagina 5)

[3] Github. (2021) Github octoverse report. [Online]. Available: https://octoverse.github.com/ (Citato a pagina 9)

[4] J. Coelho and M. T. Valente, "Why modern open source projects fail," in *Proceedings of the 2017 11th Joint meeting on foundations of software engineering*, 2017, pp. 186–196. (Citato a pagina 12)

[5] G. Iaffaldano, I. Steinmacher, F. Calefato, M. Gerosa, and F. Lanubile, "Why do developers take breaks from contributing to oss projects? a preliminary analysis," *arXiv preprint arXiv:1903.09528*, 2019. (Citato a pagina 12)

[6] D. Kawrykow and M. P. Robillard, "Non-essential changes in version histories," in *2011 33rd International Conference on Software Engineering (ICSE)*. IEEE, 2011, pp. 351–360. (Citato alle pagine 17 e 18)

[7] G. Soares, R. Gheyi, E. Murphy-Hill, and B. Johnson, "Comparing approaches to analyze refactoring activity on software repositories," *Journal of Systems and Software*, vol. 86, no. 4, pp. 1006–1022, 2013. (Citato a pagina 20)

[8] P. Hegedűs, I. Kádár, R. Ferenc, and T. Gyimóthy, "Empirical evaluation of software maintainability based on a manually validated refactoring dataset," *Information and Software Technology*, vol. 95, pp. 313–327, 2018. (Citato a pagina 20)

[9] I. Kádár, P. Hegedus, R. Ferenc, and T. Gyimóthy, "A code refactoring dataset and its assessment regarding software maintainability," in *2016 IEEE 23rd International conference on software analysis, Evolution, and Reengineering (SANER)*, vol. 1. IEEE, 2016, pp. 599–603. (Citato a pagina 20)

[10] N. Tsantalis, M. Mansouri, L. Eshkevari, D. Mazinanian, and D. Dig, "Accurate and efficient refactoring detection in commit history," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 483–494. (Citato a pagina 20)

[11] F. Rahman and P. Devanbu, "How, and why, process metrics are better," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 432–441. (Citato a pagina 22)

[12] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, no. 3, pp. 309–346, 2002. (Citato a pagina 22)

[13] E. Oliveira, E. Fernandes, I. Steinmacher, M. Cristo, T. Conte, and A. Garcia, "Code and commit metrics of developer productivity: a study on team leaders perceptions," *Empirical Software Engineering*, vol. 25, no. 4, pp. 2519–2549, 2020. (Citato a pagina 28)

[14] M. F. William. (2020) Il rasoio di occam. [Online]. Available: https://thinkinpark.it/2020/11/02/il-rasoio-di-occam-e-il-principio-della-parsimonia/ (Citato a pagina 38)