REPORT – PACI A.

Nota Preliminare

Il seguente report è una **bozza provvisoria** e include il report sulla **Delivarable 1** e parzialmente il report sulla **Delivarable 2**. Il seguente report verrà diviso in due sezioni differenti, una relativa alla prima Deliverable e una relativa alla seconda.

Entrambi i Deliverable sono hostati sulla piattaforma di **GitHub** e sono connessi a **Travis CI** per il build automatico mediante **Ant** e **SonarCloud** per l'analisi del codice.

DELIVERABLE 1

Descrizione Iniziale

Il primo Delivarable ha come scopo quello di **misurare la stabilità** di un **attributo** di **progetto**, in particolare il numero di **bug fixati per ogni mese** dalla data di creazione del progetto fino ad oggi. I progetti presi in analisi sono stati **PARQUET** e **PDFBOX**. La scelta di usare due progetti proviene dal voler mettere a paragone i due progetti. Una trattazione riguardante la **stabilità** di un **attributo di progetto** è essenziale nel momento in cui si vuole **monitorare**, **controllare** e **predirre** il comportamento dello stesso. Molteplici ricerche suggeriscono che un **progetto stabile** permette di migliorare notevoli aspetti dello sviluppo, e quindi fornire un **prodotto** di **qualità maggiore**. L'aspetto di misurare permette di **prendere decisioni** basate **su evidenze oggettive**, e quindi assicurare una *certa qualità* sia per quanto riguarda il progetto che per quanto riguarda il **processo** che ha come prodotto il progetto.

Analisi del sorgente e assunzioni

Il caloclo della metrica di *bugFix/mese* è stata effettuata usando un programma Java (1.8) che automatizza il processo dal *raccolgimento dei dati* dalle varie fonti (JIRA e GitHub) fino all'*estrapolazione* di un file **.csv** sulla quale vengono riportati il numero di bug fixati per ogni mese insieme ad *ulteriori statistiche* che ci permettono di estrapolare informazioni preziose sui risultati ottenuti. Il **.csv** è stato construito in modo da velocizzare la creazione del grafico che mostra l'andamento e la stabilità del progetto, seppur rimanendo leggibile e comprensibile anche senza usare un grafico.

Project name	PARQUET			
GitHub repository link	https://github.com/apache/parquet-mr/			
Non-linked commits	317			
Non-linked commits Percentage	57,427536			
MONTH-YEAR	NUMBER BUG F	MEAN	MEAN+3STD	MEAN-3STD
giu-14	1	3,263889	13,035936	0
lug-14	6	3,263889	13,035936	0
ago-14	9	3,263889	13,035936	0
set-14	10	3,263889	13,035936	0
ott-14	3	3,263889	13,035936	0
nov-14	4	3,263889	13,035936	0
dic-14	2	3,263889	13,035936	0
gen-15	7	3,263889	13,035936	0
feb-15	2	3,263889	13,035936	0
mar-15	13	3,263889	13,035936	0
apr-15	8	3,263889	13,035936	0

*Esempio del file **.csv** estrapolato usando **PARQUET.***

Per trovare il numero di *bugFix/mese* si è usata la piattaforma **JIRA**, da cui sono stati estrapolati i **ticket chiusi** relativi ai bug dei due progetti, per poi usare le informazioni ricavate dai ticket su **GitHub** per trovare la data relativa al **fix** del **bug**. Fatto ciò, i bug sono stati raggruppati per mese e poi conteggiato il numero di bug per ogni mese.

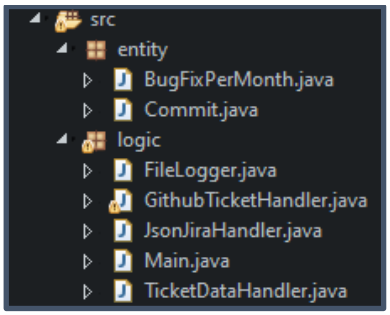
E' stata posta particolare attenzione sulla **validità** dell'output, infatti in fase di *runtime* vengono effettuati molteplici check al fine di garantire che i dati acquisiti siano *corretti* e *consistenti* per assicurarsi di fare la successiva analisi su dati veritieri.

*Un esempio di **check aggiuntivo** che è stato implementato nella funzione* `retreiveBugFixPerMonth` *è quello di contare che il numero di **commit** relativo a tutti i ticket sia uguale alla **somma** dei **bug fixati** per **tutti i mesi**. Ovviamente un controllo di questo tipo rallenta l'esecuzione dell'eseguibile, ma ci da una ulteriore garanzia che i dati raccolti siano consistenti*

Non sono state effettuate particolari operazioni di **ottimizzazione** del codice poiché il tempo di esecuzione varia dai 30 secondi a 1 minuto massimo, di cui la maggior parte del tempo viene effettuato per il *clone* della *repository* del progetto (passo necessario se si vuole accedere al log delle commit), che viene prontamente eliminata alla fine dell'operazione per evitare di memorizzare dati non più neccessari sulla memoria. Inoltre, per fornire dei dati sull'esecuzione del nostro *parser* è stato implementato un **logger** diviso su livello di **gravità** che scrive su di un file le operazioni e informazioni di contesto durante il raccoglimento dei dati. L'interfaccia con **GitHub** è stata creata usando l'**API JGit** insieme ad altre librerie di

supporto, tutte **embedded** nel sorgente per garantire massima esportabilità ed evitare di dover configurare l'ambiente di esecuzione se viene eseguito su una macchina nuova.

Il codice è stato creato pensando ad **un'alta modularità** al fine di poter **riutilizzare** le componenti, evitare **design** considerati **problematici** (es. Metodi estremamente grandi, Classi *tuttofare*, ecc.), fornire un buon livello di **leggibilità** del codice (da che ne consegue un mantenimento/aggiornamento del codice più semplice) e un **organizzazione** basata su **funzionalità** delle classi.



Gerachia delle classi.

Le **assunzioni** che sono state fatte relativamente al **raccoglimento** e all'**interpretazione** dei dati sono le seguenti:

- La data del **fix** di ogni **bug ticket** su JIRA è stata assunta *meno affidabile* di quella su **GitHub**.
- Nel caso vi sia presente un ticket su JIRA ma non su GitHub, il ticket viene **scartato**
- Tutte le operazioni di analisi devono essere fatte direttamente a **run time** invece che usando il foglio di calcolo (vedi *media, deviazione std, ecc.*)
- Se il **lower limit** è sotto lo **zero**, allora viene automaticamente impostato a 0
- Se uno dei punti del grafico non è nell'intervallo [lowerLimit, upperLimit] il progetto (o meglio, il processo) è considerato **instabile**.

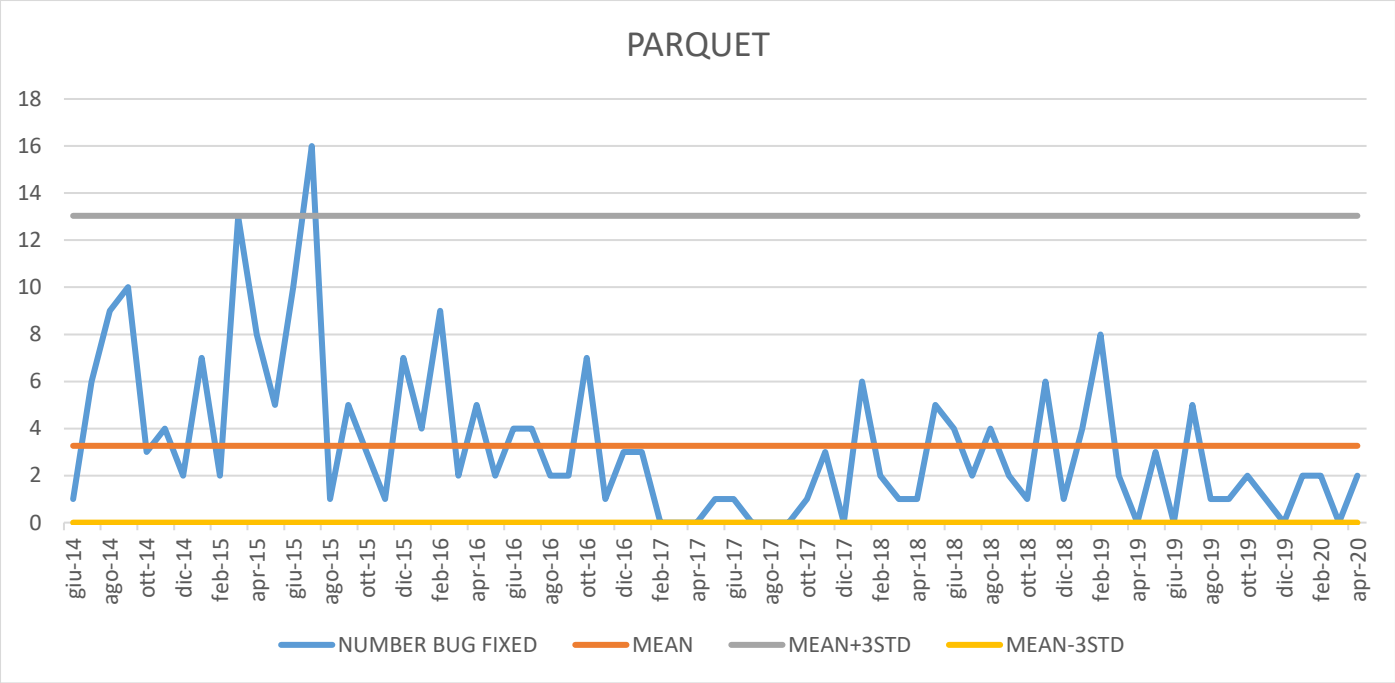
Analisi dei dati e del grafico

Oltre al **conteggio** dei bug fixati ogni mese, vengono anche calcolati **media** e **lower/upper limit**. Il lower/upper limit sono calcolati come segue:

```
lower/upper limit = mean -/+ 3 * standard_deviation
```

Inoltre viene anche calcolata la **percentuale** di **ticket** che non hanno *nessun riscontro* su **GitHub**.

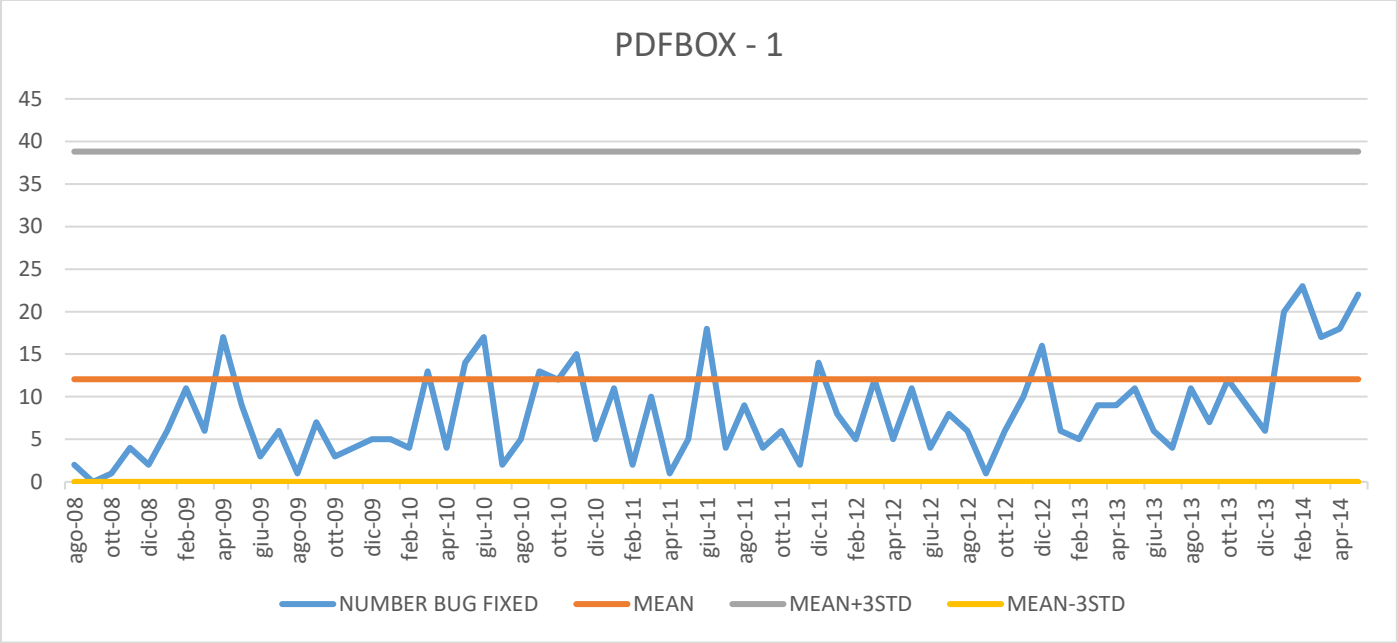
Per quanto riguarda il progetto **PARQUET** abbiamo il seguente andamento:

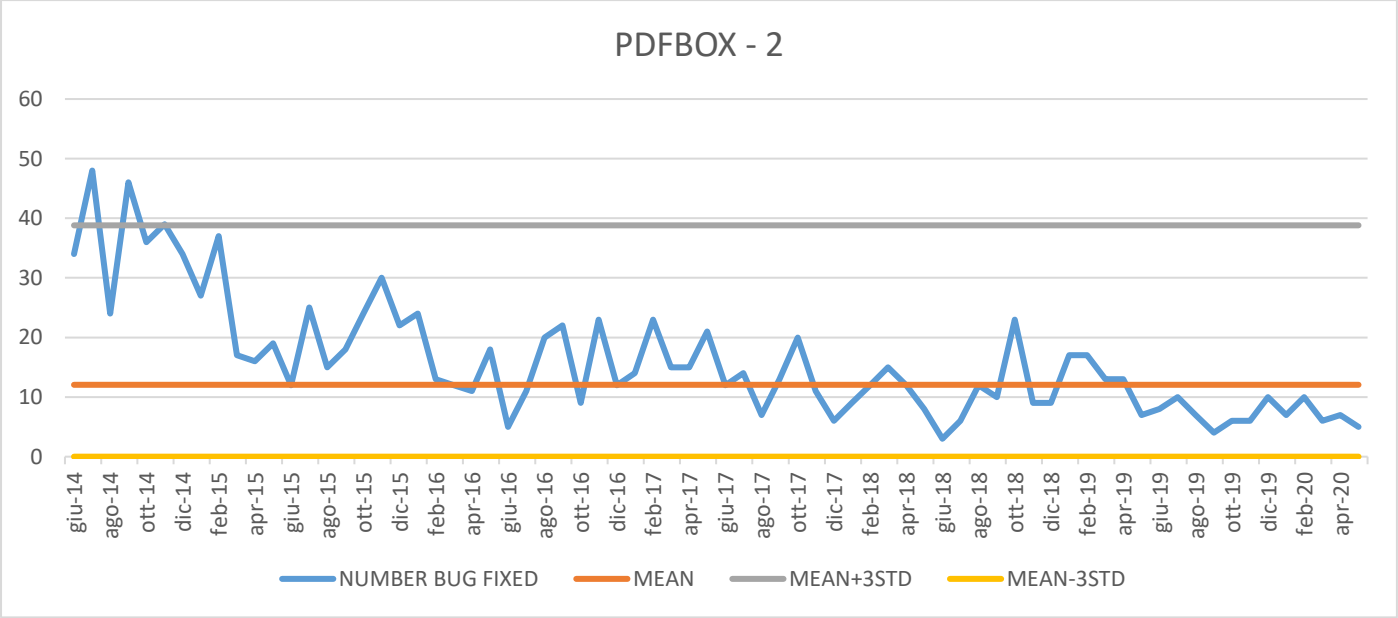


Con una **percentuale di ticket non linkata** su Github del 57,42% (317 tickets non linkati), una percentuale molto alta, è come se ci fossimo persi *più della metà* dei dati.

E' evidente come nelle fasi iniziali non abbiamo un grafico stabile, infatti presenta molteplici oscillazioni, di cui una al di fuori dei limiti inferiori/superiori, sintomo di **instabilità**. Già dall'inizio del 2018 si evidenzia una stabilità più alta e una maggiore costanza, con i valori più vicini alla media e tutti abbondandemente dentro i limiti superiori/inferiori, il che sta a significare una maggiore *maturità* del progetto e dell'organizzazione.

Per quanto invece riguarda il progetto **PDFBOX** abbiamo il seguente andamento (il grafico è stato diviso in due parti per una migliore leggibilità:





Con una **percentuale di ticket non linkata** su Github del 19,62% (421 tickets non linkati), una percentuale decisamente più bassa e accettabile di PARQUET, anche se corrisponde a *circa 1/5* dei *dati*.

Il progetto è stabile per la maggior parte del tempo, ma tra il 2014 e il 2015 ha un picco di bug fix (di cui alcuni anche sopra il limite superiore), il che può significare l'implementazione di nuove feature, una reimplentazione da capo dell'architettura o semplicemente un recupero di molteplici bug arretrati.

DELIVERABLE 2

Descrizione Iniziale

Nota: La parte di Machine Learning non è completa, perciò è stata omessa dal report provvisorio

Il secondo Delivarable ha come scopo quello di **eseguire uno studio empirico** basato su modelli di **Machine Learning** al fine di identificare le **Classi Buggy** di un progetto con tecniche di **Sampling** e **feature selection** offerta dall'API di **Weka**.

Il deliverable si divide in due fasi:

- **Acquisizione** dei dati della **bugginess** mediante **JIRA** e **GitHub**
- **Analisi** dei dati usando il **Machine Learning**

La prima fase consiste nell'acquisire la conoscenza se una determinata classe è **Buggy** o no su una determinata **Release,** insieme all'acquisizione di altre metriche, la cui scelta è direttamente collegata alla Bugginess della classe (es. una metrica come la **size** è probabilmente molto più legata alla bugginess della stessa piuttosto che il **numero di commenti**). I progetti presi in analisi sono **BOOKKEEPER** e **SYNCOPE**.

L'acquisizione di questi dati, insieme all'analisi mediante modelli **statistici** o, come nel nostro caso, mediante modelli di **ML**, permette di fare operazioni di **predizione** di dati futuri o comunque non noti. La **bugginess** è un attributo non facilmente attribuibile, specialmente in mancanza di dati essenziali come **AV, IV, OV** e **FV** di un bug, infatti nonostante la presenza di molteplici metodi per lo studio della bugginess, non è possibile stabilire con assoluta certezza se si ha a che fare con una classe buggy o meno (prendendo ad esempio il metodo **SZZ**, il quale fa uso del **blame**, oltre ad essere molto complesso da implementare, è soggetto a limitazioni importanti come una **accuratezza non perfetta**, non rileva bug di tipo **regressivo** e bug dove il fix è stato ottenuto con **l'aggiunta di codice**). Mentre la prima fase si occupa di ottenere i dati mediante tecniche di **Proportion**, la seconda parte del deliverable si occupa dell'impiego di **ML** al fine di poter usare i dati raccolti per predirre la **bugginess** con un certo livello di **confidenza**. Uno dei vantaggi chiave del ML è che permette di creare un modello sufficientemente accurato **automaticamente** a partire dai dati presenti

Analisi del sorgente e assunzioni – Parte 1

La prima parte del Deliverable consiste nel ottenimento delle **metriche** e quello di calcolare la **bugginess** di una **classe**.
Le **metriche** scelte sono le seguenti:

- Size
- Age
- Number of Revision
- Number of Authors
- LOC Touched
- LOC Added
- Max LOC Added
- Average LOC Added
- Change Set Size
- Max Change Set Size
- Average Change Set Size

Per un totale di **11 metriche**. Le metriche sono state scelte cercando di capire quali sonom i fattori che più possono influenzare la bugginess di una classe.

La **Bugginess** invece è stata calcolata, quando non presente **AV e FV** nei ticket raccolti su JIRA, usando un inseme di **Proportion Cold Start** e **Proportion Increment**. Nelle fasi iniziali (cioè le prime versioni) non si hanno sufficienti dati per poter utilizzare il **Proportion Increment**, perciò viene applicato il **Cold Start** su **20 altri progetti** presenti su JIRA, calcolando **P** (La proporzione tra (FV – IV) e (FV – OV)) come la **mediana** tra tutti gli P di tutti i ticket (trovando un valore di **P = 1.75**), andando a scartare ovviamente quei ticket la cui informazione è ambigua (es. IV = OV = FV).

Proprio come per il primo deliverable, è stata posta più **enfasi** sulla **validità** dei dati piuttosto che **dell'ottimizzazione** (anche qui, il tempo di esecuzione si aggira intorno ai 2 minuti, il quale è un tempo ragionevole considerando che va eseguito una sola volta per raccolgiere i dati, e che la maggior parte del tempo è speso per fare il *clone*, il quale non può essere ottimizzato), infatti vengono fatti molti controlli per evitare dati **falsati.**

*Un esempio di **check aggiuntivo** che è stato implementato è che i file **presenti in una versione** (cioè l'ultima commit prima della release) sono gli stessi che vengono trovati se si segue il flusso di **git diff** tra una commit e un altra*

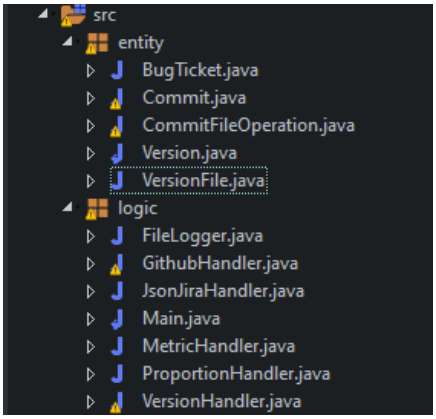
I dati recuperati da **JIRA** sono relativi alle **versioni** del progetto rilasciate e i **ticket dei bug** con annesse **AV e FV** quando presenti, mentre il restante dei dati come le **metriche** vengono estratte e calcolate mediante GitHub, interfacciata sempre usando la API di **JGit.** Come già detto per la prima deliverable, tutte le librerie necessarie sono state **embedded** nel sorgente per garantire massima **portabilità** e l'implentazione di un **logger** per tenere traccia delle operazioni effettuate.

Come per la prima delivarabel, l'**output** principale è un **file .csv** contenente per ogni **file** di ogni **versione** le **metriche** e la **bugginess**, espressa come valore **booleano**

Project name	BOOKKEEPER												
Version	FileName	Size	Age	Number of revision	Number of Author	Loc Touched	Loc Added	Max Loc Added	Avg Loc Added	Change Set siz	Max ChgSetSiz	Avg ChgSetSize	Defects
1	bookkeeper-benchmark/src/mai	121	35	4	2	333	235	137	58.0	559	281	139.0	0
1	bookkeeper-benchmark/src/mai	215	35	5	2	436	340	252	68.0	560	281	112.0	1
1	formatter.xml	286	35	1	1	286	286	286	286.0	281	281	281.0	1
1	hedwig-client/pom.xml	126	35	10	3	200	165	86	16.0	428	281	42.0	0
1	hedwig-client/src/main/cpp/Ma	24	35	1	1	29	29	29	29.0	281	281	281.0	1
1	hedwig-client/src/main/cpp/am	135	35	1	1	186	186	186	186.0	281	281	281.0	0
1	hedwig-client/src/main/cpp/c-d	1267	35	2	2	1267	1267	1252	633.0	294	281	147.0	1
1	hedwig-client/src/main/cpp/cor	39	35	2	2	49	48	40	24.0	311	281	155.0	0
1	hedwig-client/src/main/cpp/hec	27	35	1	1	30	30	30	30.0	281	281	281.0	1
1	hedwig-client/src/main/cpp/inc	45	35	2	2	50	50	45	25.0	311	281	155.0	1
1	hedwig-client/src/main/cpp/inc	71	35	3	2	86	86	80	28.0	322	281	107.0	1
1	hedwig-client/src/main/cpp/inc	42	35	1	1	51	51	51	51.0	281	281	281.0	0
1	hedwig-client/src/main/cpp/inc	55	35	1	1	61	61	61	61.0	281	281	281.0	0
1	hedwig-client/src/main/cpp/inc	43	35	1	1	52	52	52	52.0	281	281	281.0	0
1	hedwig-client/src/main/cpp/lib/	28	35	2	2	34	33	32	16.0	311	281	155.0	1
1	hedwig-client/src/main/cpp/lib/	421	35	4	2	525	507	420	126.0	327	281	81.0	0
1	hedwig-client/src/main/cpp/lib/	136	35	4	2	169	169	156	42.0	327	281	81.0	1
1	hedwig-client/src/main/cpp/lib/	48	35	3	2	62	62	57	20.0	322	281	107.0	1
1	hedwig-client/src/main/cpp/lib/	341	35	5	2	414	406	376	81.0	327	281	65.0	0
1	hedwig-client/src/main/cpp/lib/	131	35	4	2	159	158	150	39.0	326	281	81.0	1

*Esempio del file **.csv** estrapolato usando **BOOKKEEPER.***

Il codice è stato creato pensando ad **un'alta modularità** al fine di poter **riutilizzare** le componenti, infatti moltemplici componenti sono state riprese dalla prima deliverable, e per fornire un **organizzazione** basata su **funzionalità** delle classi.



*Struttura delle **Classi** del sorgente.*

Le **assunzioni** che sono state fatte relativamente al **raccoglimento** e all'**interpretazione** dei dati sono le seguenti:

- Le **versioni** vanno estrapolate da **JIRA** e non da **GitHub**
- Le metriche come **Number of Authors** o **Number of revision** non sono **incrementali**, bensì vengono ricalcolate per ogni **release**
- Ha senso studiare la **bugginess** solo di certi tipi di file tipo **.xml, .java, ecc.** che sono relativi alla parte **funzionale** della *repository* (vengono per esempio evitati file .log o .txt)
- A seguito di un **COPY**, le metriche e la bugginess del file copiato vanno calcolate da 0, viene infatti visto come un file a parte