UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II
Scuola Politecnica delle Scienze di Base
Corso di Laurea in Ingegneria Informatica

Elaborato finale in **Ingegneria del Software**

Utilizzo di SonarQube per il monitoraggio continuo della qualità dello sviluppo software

Anno accademico 2016/2017

Candidata:
Martina Pappalardo
matr. N46001794

Indice

Indice delle figure

1 Introduzione

La qualità del software, intesa come conformità ai requisiti funzionali e prestazionali enunciati esplicitamente dal committente, è di vitale importanza nei progetti di sviluppo del software. Influisce ogni aspetto del sistema, come la funzionalità, l'affidabilità, la disponibilità, la manutenzione e la sicurezza.

In svariati progetti software si sviluppano delle grandi quantità di codice sorgente, che vengono poi tenute operative e mantenute per lunghi periodi di tempo. Durante questi periodi, a causa delle attività di manutenzione, la struttura interna del codice sorgente si modifica, spesso degradando. Questo accade per molte ragioni, ad esempio, il codice viene modificato senza prestare la necessaria attenzione al "refactoring", oppure, senza verificare la coerenza di tali modifiche con la relativa documentazione a livello di requisiti di progettazione. Come altro esempio, alcune porzioni di codice possono diventare così complicate nel tempo da non poter essere più facilmente capite e modificate. Nei progetti software critici, la garanzia della qualità del codice, identificata anche come l'insieme di metriche e di standard che un software deve avere per il suo corretto funzionamento, deve essere considerata ad ogni livello partendo dal concetto iniziale fino ad arrivare al processo di ingegneria del software: dalla specifica alla codifica e all'integrazione. Al livello di codifica più basso, esistono diversi strumenti che consentono il monitoraggio e il controllo della qualità del software. Uno di questi è SonarQube, una piattaforma open source per la gestione della qualità del codice, utilizzata per analizzare e misurare la qualità tecnica.

2 Qualità del software e analisi statica

Lo sviluppo del software è un'attività creativa e tecnica allo stesso tempo; un'attività ad altissimo contenuto intellettuale e quindi con un grande margine di errore umano. In estrema sintesi, si tratta di tradurre esigenze (spesso non ben identificate) in una soluzione tecnica deterministicamente corretta. Quello che possiamo fare è, da un lato, ridurre il numero di errori inevitabilmente commessi nelle singole fasi e, dall'altro, potenziare la rimozione degli errori commessi individuandoli il più presto possibile.

Ci sono quindi due fasi cruciali per la qualità del software:

1. l'immissione di errori nella fase di realizzazione (analisi, disegno e codifica);
2. la rimozione degli errori, prima attraverso la revisione dei requisiti e della progettazione, poi attraverso la verifica (test) del codice sviluppato.

Secondo alcune statistiche, un errore commesso in una fase iniziale del ciclo di sviluppo software genera altri errori (da 3 a 5) nella fase successiva, e così via. Un singolo errore di interpretazione di un requisito può generare anche 5 errori nelle specifiche successive, fino a 25 errori di disegno e fino a 125 errori di codifica. L'analisi condotta sugli errori rilevati nel codice, inoltre, ha dimostrato che essi sono imputabili solo in minima parte ad una errata codifica, mentre la maggior parte di essi è da imputare a specifiche e a requisiti incompleti o errati, a errori nel disegno dei dati, nella progettazione o nei test.

Un altro elemento che è cresciuto notevolmente negli ultimi decenni è la complessità del software e la conseguente difficoltà ad effettuarne la manutenzione. In pratica, il codice è continuamente modificato per rispondere a diverse esigenze: nuovi requisiti, adeguamento a nuove normative, miglioramento di aspetti funzionali, prestazionali e gestionali, ecc. Intervenire sul codice esistente può risultare, a volte, difficile e costoso quando esso non sia stato già progettato e realizzato correttamente per facilitare la sua manutenzione, sia correttiva, evolutiva e migliorativa.

Uno degli scopi fondamentali dell'Ingegneria del Software è diventato in poco tempo quello dell'analisi della qualità del codice sorgente, identificata anche come l'insieme di metriche e standard che il software d'interesse deve soddisfare per il suo corretto funzionamento. Nel corso degli anni, quando ci si è accorti che l'analisi statica di un software era uno dei passi fondamentali per la realizzazione di un buon progetto, sono state inventate tantissime metriche riguardanti gli aspetti più svariati. L'analisi statica è un processo di valutazione di un sistema software o di un suo componente basato sulla sua forma, struttura, contenuto, documentazione senza che esso sia eseguito. Questo è diverso dall'analisi dinamica che invece è un processo di valutazione basato sulla osservazione del suo comportamento in esecuzione. Il concetto di analisi statica include vari aspetti legati alla qualità del software: studio delle proprietà del programma, ricerca di errori nel codice, prova della sua fattibilità, ecc.

L'utilizzo di analizzatori statici consente di ridurre il costo dello sviluppo dei prodotti dovuto al rilevamento precoce degli errori, nella fase in cui l'errore non è legato a dipendenze complesse e quando il valore di correzione è minima. Ciò consente di risparmiare tempo agli sviluppatori e ai tester. Il risultato dell'analisi statica è quello di identificare i pezzi di codice considerati sospetti, ossia affetti da errori, segnalandoli opportunamente. Gli allarmi dell'analizzatore possono essere classificati nel seguente modo[1]:

- *Useful* - avviso dell'analizzatore che indica l'errore effettivo nel codice del programma. Quanto più tali messaggi sono presenti nell'output dell'analisi, maggiore è la sua efficacia.
- *Noise* – "rumore" che influenza il risultato dell'analizzatore. Riduce l'output utile dell'analizzatore.
- *False positive* – falso positivo che innesca l'attivazione del pezzo di codice che l'analizzatore ha identificato errato, anche se il codice in realtà è corretto. Riduce l'output utile dell'analizzatore ritirando gli allarmi sul codice corretto.
- *False negative* – falso negativo che attiva l'analisi che non è stata evidenziata sul codice errato. Nell'output risultante non compaiono, ma a causa della loro natura (le aree problematiche mancanti) riducono effettivamente l'output utile.

Per verificare gli errori con la metodologia proposta di analisi statica sono richieste tre componenti[1]:

- Informazioni sintattiche;
- Informazioni semantiche;
- Set di regole diagnostiche.

Le informazioni sulla sintassi sono rappresentate dal codice del programma stesso e sono necessarie per effettuare l'analisi. Prima che l'analizzatore inizi la ricerca esso produce un albero sintattico, con il quale in futuro viene svolto il lavoro. La costruzione errata dell'albero sintattico riduce l'efficacia dell'analisi statica. Per quanto riguarda le informazioni semantiche non sono obbligatorie per l'analisi statica, ma hanno aumentato notevolmente l'efficienza grazie all'analisi più approfondita delle espressioni. Le regole di diagnostica implementano la funzionalità dell'analizzatore del codice statico per rilevare frammenti di codice difettosi. Maggiore è il numero di regole maggiore è la possibilità di trovare un frammento di codice molto problematico.

Poiché è difficile tenere sotto controllo lo sviluppo del codice attraverso semplici analisi visive, avere a disposizione dei software che effettuano automaticamente dei test e delle analisi approfondite è un grande aiuto per il team di sviluppo. Lo strumento di cui ci occuperemo è open source, cioè software il cui codice sorgente è messo a disposizione di tutti permettendone lo studio e la modifica, se ritenuta necessaria. Tale software effettua una scansione del codice e ne individua gli errori in maniera automatica con un elevato grado di fiducia. Esso consente, attraverso numerosi strumenti, di analizzare il codice in base ad un vasto numero di metriche e, di conseguenza, di garantire una copertura quasi totale degli aspetti principali riguardanti il grado di qualità del codice. Nella maggior parte dei casi gli strumenti di analisi vengono utilizzati come supporto all'analista per individuare delle falle all'interno del codice ed indirizzarlo verso modifiche appropriate. Alcuni di questi strumenti agiscono all'interno degli IDE (ambienti di sviluppo integrato), il che permette di effettuare l'analisi durante la fase di sviluppo. Questo è effettivamente il momento migliore per consentire al team di tornare indietro (azione di feedback) e aggiustare parti di codice che potrebbero, nello sviluppo futuro, provocare danni irreparabili. Esistono diversi strumenti di analisi (es. Checkstyle[2], Stan4j[3], ecc.), ma noi ci soffermeremo sulla piattaforma SonarQube[4].

3 Ambiente di sviluppo

3.1 Architettura e integrazione

SonarQube è un'applicazione web che produce report sul codice duplicato, sugli standard di programmazione, i test di unità, il code coverage, la complessità, i bug potenziali, i commenti, la progettazione e l'architettura. La piattaforma SonarQube è composta da quattro componenti:

1. SonarQube server, il quale avvia tre processi fondamentali:
   - un server web per gli sviluppatori per configurare l'istanza SonarQube ed esplorare le istantanee di qualità;
   - un server di ricerca per ripristinare le ricerche dell'interfaccia utente;
   - un server di calcolo incaricato di elaborare i rapporti di analisi dei codici e salvarli nel database SonarQube.
2. SonarQube Database per archiviare:
   - le istantanee di qualità di progetti, opinioni, ecc.;
   - la configurazione dell'istanza di SonarQube (sicurezza, impostazioni dei plugin, ecc.)
3. Plugin di SonarQube installati sul server, possibilmente inclusi plugin di linguaggi, integrazione, autenticazione e governace-plugin.
4. Uno o più scanner SonarQube per analizzare i progetti.

La Figura 1 mostra come SonarQube si integra con altri strumenti ALM (Application Lifecycle Management). In particolare i vari componenti di SonarQube vengono usati in 7 fasi distinte:

1. Gli sviluppatori codificano nei loro IDE e utilizzano SonarLint per eseguire l'analisi locale.
2. Gli sviluppatori spingono il loro codice nel loro SCM (Software Configuration Management) preferito (es. git, SVN, TFVC, ecc.)
3. Il server di integrazione continua CIS (Continuos Integration Servers) innesca una creazione automatica del progetto con tutte le sue proprietà e l'esecuzione dello Scanner SonarQube necessaria per eseguire l'analisi.

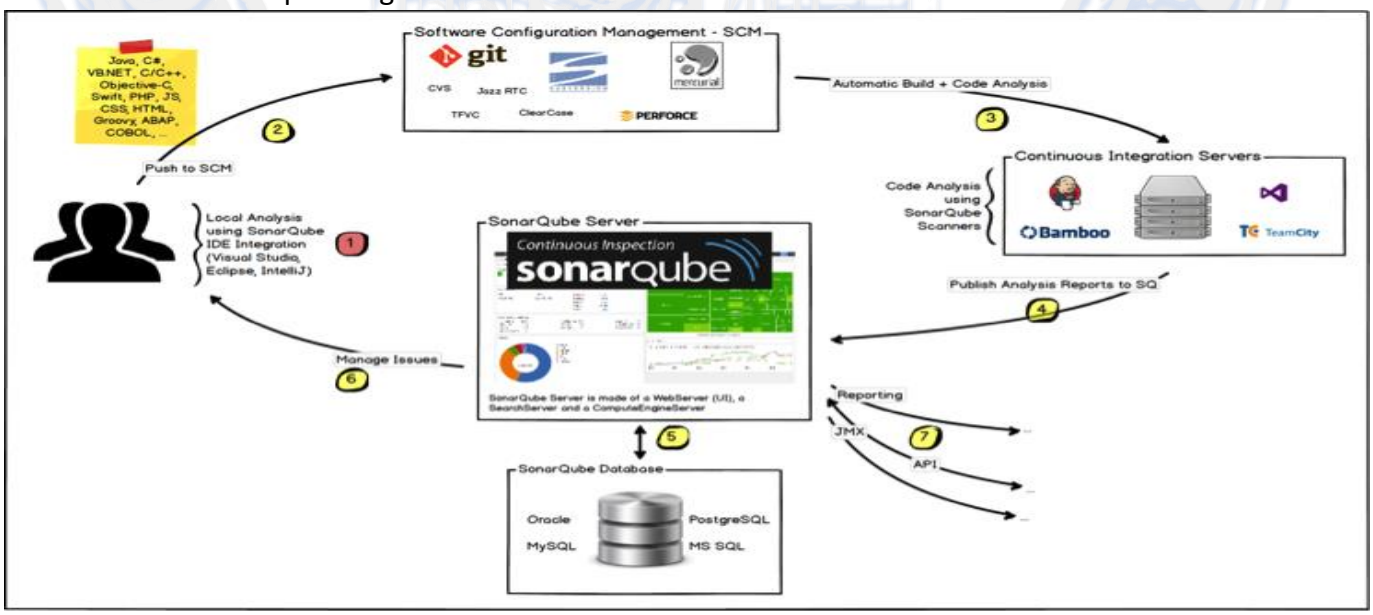

Figura 1.Architettura SonarQube

5. SonarQube Server elabora e memorizza i risultati del rapporto di analisi nel database SonarQube e visualizza i risultati nell'interfaccia utente.
6. Gli sviluppatori riesaminano, commentano e affrontano i loro problemi per gestire e ridurre il loro debito tecnico attraverso l'interfaccia utente SonarQube.
7. I gestori ricevono i rapporti dall'analisi.

Per completezza si riportano alcune informazioni sulle componenti e sulla loro localizzazione:

1. La piattaforma SonarQube non può avere più di un SonarQube Server e di un database SonarQube.
2. Per prestazioni ottimali, ogni componente (server, database, scanner) dovrebbe essere installato su una macchina separata e la macchina server deve essere dedicata.
3. Non esiste alcuna comunicazione tra SonarQube Scanner e il database SonarQube.
4. Gli scanner SonarQube non devono essere sulla stessa rete del server SonarQube.

3.2 Analisi del codice sorgente

Una volta installata la piattaforma SonarQube si procede istallando un analizzatore, iniziando così a creare progetti. Un progetto viene creato automaticamente nella piattaforma procedendo con la prima analisi. SonarQube può eseguire analisi su 20 e più linguaggi di programmazione. I risultati di questa analisi saranno misure e problemi di qualità. Tuttavia, ciò che viene analizzato varia a seconda della lingua:

- In tutte i linguaggi viene eseguita un'analisi statica del codice sorgente (file Java, programmi COBOL, ecc.).
- È possibile eseguire un'analisi dinamica del codice con determinati linguaggi.

3.2.1 File non riconosciuti

Per impostazione predefinita, solo i file riconosciuti da un plugin di linguaggio vengono caricati nel progetto durante l'analisi. Ad esempio, se l'istanza SonarQube dispone dei plugin Java e JavaScript, tutti i file .java e .js verranno caricati, mentre i file .xml, ad esempio, saranno ignorati. Tuttavia, è possibile importare tutti i file di testo nella codifica di analisi in un progetto, impostando il path: *Settings > Exclusions > Files > Import unknown files to true*.

3.2.2 Durante l'analisi

Durante l'analisi, i dati vengono richiesti dal server. I file forniti vengono analizzati e i dati risultanti vengono restituiti al server alla fine sotto forma di un rapporto, che viene quindi analizzato in modo asincrono sul lato server. I rapporti di analisi sono in coda e vengono elaborati in sequenza, quindi è abbastanza possibile che per un breve periodo, dopo il completamento del registro di analisi, i valori aggiornati non siano visibili nel progetto SonarQube. Tuttavia, sarà in grado di dire cosa succede perché l'icona "*In Progress*" verrà aggiunta accanto al nome del progetto.

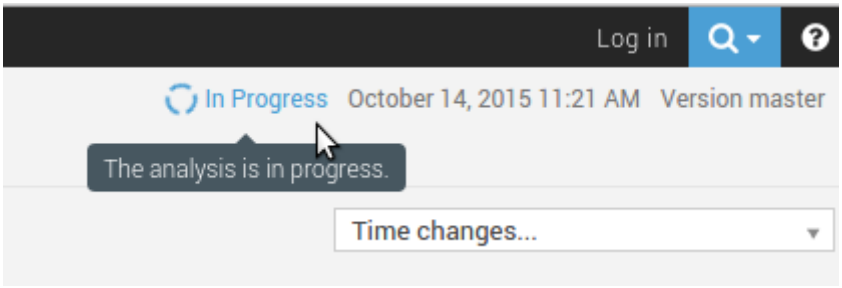

Figura 2.Icona elaborazione progetto

L'icona si spegne una volta che l'elaborazione è completa, ma se l'elaborazione dei rapporti di analisi non riesce per qualche motivo, l'icona cambia in *"Failed"* (Figura 3):

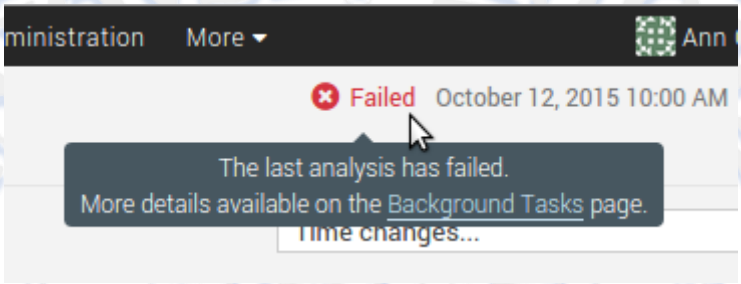

Figura 3.Icona a seguito di fallimento

L'analisi non è completa finché non è stata completata la relativa attività di background. Anche se il registro di SonarQube scanner mostra *EXECUTION SUCCESS*, i risultati delle analisi non saranno visibili nel progetto SonarQube fino al completamento dell'attività di background. Dopo che SonarQube scanner ha terminato l'analisi del codice, il risultato (o rapporto) dell'analisi (sources, issues, metrics) viene inviato a SonarQube server per l'elaborazione finale da parte del motore di calcolo. I rapporti di analisi sono in coda e vengono elaborati in serie. A livello di progetto, quando esiste un rapporto di analisi in attesa di essere consumato, compare un avviso "*Pending*" nell'intestazione.

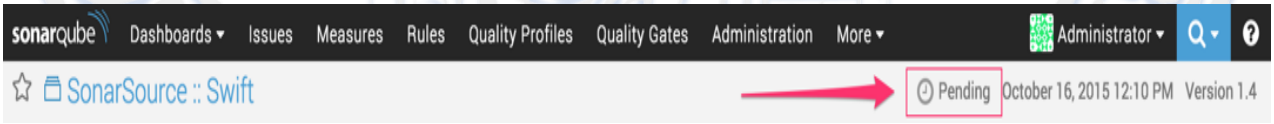

Figura 4.Rapporto di analisi

Gli amministratori globali possono visualizzare la coda corrente in *Administration > Projects > Background Tasks*.

Se un'attività di sfondo è in fase di elaborazione, verrà contrassegnata con un'icona "*wait*" in questo elenco.

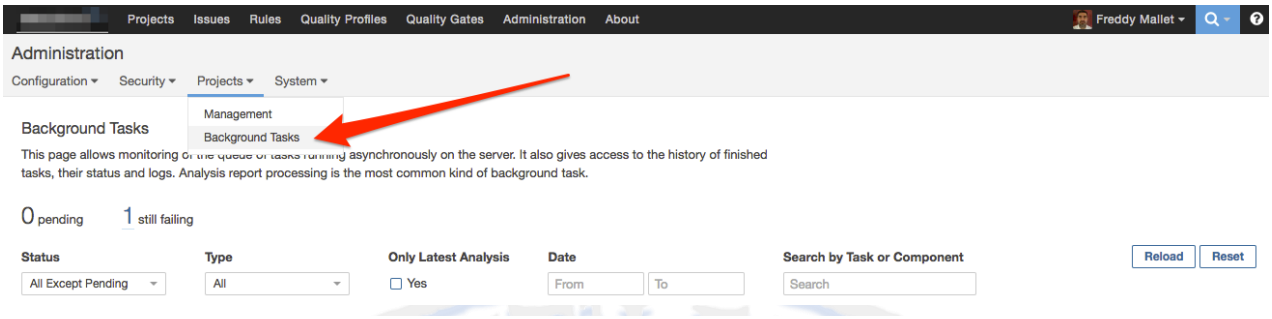

Figura 5. Coda di progetti

Le attività di background generalmente hanno successo, ma a volte circostanze insolite causano il fallimento dell'elaborazione. Ad esempio:

- esaurimento della memoria durante l'elaborazione di un report di un progetto molto grande;
- "scontro" tra una chiave (identificatore univoco del progetto, vedi 3.2.4) di un modulo o di un progetto già esistente e un report.

Quando ciò accade, lo stato fallito è riflesso sulla home page del progetto, ma richiede che qualcuno lo noti. Vi è la possibilità di ricevere una notifica via e-mail quando le attività in background falliscono.

Come mostrato in Figura 6, per ogni rapporto di analisi è disponibile un menu a tendina che consente di accedere allo "*Scanner Context*" che mostra la configurazione dello scanner al momento in cui è stata eseguita la scansione dei codici. Se l'elaborazione non è riuscita per l'attività, sarà disponibile un'opzione aggiuntiva: "*Show Error Details*" per ottenere i dettagli tecnici relativi al fallimento dell'elaborazione.

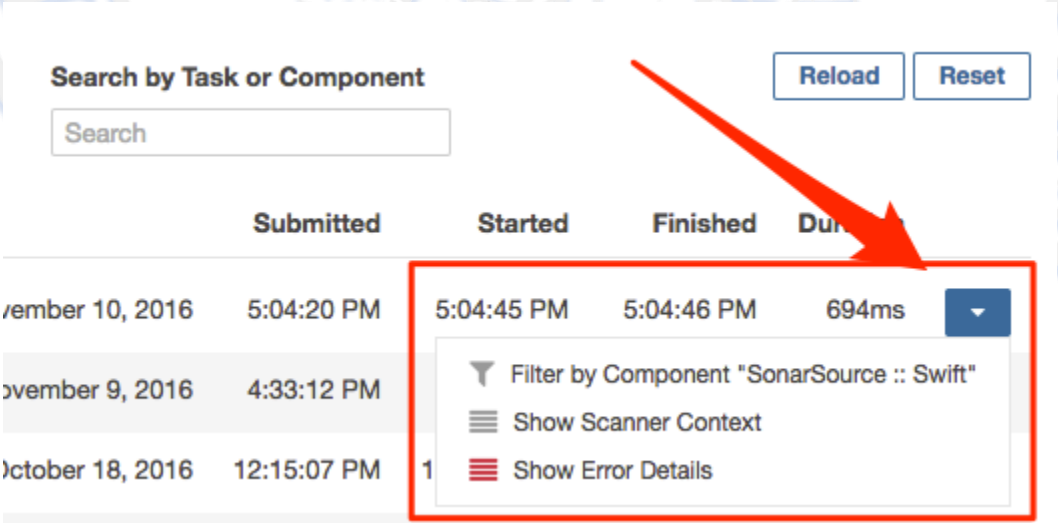

Figura 6.Menu associato al rapporto di analisi

Gli amministratori possono annullare l'elaborazione di un'attività in sospeso facendo clic su:

- Sulla 'x' disponibile su ogni riga di un lavoro sospeso, ciò permette l'eliminazione del singolo lavoro, vedi Figura 7 parte inferiore.

- sull'opzione "bulk cancel " rosso accanto al conteggio dei lavori in sospeso. Questo pulsante consente di annullare tutte le attività in sospeso, vedi Figura 7 parte superiore.

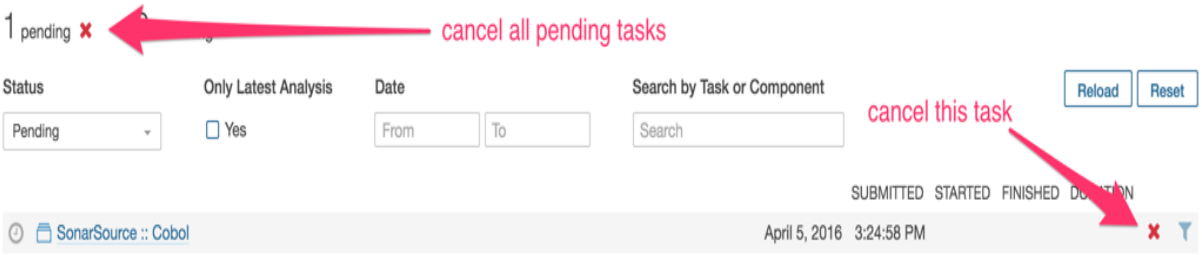

Figura 7.Annullamento elaborazione attività

3.2.2.1 Filtri

È possibile filtrare le attività di sfondo in base al loro stato: *Pending*, Success, *Failed* o *Canceled*. Il pulsante " *Only Latest Analysis*" filtrerà le attività di sfondo mostrando solo l'ultima importazione per ogni progetto. È inoltre possibile filtrare le attività di sfondo in base alla data di inizio dell'importazione. Nella parte superiore della pagina ci sono due o tre contatori:

- pending - mostra il numero di rapporti di analisi in coda e in attesa di essere elaborati.
- failures - mostrano il numero di progetti in cui il trattamento di un rapporto di analisi più recente del progetto è fallito.
- il terzo contatore è presente solo durante l'elaborazione di un rapporto di analisi. Esso mostra la durata della data dell'attività in fase di elaborazione.

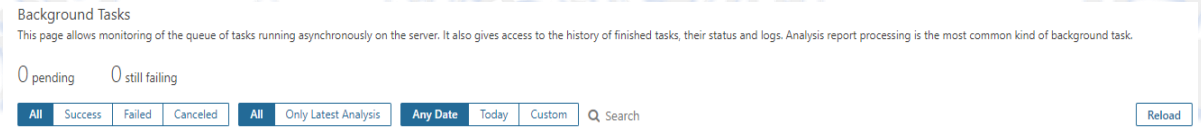

Figura 8. Filtraggio attività

3.2.3 Impostazioni progetto

L'amministrazione del progetto è accessibile tramite il menu *Administration*. Solo gli amministratori di progetto possono accedere alle impostazioni del progetto. Un progetto viene aggiunto automaticamente alla sua prima analisi. Selezionando *Administration > Tasks* è possibile consentire agli amministratori del progetto di controllare l'elaborazione dei loro progetti. È possibile eliminare un progetto tramite *Administration > Deletion*. Per settare il leak period bisogna seguire questo cammino *Administration > General Settings > Leak > Leak Period*. La chiave del progetto può essere aggiornata (senza perdere la storia del progetto) in *Administration > Update Key*. La nuova chiave deve contenere almeno un carattere non cifrato. I caratteri ammessi sono: da 'a' a 'z', da 'A' a 'Z', '-', '_', '.', ':' e cifre da '0' a '9'. Gli amministratori del progetto possono scegliere quali:

- **Quality profiles** *(Administration > Quality Profiles*);
- **Quality gate** (*Administration > Quality Gate*);

utilizzare per il loro progetto.

Il *Quality profiles* è uno dei servizi più importanti offerti dalla piattaforma. Esso permette all'utente di definire un set di regole, ad esempio pretendere che il proprio progetto non abbia metodi con complessità superiore a 10. Qualsiasi utente può accedere al servizio Quality profiles, quindi tutti possono vedere quali regole sono incluse in un progetto e quali invece sono state lasciate fuori. Per creare o modificare questo set di regole è necessario invece che gli utenti dispongano di permessi *Administer Quality Profiles and Gates.*

Il *Quality Gate* è il modo migliore di applicare una politica di qualità nell'organizzazione. Risponde alla domanda "posso consegnare il mio progetto di produzione oggi oppure no?". Per rispondere a questa domanda si definisce un insieme di condizioni booleane basate su soglie di misura rispetto alle quali vengono misurati i progetti, ad esempio nessun new blocker issue o code coverage superiore al 80%.

3.2.4 Parametri dell'analisi

I parametri per configurare l'analisi del progetto possono essere impostati in più punti. Ecco la gerarchia dei parametri:

- I parametri di analisi globali, definiti nell'interfaccia utente, si applicano a tutti i progetti.
- I parametri di analisi del progetto, definiti nell'interfaccia utente, sovrascrivono i parametri globali.
- I parametri di analisi del progetto, definiti in un file di configurazione di analisi di progetto o un file di configurazione dell'analizzatore, sovrascrivono quelli definiti nell'interfaccia utente
- Linea di comando dei parametri, definiti quando si lancia un'analisi, sovrascrivono i parametri di analisi di progetto.

Si noti che solo i parametri impostati tramite l'interfaccia utente sono memorizzati nel database.

3.2.4.1 Parametri obbligatori

Server

Parametro	Descrizione	Valore di Default
sonar.host.url	Server URL	http://localhost:9000

Configurazione del progetto

Parametro	Descrizione	Valore di Default
sonar.projectKey	La chiave del progetto è unica per ogni progetto	
sonar.sources	Percorso per la directory contenente il file di origine	

3.2.4.2 Parametri opzionali

Identità del progetto

Parametro	Descrizione	Valore di Default
sonar.projectName	Nome del progetto che verrà visualizzato nell'interfaccia web.	• Se non è ancora stato definito un nome, utilizzare la key del progetto • Se nel DB è già presente un tale nome, non sovrascrivere
sonar.projectVersion	La versione del progetto	Non fornito

Autenticazione

Parametro	Descrizione	Valore di Default
sonar.login	Il token di accesso o di autenticazione di un utente SonarQube con autorizzazione di eseguire un 'analisi	
sonar.password	La password associata al nome utente sonar.login. Questo deve essere lasciato vuoto se viene utilizzato un token di autenticazione.	

Configurazione del progetto

Parametro	Descrizione	Valore di Default
sonar.projectDescription	La descrizione del progetto	
sonar.language	Impostare il linguaggio del codice sorgente da analizzale. Consultare la pagina delle librerie dei plugin per ottenere tutti i linguaggi disponibili. Se non è impostato, verrà attivata un'analisi multilingua.	
sonar.projectDate	Assegnare una data all'analisi	Data corrente

3.3 Scanner

Analisi con SonarQube Scanner

Lo scanner SonarQube è consigliato come launcher predefinito per analizzare un progetto con SonarQube. Dopo aver scaricato lo scanner è possibile verificare l'avvenuta installazione aprendo la shell ed eseguendo il comando sonar-scanner.bat ottenendo questo tipo di output:

```
usage: sonar-scanner [options]

Options:
 -D,--define <arg>     Define property
 -h,--help             Display help information
 -v,--version          Display version information
 -X,--debug            Produce execution debug output
```

Per quanto riguardo l'uso dello scanner, bisogna creare un file di configurazione nella directory principale del progetto: sonar-project.properties

sonar-project.properties

```
# must be unique in a given SonarQube instance
sonar.projectKey=my:project
# this is the name and version displayed in the SonarQube UI. Was mandatory prior to SonarQube 6.1.
sonar.projectName=My project
sonar.projectVersion=1.0

# Path is relative to the sonar-project.properties file. Replace "\" by "/" on Windows.
# This property is optional if sonar.modules is set.
sonar.sources=.

# Encoding of the source code. Default is default system encoding
#sonar.sourceEncoding=UTF-8
```

Figura 9.Proprietà del progetto

Eseguire il seguente comando dalla directory di base del progetto per avviare l'analisi:

```
sonar-scanner
```

3.4 Plugin

Ci sono due opzioni per installare i plugin in SonarQube:

- Marketplace - Installa i plugin automaticamente, dall'interfaccia utente di SonarQube.
- Manual Installation – Questo metodo viene utilizzato se l'istanza SonarQube non ha accesso a Internet.

3.4.1 Marketplace

Se si è connessi a Internet e si ha accesso alla piattaforma con l'autorizzazione globale " *Administer System*", è possibile accedere a *Administration* > *Marketplace (*Figura 10*)*.

- Trova il plugin che si desidera installare.
- Fare clic su *Install* e attendere che il download sia terminato.

Una volta completato il download, sarà disponibile un pulsante "*Restart*" per riavviare l'istanza.

3.4.2 Manual Installation

Per installare manualmente un plugin, vedere *SonarSource Editions*. Nella pagina dedicata al plugin che si desidera installare (ad esempio: per Python: SonarPython), fare clic sul collegamento "*Download*" della versione compatibile con la versione di SonarQube. Inserire il jar scaricato in SONARQUBE_HOME / extensions / plugins, rimuovendo eventuali versioni precedenti degli stessi plugin. Una volta terminato, sarà necessario riavviare il server SonarQube.

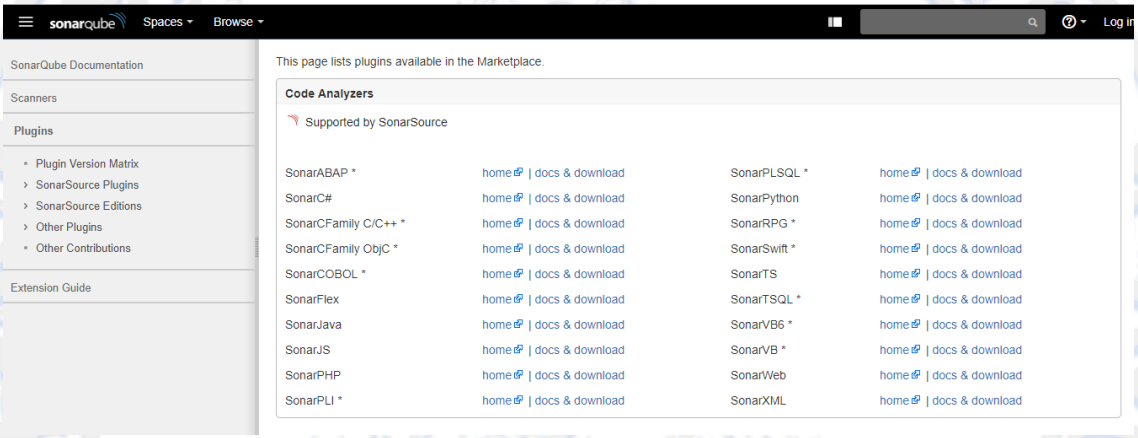

Figura 10. Plugin Library

4 Caso di studio

Analizziamo adesso un progetto e soffermiamoci sui risultati ottenuti. Dall'analisi della prima versione del progetto considerato, un semplice codice che data una parola ingresso restituisce tutte le parole palindrome ottenute dalle lettere presenti nella stessa. Abbiamo ottenuto il seguente responso:

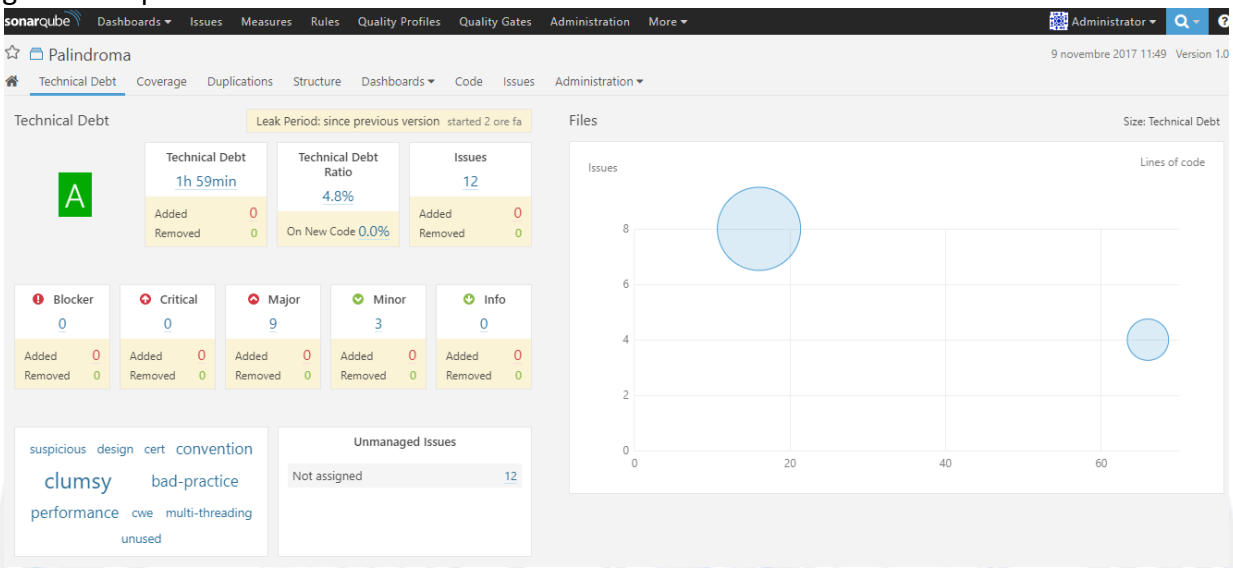

Figura 11.Technical Debt del progetto Palindroma Prima Versione

Technical Debt: il debito tecnico riflette il costo implicito di un lavoro aggiuntivo causato da una scelta di una soluzione facile invece di utilizzare un approccio migliore che richiederebbe più tempo. Il debito tecnico può essere confrontato con l'indebitamento monetario, se il debito tecnico non viene rimborsato può accumulare "interesse" rendendo più difficile applicare le successive modifiche. Tale debito non trattato aumenta l'entropia del software. Le cause comuni del debito tecnico sono la combinazione:

- Pressioni imprenditoriali, l'impresa vuole ottenere una prima versione del prodotto prima che tutti i cambiamenti necessari siano completi. Ciò crea un indebitamento tecnico che comprende quelle modifiche non completate.
- Mancanza di una suite di test.
- Mancanza di documentazione, il codice viene creato senza necessaria documentazione di supporto. Il lavoro per creare qualsiasi tipo di documentazione di supporto rappresenta un debito che deve essere pagato.
- Mancanza di collaborazione, la conoscenza non è condivisa in tutta l'organizzazione e di conseguenza l'efficienza dell'azienda soffre.
- Mancanza di conoscenza, lo sviluppatore non sa scrivere del codice "elegante".
- Mancanza di allineamento agli standard.
- Modifiche dell'ultimo minuto.
- Sviluppo in parallelo, più cambiamenti vengono fatti in "isolamento" più debito è accumulato.

Technical Debt Ratio: il rapporto di debito tecnico è una metrica che fornisce appunto un rapporto tra il debito tecnico e lo sforzo che ci sarebbe voluto per riscrivere tutto il codice sorgente da zero. Quest'ultimo è stimato in base al numero di linee di codice o alla complessità complessiva, il che

significa che il valore di questa metrica dipende dalla dimensione del progetto. Il progetto riceve un voto che va da A (grado migliore) ad E (grado peggiore).

Issue: SonarQube mentre esegue l'analisi solleva un problema ogni volta che un pezzo di codice viola una regola di codifica. L'insieme di regole di codifiche è definito attraverso il profilo di qualità associato al progetto. Ad ogni problema viene associato una delle cinque severità:

- BLOCKER: Bug con alta probabilità di influenzare il comportamento dell'applicazione nella produzione (perdita di memoria, connessione JDBC non chiusa, ecc..). Il codice deve essere immediatamente risolto.
- CRITICAL: Bug con minima probabilità di influenzare il comportamento dell'applicazione nella produzione o semplicemente un problema che rappresenta un difetto di protezione. Il codice deve essere immediatamente riveduto.
- MAJOR: Difetto di qualità che può influenzare notevolmente la produttività dello sviluppatore (pezzo di codice scoperto, blocchi duplicati, parametri non utilizzati).
- MINOR: Difetto di qualità che può avere un leggero impatto sulla produttività degli sviluppatori (le linee di codice non devono essere troppo lunghe, le dichiarazioni "switch" dovrebbero avere almeno tre casi, ecc..).
- INFO: Né un bug né un difetto di qualità ma semplicemente una scoperta.

Confirm, False Positive, Won't Fix, Change Severity rientrano in una nuova categoria di Issue, che presuppone una prima revisione di un problema per verificarne la validità.

- CONFIRM: Problema confermato, si sta dicendo fondamentalmente "Si, questo è un problema".
- FALSE POSITIVE: Guardando il problema nel contesto emerge che in realtà non risulta essere un reale problema, lo si contrassegna come falso positivo e si prosegue.
- WON'T FIX: Rappresenta il debito tecnico accettato, quindi non si corregge.
- CHANGE SEVERITY: Questa è una via di mezzo tra le prime due opzioni, ovvero è un problema ma non uno così grave come le regole di gravità predefinite lo catalogano. O forse è anche peggio.
  Dopo l'analisi della prima versione del progetto preso in considerazione gli issue risultano questi sotto illustrati:

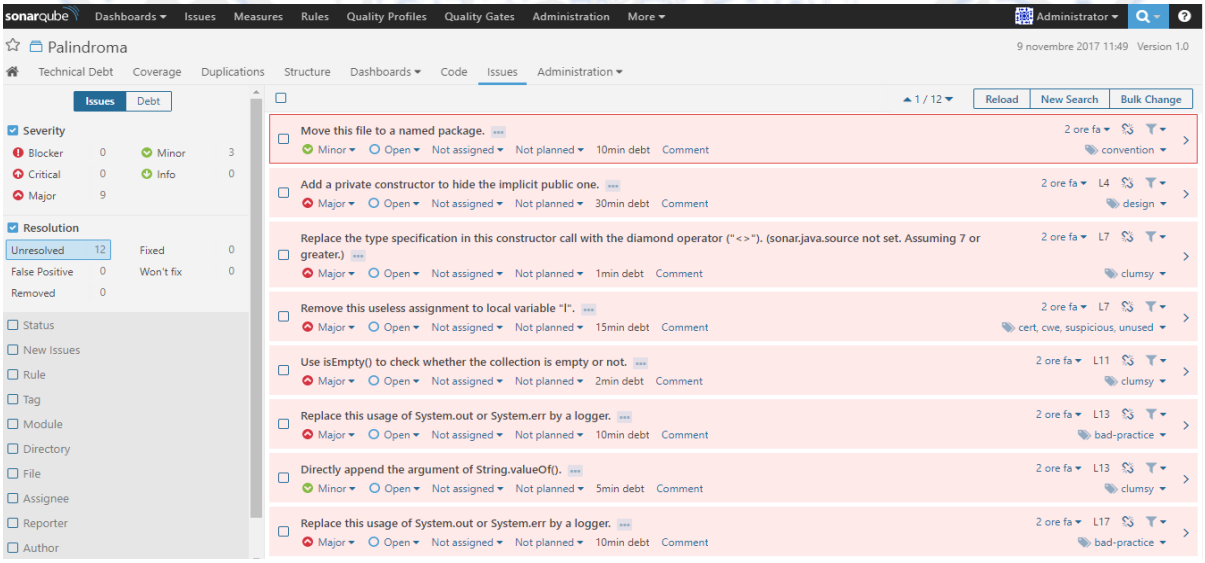

Figura 12. Issue del progetto Palindroma Prima Versione parte1

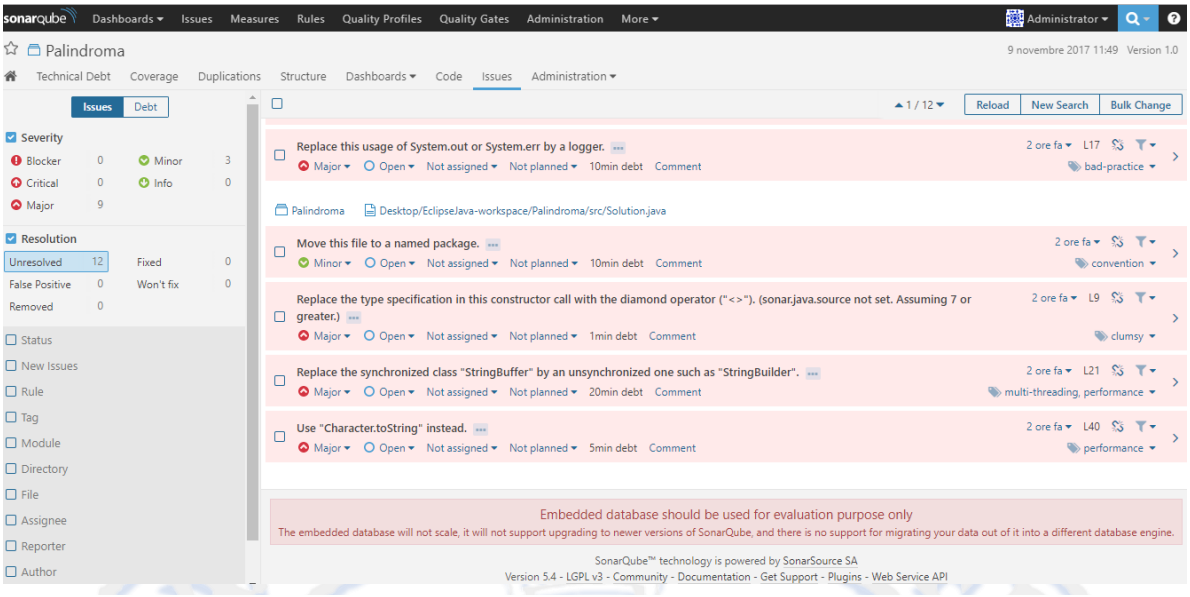

Figura 13. Issue del progetto Palindroma Prima Versione parte2

Per quanto riguarda lo studio dei duplicati, intesi come duplicati di file, linee di codice o di blocchi di codice dalla prima analisi è emerso questo, vedi Figura 14*:*

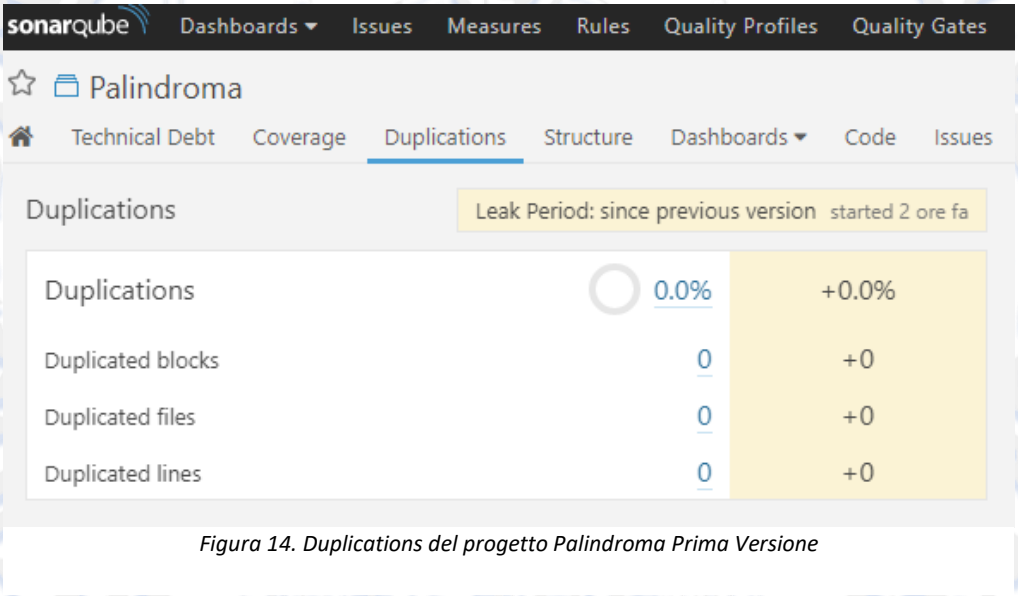

Figura 14. Duplications del progetto Palindroma Prima Versione

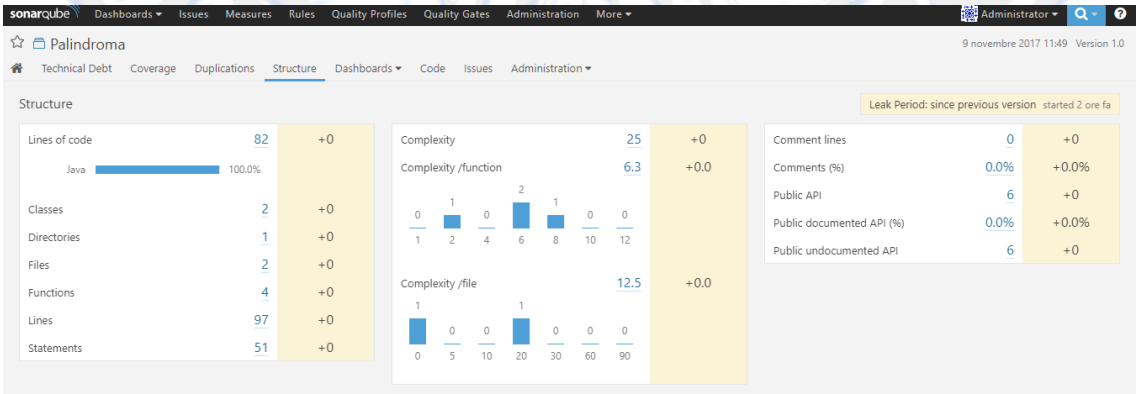

Figura 15. Structure del progetto Palindroma Prima Versione

La complessità, vedi Figura 15, è calcolata in base al numero di percorsi linearmente indipendenti attraverso il codice. Ogni volta che il flusso di una funzione si divide, il contatore di complessità viene incrementato di uno. Ogni funzione ha complessità minima di uno. Questo calcolo varia leggermente in base al linguaggio adottato.

Language	**Notes**
C/C++	La complessità viene incrementata da definizioni di funzioni, while, do while, for, switch, case, || operator, catch, continue, break, goto
Java	La complessità viene incrementata da if, for, while, case, catch, throw, return, &&, ||, ?.
Cobol	I seguenti comandi aumentano la complessità: ALSO, ALTER, AND, DEPENDING, END_OF_PAGE, ENTRY, EOP, EXCEPTION, EXIT, GOBACK, CONTINUE, IF, INVALID, OR, OVERFLOW, SIZE, STOP, TIMES, UNTIL, USE, VARYING, WHEN, EXEC CICS HANDLE, EXEC CICS LINK, EXEC CICS XCTL, EXEC CICS RETURN

Dall'analisi emerge anche lo studio basato su:

- Comment Lines: linee che hanno lo scopo di descrivere le caratteristiche funzionali.
- Comment (%): density of comment lines =Comment Lines/ (Lines of code+ Comment Lines) *100.
- Public API: Application programming interface.
- Public documented API (%): density of public documented API = (Public API - Public undocumented API) / Public API * 100.
- Public undocumented API: public API senza intestazione dei commenti.

Dopo aver effettuato l'analisi sulla prima versione del progetto andiamo ad apportare le modifiche necessarie in base ai risultati ottenuti. Rianalizzando subito dopo il progetto si è ottenuto il risultato illustrato nella Figura 16. Appare molto evidente che con le modifiche apportate il Technical Debt si sia ridotto di 1h, il Technical Debt Ratio sia passato dal 4,8% al 1,0%, gli Issue altrettanto si sono ridotti a 3.

In questa seconda versione del progetto gli Issue riscontrati sono tre con severità associata MAJOR(Figura 17).

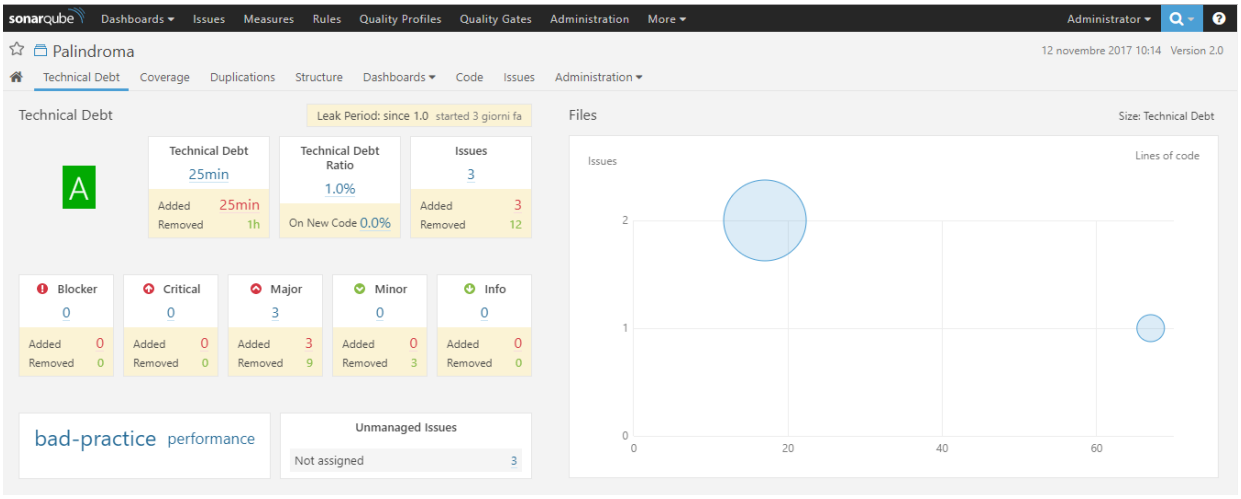

Figura 16.Technical Debt del progetto Palindroma Seconda Versione

Ovviamente in un ulteriore revisione del progetto risulta possibile eliminare totalmente gli issue del progetto. SonarQube rende estremamente semplice sia individuare dove l'errore si manifesta, sia risolvere il problema grazie alla spiegazione e alla rispettiva possibile risoluzione di tale

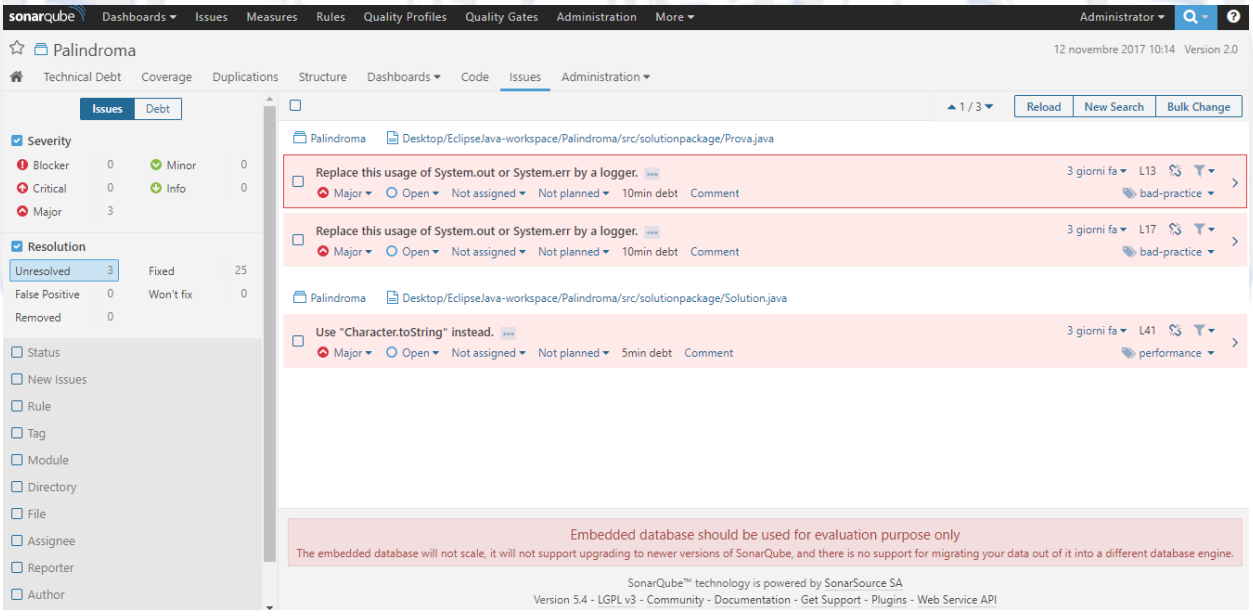

Figura 17.Issue del progetto Palindroma Seconda Versione

problematica. L'analisi dei duplicati è rimasta invariata in questa seconda versione come si evince dalla Figura 19. Il conteggio delle linee di codice ha subito una piccolissima variazione così come il numero di funzioni del progetto, mentre il numero di classi e di file è rimasto inalterato. La Figura 18 mostra i cambiamenti riscontrati riguardo la complessità e la documentazione del caso di studio sotto esame. Un aspetto molto interessante di questo strumento di analisi è che dispone di un'opzione di confronto fra progetti o versioni diverse del medesimo. Nella Figura 20 viene mostrato il confronto tra le due versioni del progetto.

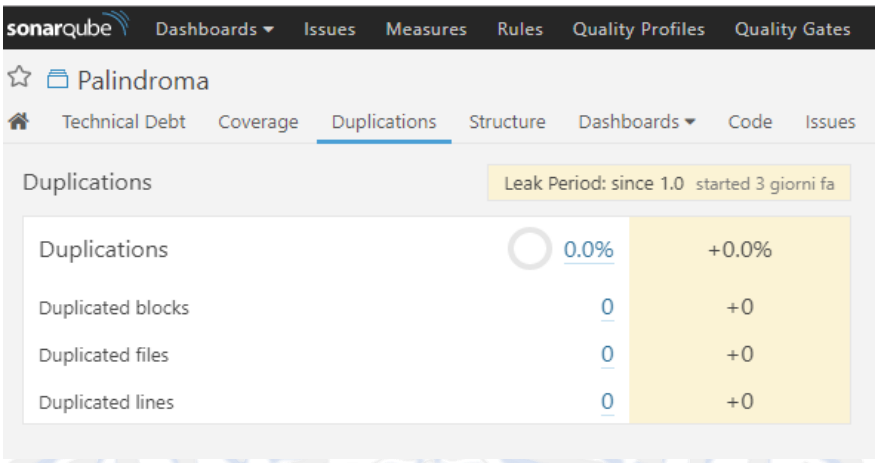

Figura 19. Duplications Palindroma Seconda Versione

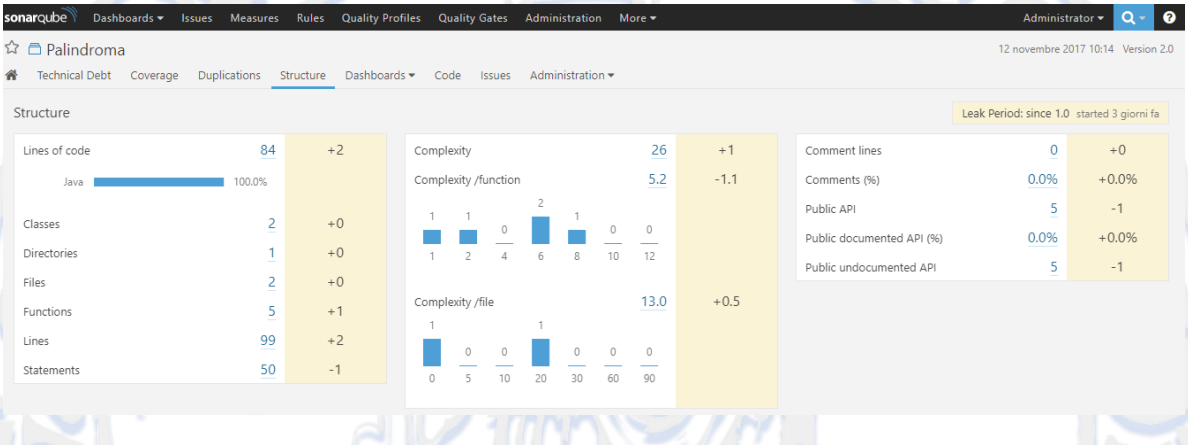

Figura 18. Structure Palindroma Seconda Versione

Nel caso di studio considerato in precedenza siamo partiti da un progetto funzionante dove però vi erano delle problematiche risolte successivamente nella seconda versione. Ora invece ragioniamo in modo inverso prendiamo un altro progetto lo poniamo sotto l'analisi dello strumento e nella seconda versione cercheremo di "sporcarlo", mostrando così le dovute differenze tra le due versioni.

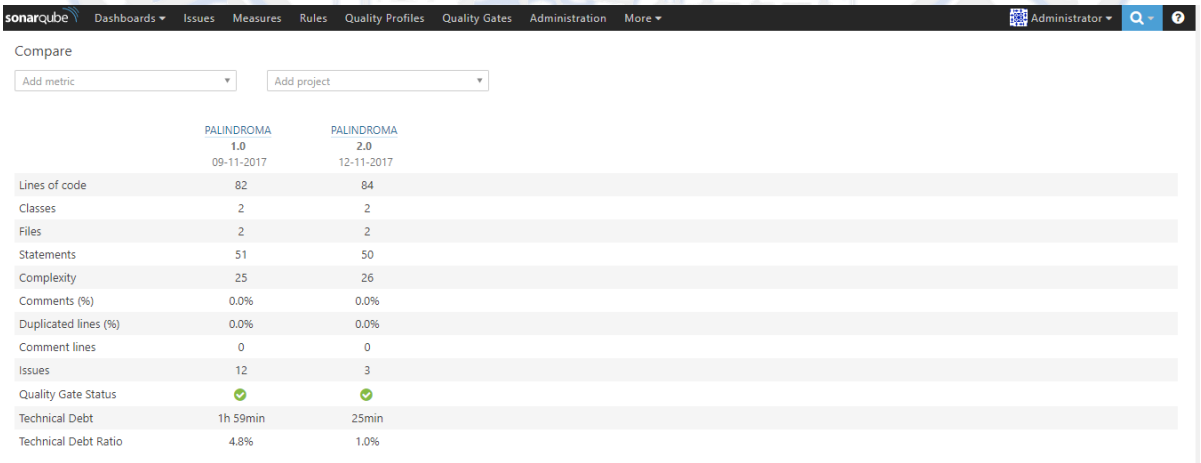

Figura 20. Compare Palindroma

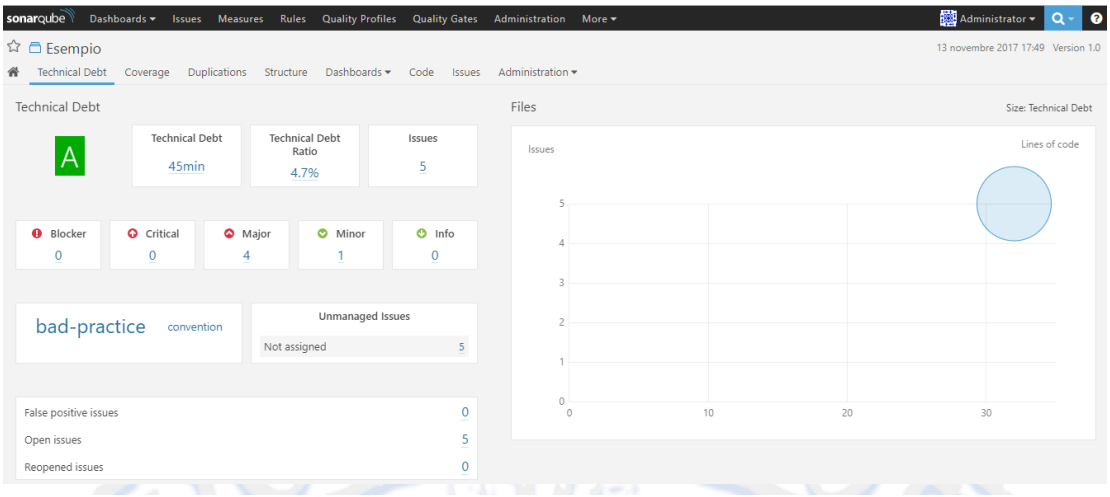

Figura 21. Technical Debt progetto Esempio Prima Versione

La prima analisi di questo progetto, che non è altro che un semplice algoritmo di ordinamento, mostra che non sono presenti "pericolose" problematiche ma piuttosto "cattive abitudini". Il Technical Debt è di 45 minuti, ma cosa importante SonarQube ha giudicato in modo positivo il codice restituendo A. Per quanto riguarda l'analisi dei duplicati essa ha rivelato il responso in Figura 22.

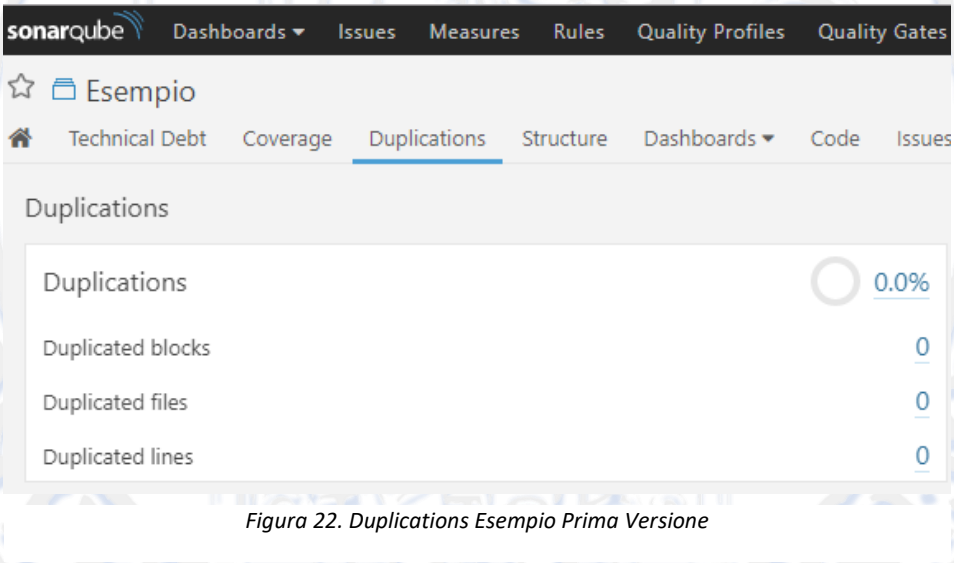

Figura 22. Duplications Esempio Prima Versione

Gli Issue riscontrati, Figura 23, sono cinque tra cui uno soltanto risulta avere severità Minor, i restanti invece Major. Dalla figura sotto illustrata è chiaramente visibile, in basso a sinistra di ogni Issue, il debito tecnico di ciascuno, ed inoltre cliccando sull'icona contenente tre puntini, la piattaforma mostrerà la natura del problema e la possibile soluzione di esso.

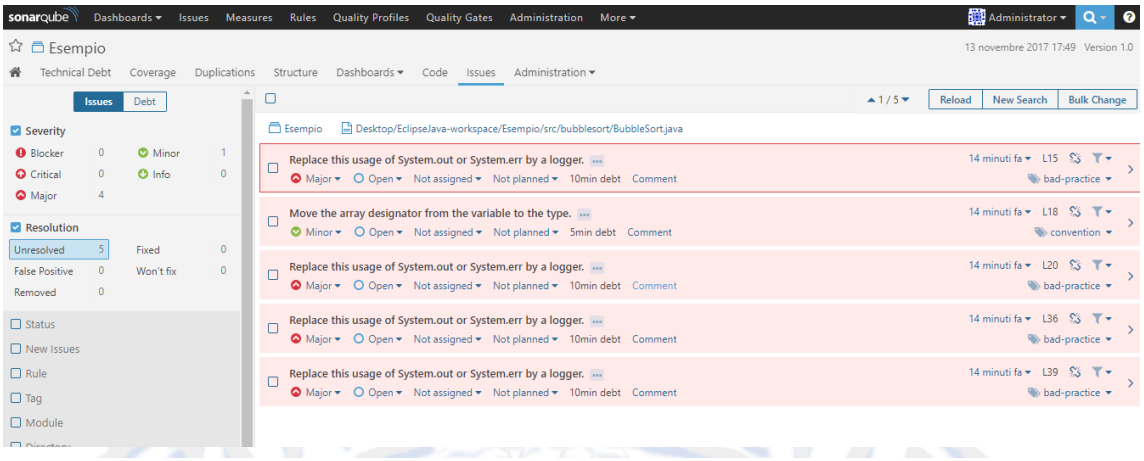

Figura 23. Issue Esempio Prima Versione

Dopo il risultato di questa prima analisi procediamo con la contaminazione del codice (Figura 25, Figura 26) Ovviamente dobbiamo sempre assicurarci che il codice risponda alle esigenze dichiarate.

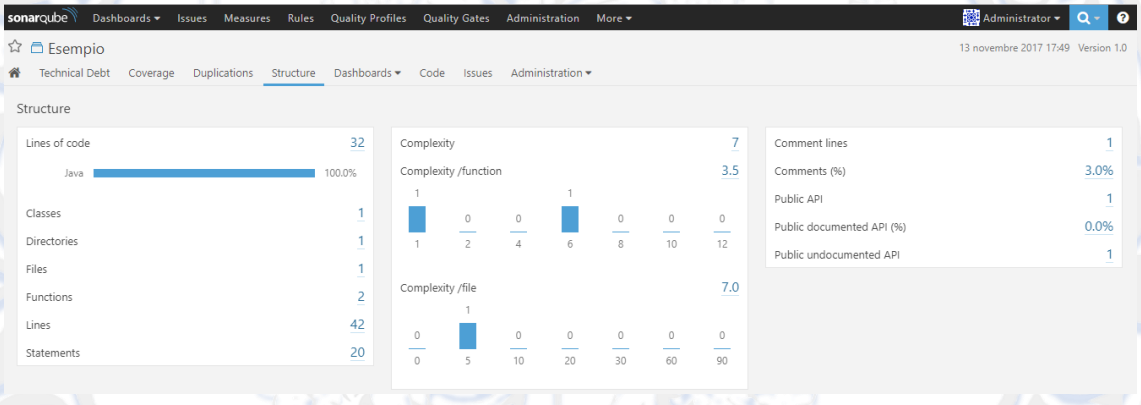

Figura 24. Structure Esempio Prima Versione

```
public static void main(String []args) {
        int n;
        int c;
        int  d;
        int swap;
```

Figura 26.A)Codice sorgente Prima Versione

```
public static void main(String []args) {
        int n, c, d, swap;
```

Figura 25.A)Codice sorgente Seconda Versione

```
class BubbleSort {
        private BubbleSort(){

        }
```

```
public class BubbleSort {
```

Figura 30.B) Codice sorgente Prima Versione

Figura 29.B) Codice sorgente Seconda Versione

```
package bubblesort;
import java.util.Scanner;
```

```
import java.util.Scanner;
```

Figura 28.C) Codice sorgente Prima Versione

Figura 27.C) Codice sorgente Seconda Versione

In seguito a queste variazioni nel codice sorgente rianalizziamo il progetto e attendiamo il responso della piattaforma. Poiché stiamo ragionando in modo inverso, il nostro intento è di ottenere un giudizio negativo o quanto meno differente dalla prima versione.

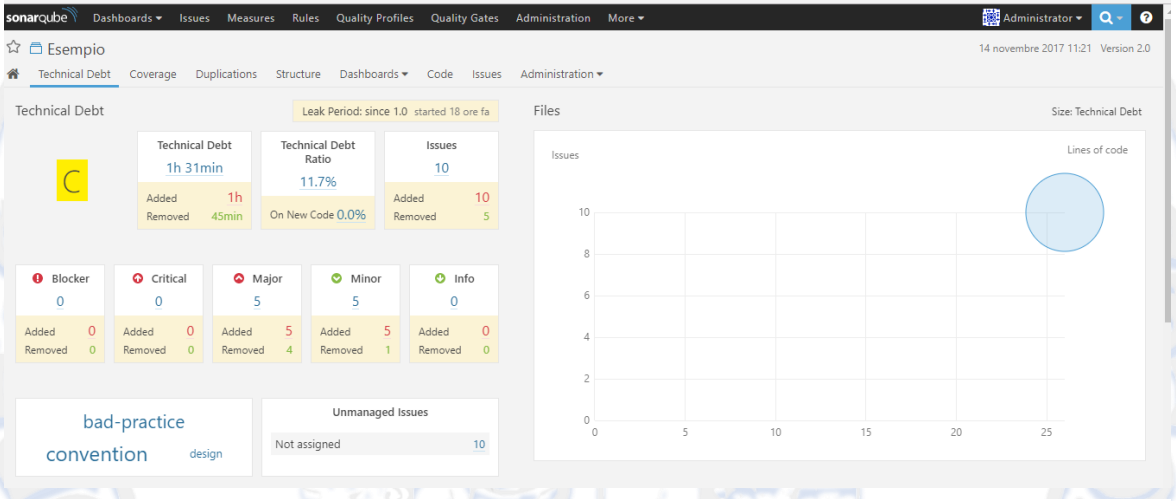

Figura 31. Technical Debt Esempio Seconda Versione

La Figura 31 mostra ciò che ci desideravamo, ovvero il responso è declassato. Inoltre il Technical Debt è notevolmente aumentato così come gli Issue. SonarQube mostra anche graficamente come il Technical Debt sia variato, Figura 33. Aprendo la finestra degli Issue riscontriamo che secondo le modifiche apportate, tra la prima versione Figura 26 e la seconda versione Figura 25, la piattaforma porta alla luce, Figura 35, proprio come primo problema la mancata assegnazione di un nome al package dentro il quale il file è situato. La trasformazione illustrata in Figura 30 e Figura 29 solleva una problematica, ovvero l'assenza di un costruttore privato che nasconda quello implicito pubblico, che ha come Technical Debt un tempo di 30 minuti. Questa modifica apportata al codice sorgente risulta essere davvero svantaggiosa. Visionando gli Issue notiamo che anche le modifiche, Figura 28 e Figura 27, hanno sollevato errori, in questo caso non particolarmente rilevanti ma ovviamente sommati ai restanti fanno sì che il Technical Debt sia notevolmente aumentato. Per quanto riguarda l'analisi dei duplicati siccome non sono state inseriti appositamente né duplicati di codice, né di blocchi di codice o file il risultato è rimasto invariato dalla prima versione, Figura 36.

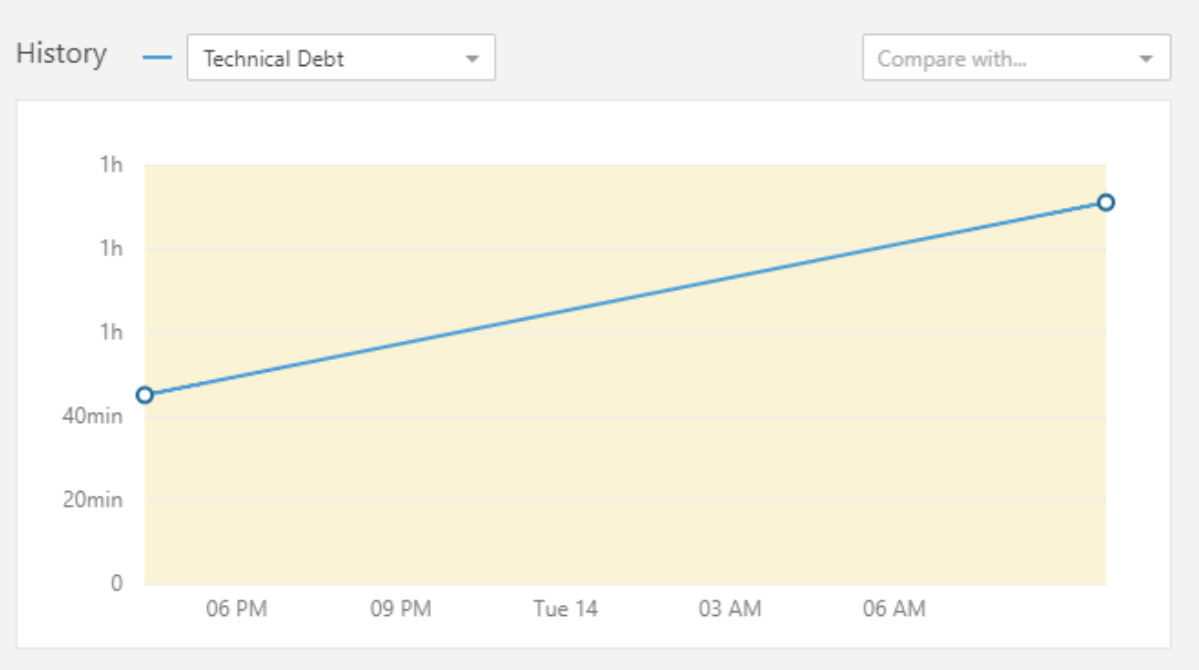

Figura 33.History Technical Debt

La complessità risulta essere diminuita di un'unità, ciò comporta una variazione anche per quanto riguarda il campo "*Complexity/function*" e "*Complexity/file*".

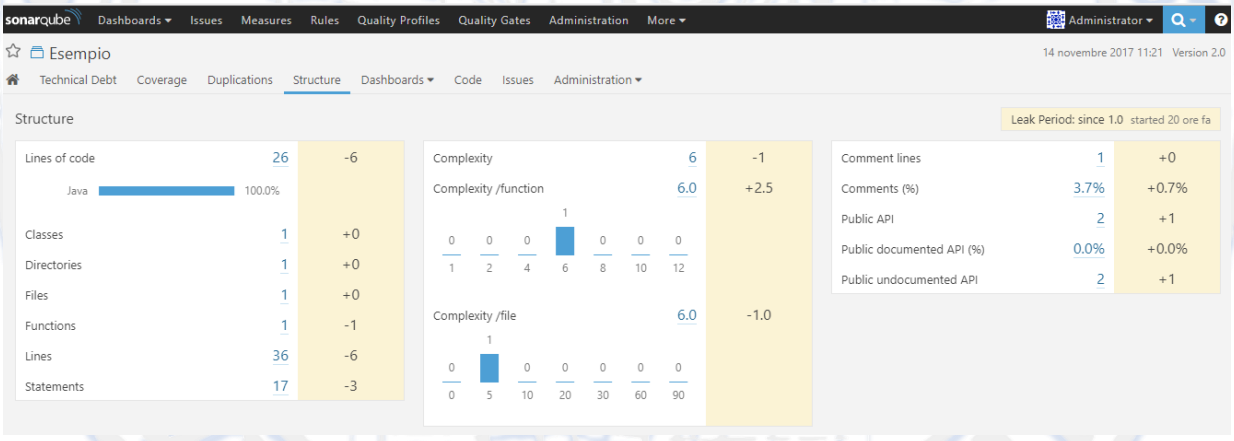

Figura 32.Structure Esempio Seconda Versione

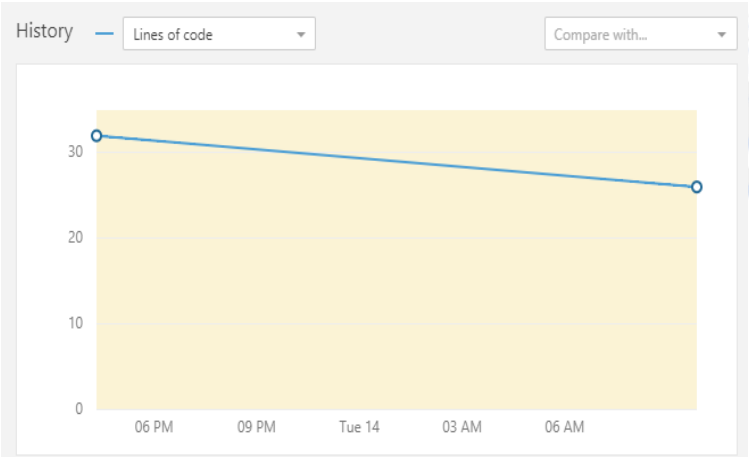

Figura 34. History Lines of Code Esempio

Così come per il Technical Debt anche per le Lines of code è possibile visualizzare l'andamento assunto fra le due versioni del medesimo progetto. In realtà basta variare parametro all'interno del campo History (Duplications, Comments, ecc..) e la piattaforma risponde con l'andamento associato al parametro selezionato.

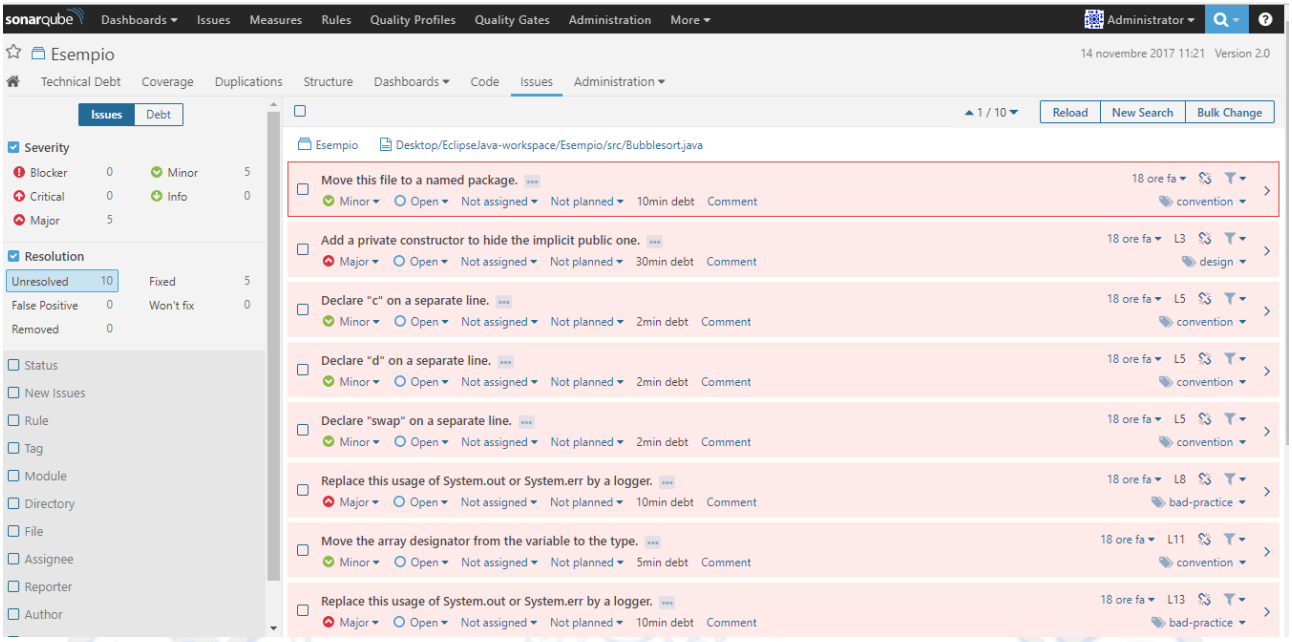

Figura 35.Issue Esempio Seconda Versione Parte1

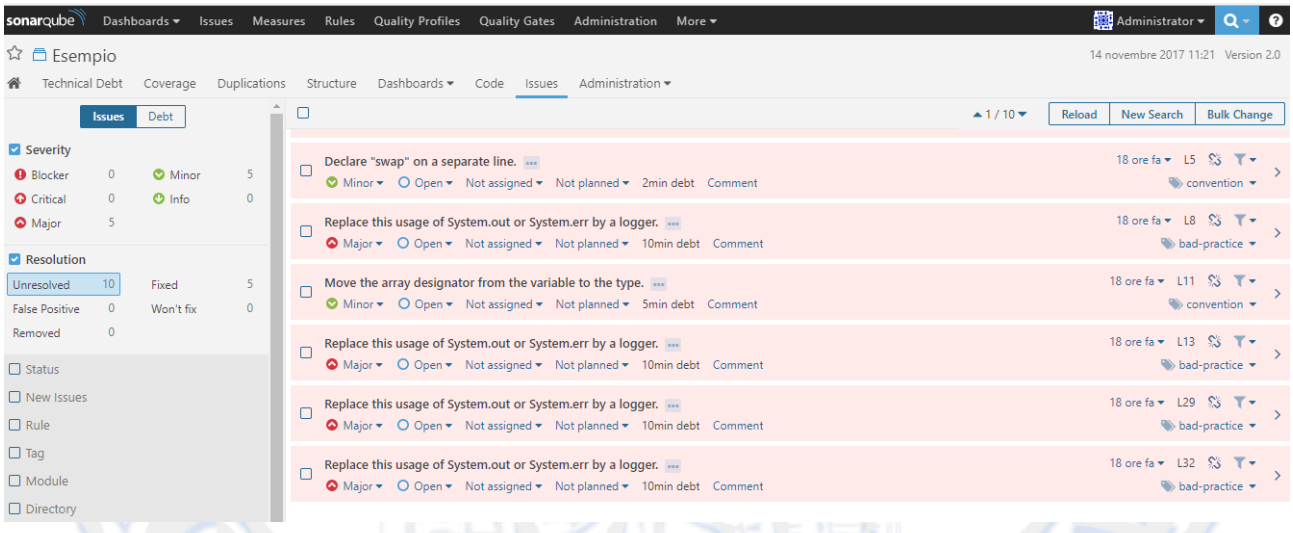

Figura 37. Issue Esempio Seconda Versione Parte2

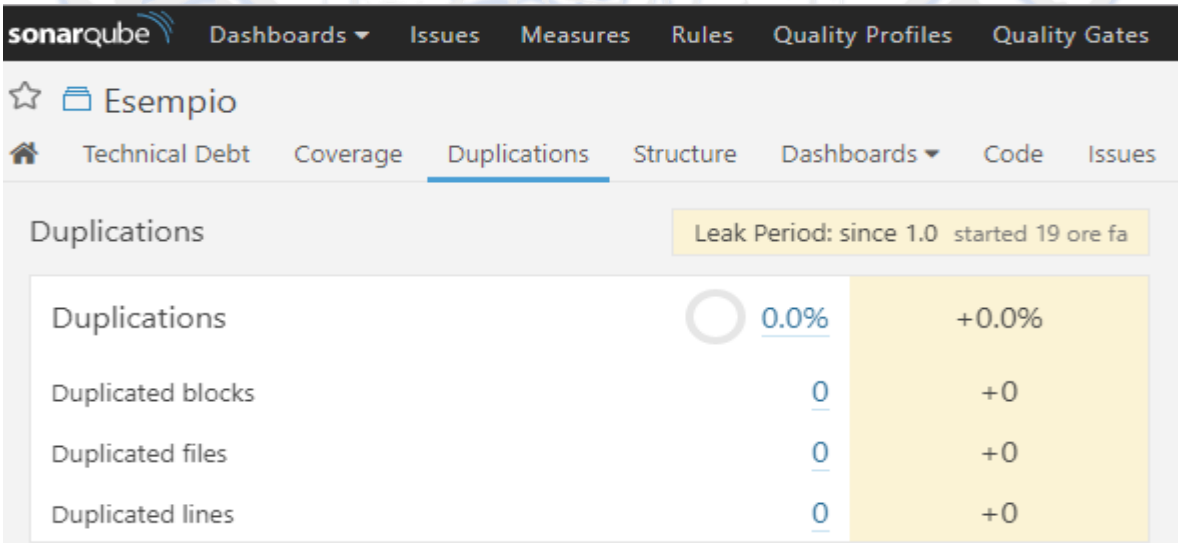

Figura 36. Duplications Esempio Seconda Versione

Per vedere esplicitamente le notevoli differenze fra le due versioni procediamo come nel progetto studiato in precedenza, ovvero utilizziamo la funzionalità di comparazione di SonarQube, Figura 38.

Figura 38. Compare Esempio

5 Conclusioni

La qualità del software è migliorata moltissimo negli ultimi quindici anni, grazie all'introduzione di nuove tecniche e tecnologie, e nell'industria del software è aumentata la consapevolezza dell'importanza della gestione della qualità del codice sorgente. L'utilizzo di strumenti come SonarQube che analizzano automaticamente il codice sorgente, ha permesso ai progettisti di creare in maniera sempre più rapida dei prodotti pronti all'uso, evitando continue fasi di refactoring come in passato, causate dalla scarsa qualità del progetto. Di seguito, i motivi che hanno portato i progettisti a servirsi di tali strumenti:

- Rendere maggiormente industrializzato un processo che è ancora molto artigianale
- Evitare il fallimento di progetti di grandi dimensioni
- Minimizzare i costi di produzione e manutenzione per consentire un facile intervento da parte di persone diverse
- Riciclare il software per permettere il riutilizzo di un lavoro già testato

Purtroppo ci sono ancora molte società che non fanno uso di misurazioni sistematiche del software per valutarne la qualità, essendo i loro prodotti mal definiti e controllati, ma il continuo uso di tali prodotti open source promette uno sviluppo positivo in questo senso. Nonostante ciò il concetto di qualità del software è ancora molto complesso e non basta seguire standard e metriche per ottenere un prodotto ottimale. I manager dello sviluppo puntano a infondere una "cultura della qualità" incoraggiando il team a lavorare nel migliore dei modi, cercando di non affidare del tutto il miglioramento del progetto a questi software. Un altro punto da mettere in risalto è quello dell'open source. I programmatori che lavorano per l'open source incoraggiano lo studio, l'implementazione e la modifica dei programmi da loro generati. Questo perché è importante condividere qualsiasi tipo di conoscenza in una società dove il profitto è diventato la nuova filosofia di vita. Il semplice ed immediato utilizzo di tali strumenti ha prodotto tre benefici:

- Hanno incoraggiato gli sviluppatori nell'adozione di tecniche di analisi sempre migliori
- Hanno consentito l'aumento delle conoscenze del codice sorgente da parte del team di sviluppo
- Hanno ridotto il costo di manutenzione

Quindi sia che si tratti di soddisfare le richieste di un cliente, o gli standard imposti da modelli generali, avere un codice sorgente di più elevata qualità può aiutare le imprese a soddisfare i requisiti di conformità, e aiuta i progettisti a lavorare in maniera più efficiente e motivata.

6 Bibliografia

[1] The Approach to Finding Errors in Program Code Based on Static Analysis Methodology, Alexander S. Novikov, Alexey N. Ivutin, Anna G. Troshina, Sergey N. Vasiliev, 2017.
[2] Checkstyle, http://checkstyle.sourceforge.net/.
[3] Stan4j, http://stan4j.com/.
[4] Sonar, http://www.sonarsource.org/.