Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

DASHBOARD PER MONITORARE LO SVILUPPO SOFTWARE: CRITICITÀ, VANTAGGI

Relatore:
Chiar.mo Prof.
PAOLO CIANCARINI

Presentata da:
ANDREA LORETTI

II Sessione
Anno Accademico 2021/2022

5 parole chiave per caratterizzare il contenuto della dissertazione:

DASHBOARD,

SOFTWARE ENGINEERING,

COMPUTER SCIENCE,

AGILE,

LOGGER

A mia Madre & a mio Padre
che mi hanno sempre incoraggiato
a seguire le mie passioni

Abstract

Lo scopo di questa tesi è studiare l'uso di "cruscotti" (in inglese *Dashboard*) per il monitoraggio dello sviluppo software, approfondendo i metodi di raccolta delle metriche e come esse vengono gestite.
Nello specifico, analizzerò l'ambiente di sviluppo *Compositional Agile System* (CAS), sviluppando un nuovo plugin per l'IDE Microsoft Visual Studio Code, che è open source.

Verranno proposti nuovi metodi di implementazione e utilizzo delle Dashboard e possibili miglioramenti dell'ambiente CAS.

Indice

Capitolo 0

INTRODUZIONE

0.1 Perché misurare la produttività di uno sviluppatore?

Una possibile motivazione per misurare la produttività degli sviluppatori è aiutare sia gli individui sia i team di sviluppo ad analizzare il proprio rendimento. Un'altra motivazione riguarda la possibilità che un'azienda voglia raccogliere dati di produttività-
I motivi per cui un'azienda potrebbe voler misurare la produttività dei propri dipendenti sono molteplici.
Le motivazioni includono l'emergere, in azienda, di nuove esigenze: la valutazione dell'efficacia di diversi strumenti o pratiche, l'esecuzione di confronti per un intervento inteso a migliorare la produttività ed evidenziare inefficienze ove la produttività possa essere migliorata.
Sebbene ciascuno di questi scenari abbia bisogno di misurare la produttività, le metriche, le aggregazioni e le segnalazioni sono diverse.
Ad esempio, identificare individui più o meno performanti significa aggregare una metrica a livello individuale, mentre eseguire un confronto tra team significherebbe aggregare gruppi di sviluppatori.

Va comunque tenuto presente che il tipo di metriche utilizzate in questi scenari è differente e quindi non possono essere trattate tutte alla stessa maniera.
Ci sono diversi stakeholders che potrebbero essere interessati a misurare la produttività con finalità diverse.
Nel caso in cui l'obiettivo fosse identificare performances basse o far emergere tendenze globali, gli stakeholder interessati cercheranno metriche che misurino il completamento delle tasks.

Se, invece, l'obiettivo fosse eseguire un confronto per un intervento specifico o evidenziare le inefficienze interne di un processo, le metriche di produttività utilizzate misureranno le attività secondarie che affrontano gli obiettivi dell'intervento o del processo in questione.
Inoltre, bisogna prestare attenzione al fatto che ciò che è perseguibile per un individuo non è detto che sia perseguibile per un team.
In ogni caso, per riuscire ad aggregare le misurazioni raccolte e riconoscere dei pattern vengono spesso utilizzate le dashboard.

0.2 Lavori precedenti

Il mio lavoro di tesi è basato su tre precedenti lavori di tesi di laurea triennale da cui ho tratto diverse informazioni e dati utili.
In particolare:

- Dalla tesi "Progettazione di una dashboard per sviluppatori agili" [13] di Salvatore Perri ho tratto gli elementi della Dashboard di CAS.
- Dalla tesi "Compositional agile system: un ambiente di "Extreme Development"" [15] di Stefano Propato ho ottenuto la lista di microservizi CAS , che avevo usato come studente del corso di Ingegneria del Software.
- Dalla tesi "La misurazione del teamwork nei progetti agili" [17] di Andrea Schinoppi ho estratto alcuni dati statistici inerenti l'utilizzo di CAS da parte dei team di studenti del mio corso.

0.3 Struttura della tesi

L'elaborato è diviso in quattro capitoli, oltre al capitolo introduttivo:

- Il primo capitolo illustra l'ambiente CAS [1].
- Il secondo capitolo discute i vantaggi e gli svantaggi nell'utilizzo di dashboard nell'ingegneria del software [2].
- Il terzo capitolo analizza le dashboard implementate in CAS [3].
- Il quarto e ultimo capitolo presenta le conclusioni e i possibili lavori futuri per CAS [4].

Capitolo 1

Compositional Agile System

In questo capitolo si parlerà di CAS (Compositional Agile System): un ambiente di sviluppo totalmente *open source* (OS) ideato per utilizzare un approccio agile[11] seguendo iAgile, un modello di processo derivante da Scrum, introdotto appositamente per lo sviluppo di sistemi critici.

L'implementazione di CAS consiste nel fornire un ambiente di sviluppo autonomo realizzabile su un server condiviso privato o in un cloud ibrido, mantenendo completo controllo su dati e codice di sviluppo in esso contenuti.

La struttura di CAS è composta da:

- CAS Client, l'ambiente lato sviluppatore;
- CAS Server, l'insieme dei servizi condivisi.

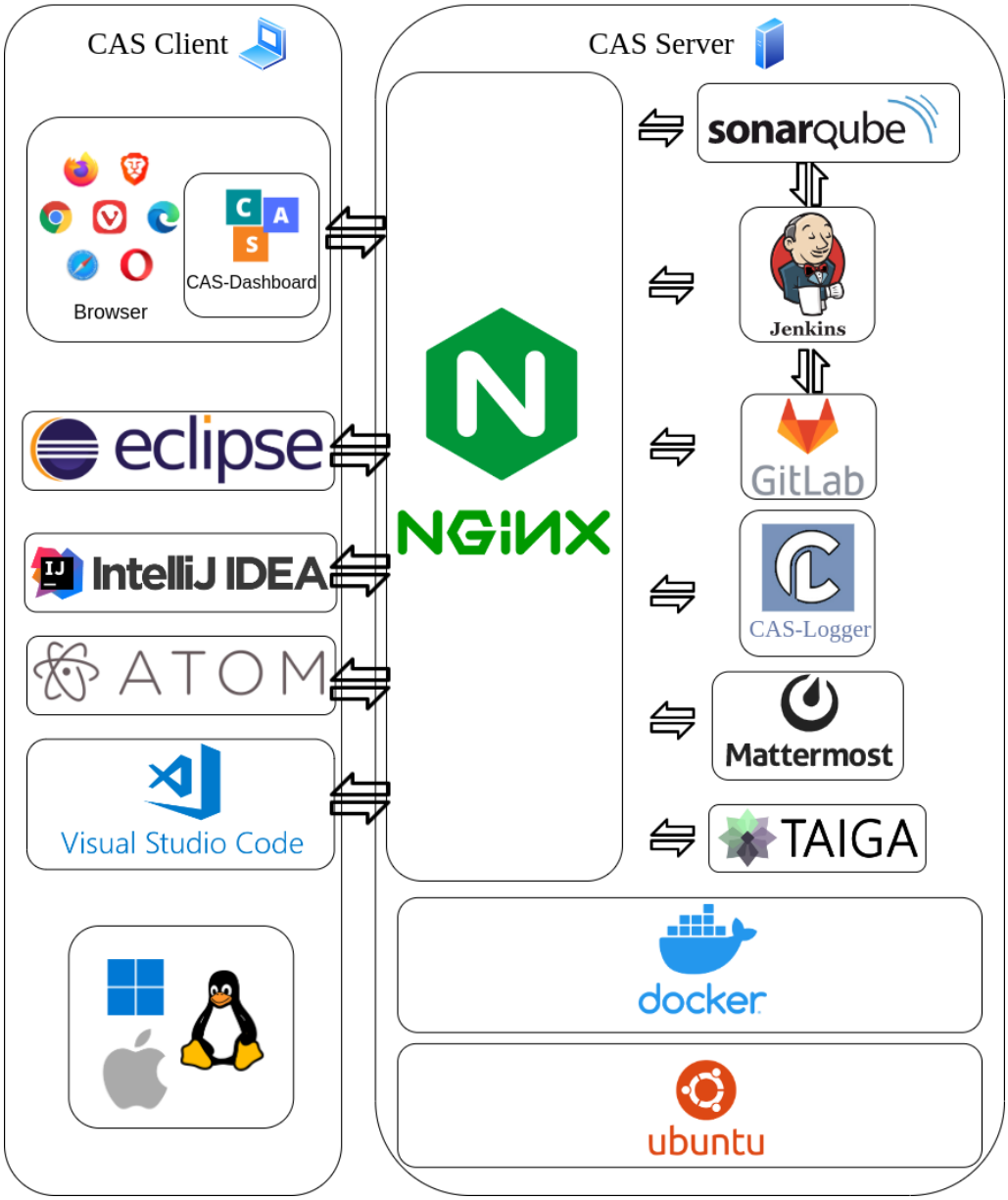


Figura 1.1: Struttura dei servizi CAS

1.1 CAS Client

Il CAS Client si istalla lato sviluppatore assieme ad un IDE a scelta e comprende due parti distinte:

- Un IDE integrato da un plug-in utile nel monitorare l'attività di sviluppo registrandone le misurazioni che successivamente verranno inviate al servizio Logger sul CAS Server.

  Attualmente esistono quattro IDE compatibili con il servizio Logger di CAS: Eclipse, IntelliJ, Atom, Visual Studio Code (quest'ultimo sviluppato per questa tesi).

- Un Browser capace di interfacciarsi con il logger sul server CAS attraverso la Dashboard, permettendo di recuperare e visualizzare tutte le misurazioni presenti sul server.

Di seguito si propone una breve descrizione di ogni IDE compatibile attualmente con CAS.

1.1.1 Eclipse-Logger

È reperibile presso `https://gitlab.com/Siber93/cas-eclipse-plugin`.
Una volta scaricato, tramite il menu Help di Eclipse, selezionare la voce Install New Software e successivamente indicare il file .jar appena scaricato.

1.1.2 IntelliJ-Logger

È sufficiente reperire il file .jar nel repository `https://gitlab.com/fulvio1993/logger-intellij` ed installarlo come qualsiasi altro plugin di IntelliJ.

1.1.3 Atom-Logger

È sufficiente cercarlo tra i pacchetti direttamente dentro l'IDE `https://atom.io/packages/`.
Oppure attraverso `https://github.com/elPeroN/atom-logger`

1.1.4 VScode-Logger

VScode-Logger è stato realizzato effetuando un porting dell'estensione Atom-Logger da Atom a Visual Studio Code.
Durante il processo di sviluppo, il cambio di IDE ha reso necessario rivedere tutte le API precedentemente create, impiegando librerie più efficienti e moderne.
Sono state risolte le problematiche di compatibilità con MacOS e sono stati eseguiti alcuni bug fix in seguito alle segnalazioni degli utenti nel corso dell'anno accademico 2021/2022.
Tutte le statistiche e le informazioni dell'ultima settimana saranno mostrate in una Dashboard minimale che presenta 3 grafici a ciambella.
Le funzionalità sono le medesime:

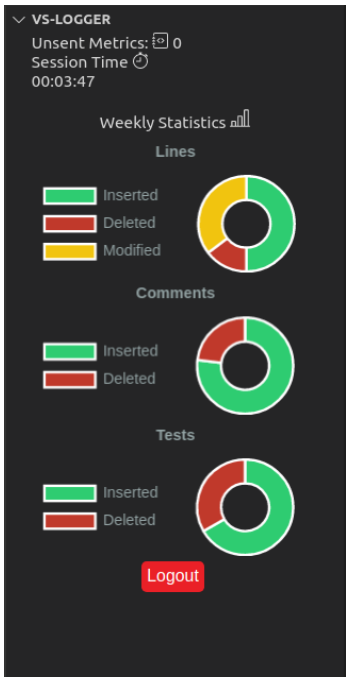


Figura 1.2: Immagine VScode-Logger

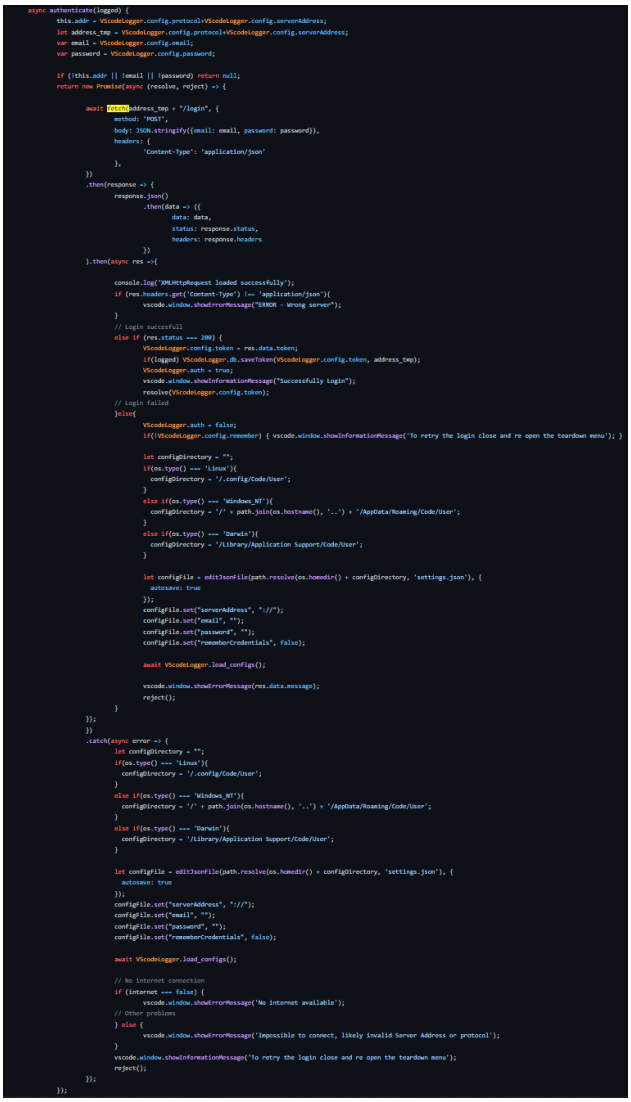
- Tempo trascorso dall'inizio della sessione di coding corrente.
- Statistiche sulle linee di codice (aggiunte, rimosse, modificate).
- Statistiche sui commenti del codice (aggiunti, rimossi, modificati).
- Statistiche sui file di test del progetto.

Per installare l'estensione bisogna cercare "VScode-Logger" nel marketplace delle estensioni di Visual Studio Code; in alternativa è possibile installare manualmente l'estensione scaricandolo il pacchetto .vsix dal GitHub dell'estensione `https://github.com/Stintipacchio/VScode-Logger.git`.
le istruzioni per l'installazione sono presenti nel README.md.

API logger

Il servizio logger permettere di effettuare 3 richieste API al server:

1. La prima è una chiamata POST e serve per autenticarsi attraverso uno user-rid(in questo caso la mail) e una password.
   Nel caso di risposta positiva da parte del server l'utente verrà autenticato, altrimenti verrà mostrato un messaggio di errore che varierà a seconda del motivo per cui l'autenticazione non è andata a buon fine.



```
async authenticate(logged) {
        this.addr = VScodeLogger.config.protocol+VScodeLogger.config.serverAddress;
        let address_tmp = VScodeLogger.config.protocol+VScodeLogger.config.serverAddress;
        var email = VScodeLogger.config.email;
        var password = VScodeLogger.config.password;

        if (!this.addr || !email || !password) return null;
        return new Promise(async (resolve, reject) => {

                await fetch(address_tmp + "/login", {
                        method: 'POST',
                        body: JSON.stringify({email: email, password: password}),
                        headers: {
                                'Content-Type': 'application/json'
                        },
                })
                .then(response => {
                        response.json()
                                .then(data => ({
                                        data: data,
                                        status: response.status,
                                        headers: response.headers
                                }))
                }).then(async res =>{

                        console.log('XMLHttpRequest loaded successfully');
                        if (res.headers.get('Content-Type') != 'application/json'){
                                vscode.window.showErrorMessage("ERROR - Wrong server");
                        }
                        // Login succesfull
                        else if (res.status === 200) {
                                VScodeLogger.config.token = res.data.token;
                                if(logged) VScodeLogger.db.saveToken(VScodeLogger.config.token, address_tmp);
                                VScodeLogger.auth = true;
                                vscode.window.showInformationMessage("Successfully Login");
                                resolve(VScodeLogger.config.token);
                        // Login failed
                        }else{
                                VScodeLogger.auth = false;
                                if(!VScodeLogger.config.remember) { vscode.window.showInformationMessage('To retry the login close and re open the teardown menu'); }

                                let configDirectory = "";
                                if(os.type() === 'Linux'){
                                  configDirectory = '/.config/Code/User';
                                }
                                else if(os.type() === 'Windows_NT'){
                                  configDirectory = '/' + path.join(os.hostname(), '..') + '/AppData/Roaming/Code/User';
                                }
                                else if(os.type() === 'Darwin'){
                                  configDirectory = '/Library/Application Support/Code/User';
                                }

                                let configFile = editJsonFile(path.resolve(os.homedir() + configDirectory, 'settings.json'), {
                                  autosave: true
                                });
                                configFile.set("serverAddress", "://");
                                configFile.set("email", "");
                                configFile.set("password", "");
                                configFile.set("rememberCredentials", false);

                                await VScodeLogger.load_configs();

                                vscode.window.showErrorMessage(res.data.message);
                                reject();
                        }
                });
                })
                .catch(async error => {
                        let configDirectory = "";
                        if(os.type() === 'Linux'){
                          configDirectory = '/.config/Code/User';
                        }
                        else if(os.type() === 'Windows_NT'){
                          configDirectory = '/' + path.join(os.hostname(), '..') + '/AppData/Roaming/Code/User';
                        }
                        else if(os.type() === 'Darwin'){
                          configDirectory = '/Library/Application Support/Code/User';
                        }

                        let configFile = editJsonFile(path.resolve(os.homedir() + configDirectory, 'settings.json'), {
                          autosave: true
                        });
                        configFile.set("serverAddress", "://");
                        configFile.set("email", "");
                        configFile.set("password", "");
                        configFile.set("rememberCredentials", false);

                        await VScodeLogger.load_configs();

                        // No internet connection
                        if (internet === false) {
                                vscode.window.showErrorMessage('No internet available');
                        // Other problems
                        } else {
                                vscode.window.showErrorMessage('Impossible to connect, likely invalid Server Address or protocol');
                        }
                        vscode.window.showInformationMessage('To retry the login close and re open the teardown menu');
                        reject();
                });
        });
```

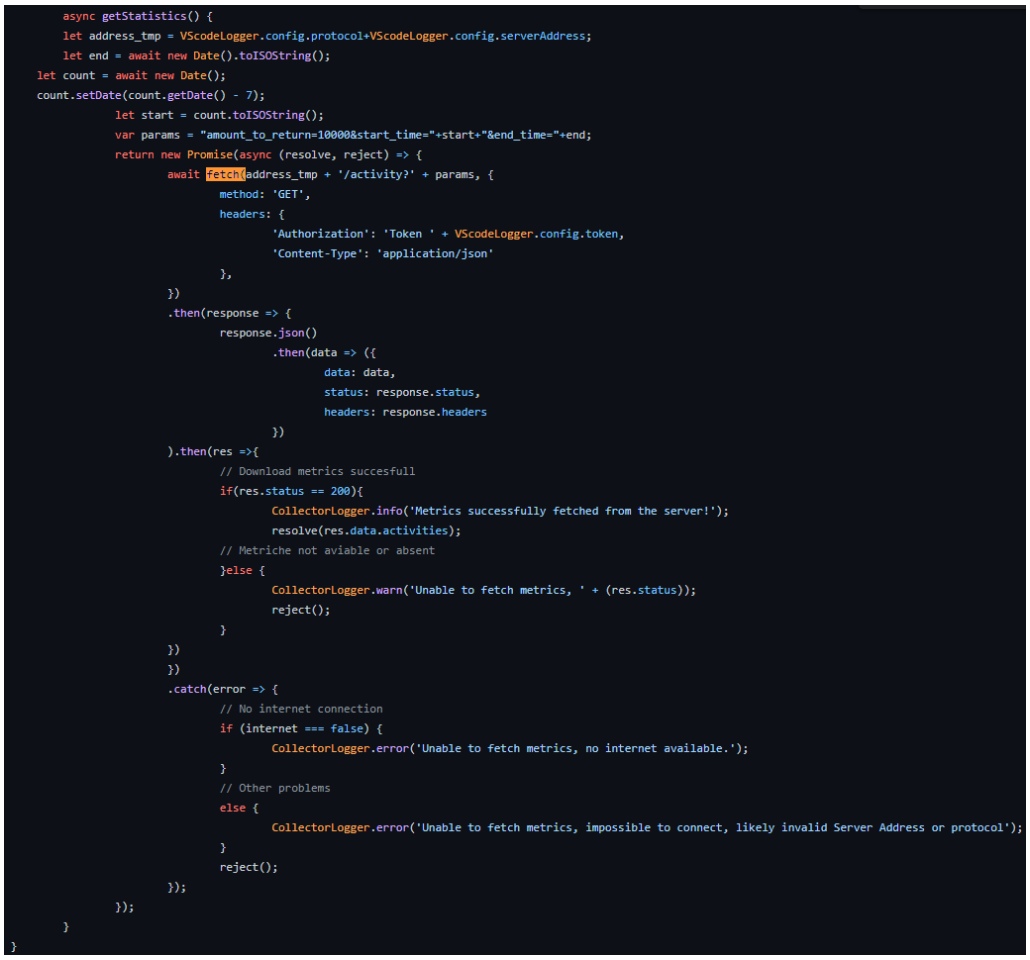
Figura 1.3: Codice API per autenticarsi

2. La seconda è una chiamata POST e serve per inviare le metrics raccolte al server.
   Nel caso di risposta positiva da parte del server le metrics verranno salvate nel database del server, altrimenti verrà mostrato un messaggio di errore che varierà a seconda del motivo per cui l'operazione non è andata a buon fine.

```
async sendMetrics(metrics) {
        if (!VScodeLogger.config.token) return false;

        var activities = [];
        for (var i in metrics) {
                var metric = {};
                metric.executable_name = metrics[i].tabName;
                metric.start_time = metrics[i].startDate;
                metric.end_time = metrics[i].endDate;
                metric.ip_address = metrics[i].session.ipAddr;
                metric.mac_address = metrics[i].session.macAddr;
                metric.activity_type = metrics[i].activity_type;
                metric.value = metrics[i].value;
                activities.push(metric);
        }

        var json = JSON.stringify({activities: activities});
        var formData = new FormData(); // Currently empty
        formData.append("activity", json);

        return new Promise(async (resolve, reject) => {
                let address_tmp = VScodeLogger.config.protocol+VScodeLogger.config.serverAddress;
                await fetch(address_tmp + "/activity", {
                        method: 'POST',
                        body: formData,
                        headers: {
                                'Authorization': 'Token ' + VScodeLogger.config.token,

                        },
                })
                .then(response => {
                        response.json()
                                .then(data => ({
                                        data: data,
                                        status: response.status,
                                        headers: response.headers
                                }))
                ).then(res =>{
                        // Upload metrics succesfull
                        if (res.status == 200 || res.status == 201) {
                                CollectorLogger.info('Metrics successfully sent to server!');
                                resolve(true);
                        // Upload metrics failed
                        }else{
                                CollectorLogger.warn('Unable to send metrics. ' + res.status);
                                reject(false);
                        }
                });
                })
                .catch(error => {
                        // No internet connection
                        if (internet === false) {
                                CollectorLogger.error('Unable to send metrics, no internet available');
                        }
                        // Other problems
                        else {
                                CollectorLogger.error('Unable to send metrics, impossible to connect, likely invalid Server Address or protocol');
                        }
                        reject(false);
                });
        });
}
```

Figura 1.4: Codice API per inviare metrics al server

3. La terza è una chiamata GET e serve per ricevere metrics dal server.
   In caso di risposta positiva da parte del server le misurazioni presenti nel database riguardante l'ultima settimana di attività verranno mostrate nella dashboard dell'estensione, altrimenti verrà mostrato un messaggio di errore che varierà a seconda del motivo per cui l'operazione non è andata a buon fine.



```
    async getStatistics() {
        let address_tmp = VScodeLogger.config.protocol+VScodeLogger.config.serverAddress;
        let end = await new Date().toISOString();
    let count = await new Date();
    count.setDate(count.getDate() - 7);
                let start = count.toISOString();
                var params = "amount_to_return=10000&start_time="+start+"&end_time="+end;
                return new Promise(async (resolve, reject) => {
                        await fetch(address_tmp + '/activity?' + params, {
                                method: 'GET',
                                headers: {
                                        'Authorization': 'Token ' + VScodeLogger.config.token,
                                        'Content-Type': 'application/json'
                                },
                        })
                        .then(response => {
                                response.json()
                                        .then(data => ({
                                                data: data,
                                                status: response.status,
                                                headers: response.headers
                                        })
                        ).then(res =>{
                                // Download metrics succesfull
                                if(res.status == 200){
                                        CollectorLogger.info('Metrics successfully fetched from the server!');
                                        resolve(res.data.activities);
                                // Metriche not aviable or absent
                                }else {
                                        CollectorLogger.warn('Unable to fetch metrics, ' + (res.status));
                                        reject();
                                }
                        })
                        })
                        .catch(error => {
                                // No internet connection
                                if (internet === false) {
                                        CollectorLogger.error('Unable to fetch metrics, no internet available.');
                                }
                                // Other problems
                                else {
                                        CollectorLogger.error('Unable to fetch metrics, impossible to connect, likely invalid Server Address or protocol');
                                }
                                reject();
                        });
                });
        }
}
```

Figura 1.5: Codice API per ricevere metrics dal server

1.1.5 Tipologia di misurazioni

I logger permettono di effettuare cinque diversi tipi di misurazioni:

- **Linee di codice**: i plugin determinano, attraverso l'utilizzo di librerie diff in Java e JavaScript(ECMAScript), il numero di righe aggiunte, modificate ed eliminate ogni qualvolta un file viene salvato.
- **Commenti**: utilizzando le stesse librerie usate per il controllo delle linee di codice, mediante il pattern matching viene stabilito in che quantità sono stati aggiunti o eliminati commenti.
- **Linee di testing**: il medesimo procedimento viene utilizzato per individuare le linee relative al testing.
- **Refactoring**: ogni editor emette dei segnali per gli eventi generati mediante l'uso dell'IDE stesso.
  I plugin catturano metriche determinando la durata ed il tipo di ogni singola operazione e ignorando gli eventi non considerati significativi dell'attività, come ad esempio cut, paste, copy, backspace, enter, save, undo, redo.
- **Sessione**: un utente autenticato genera anche delle metriche di attività che comprendono dettagli hardware della macchina sulla quale sta lavorando, in particolare indirizzo IP e MAC, che verranno associati alla tipologia di attività che sta svolgendo.

1.2 CAS Server

Tutti i servizi CAS sono open source e ognuno di essi ha una sua utilità nel contribuire a diverse attività nello sviluppo agile [18].
Di seguito, una breve analisi di tutti i servizi.

1.2.1 Taiga

Taiga [7] è un servizio di project managment ideato per gestire sviluppatori agili che utilizzano Scrum e Kanban.
È possibile decidere, alla creazione del progetto, se fare uso di Backlog, Kanban o Epiche.
Inoltre dispone di sezioni Issue, Wiki e permette di impostare un servizio di videoconferenza di default, oltre alla possibilità di integrare sistemi VCS in caso di necessità.
Riassumendo, Taiga permette di visualizzare in maniera chiara e completa l'andamento di un progetto permettendo anche di analizzare ripartizioni dei compiti ed eventuali problemi riscontrati durante lo sviluppo.

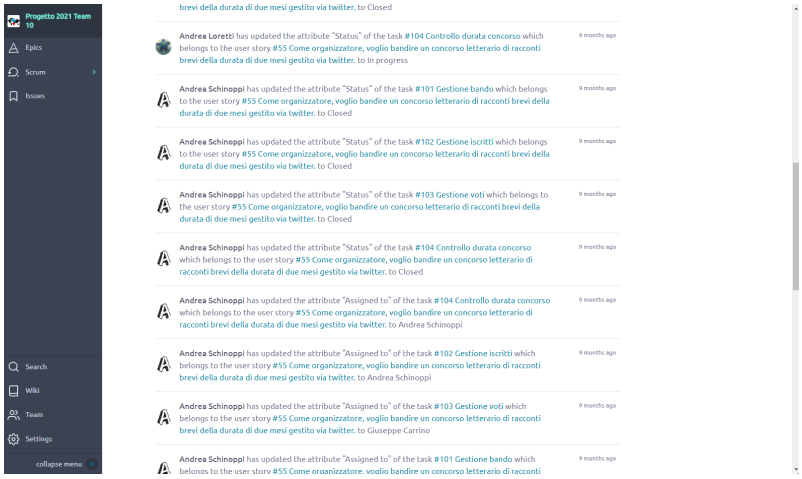


Figura 1.6: Taiga interface

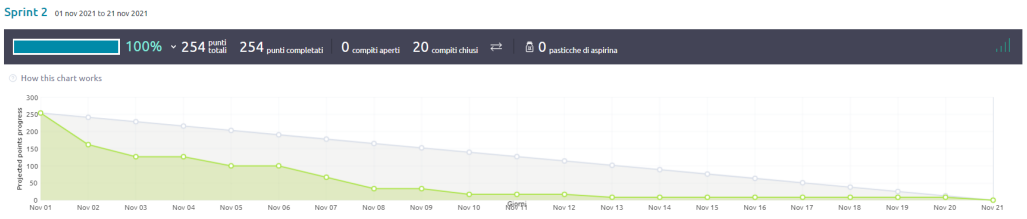


Figura 1.7: Taiga burndown

1.2.2 GitLab

GitLab [2] è una piattaforma per il version control che permette la gestione di repository git. Fornisce, inoltre, delle funzionalità aggiuntive come una Wiki, issue-tracking e CI/CD.
Permette l'utilizzo di operazioni push, pull e merge in modo da facilitare la collaborazione tra sviluppatori.

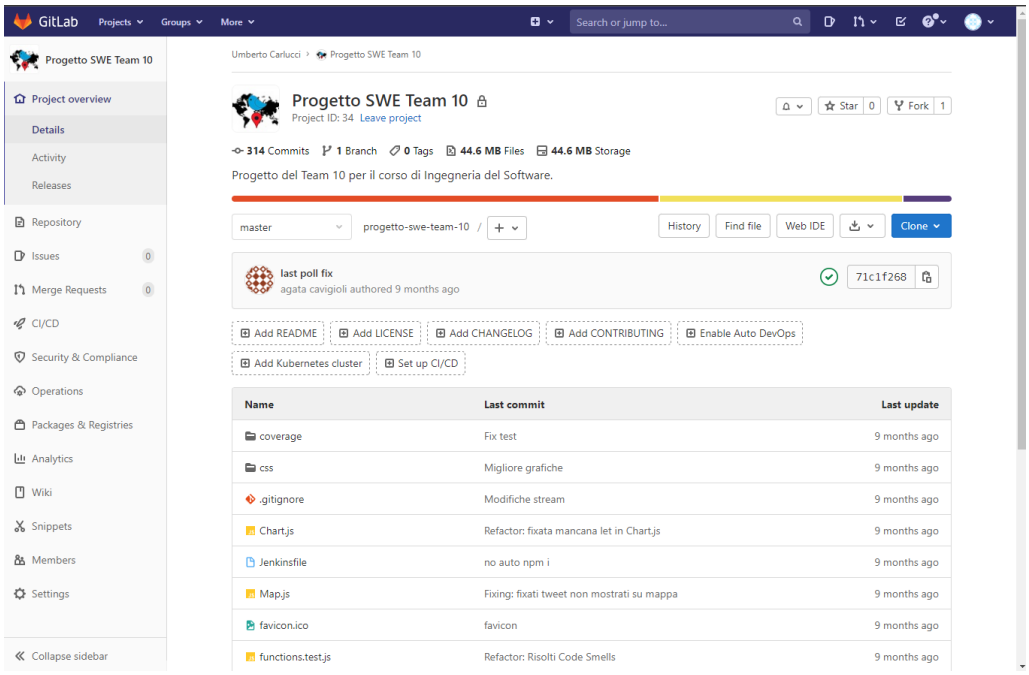


Figura 1.8: GitLab interface

1.2.3 Jenkins

Jenkins[3] è uno strumento di automazione per continuous integration/continuous delivery and deployment (CI/CD) scritto in Java.
Viene utilizzato per l'implementazione di workflows CI/CD, chiamati *pipeline*.

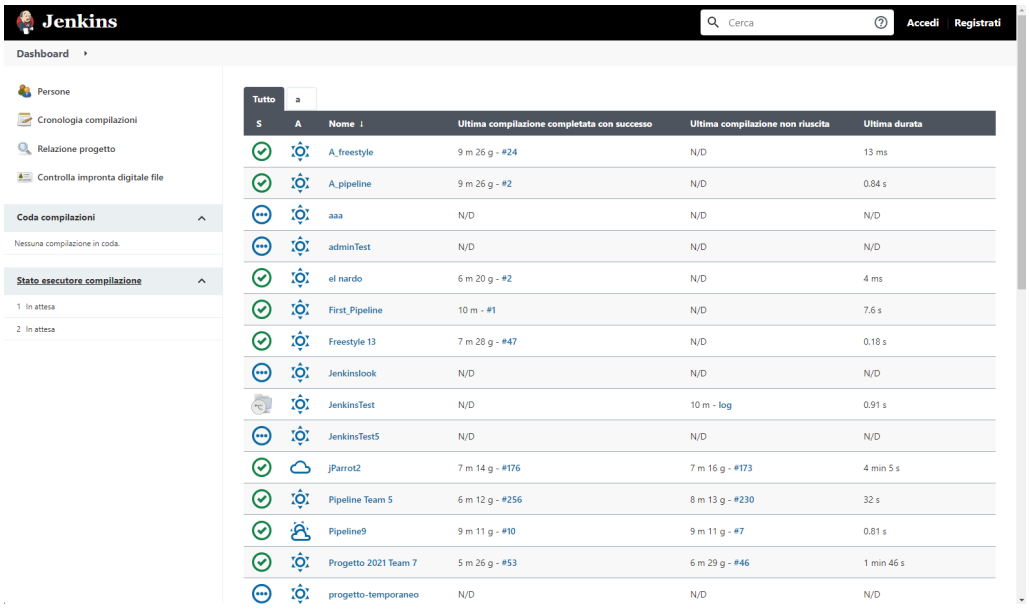


Figura 1.9: Jenkins interface

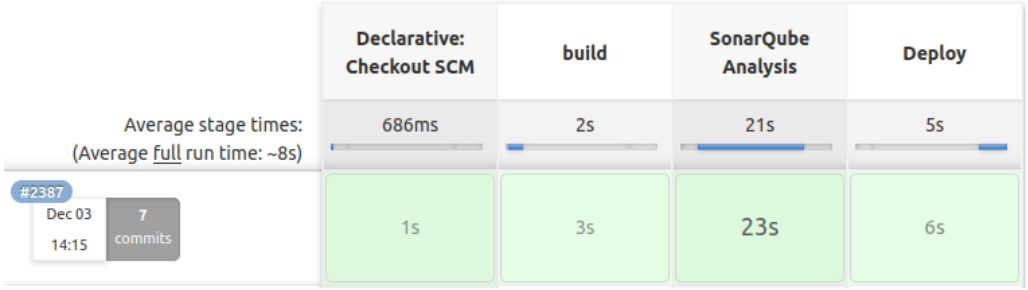


Figura 1.10: Jenkins pipeline

1.2.4 SonarQube

SonarQube[6] è una piattaforma che permette di effettuare un analisi statica del codice.
A seguito dell'analisi vengono generati dei report di qualità del software che evidenziano eventuali bug, code smell e vulnerabilità.
Una corretta inizializzazione dei parametri da tenere in considerazione per la scansione del codice sorgente (affidabilità, sicurezza, manutenibilità, dimensione, complessità) renderà più precisa l'interpretazione dei risultati.
Se ad ogni modifica rilevante del codice viene eseguita un'analisi, è possibile osservare il cambiamento nel corso del tempo degli indici di qualità.

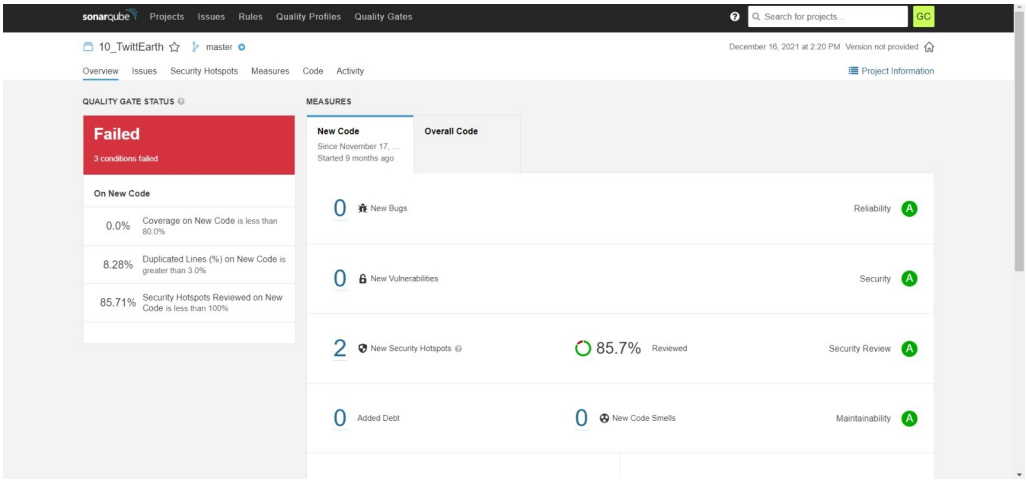


Figura 1.11: Sonarqube interface

1.2.5 Mattermost

Mattermost [4] è un sistema di comunicazione self-hosted simile al servizi proprietari Slack o Discord che permette di creare canali testuali pubblici o privati, avviare audio o video conferenze e condividere file.
È possibile aggiungere funzionalità attraverso le integrazioni con altri servizi.
Ad esempio, è possibile ricevere un messaggio automatico ogni qualvolta viene effettuato un push sul main branch di un progetto in git.

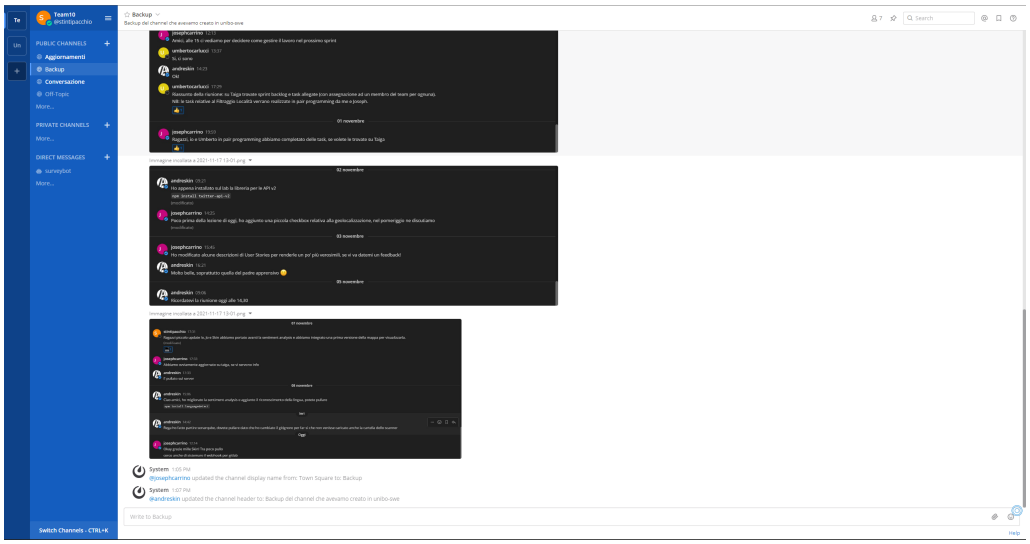


Figura 1.12: Mattermost interface

1.2.6 CAS-Logger

CAS-Logger [14] è un servizio di self tracking che permette a ogni utente di visualizzare le proprie statistiche generate dalle azioni effettuate su un IDE dotato di un plugin che lo rende compatibile con i servizi CAS.
Per poter consultare nel dettaglio le statistiche è sufficiente effettuare il login alla piattaforma CAS.

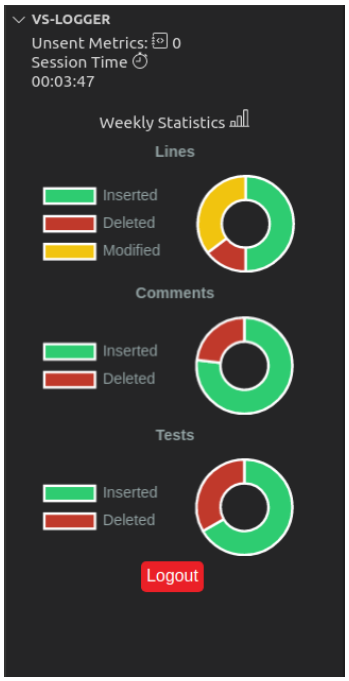


Figura 1.13: Immagine VScode-Logger

1.2.7 CAS-Dashboard

CAS-Dashboard [16] è una web app, di tipo operazionale, che funziona da "cruscotto" per CAS aggregando i dati, provenienti da tutti i servizi di CAS, di un singolo sviluppatore.
Le tecnologie impiegate sono framework Javascript quali React e Redux, scelti per la loro natura modulare.
Essendo stata concepita come hub per i servizi CAS, non esiste un unico pannello che raggruppi tutti i dati aggregati; al contrario, sono presenti diverse sezioni, una per ogni servizio.
Ogni sezione interagisce con il servizio associato attraverso le API messe a disposizione dal servizio stesso.
I dati estrapolati sono manipolati per restituire delle tabelle o dei grafici. I grafici sono stati implementati con ChartJS, che agisce da wrapper della libreria per React.

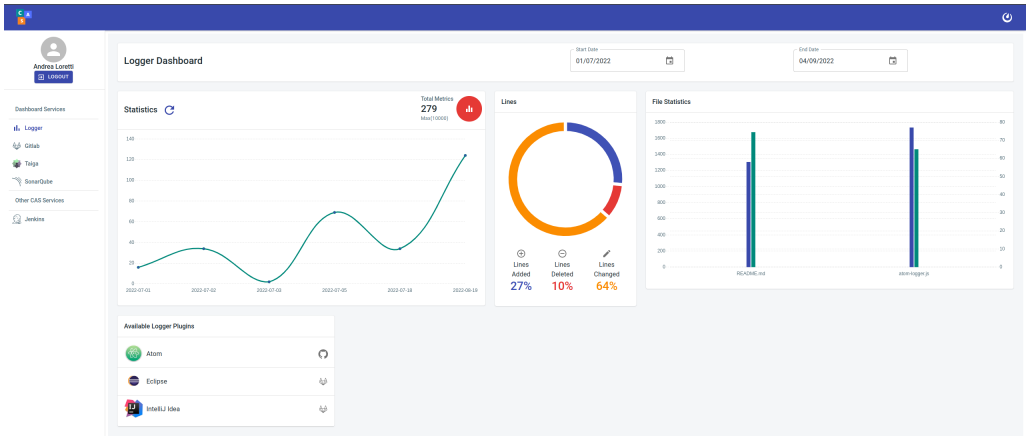


Figura 1.14: CAS-Dasboard

1.2.8 Componenti dell'infrastruttura di CAS

Docker e Nginx vengono utilizzati come Wrapper dei servizi CAS.

Docker

Docker [1] è una piattaforma open source per lo sviluppo, la spedizione e l'esecuzione di applicazioni.
Docker consente di separare le applicazioni dall'infrastruttura in modo da poter distribuire rapidamente il software.
Docker può gestire l'infrastruttura nello stesso modo in cui vengono gestite normalmente le applicazioni.
Sfruttando le metodologie Docker per la spedizione, il test e la distribuzione rapida del codice, si può ridurre significativamente il ritardo tra la scrittura del codice e l'esecuzione in produzione.

Nginx

Nginx [5] è un server HTTP e proxy inverso gratuito, open source e ad alte prestazioni, nonché un server proxy IMAP/POP3.
Nginx è noto per le sue elevate prestazioni, per la sua stabilità, per la ricchezza del suo set di funzionalità, per la sua configurazione semplice e per il basso consumo di risorse.
Nginx è uno dei pochi server scritti per risolvere il problema C10K.
A differenza dei server tradizionali, Nginx non fa affidamento sui thread per gestire le richieste.
Al contrario, utilizza un'architettura (asincrona) basata su eventi molto più scalabili.
Questa architettura utilizza piccole, ma soprattutto prevedibili, quantità di memoria sotto carico.
Anche se non occorre gestire migliaia di richieste simultanee, si può comunque trarre vantaggio dalle prestazioni elevate e dal footprint di memoria ridotto di Nginx.
Nginx è scalabile in tutte le direzioni: dal più piccolo server virtual privato (VPS) fino ai grandi cluster di server.

1.3 Problemi riscontrati in CAS 2.0

I problemi riscontrati dall'uso di CAS nel progetto del corso di Ingegneria del software riguardano tre microservizi: Jenkins, Mattermost e il Logger.

Per quanto riguarda **Jenkins**, molti team durante l'esame di ingegneria del software nell'anno accademico 2021/2022 non sono riusciti ad utilizzare il servizio. Alcuni lo hanno hostato su una macchina privata perché non era possibile mantenere privati i token utilizzati per il riconoscimento del team, rendendo molto vulnerabile la pipeline, come evidenziato nei dati raccolti[17].

Mattermost, invece, nonostante funzionasse bene come servizio di messaggistica e fosse possibile integrarlo alla pipeline, inviando un messaggio al team ogni qual volta veniva prodotta una release, non aveva alcun altro servizio abilitato (ad esempio non era possibile effettuare call, video e non)
Per questo motivo si è preferito, molto spesso, utilizzare canali alternativi (Telegram, Discord, ecc.).

I problemi riguardanti il **logger** sono dovuti ai plug-in utilizzati dagli IDE compatibili.
I problemi sono i seguenti:

- L'IntelliJ-Logger è compatibile solo con una versione specifica di IntelliJ Community Edition.
- L'Atom-Logger non funziona su dispositivi Apple(MacOS), questo bug è dovuto al modo in cui Atom è concepito.
  Atom esegue richieste che violano i protocolli di sicurezza Apple, di conseguenza l'S.O. termina il processo.
  Questa problematica non si presenta in VisualStudioCode perché i protocolli utilizzati per la gestione delle estensioni sono molto rigidi e rispettano le policy Apple.
  Inoltre, non è possibile salvare le credenziali per non rieffettuare il log-in ad ogni avvio, le richieste API sono scritte con librerie datate e presenta diversi bug.

Queste problematiche sono state tutte risolte nel plug-in per VisualStudioCode (Figure 1.2).

Capitolo 2

Dashboard

In questo capitolo si andrà ad analizzare cosa sono le dashboard, perché sono utili, quali sono i limiti alle quali sono soggette e come si evolveranno in futuro.

Figura 2.1: Esempio di una Dashboard

2.1 Definizione

"Le dashboard sono strumenti di comunicazione, visibilità e consapevolezza progettati per aiutare le persone a identificare tendenze, pattern e anomalie di modo da prendere decisioni informate ed efficaci [9].
Il loro valore è uno dei motivi principali della loro popolarità è la loro capacità di "replace hunt-and-peck data-gathering techniques with a tireless, adaptable, information flow mechanism" [12]." [10]

2.2 Scopo

Lo scopo delle dashboard è di trasformare i dati grezzi, contenuti nei repository di un'organizzazione, in informazioni di consumo.
Nell'ingegneria del software le dashboard sono usate per fornire informazioni riguardanti l'andamento dello sviluppo di un progetto, rispondendo a domande del tipo: "Questo progetto è nei tempi previsti?", "Quali sono i colli di bottiglia attuali?", "Come sta andando il progresso di altri team?".

Prima di poter valutare i vantaggi dell'utilizzo delle dashboard e i vari contesti nelle quali risultano maggiormente utili nell'ingegneria del software, è doverosa una premessa.
Il collegamento tra produttività e dashboard diventa evidente se si osserva la tipologia di misurazioni utilizzate.
Sebbene non sia sempre inteso in questo modo, gran parte dei dati presentati nelle dashboard possono anche essere interpretati come una misura della produttività degli sviluppatori.
Ad esempio, un grafico a barre che mostra gli open issues raggruppati per team può essere facilmente frainteso come un grafico che evidenzia i team più produttivi.
Il rapporto tra produttività di un team di sviluppo e il numero di open issues è ovviamente molto più complesso, "Just because one team has a lot more defects than another that doesn't necessarily mean that the quality of that component is any worse" [21].
Dato che gli sviluppatori di software producono molti artefatti testuali, come codice sorgente, documentazione, segnalazioni di bug e revisioni del codice, non è sorprendente che le dashboard utilizzate nei progetti software spesso combinino diversi tipi di dati, ad es. dati qualitativi e quantitativi.
Un altro parametro da osservare è l'ampiezza dei dati.
Durante la creazione di una dashboard per un progetto software, molte variabili devono essere prese in considerazione: ad esempio ci si potrebbe domandare: "la dashboard dovrebbe contenere dati a livello aziendale o solo dati di un singolo pro-

getto (tenendo presente che i progetti tendono a non essere indipendenti)?"
"Ogni sviluppatore dovrebbe avere la propria dashboard personalizzata o tutte le dashboard di un progetto devono avere lo stesso aspetto?"

Inoltre, le dashboard possono coprire diversi periodi di tempo, come l'intera durata di un progetto, la versione corrente o l'ultima settimana.

2.3 Vantaggi

Come constatato in precedenza, le dashboard nell'ingegneria del software sono utilizzate per fornire informazioni e metriche sul prodotto che si sta sviluppando, nonché per visualizzare informazioni e per supportare l'analisi del processo di sviluppo.
In genere, le dashboard sono progettate con uno specifico stakeholder e obiettivo in mente. Molti di questi obiettivi si riferiscono direttamente o implicitamente ad alcuni aspetti della produttività, inclusa la qualità del prodotto, la velocità di lavoro o la soddisfazione dello stakeholder.
Ci sono diversi ambiti in cui le dashboard possono offrire vantaggi durante il processo di sviluppo software.
Di seguito verranno presentati alcuni di questi ambiti.

Figura 2.2: Immagine sul miglioramento della produttività

2.3.1 Attività degli sviluppatori

Le dashboard possono essere utilizzate per visualizzare l'attività e le prestazioni dei singoli sviluppatori, ad esempio come viene speso il coding time (creazione, debugging, test, ricerca, ecc.), quanto tempo gli sviluppatori sono stati concetrati in un dato periodo di tempo, il numero e la natura di interruzioni che possono incontrare, il tempo trascorso utilizzando altri strumenti ausiliari, coding behaviors (ad esempio, velocità di correzione degli errori sintattici) e metriche che indicano quante righe di codice o funzionalità hanno contribuito a un repository.
Queste informazioni, se utilizzate dagli stessi sviluppatori, possono anche aiutare nel monitoraggio delle prestazioni personali per miglioramenti della produttività personale soprattutto quando le dashboard consentono un confronto di tali informazioni nel tempo.
Tali dashboard aiutano anche gli sviluppatori a rivelare colli di bottiglia nel codice del progetto stesso o dal proprio processo di sviluppo.

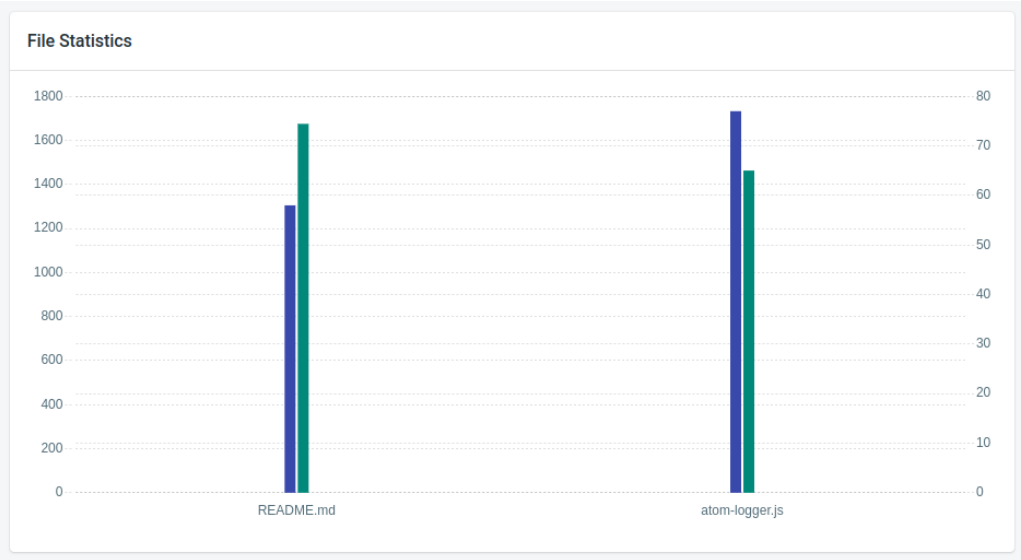


Figura 2.3: Grafico a barre di tutte le interazioni con i file di CAS-Dashboard

2.3.2 Performance del team

Sebbene diverse dashboard siano progettate principalmente per consentire agli sviluppatori di ottenere informazioni dettagliate sulle proprie attività e comportamenti, molte mostrano o aggregano informazioni all'interno di un team per altre parti interessate, come team leader, manager, analisti aziendali o ricercatori.
Queste informazioni a livello di team possono essere utilizzate per migliorare l'ambiente di lavoro, il processo di sviluppo o gli strumenti che utilizzano.
Alcuni servizi forniscono anche supporto ai team per migliorare attivamente l'insieme della loro performance.
Tenere traccia e monitorare il lavoro a livello di team è particolarmente importante per i cosiddetti distributed teams (team in cui tutti i membri si trovano fisicamente distanti l'uno dall'altro, operando anche in fusi orari differenti).

I team Agili utilizzano molti strumenti per tenere traccia delle attività del progetto che li aiutano a gestire e riflettere sul loro processo, in particolare monitorando le loro performance attraverso i vari sprint.
Nei team agili, in particolare, le dashboard svolgono un ruolo importante per i manager, dato che hanno la responsabilità di tenere traccia di tutte le cose in sviluppo durante uno sprint.
Molto spesso i manager fanno affidamento su dashboard che visualizzano tutti gli open issues per un particolare progetto e visualizzano a chi sono stati assegnati e con quale priorità.
I grafici di burndown, nelle dashboard, possono mostrare l'andamento del team rispetto alla linea di burndown prevista.

Figura 2.4: Taiga Burndown

2.3.3 Monitoraggio del progetto e performance

Per monitorare l'attività a un livello di progetto specifico, GitHub, come altri servizi di repository, utilizza ampiamente le dashboard per fornire approfondimenti a manager, proprietari di progetti e altri sviluppatori che potrebbero voler decidere il valore dell'utilizzo, a seconda del contributo a progetti particolari.
Poiché molti progetti oggigiorno si basano su servizi di continuous integration e deployment, molte dashboard visualizzano come il codice si sta muovendo attraverso la pipeline, in particolare come le nuove funzionalità vengono utilizzate negli esperimenti di test.
Il supporto DevOps aggiuntivo potrebbe essere fornito visualizzando le prestazioni dei servizi in esecuzione, monitorando le interruzioni, ecc.

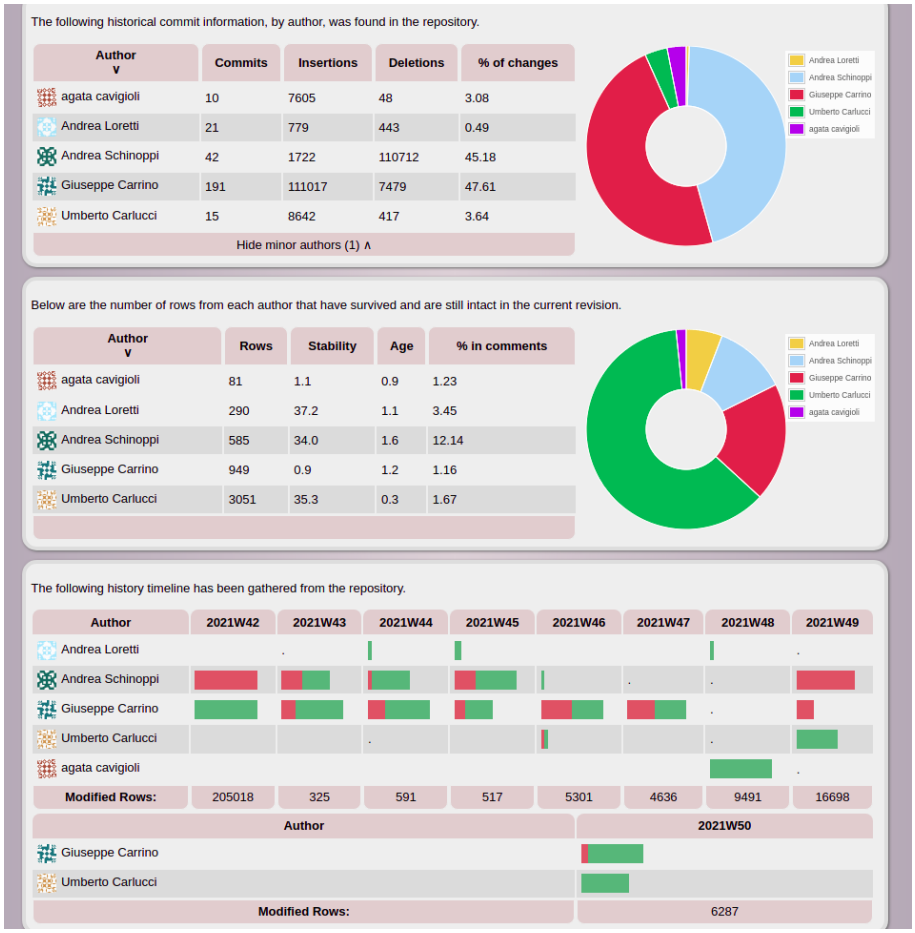


Figura 2.5: Dashboard GitInspector

2.3.4 Salute della Community

Strettamente correlati alle dashboard a livello di progetto, altri servizi di dashboard mirano in modo specifico a visualizzare i dati a livello di comunità o ecosistema.
Ad esempio, CHAOSS, un software open source della Linux Foundation, raccoglie e visualizza i dati per supportare l'analisi della salute delle community, ad esempio community open source come Linux.
Per Linux, CHAOSS definisce metriche di salute interessanti come il numero di licenze utilizzate[1].

[1] `https://github.com/chaoss/metrics/`

2.4 Criticità

Come visto nella sezione precedente, le dashboard sono molto utili per trasformare raw data in informazioni fruibili e sfruttabili per migliorare il processo di sviluppo software in tutte le sue sfaccettature, ma come tutti gli strumenti di monitoraggio presentano dei problemi se ne si abusa senza riconoscerne le limitazioni.
Di seguito sono esposte le principali limitazioni.

Figura 2.6: Immagine pericolosità delle dashboard

2.4.1 Le dashboard prediligono i numeri rispetto al testo

Mentre molti degli artefatti con cui gli sviluppatori di software lavorano sono testuali, come i requisiti di specifiche, messaggi di commit o bug report, presentare il contenuto di questi artefatti testuali su una dashboard non è banale.
Per risolvere questo problema spesso vengono utilizzate tecniche di aggregamento di informazioni testuali, come algoritmi topic modeling o di sintesi, però queste tecniche non sempre producono risultati ottimali, ed è quindi spesso più facile presentare dei numeri invece del testo su una dashboard.
Di conseguenza, una dashboard per sviluppatori è più probabile che contenga informazioni su quanti problemi sono stati chiusi rispetto alle informazioni su quale caratteristica è la più citata in bug report.

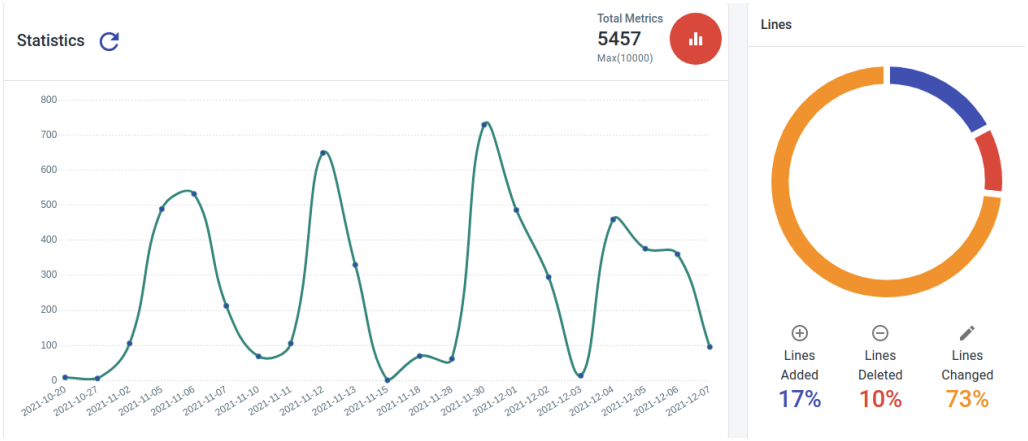


Figura 2.7: Overview delle metriche totali prodotte di CAS-Dashboard

2.4.2 Le dashboard spesso non spiegano

Una dashboard potrebbe essere in grado di mostrare che un team ha meno problemi aperti di un altro team, che una certa componente ha meno bug di un'altra componente, o che lo sviluppatore ha trascorso più tempo nell'IDE rispetto al precedente mese.
Tuttavia, molte dashboard non forniscono spiegazioni per tali osservazioni, e, senza spiegazioni, queste informazioni potrebbero risultare inutili.
Ad esempio, un team potrebbe non sapere cosa fare per diminuire il numero di open issue attuale, il motivo per cui un componente ha più problemi di un altro non è ovvio, e uno sviluppatore potrebbe non sapere cosa può fare per migliorare la propria produttività.

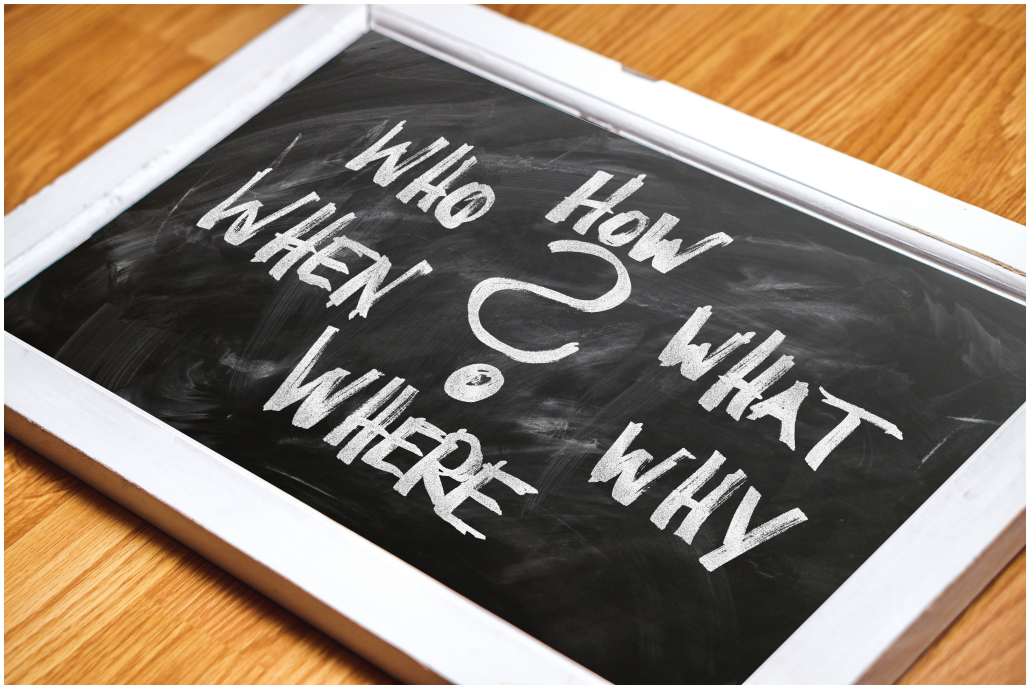


Figura 2.8: Immagine cosa fare con le dashboard che non spiegano

2.4.3 Le dashboard potrebbero non mostrare un contesto pertinente

L'aggregazione delle informazioni implica la mancanza di alcuni dettagli, il che spesso significa che non tutte le informazioni contestuali sono disponibili.
Una dashboard che visualizza le informazioni su una correzione di bug critici potrebbe non contenere tutti i dettagli su come il bug è stato corretto e una dashboard che confronta il tempo trascorso in un browser rispetto al tempo trascorso in un IDE potrebbe non contenere informazioni riguardo quali delle attività siano effettivamente legate allo sviluppo del software.
Inoltre, seguendo un ragionamento intuitivo, la presentazione di informazioni aggregate su dashboard potrebbe invitare gli utenti al confronto tra progetti e aziende.
Questi confronti però non prendono in considerazione due fattori molto importanti:

- Non esistono due progetti software uguali.
- Molto spesso mancano di contesto per poter comprendere la situazione.

In una certa misura, questi problemi possono essere affrontati rendendo una dashboard interattiva e consentendo ai suoi utenti di usufruire di informazioni più complete.

2.4.4 Ottieni ciò che misuri

La legge di Goodhart, di solito citata come "*When a measure becomes a target, it ceases to be a good measure*", descrive un altro rischio dell'uso di dashboard nello sviluppo di progetti software.
Ad esempio, se una dashboard enfatizza il numero di open issue, gli sviluppatori saranno più cauti nel generare altri open issue, combinando diversi problemi minori in uno.
Allo stesso modo, se una dashboard concettualizza la produttività come tempo speso nell'IDE, gli sviluppatori potrebbero esitare a cercare informazioni al di fuori dell'IDE stesso.
In entrambi gli esempi probabilmente non era questo l'intento della dashboard, eppure decenni di ricerca sulla gamification hanno dimostrato che gli esseri umani tendono a giocare con tali sistemi.

"Developers are the most capable people on Earth to game any system you create." [20].

2.4.5 La validità di una dashboard è strettamente correlata alla qualità dei dati impiegati

Molti studi hanno scoperto che i dati acquisiti nei repository di software non sempre rispecchiano accuratamente la realtà di sviluppo.
Per esempio, Aranda e Venolia [8] hanno scoperto che il coordinamento che si verifica intorno ai bug del software non può essere estratto esclusivamente dai repository poiché porterebbe a resoconti incompleti e spesso errati del coordinamento.
In tal caso, una dashboard basata su tali dati non sarà in grado di visualizzare informazioni accurate.

Figura 2.9: Immagine sulla qualità dei dati

2.4.6 Le dashboard possono visualizzare solo i dati che sono stati tracciati da qualche parte

Mentre i repository software odierni sono in grado di acquisire molte delle azioni intraprese dagli sviluppatori software, ci sono ancora molte attività che non vengono misurate.
Ad esempio, un repository non sarebbe in grado di catturare una conversazione tra sviluppatori su un nuovo processore, anche se potrebbe fornire un'informazione cruciale per la risoluzione di un particolare bug.
Sostanzialmente è impossibile catturare informazioni non divulgate in un ambiente di sviluppo software.
Gli utenti delle dashboard devono quindi essere consapevoli che la dashboard potrebbe non fornire sempre il quadro completo.

2.4.7 I dati relativi alle prestazioni sulle dashboard possono essere facilmente interpretati erroneamente come dati sulla produttività

Molte delle metriche che possono essere facilmente visualizzate su una dashboard, come il numero di open issue o il numero di righe di codice, possono essere interpretati come misure di produttività, consentendo confronti tra sviluppatori, team o componenti che ignorano le molte complessità di sviluppo software.
Gli sviluppatori nutrono molte riserve su tali misure di produttività, di conseguenza, accetteranno solo dashboard che non tentino di ridurre la complessità del contributo di uno sviluppatore a un singolo numero.

2.4.8 Le dashboard spesso non definiscono degli obbiettivi chiari

Può esserci uno scontro tra gli obiettivi di un'organizzazione di sviluppo software e gli elementi che emergono da una dashboard.
Mentre l'obiettivo di un organizzazione potrebbe essere la creazione di valore a lungo termine, spesso le dashboard utilizzano intervalli di tempo relativamente brevi.

Valori come la soddisfazione del cliente non sono facilmente estraibili da un repository software, anche se potrebbero effettivamente allinearsi molto meglio con l'obiettivo dell'organizzazione rispetto al numero di questioni aperte in un progetto o al tempo trascorso nell'IDE.

Figura 2.10: Immagine incertezza degli obbiettivi

2.5 Come migliorare la progettazione di Dashboard in futuro

Man mano che l'ingegneria del software diventa sempre più basata su dati e gli strumenti per la creazione di dashboard diventano più facili da usare, è auspicabile una crescita del ruolo svolto dalle dashboard nell'ingegneria del software e un aumento del numero di funzionalità che forniscono.
Per i singoli sviluppatori, le dashboard forniscono approfondimenti sulla produttività personale.
I team e i progetti le utilizzano per monitorare le prestazioni.
I manager e i leader della comunità le utilizzano per il processo decisionale.
L'intelligenza artificiale (I.A.), l'elaborazione del linguaggio naturale e i robot software[19] avranno un impatto anche sul design delle dashboard e sulle funzionalità che forniranno in un prossimo futuro.
Inoltre, le I.A. potrebbero essere utilizzate per raccogliere informazioni su come e quando vengono utilizzati i dashboard, sull'impatto che possono avere sui progetti software e su come la loro progettazione potrebbe essere migliorata nel tempo.
C'è sicuramente l'opportunità di automatizzare la visualizzazione di sempre più approfondimenti sui dati, ma anche di migliorare il modo in cui sviluppatori e altri stakeholder collaborano tra loro attraverso le dashboard.
Ci si può anche chiedere se le dashboard possano sostituire anche parzialmente altre modalità di scambio di informazioni (ad es. diapositive PowerPoint).
Dopo aver preso nota dei dati riportati dalle dashboard, gli stakeholder li interpreteranno come "verità" anche se i dati sottostanti o il modo in cui vengono analizzati e presentati possono essere imprecisi, distorti o fuorvianti?
È probabile che le dashboard e le tecnologie per crearle diventino onnipresenti e più facili da usare nel tempo.
Resta da vedere se aumenteranno o eventualmente danneggeranno e sminuiranno la produttività o se possano semplicemente fornire informazioni sulla produttività, ma è necessario prestare attenzione a come vengono create e utilizzate.

Capitolo 3

Valutazione

In questo capitolo verrà mostrato il tipo di dashboard utilizzate in CAS mostrando perché sono coerenti con la letteratura attualmente disponibile per quanto riguarda la progettazione di dashboard.

3.1 Esposizione e Analisi Dashboard utilizzate in CAS

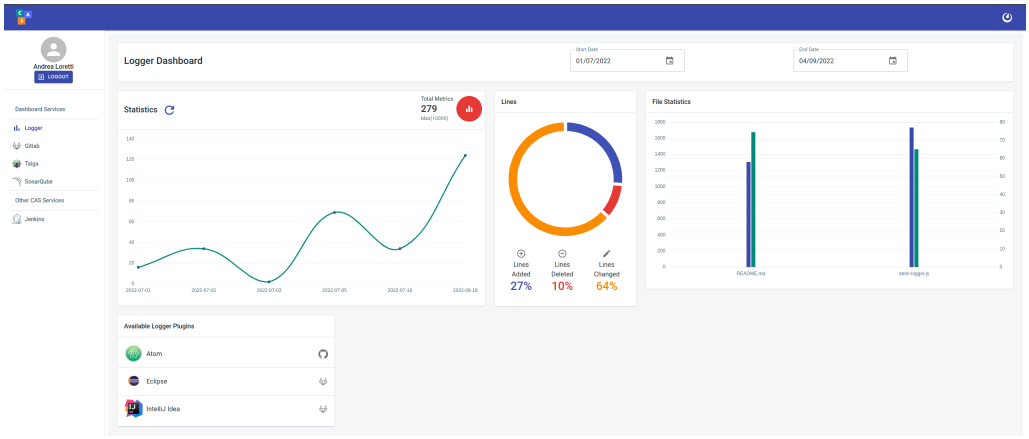


Figura 3.1: CAS-Dashboard

Le dashboard utilizzate nell'ambiente CAS sono due: la prima è visualizzabile attraverso un IDE compatibile (Figure 1.2), l'altra attraverso un browser collegandosi a `https://aminsep.disi.unibo.it/dashboard/logger` (Figure 3.1).

Come osservato nel capitolo precedente (chapter 2), le dashboard sono strumenti molto potenti, ma la loro implementazione è soggetta al contesto nel quale vengono utilizzate e al tipo di dataset che vengono elaborati, quindi è molto comune realizzare dashboard che all'apparenza forniscono molte informazioni, ma che nel concreto non siano impattanti nel processo di sviluppo software.

Lo scopo di questa sezione è quindi mostrare che le dashboard utilizzate in CAS sono coerenti con i concetti precedentemente esposti.

La prima dashboard è stata già discussa nel capitolo 1, nella sezione dedicata al logger per Visual Studio Code (subsection 1.1.4).

La seconda dashboard permette di monitorare quattro microservizi CAS ed è possibile accedervi attraverso una Navbar posizionata sulla sinistra.

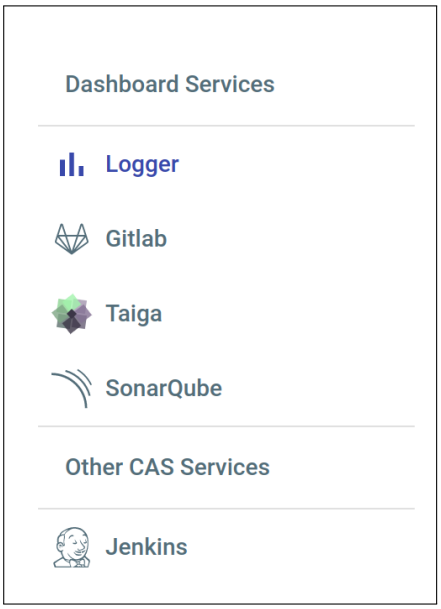


Figura 3.2: Navbar CAS-Dashboard

I quattro servizi disponibili sono:

1. Logger
2. Gitlab
3. Taiga
4. SonarQube

3.1.1 Logger

Permette di selezionare un arco temporale da poter analizzare ed è diviso in tre grafici:

- Un grafico a linea che mostra il totale della produzione di metriche nell'arco di tempo selezionato.
  Qui è possibile analizzare quanto e quando uno sviluppatore è stato attivo e produttivo, individuandone quindi l'andamento delle performance.
  Questa analisi è molto utile per stimare la consistenza del lavoro svolto da un singolo sviluppatore, permettendo di capire se un membro del team sia adatto alla task alla quale è stato assegnato.
  Inoltre, uno sviluppatore può sfruttare questi dati per migliorare le proprie performance personali; può essere anche utile per individuare eventuali burnout.

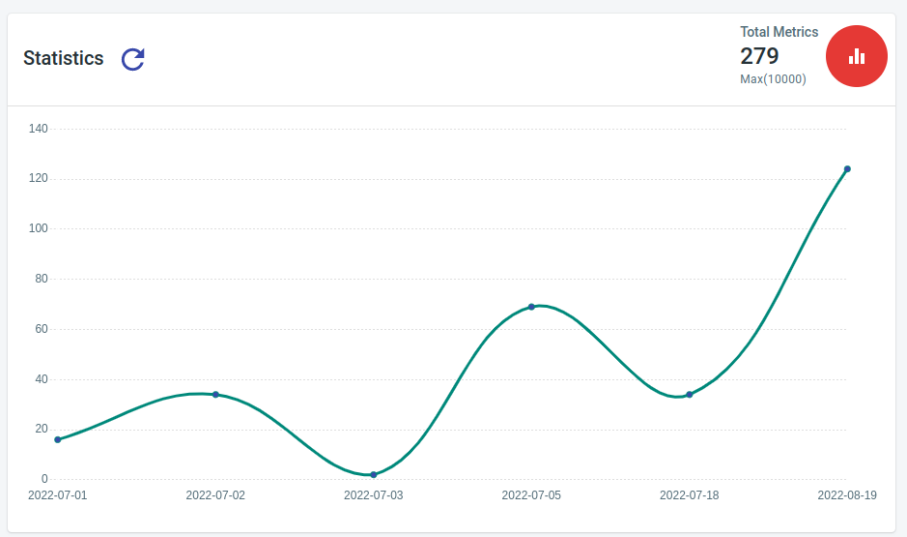


Figura 3.3: Grafico a linea delle metriche totali prodotte

- Un grafico a ciambella che mostra il tipo di interazioni che sono avvenute durante la scrittura del codice nell'arco di tempo selezionato.
  È possibile osservare il tipo e la quantità di interazioni avvenute durante la scrittura del codice: aggiunta, modifica e cancellazione di linee di codice. È molto utile per capire quanto codice effettivamente viene prodotto e la relativa qualità.
  Ad esempio, se la percentuale di linee cambiate è maggiore delle linee aggiunte molto probabilmente il codice già scritto presentava dei problemi o non era scritto seguendo dei design pattern, quindi ha necessitato di molte modifiche e refactoring.

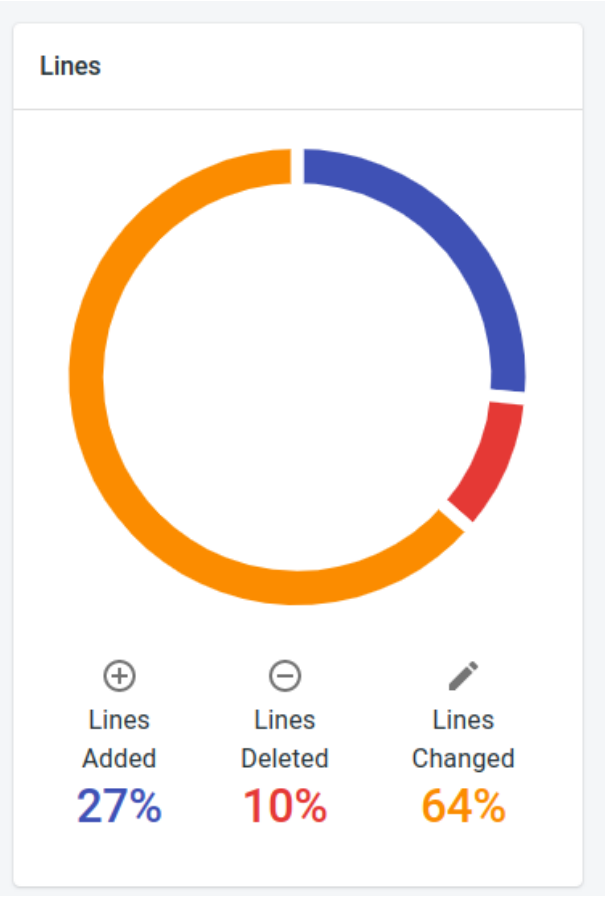


Figura 3.4: Grafico a ciambella di tutte le interazioni con il codice

- Un grafico a barre che espone i file sui quali si è lavorato mostrando il numero di metriche per file e il tempo di interazione nel periodo selezionato.
  Le metriche riportate in questo grafico sono interessanti perché permettono di osservare su quali file l'attività di sviluppo si è concentrata, consentendo di individuare le parti del codice più complesse e suggerendo di conseguenza dove è più probabile la presenza di bug o eventuali bottleneck.

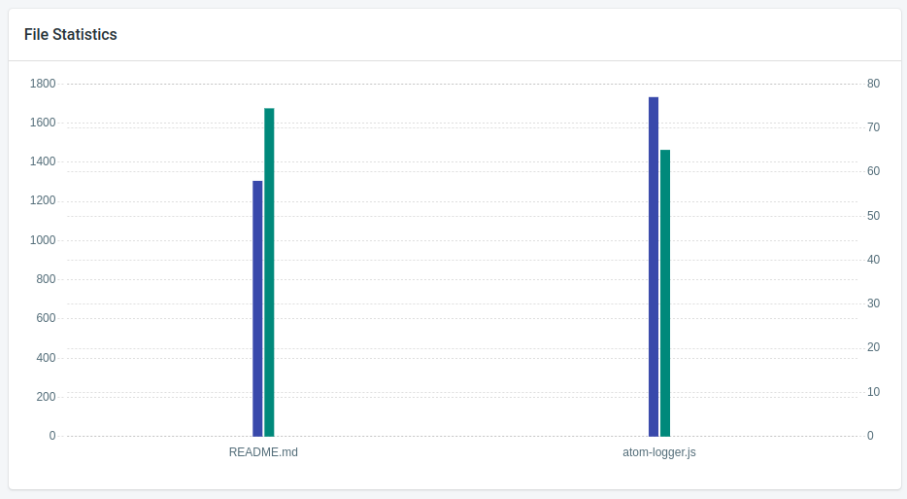


Figura 3.5: Grafico a barre di tutte le interazioni con i file

3.1.2 GitLab

Nelle sezione GitLab gli utenti possono analizzare le statistiche dei diversi repository di cui sono collaboratori.
Questa sezione presenta sulla sinistra un grafico a ciambella che mostra il contributo di ogni utente alla repository visualizzando il numero di commit pro capite. Sulla destra, invece, è mostrata una lista dei membri del gruppo, con un link diretto al loro profilo GitLab.

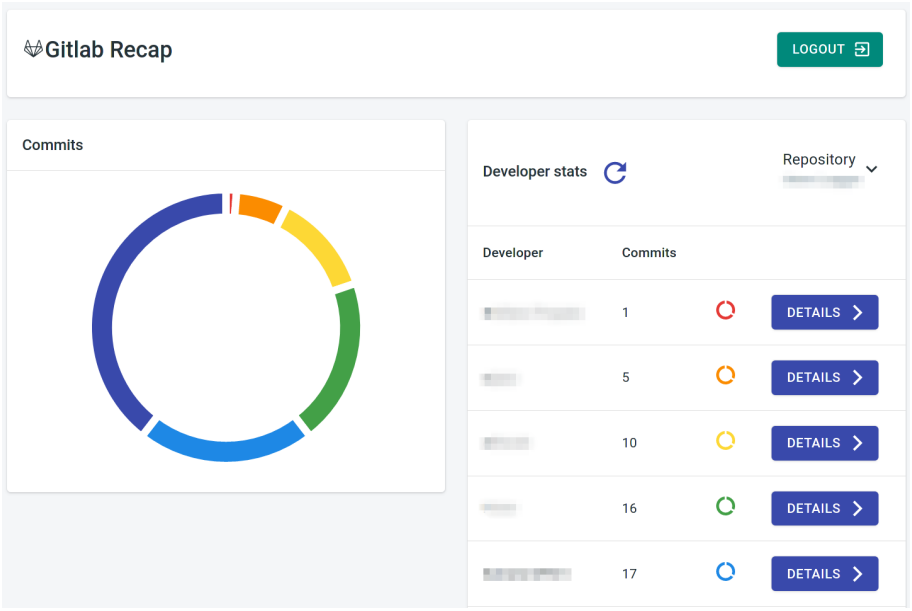


Figura 3.6: Sezione Gitlab di CAS-Dashboard

3.1.3 Taiga

Per quanto riguarda Taiga vengono mostrate due sezioni:
La prima presenta due elementi:

- un grafico a ciambella che mostra la percentuale di task e user stories completate;
- I ruoli dello sviluppatore nei progetti di cui fa parte;
- Link diretti ai progetti;

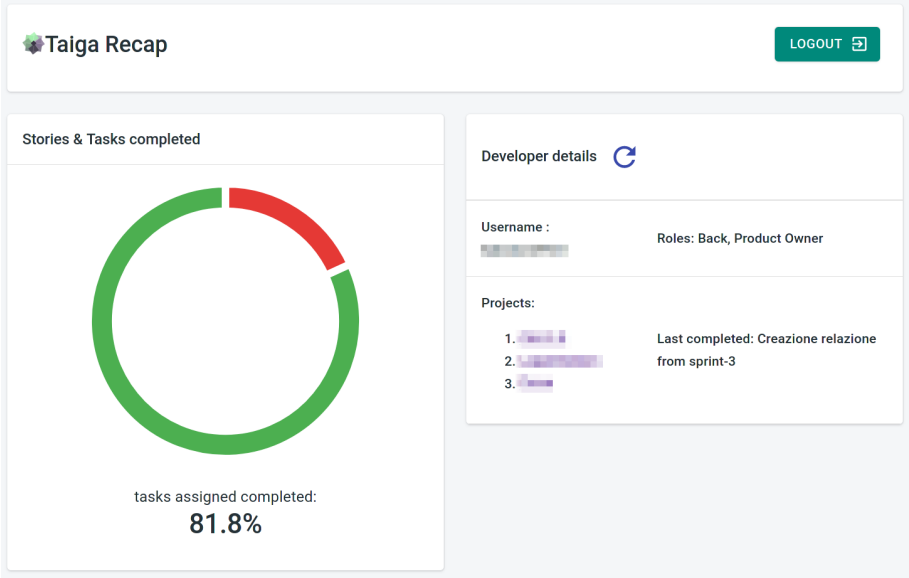


Figura 3.7: Sezione Taiga di CAS-Dashboard

La seconda sezione presenta un grafico con il numero di compiti completati giornalmente.

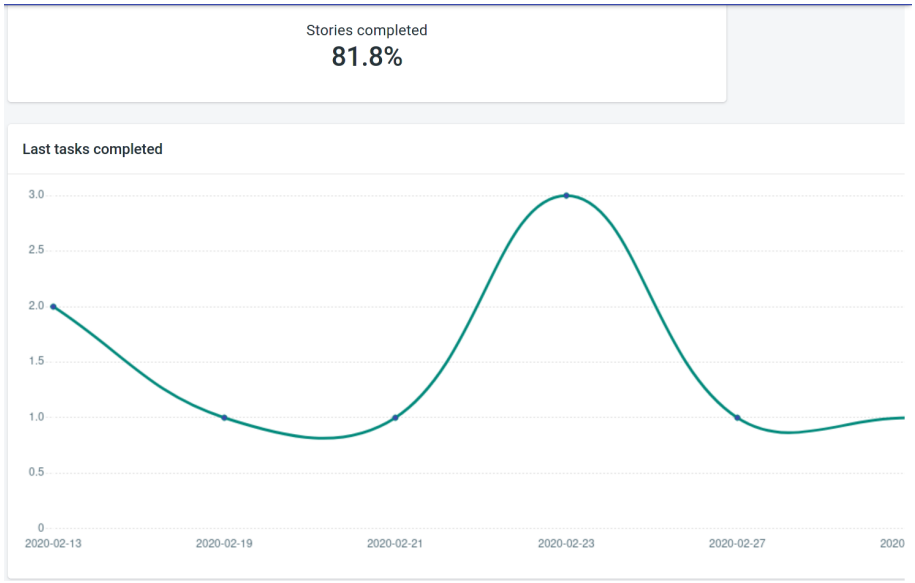


Figura 3.8: Sezione Taiga di CAS-Dashboard 2

3.1.4 SonarQube

Nella pagina di SonarQube vengono riportati i dati di ogni progetto che l'utente ha sottoposto ad analisi:

- Nome del progetto;
- Quality gate di SonarQube;
- Data dell'ultima analisi;
- Debito tecnico calcolato;
- Link alla pagina SonarQube del progetto;

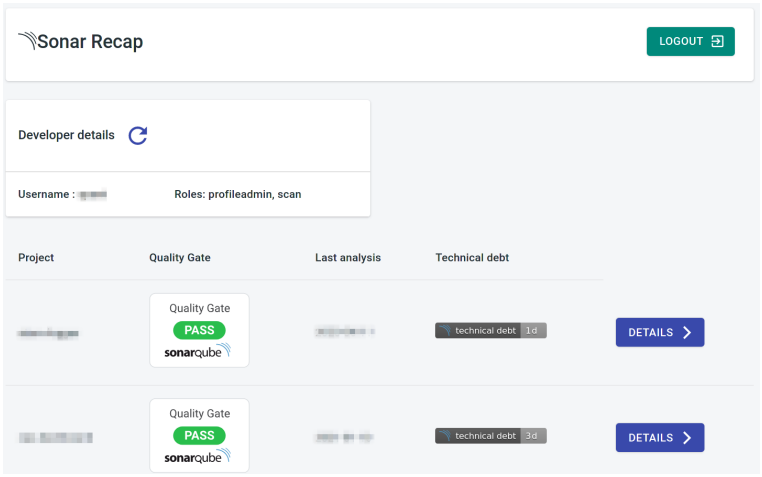


Figura 3.9: Sezione SonarQube di CAS-Dashboard

L'obiettivo di questa dashboard è di fornire una overview su tutto il lavoro svolto usando l'IDE nell'arco di tempo selezionato, permettendo a ogni sviluppatore di individuare più facilmente possibili margini di miglioramento o di scoprire problematiche dovute al processo di sviluppo software.
Ogni statistica fornita è volutamente non troppo specifica sulle attività svolte per impedire un'interpretazione vincolata da bias cognitivi e permette di non basarsi unicamente sulle metriche raccolte per trarre conclusioni, fornendo solo indizi utili per ulteriori indagini che il team o lo sviluppatore singolo svolgeranno in seguito.

Capitolo 4

Conclusioni

4.1 Contributo di questa tesi

Le dashboard sono strumenti più complessi di quanto sembri all'apparenza.
In questo elaborato si è discussa la loro utilità e i limiti alle quali sono sottoposte nell'ingegneria del software.
Inoltre è stata ipotizzata una loro possibile evoluzione nel prossimo futuro, data la loro popolarità in costante crescita.
È stato descritto l'ambiente CAS 2.0, Client e Server, definendo ogni microservizio, in particolare il logger esponendone le API attualmente disponibili.
Sono state analizzate le dashboard utilizzate in CAS e come si interfacciano con gli altri microservizi disponibili in CAS.

4.2 Possibili lavori futuri

Di seguito alcune proposte di funzionalità aggiuntive implementabili in CAS.

4.2.1 Aggregare i database

Attualmente ogni servizio utilizza il proprio database attraverso un altro container. Si potrebbe utilizzare un unico database, così facendo diminuirebbe il numero di processi attivi e renderebbe più facile esportare tutti i dati sensibili del server.

4.2.2 Ampliare la portata di CAS-dashboard

Attualmente la dashboard visualizza solo le metriche individuali di uno sviluppatore, si potrebbero aggregare i dati per concedere una visione d'insieme a un team o progetto.

4.2.3 Inserire visione PO in CAS-Dashboard

Poiché questo sistema è stato pensato per essere utilizzato durante il corso di Ingegneria del Software, dovrà essere possibile per il PO, in questo caso il professore, avere sotto controllo i dati relativi ai team e ai singoli studenti.
Si può pensare di inserire una visione Amministratore della Dashboard ampliando il backend del logger o svilupparne uno che lavori esclusivamente con la Dashboard.

4.2.4 Estendere il supporto ad altri IDE

CAS è compatibile con i principali IDE, ma per rendere il servizio ancora più capillare si potrebbero sviluppare dei plug-in per IDE e O.S. molto diffusi, come CodeBlocks, NEtBeans, AndroidStudio, Vim ecc.

4.2.5 Semplificare la fase di autenticazione

Attualmente ogni nuovo utente che voglia usufruire dei servizi di CAS deve registrare un account su ciascuno di essi.
Rendere sufficiente un unico account, ad esempio implementando un meccanismo quale il single-sign-on o sfruttando le credenziali istituzionali, comporterebbe un notevole miglioramento per la facilità d'utilizzo da parte dell'utente finale.

4.2.6 Implementare aggiornamenti automatici

Il server dovrebbe essere capace di effettuare aggiornamenti automatici attraverso il proprio microservizio Jenkins.
Andrebbe fatta una valutazione per valutare se tutti i servizi possano aggiornarsi autonomamente o creare uno script che modifichi il dockercompose per aggiornamenti "manuali".

Riferimenti bibliografici

[1] AAVV. Docker documentation. `https://docs.docker.com`.

[2] AAVV. Gitlab documentation. `https://docs.gitlab.com/`.

[3] AAVV. Jenkins documentation. `https://www.jenkins.io/doc/`.

[4] AAVV. Mattermost documentation. `https://docs.mattermost.com/`.

[5] AAVV. Nginx documentation. `https://nginx.org/en/docs/`.

[6] AAVV. Sonarqube documentation. `https://docs.sonarqube.org/latest/`.

[7] AAVV. Taiga documentation. `https://taigaio.github.io/taiga-doc/dist/`.

[8] Jorge Aranda and Gina Venolia. The secret life of bugs: Going past the errors and omissions in software repositories. In *Proc. IEEE 31st International conference on Software engineering*, pages 298–308. IEEE, 2009.

[9] Richard Brath and Michael Peters. Dashboard design: Why design is important. *DM Direct*, 85:1011285–1, 2004.

[10] Thomas Zimmermann Caitlin Sadowski. *Rethinking Productivity in Software Engineering*. APress Open, 2019.

[11] Paolo Ciancarini, Marcello Missiroli, Francesco Poggi, and Daniel Russo. An open source environment for an agile development model. In *IFIP International Conference on Open Source Systems*, pages 148–162. Springer, 2020.

[12] Gregory L Hovis. Stop searching for information–monitor it with dashboard technology. *DM Direct, February*, 20, 2002.

[13] Salvatore Perri. Progettazione di una dashboard per sviluppatori agili. Tesi di Laurea Triennale in Informatica, Università di Bologna, 2021.

[14] Stefano Propato. logger-backend github repository. `https://github.com/elPeroN/logger-backend.`

[15] Stefano Propato. Compositional agile system: un ambiente di extreme development. Tesi di Laurea Triennale in Informatica, Università di Bologna, 2021.

[16] Stefano Propato and Salvatore Perri. Cas-dashboard github repository. `https://github.com/elPeroN/CAS-dashboard`.

[17] Andrea Schinoppi. La misurazione del teamwork nei progetti agili. Tesi di Laurea Triennale in Informatica, Università di Bologna, 2022.

[18] Salvatore Perri Stefano Propato. Cas-server documentation. `https://github.com/elPeroN/CAS-Server`.

[19] Margaret-Anne Storey and Alexey Zagalsky. Disrupting developer productivity one bot at a time. In *Proceedings of 24th ACM SIGSOFT International symposium on Foundations of Software engineering*, pages 928–931, 2016.

[20] Christoph Treude, Fernando Figueira Filho, and Uirá Kulesza. Summarizing and measuring development activity. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*, pages 625–636, 2015.

[21] Christoph Treude and Margaret-Anne Storey. Awareness 2.0: staying aware of projects, developers and tasks using dashboards and feeds. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 365–374, 2010.

Ringraziamenti

Primo tra tutti ringrazio il professor Paolo Ciancarini per avermi concesso l'opportunità di mettermi alla prova nello sviluppo del software che ha dato origine a questa tesi.

Ringrazio la mia famiglia, i miei colleghi e tutti gli amici che mi hanno sostenuto durante tutto il periodo universitario senza i quali non sarei dove sono ora.

Ringrazio i gruppi "ER KLAN" e "Queen's Empire" per essere degli amici sempre disponibili e affidabili.

Infine un particolare ringraziamento lo dedico al mio stimato amico e collega Andrea Schinoppi con il quale ho sempre lavorato in maniera impeccabile e spero di continuare in futuro.