UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Software Architecture Design

Corso di Laurea Magistrale in Ingegneria Informatica

Dipartimento di Ingegneria Elettrica e delle tecnologie dell'informazione

GRUPPO 21-TASK8

Emanuele Agostino Messuri M63001547
Teresa Di Dona M63001437
Preziosa Pietroluongo M63001456
Pasquale Carusone M63001459

ANNO ACCADEMICO 2022/2023
Secondo Semestre

Sommario

1. Descrizione del progetto

1.1 Specifiche di progetto

L'applicazione deve offrire la funzionalità di esecuzione del robot Evosuite su una data classe Java. Tale funzionalità riceverà in input un file di testo (classe da testare), dovrà lanciare il generatore di Test, restituendo in output le informazioni relative all'esito dei test del Robot. L'esito dell'esecuzione dovrà essere elaborato in maniera da estrarre da essi le informazioni rilevanti ai fini del gioco (ad esempio la copertura del codice, decisioni, etc.).

1.2 Processo di sviluppo

Sono state adottate metodologie agili al fine di avere i vantaggi di un approccio iterativo ed evolutivo, con la pianificazione di iterazioni brevi ed eventuali feedback. In particolare, si è scelto di seguire il processo di sviluppo Agile Unified Process (AUP), il quale è stato organizzato suddividendo il lavoro di sviluppo software in quattro iterazioni.

Ogni iterazione produce un sistema eseguibile, testato ed integrato con quanto prodotto nelle cinque precedenti iterazioni; comprende attività di analisi dei requisiti, progettazione ed implementazione. Il processo di sviluppo AUP evolve attraverso cicli di "Costruzione-Feedback-Adattamento" e risulta essere molto flessibile: durante ogni iterazione, il team ha aggiornato la documentazione affinché risulti coerente con il sistema in produzione. I feedback ricevuti andranno poi ad influenzare le attività da eseguire nelle future iterazioni.

Il processo di sviluppo è costituito dalle seguenti quattro fasi:

• Ideazione: fase di pianificazione degli obiettivi generali e stima dei costi e dei tempi;

• Elaborazione: fase di implementazione del nucleo dell'architettura del software da realizzare e risoluzione di rischi maggiori;

• Costruzione: fase di implementazione interattiva degli elementi rimanenti, a minor costo;

• Transizione: fase in cui avviene il testing complessivo, il completamento della documentazione ed il rilascio

1.3 Gestione del gruppo

Il team è composto da quattro membri; ogni membro ha lavorato attivamente a tutte le fasi di sviluppo dell'architettura proposta. Alcune attività sono state svolte in coppia, ma al termine di ogni fase il team si è riunito per discutere dei risultati ottenuti e delle modalità di procedimento. Per coordinare tutti i membri del gruppo abbiamo utilizzato la piattaforma **Notion**.

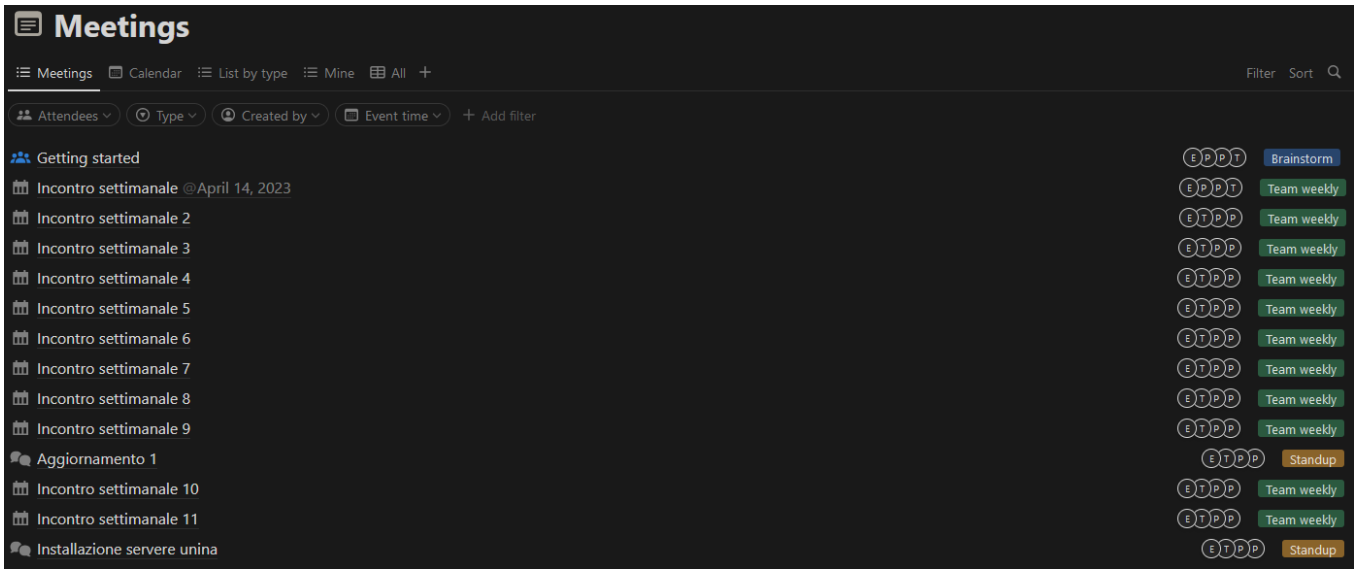


Figura 1: Meetings

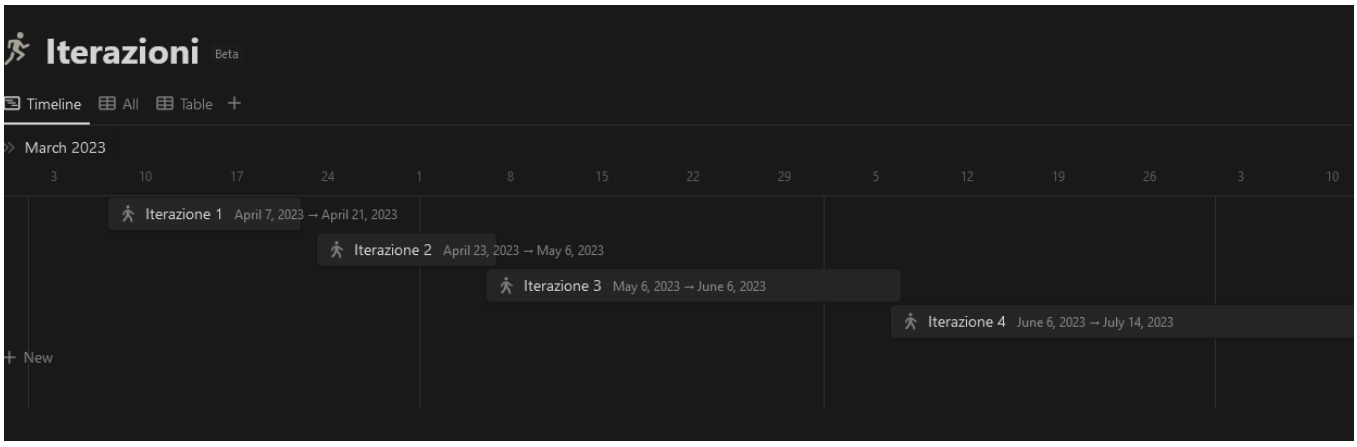
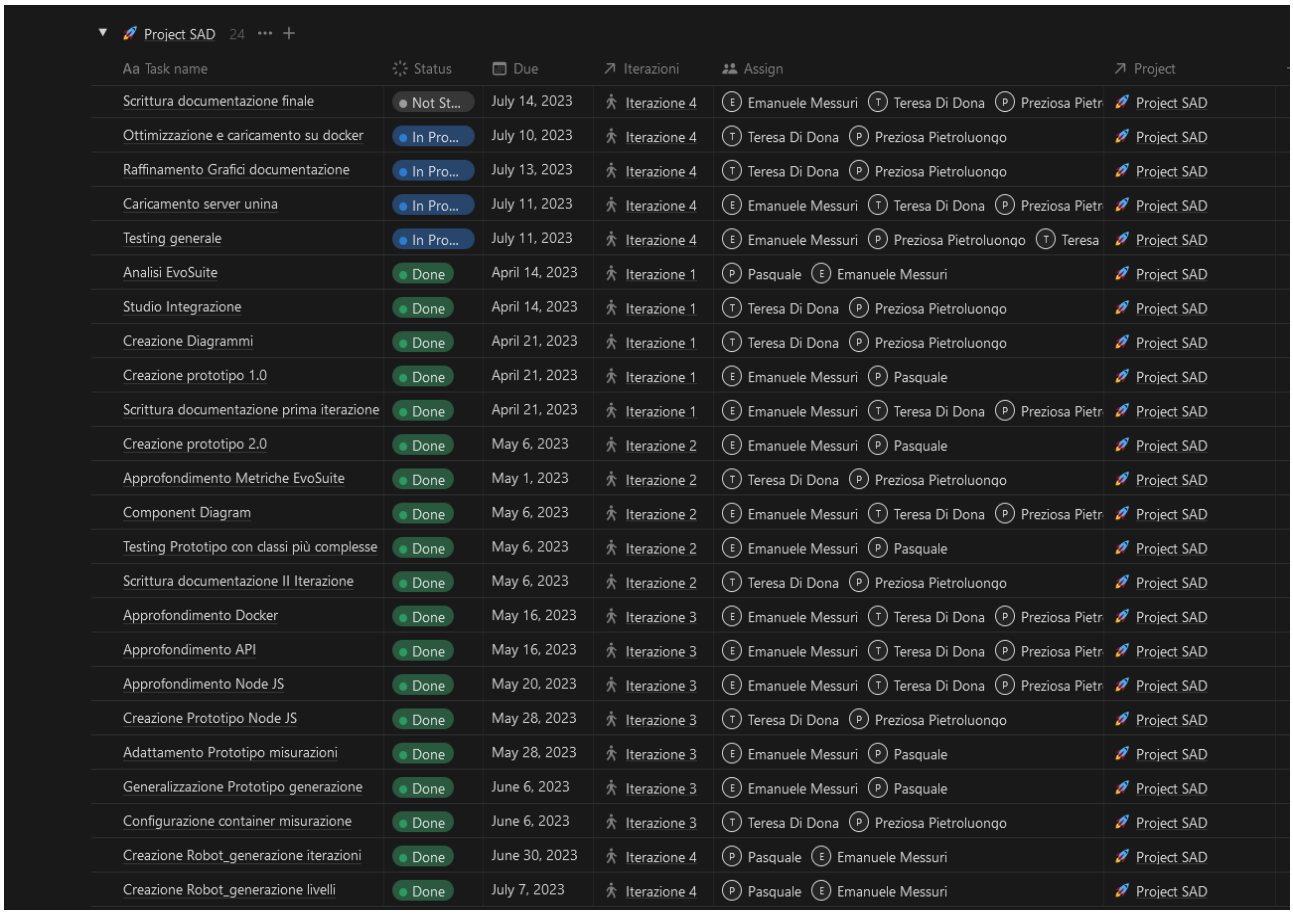


Figura 2:Iterazioni



Project SAD 24

Task name	Status	Due	Iterazioni	Assign	Project
Scrittura documentazione finale	Not St...	July 14, 2023	Iterazione 4	Emanuele Messuri, Teresa Di Dona, Preziosa Pietr	Project SAD
Ottimizzazione e caricamento su docker	In Pro...	July 10, 2023	Iterazione 4	Teresa Di Dona, Preziosa Pietroluongo	Project SAD
Raffinamento Grafici documentazione	In Pro...	July 13, 2023	Iterazione 4	Teresa Di Dona, Preziosa Pietroluongo	Project SAD
Caricamento server unina	In Pro...	July 11, 2023	Iterazione 4	Emanuele Messuri, Teresa Di Dona, Preziosa Pietr	Project SAD
Testing generale	In Pro...	July 11, 2023	Iterazione 4	Emanuele Messuri, Preziosa Pietroluongo, Teresa	Project SAD
Analisi EvoSuite	Done	April 14, 2023	Iterazione 1	Pasquale, Emanuele Messuri	Project SAD
Studio Integrazione	Done	April 14, 2023	Iterazione 1	Teresa Di Dona, Preziosa Pietroluongo	Project SAD
Creazione Diagrammi	Done	April 21, 2023	Iterazione 1	Teresa Di Dona, Preziosa Pietroluongo	Project SAD
Creazione prototipo 1.0	Done	April 21, 2023	Iterazione 1	Emanuele Messuri, Pasquale	Project SAD
Scrittura documentazione prima iterazione	Done	April 21, 2023	Iterazione 1	Emanuele Messuri, Teresa Di Dona, Preziosa Pietr	Project SAD
Creazione prototipo 2.0	Done	May 6, 2023	Iterazione 2	Emanuele Messuri, Pasquale	Project SAD
Approfondimento Metriche EvoSuite	Done	May 1, 2023	Iterazione 2	Teresa Di Dona, Preziosa Pietroluongo	Project SAD
Component Diagram	Done	May 6, 2023	Iterazione 2	Emanuele Messuri, Teresa Di Dona, Preziosa Pietr	Project SAD
Testing Prototipo con classi più complesse	Done	May 6, 2023	Iterazione 2	Emanuele Messuri, Pasquale	Project SAD
Scrittura documentazione II Iterazione	Done	May 6, 2023	Iterazione 2	Teresa Di Dona, Preziosa Pietroluongo	Project SAD
Approfondimento Docker	Done	May 16, 2023	Iterazione 3	Emanuele Messuri, Teresa Di Dona, Preziosa Pietr	Project SAD
Approfondimento API	Done	May 16, 2023	Iterazione 3	Emanuele Messuri, Teresa Di Dona, Preziosa Pietr	Project SAD
Approfondimento Node JS	Done	May 20, 2023	Iterazione 3	Emanuele Messuri, Teresa Di Dona, Preziosa Pietr	Project SAD
Creazione Prototipo Node JS	Done	May 28, 2023	Iterazione 3	Teresa Di Dona, Preziosa Pietroluongo	Project SAD
Adattamento Prototipo misurazioni	Done	May 28, 2023	Iterazione 3	Emanuele Messuri, Pasquale	Project SAD
Generalizzazione Prototipo generazione	Done	June 6, 2023	Iterazione 3	Emanuele Messuri, Pasquale	Project SAD
Configurazione container misurazione	Done	June 6, 2023	Iterazione 3	Teresa Di Dona, Preziosa Pietroluongo	Project SAD
Creazione Robot_generazione iterazioni	Done	June 30, 2023	Iterazione 4	Pasquale, Emanuele Messuri	Project SAD
Creazione Robot_generazione livelli	Done	July 7, 2023	Iterazione 4	Pasquale, Emanuele Messuri	Project SAD

Figura 3: Tasks

2. Documentazione di analisi

Il Documento di Analisi definisce l'approccio e i dettagli dell'analisi, per il sistema di misurazione e generazione che sarà sviluppato. Lo scopo di questa analisi è comprendere a fondo le esigenze degli utenti, identificare i requisiti funzionali e non funzionali e definire una solida base per la progettazione del sistema di misurazione e generazione.

2.1 Documento dei requisiti

Il **Documento dei Requisiti** definisce le specifiche e le esigenze per il sistema che sarà sviluppato. Questo sistema avrà il compito di offrire un servizio offline di generazione dei test, divisi per livelli, e uno online di misurazione della copertura di una suite di test creata dall'utente. Il sistema sarà implementato come parte di una piattaforma o di un'applicazione più ampia, che richiede un'adeguata gestione. Inoltre, esso ha lo scopo di fornire una visione chiara e dettagliata dei requisiti funzionali e non funzionali del sistema da noi creato. È destinato agli sviluppatori, ai progettisti e agli stakeholder coinvolti nel progetto, al fine di garantire una comprensione comune delle funzionalità richieste e delle aspettative nei confronti del sistema.

2.1.1 Requisiti informali

Il sistema deve sia gestire una richiesta di un utente esterno, offrendo la possibilità di compilare ed eseguire casi di test per una determinata classe, che generare i diversi livelli del gioco. In entrambi i casi, il sistema deve consentire la misura della copertura del codice.

Per soddisfare questi requisiti, si utilizzano due diversi container ubuntu docker:

- sul primo viene gestita la generazione dei livelli che comprende, per ogni livello, la creazione dei rispettivi test e la relativa misura della copertura. Tale misura (salvata in un file "statistics.csv") e, i relativi test, dovranno essere salvati nell'opportuna directory di un repository condiviso. Questo primo container, una volta generati i test, verrà spento a tempo di gioco.
- sul secondo viene soddisfatta la richiesta dell'utente. Il sistema deve prevedere un accesso ad un repository condiviso sia per prelevare la classe sotto test e i test stessi, che per il salvataggio del file "statistics.csv". Questo secondo container sarà attivo per tutta la durata del gioco.

Per gestire la richiesta dall'utente, viene realizzato un server con Node.js. La richiesta deve prevedere: percorso della classe, percorso dei test, percorso di salvataggio. In risposta all'utente verrà fornito un messaggio di corretta esecuzione dell'operazione, specificando il path di salvataggio del file ".csv".

Il servizio deve offrire una documentazione per aiutare colui che utilizza il sistema ad avviare la procedura.

2.1.2 Requisiti funzionali

R01: Il servizio deve garantire la corretta compilazione ed esecuzione della classe sotto test

R02: Il servizio deve restituire un messaggio di errore in caso di fallimento delle operazioni

R03: Il servizio deve prevedere in ingresso il nome della classe sotto test, il nome e il percorso del package contente la classe e, infine, il numero di livelli selezionato

R04: Il servizio deve garantire la corretta generazione dei test per i diversi livelli

R05: Il servizio deve selezionare il numero di livelli richiesto

R06: Il servizio deve fornire la misurazione della copertura per ogni livello richiesto, al fine di valutare l'efficacia dei test

R07: Il servizio deve garantire che la generazione dei livelli sia offline rispetto al gioco

R08: Il servizio deve garantire il corretto salvataggio dei test dei livelli nel repository condiviso

R09: Il servizio deve offrire la misura della copertura dei test su richiesta dell'utente

R10: Il servizio deve prendere in ingresso il percorso fino alla classe, il percorso del package contente i test, il percorso di salvataggio dei risultati della copertura

R11: Il servizio deve estrapolare, dal primo percorso, il nome della classe e del package che la contiene e il percorso della classe

R12: Il servizio deve notificare al chiamante la corretta esecuzione, specificando il percorso di salvataggio della misura effettuata (il percorso viene restituito per aggiornare l'utente di eventuali cambiamenti della destinazione di salvataggio rispetto a ciò che aveva specificato)

R13: Il servizio deve garantire che la misura della copertura su richiesta esterna sia online rispetto al gioco

R14: Il servizio deve garantire il corretto salvataggio dei file ".csv" nel repository condiviso

2.1.3 Requisiti non funzionali

- **Affidabilità:** il sistema deve essere in grado di funzionare in modo stabile e coerente nel tempo, minimizzando i guasti e le interruzioni e soddisfando i requisiti.
- **Usabilità:** il sistema deve essere facile da usare, comprensibile e intuitivo.
- **Manutenibilità:** il sistema deve essere facilmente manutenibile, consentendo modifiche, correzioni e aggiornamenti senza causare impatti negativi sulle funzionalità esistenti.
- **Interoperabilità:** il sistema deve essere in grado di scambiare dati e comunicare con altri sistemi in modo efficace.
- **Disponibilità:** il sistema deve essere disponibile per l'uso.

2.2 Scelte progettuali

2.2.1 Generazione livelli

Durante la fase di progettazione della parte riguardante la generazione dei livelli, si è deciso di usare un algoritmo che campionasse, a partire da 13 iterazioni, le N classi di test necessarie alla creazione dei livelli richiesti in input. Per la generazione delle iterazioni, abbiamo scelto dei set di criteri di generazione, con lo scopo di ottenere valori di copertura diversi tra loro. I set scelti per le varie iterazioni sono:

1. Tutte
2. Line
3. Branch
4. Weak Mutation
5. Output
6. Method
7. Line-branch

8. Output-line
9. Weak Mutation-branch
10. Weak Mutation-output
11. C-branch-line
12. Line-branch-exception
13. Output-line-weak mutation

I set 2-6 hanno l'obiettivo di creare dei test che vadano a verificare le caratteristiche "base" della classe da testare, mentre i set successivi sono stati creati a partire da evidenze sperimentali ottenute eseguendo i set su varie classi diverse. In generale sono stati scelti dei set che producessero dei valori di copertura diversi tra loro.

2.2.2 Misurazione Utente

In questa sezione, riportiamo una descrizione della realizzazione del server con nodejs. Per la realizzazione del servizio, sono stati realizzati due file .js: *prova_esecuzione_parametri4.js* e *child.js.*

Nel primo file, viene creato un server HTTP utilizzando la funzione *createServer* del modulo http. All'arrivo di una richiesta, viene controllato se l'URL inizia con '/api/'. In questo caso, la stringa dell'URL viene manipolata per ottenere i parametri necessari per l'avvio ed il corretto funzionamento della misurazione della copertura. Viene rimossa la parte *'/api/'*; viene estratto il primo elemento della richiesta, ovvero il percorso fino alla classe Java, e da quest'ultimo vengono estrapolati il path fino al package, il nome della classe Java e il nome del package. Viene, quindi concatenata una stringa contenente tutti i parametri (nome della classe, nome del package, percorso del package, percorso dei test e percorso di salvataggio), la quale verrà inviata al file *robot_misurazione_utente.sh,* che potrà iniziare con la sua esecuzione.

Per mantenere una coerenza temporale, abbiamo impostato il server in modo da lasciarlo in attesa durante l'esecuzione del bash che effettua la misurazione, per permettergli di terminare il proprio lavoro e di creare e salvare il file *statistics.csv*. In particolare, abbiamo creato un server figlio (ovvero il secondo file, child.js) con una fork, che attende una richiesta di terminazione da parte del bash. Una volta ricevuta, il Server "figlio" invierà un messaggio al "padre" che, nel mentre, era in attesa. A

questo punto, l’operazione del server è completata e il client, in risposta, riceverà la conferma del successo dell’operazione.

2.3 Diagramma di contesto

Il **diagramma di contesto** rappresenta una panoramica ad alto livello del sistema in analisi e delle sue interazioni con gli attori esterni. Questo diagramma fornisce una visione chiara e concisa del modo in cui il sistema si integra con l'ambiente circostante e interagisce con gli altri componenti dell’applicazione.

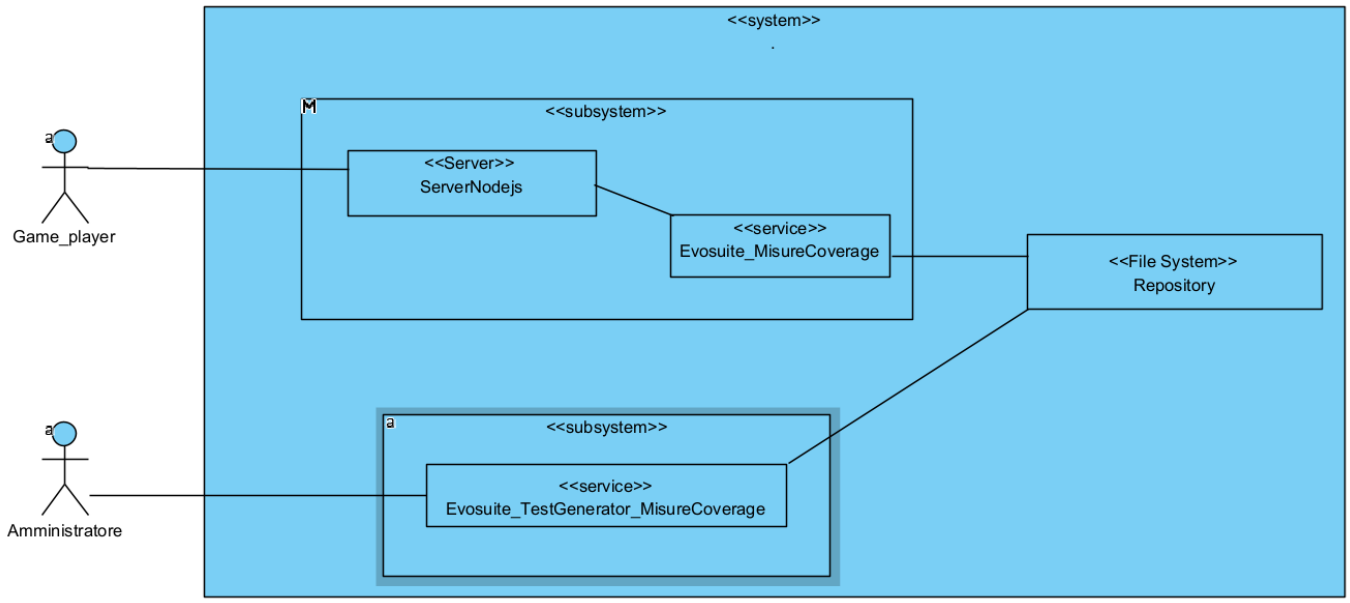


Figura 4: Diagramma di contesto

2.4 Storie Utente

Le **storie utente** hanno lo scopo di identificare e documentare in modo dettagliato le esigenze e le aspettative degli utenti finali. Esse descrivono le attività specifiche che gli utenti desiderano svolgere all'interno del sistema e offrono una guida per la progettazione e lo sviluppo delle funzionalità.

USER STORY TITLE:
MISURATION

AS I *GAMEPLAYER*

I WANT TO MAKE A REQUEST FOR THE MISURE COVERAGE AT ANY TIME

SO THAT I CAN KNOW THE EFFECTIVENESS OF MY TESTS

USER STORY TITLE:
SAVING FILE

AS I *GAMEPLAYER*

I WANT TO SPECIFY THE SAVING PATH

SO THAT I CAN CHOOSE THE DESTINATION OF THE FILE

USER STORY TITLE:
TEST GENERATION

AS I ADMINISTRATOR

I WANT TO START BUILDING THE TEST SUITES

SO THAT IS POSSIBLE TO CREATE THE LEVELS OF THE GAME

USER STORY TITLE: **LEVEL GENERATION**

AS I ADMINISTRATOR

I WANT TO SPECIFY THE NUMBER OF LEVELS OF THE GAME

SO THAT I CAN CHOOSE HOW MANY DIFFICULTY LEVELS THE GAME SHOULD HAVE

2.4.1 Criteri di accettazione

USER STORY TITLE:
MISURATION

AS I *GAMEPLAYER*

I WANT TO MAKE A REQUEST FOR THE MISURE COVERAGE AT ANY TIME

SO THAT I CAN KNOW THE EFFECTIVENESS OF MY TESTS

ACCEPTANCE CRITERION 1:

GIVEN A WRONG CLASS PATH

WHEN THE *GAMEPLAYER* SEND THE REQUEST

THEN THE SYSTEM RETURNS AN ERROR MESSAGE

AND THE MEASUREMENT CANNOT BE PERFORMED

ACCEPTANCE CRITERION 3:

GIVEN A RIGHT CLASS PATH
AND A RIGHT TEST PATH

WHEN THE *GAMEPLAYER* SEND THE REQUEST

THEN THE MEASUREMENT WAS PERFORMED SUCCESSFULLY

AND THE RESULTS ARE SAVED

ACCEPTANCE CRITERION 2:

GIVEN A WRONG TEST PATH

WHEN THE *GAMEPLAYER* SEND THE REQUEST

THEN THE SYSTEM RETURNS AN ERROR MESSAGE

AND THE MEASUREMENT CANNOT BE PERFORMED

USER STORY TITLE:
SAVING FILE

AS I *GAMEPLAYER*

I WANT TO SPECIFY THE SAVING PATH

SO THAT I CAN CHOOSE THE DESTINATION OF THE FILE

ACCEPTANCE CRITERION 1:

GIVEN A NON-EXISTENT FILE SAVE PATH

WHEN THE *GAMEPLAYER* SPECIFY THE SAVING PATH

THEN THE SYSTEM RETURNS AN ERROR MESSAGE

AND FILE SAVING CANNOT OCCUR

USER STORY TITLE:
TEST GENERATION

AS I ADMINISTRATOR

I WANT TO START BUILDING THE TEST SUITES

SO THAT IS POSSIBLE TO CREATE THE LEVELS OF THE GAME

ACCEPTANCE CRITERION 1:

GIVEN A WRONG CLASS PATH

WHEN THE ADMINISTRATOR START BUILDING THE TEST SUITES

THEN THE SYSTEM RETURNS AN ERROR MESSAGE

AND THE GENERATION DOESN'T HAPPEN

USER STORY TITLE: **LEVEL GENERATION**

AS I ADMINISTRATOR

I WANT TO SPECIFY THE NUMBER OF LEVELS OF THE GAME

SO THAT I CAN CHOOSE HOW MANY DIFFICULTY LEVELS THE GAME SHOULD HAVE

ACCEPTANCE CRITERION 1:

GIVEN A NUMBER OF LEVELS IS EQUAL TO 0

WHEN THE ADMINISTRATOR SPECIFY THE NUMBER OF LEVELS OF THE GAME

THEN THE SYSTEM RETURNS AN ERROR MESSAGE

AND LEVELS CANNOT BE GENERATED

ACCEPTANCE CRITERION 2:

GIVEN A NUMBER OF LEVELS BETWEEN 1 AND 9

WHEN THE ADMINISTRATOR SPECIFY THE NUMBER OF LEVELS OF THE GAME

THEN THE SYSTEM SUCCESSFULLY CREATE LEVELS

AND SAVE THEM

2.5 Diagramma dei Casi d'uso

Il **diagramma dei casi d'uso** rappresenta le interazioni tra gli utenti e il sistema, mettendo in evidenza le funzionalità e le azioni che possono essere eseguite dagli attori coinvolti.

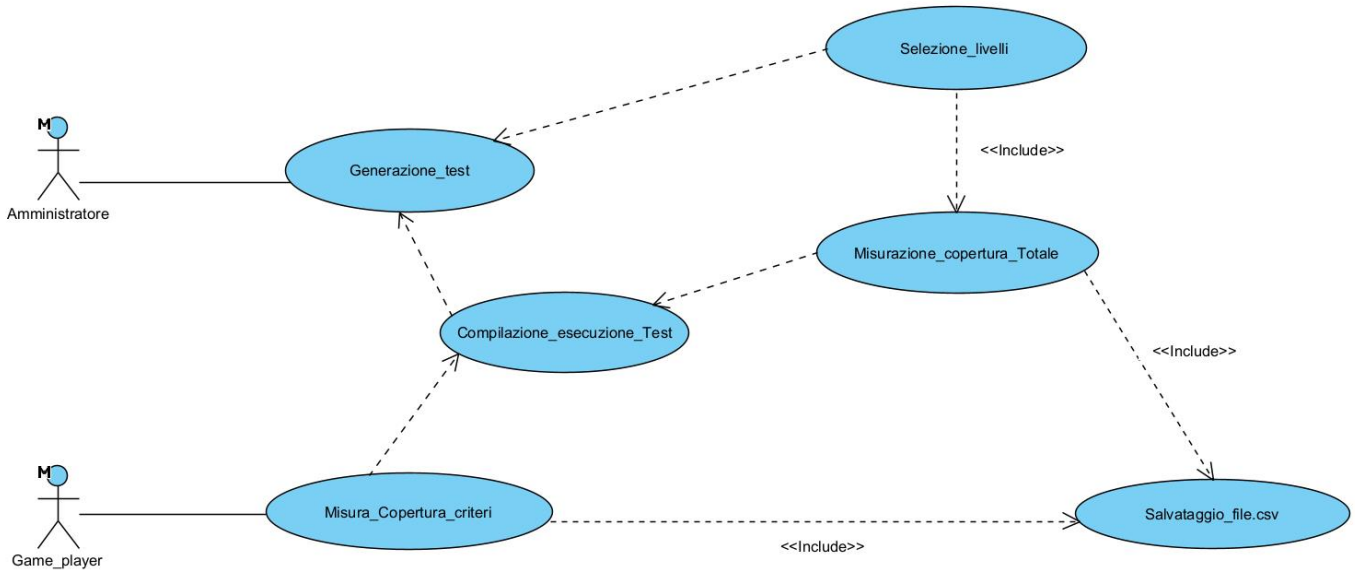


Figura 5: Diagramma dei casi d'uso

2.6 Scenari

Riportiamo, di seguito, alcuni scenari d'uso per rappresentare in maniera più esplicita il comportamento del sistema:

CASO D'USO:	***GENERAZIONE TEST***
ATTORI:	Amministratore
DESCRIZIONE:	Generazione di tredici differenti "iterazioni" di test usando combinazioni diverse dei criteri di copertura
PRECONDIZIONI:	Disporre della classe da testare e specifica dei parametri di configurazione per la generazione dei test (come, ad esempio, il percorso del codice sorgente da testare, le dipendenze richieste)
SEQUENZA DEGLI EVENTI:	1. L'amministratore avvia la generazione dei test, indicando il nome della classe, il nome del package contente la classe, il percorso del package e il numero di livelli che si vuole generare. 2. Il sistema verifica che: • la classe sia presente nel package specificato • il percorso sia corretto • siano presenti le dipendenze richieste 3. Il processo di generazione avviene attraverso Evosuite che fa uso di tecniche basate sugli algoritmi genetici. 4. Vengono prodotti 13 diversi casi di test, ottenuti attraverso la combinazione delle varie metriche di copertura.
POSTCONDIZIONE:	I vari test vengono salvati nell'opportuna directory I-iesima.
SEQUENZA ALTERNATIVA:	2.a La classe non è presente nel package 2.b Il percorso specificato non è corretto 2.c Le dipendenze richieste per il corretto funzionamento di Evosuite non sono disponibili

POSTCONDIZIONE:	Interruzione della generazione del test e visualizzazione di un messaggio di errore indicante la causa dell'interruzione.

CASO D'USO:	***SELEZIONE LIVELLI***
ATTORI:	Amministratore
DESCRIZIONE:	Selezione dei livelli scelti per il gioco
PRECONDIZIONE:	Generazione delle tredici iterazioni con misura della copertura totale del codice e scelta del numero di livelli da creare
SEQUENZA DEGLI EVENTI:	1. Creazione di una HashMap con chiave l'iterazione iesima e con valore la misura della copertura totale relativa 2. Ordinamento crescente della HashMap sulla base del valore della copertura 3. Sulla base di n livelli, si selezionano n iterazioni
POSTCONDIZIONE:	Generazione di un array che contiene nelle prime n posizione le iterazioni scelte e nelle successive il valore -1

CASO D'USO:	***COMPILAZIONE ED ESECUZIONE TEST***
ATTORI:	Amministratore/Game_player
DESCRIZIONE:	Maven gestisce la compilazione dei test e JUnit la loro esecuzione
PRECONDIZIONI:	• Per poter compilare i test con Maven, è necessario avere un progetto configurato correttamente utilizzando il file "pom.xml"(deve contenere tutte le informazioni necessarie come le dipendenze, ad esempio JUnit) per eseguire la compilazione dei test in modo adeguato. Ciò assicurerà che le librerie richieste siano scaricate e incluse nel classpath durante la compilazione e l'esecuzione dei test. • Il codice sorgente e i test devono essere organizzati in modo coerente e devono rispettare le convenzioni di Maven (ad esempio, i test dovrebbero essere collocati nella directory corretta) • Le classi di test devono avere il suffisso "_Test" o "_ESTest" nel loro nome, in modo che JUnit le riconosca come classi di test.
SEQUENZA DEGLI EVENTI:	1. Maven identifica i test automaticamente durante il processo di compilazione. 2. Durante la compilazione: • Maven compila sia il codice sorgente principale che i test • Viene controllato se il codice sorgente presenta errori 3. Vengono generati i file .class 4. Dopo la compilazione, si eseguono i test utilizzando il framework di test JUnit
POSTCONDIZIONE:	I risultati indicano quali test sono stati eseguiti correttamente (successo) e quali sono falliti
SEQUENZA ALTERNATIVA:	2.a Compilazione dei test non riuscita
POSTCONDIZIONE:	Visualizzazione di un messaggio di errore e mancata esecuzione dei test
SEQUENZA ALTERNATIVA:	4.a Esecuzione non riuscita
POSTCONDIZIONE:	Interruzione dell'esecuzione e visualizzazione di un messaggio di errore

CASO D'USO:	*MISURAZIONE COPERTURA TEST*
ATTORI:	Amministratore/Game_player
DESCRIZIONE:	Misurazione della copertura del codice di interesse, facendo uso di diverse metriche di copertura
PRECONDIZIONI:	• Codice sorgente compilato correttamente • Creazione della suite di test
SEQUENZA DEGLI EVENTI:	1. Raccolta dei dati di copertura generati durante l'esecuzione dei test (questi dati possono includere informazioni come la copertura delle istruzioni, la copertura dei rami e altre metriche di copertura specifiche). 2. Analisi di questi dati per valutare quanto il codice è stato coperto dai test.
POSTCONDIZIONE:	Creazione del file statistics.csv, contenente i risultati, e salvataggio nell'opportuna directory di destinazione.

2.7 Diagramma di attività

I **diagrammi di attività** offrono una rappresentazione visuale del flusso di lavoro o dei processi all'interno del sistema. Questi diagrammi consentono di modellare le attività, le decisioni e le scelte durante il processo, fornendo una panoramica chiara e intuitiva delle azioni eseguite dagli utenti e dal sistema.

2.7.1 Misurazione Utente

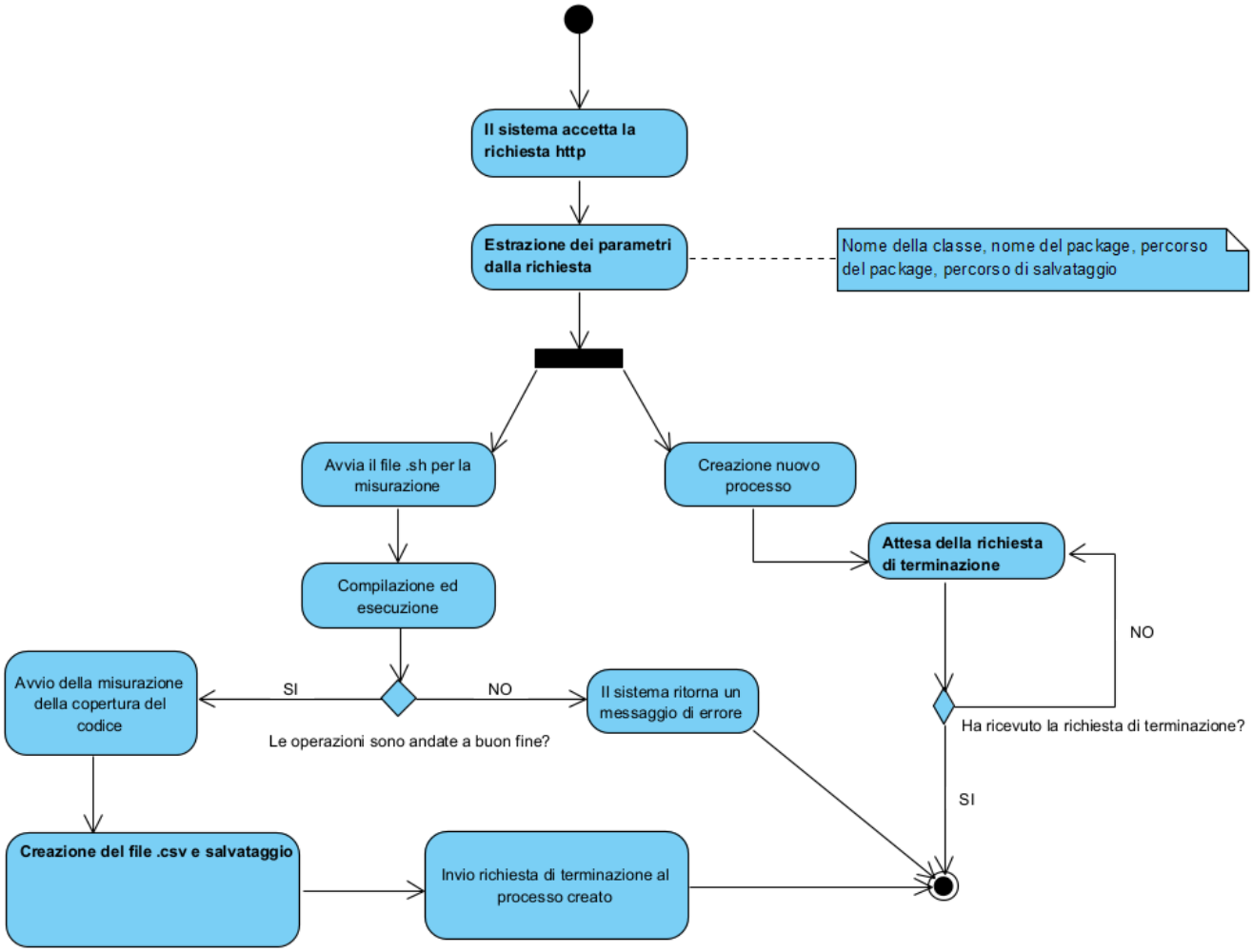


Figura 6: Diagramma di attività misurazione utente

2.7.3 Generazione livelli

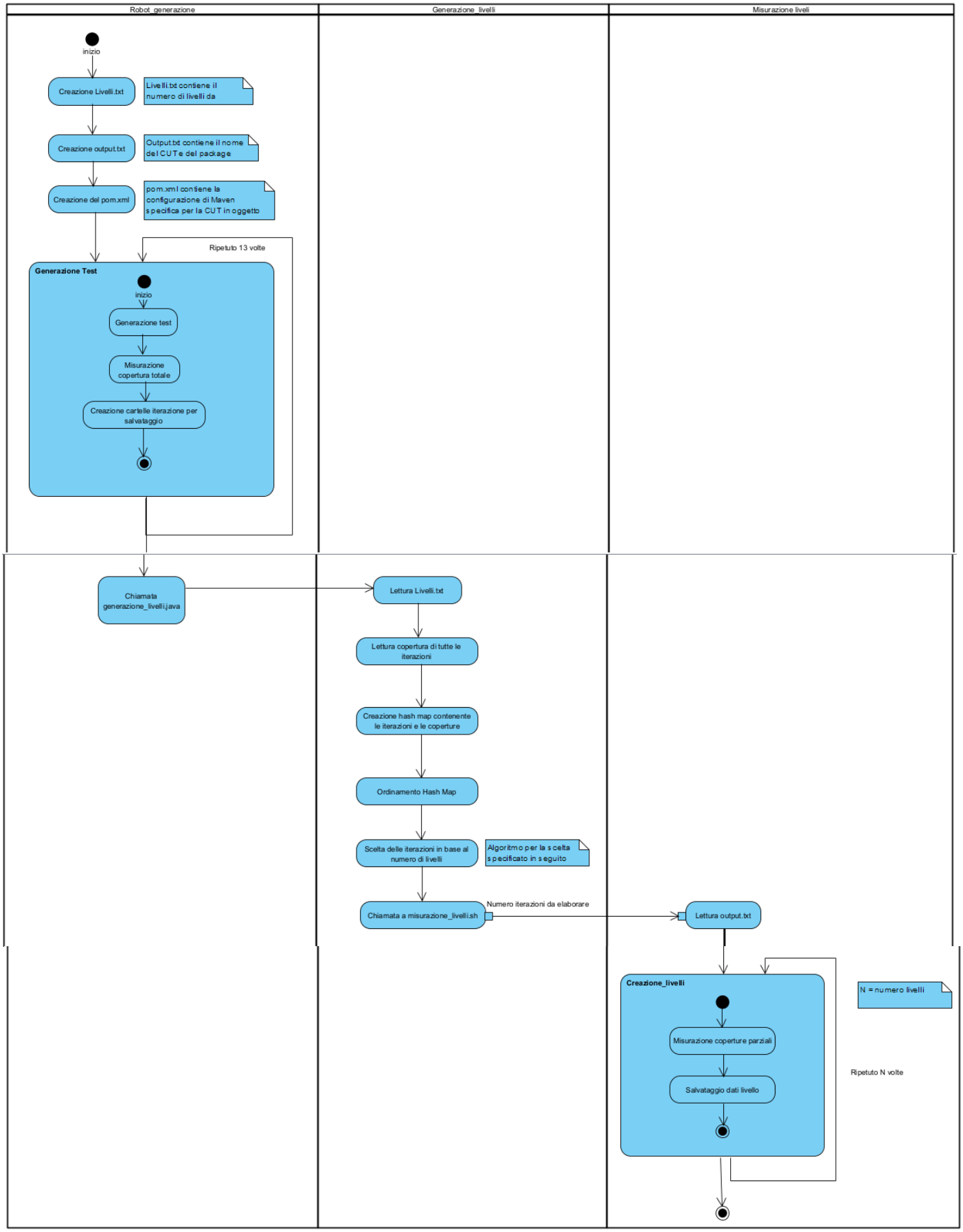


Figura 7: Diagramma di attività generazione livelli

2.8 Diagramma di comunicazione

Il **diagramma di comunicazione** è uno strumento visuale utilizzato per rappresentare le interazioni tra i diversi attori o componenti di un sistema. È particolarmente utile nella progettazione del software per modellare e comprendere le relazioni e le interazioni tra gli oggetti o le entità coinvolte nel sistema, consentendo agli sviluppatori di comprendere meglio la struttura del sistema e le dinamiche delle comunicazioni tra le diverse parti.

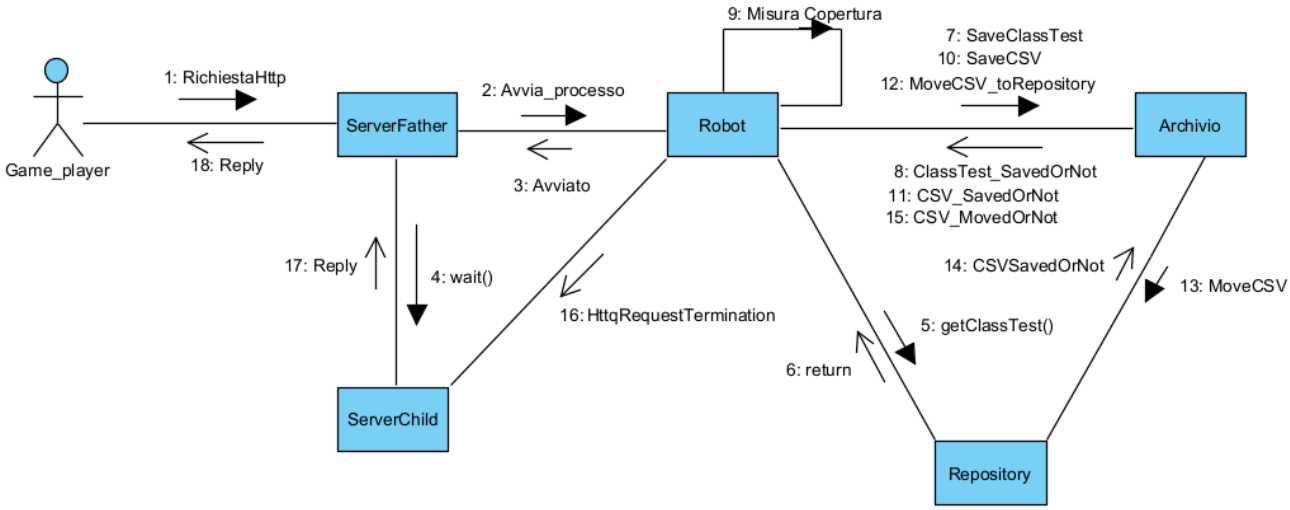


Figura 8: Diagramma di comunicazione

2.9 Diagrammi di sequenza

I **Diagrammi di sequenza** rappresenta la sequenza temporale delle interazioni tra gli oggetti principali all'interno del sistema. Questo diagramma offre una visione dettagliata dei passaggi e delle operazioni coinvolte nel processo di generazione dei livelli e di misurazione dei test.

2.9.1 Misurazione utente

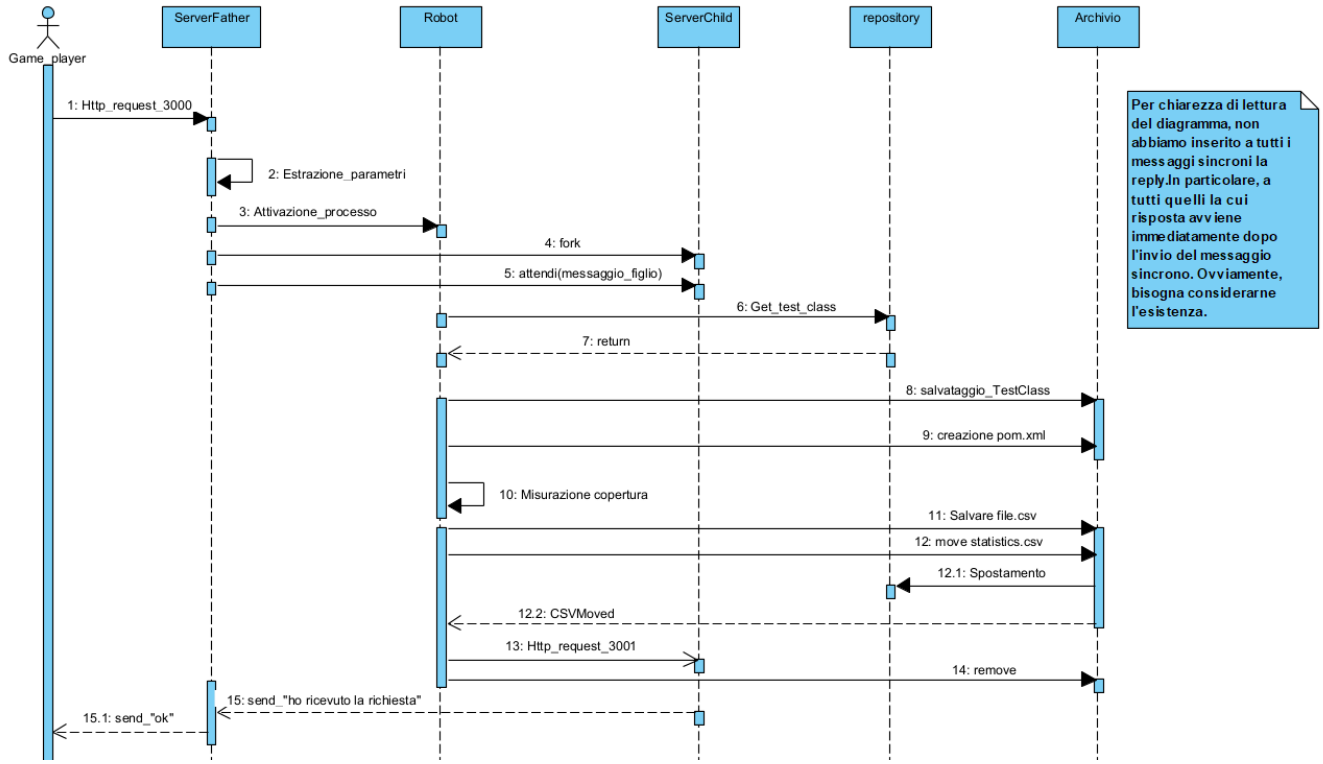


Figura 9: Diagramma di sequenza misurazione utente

2.9.2 Generazione Livelli

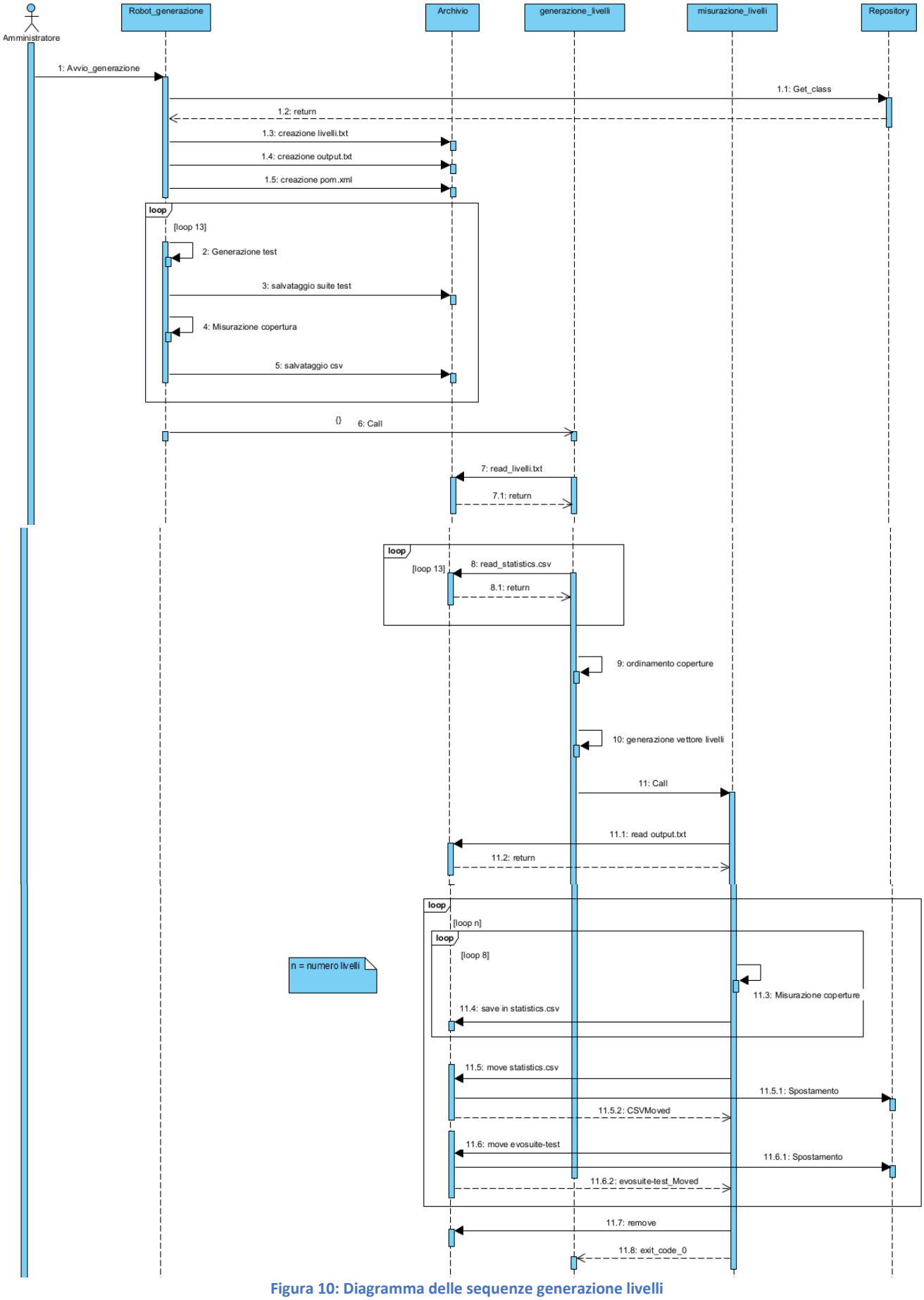


Figura 10: Diagramma delle sequenze generazione livelli

2.10 Flow chart

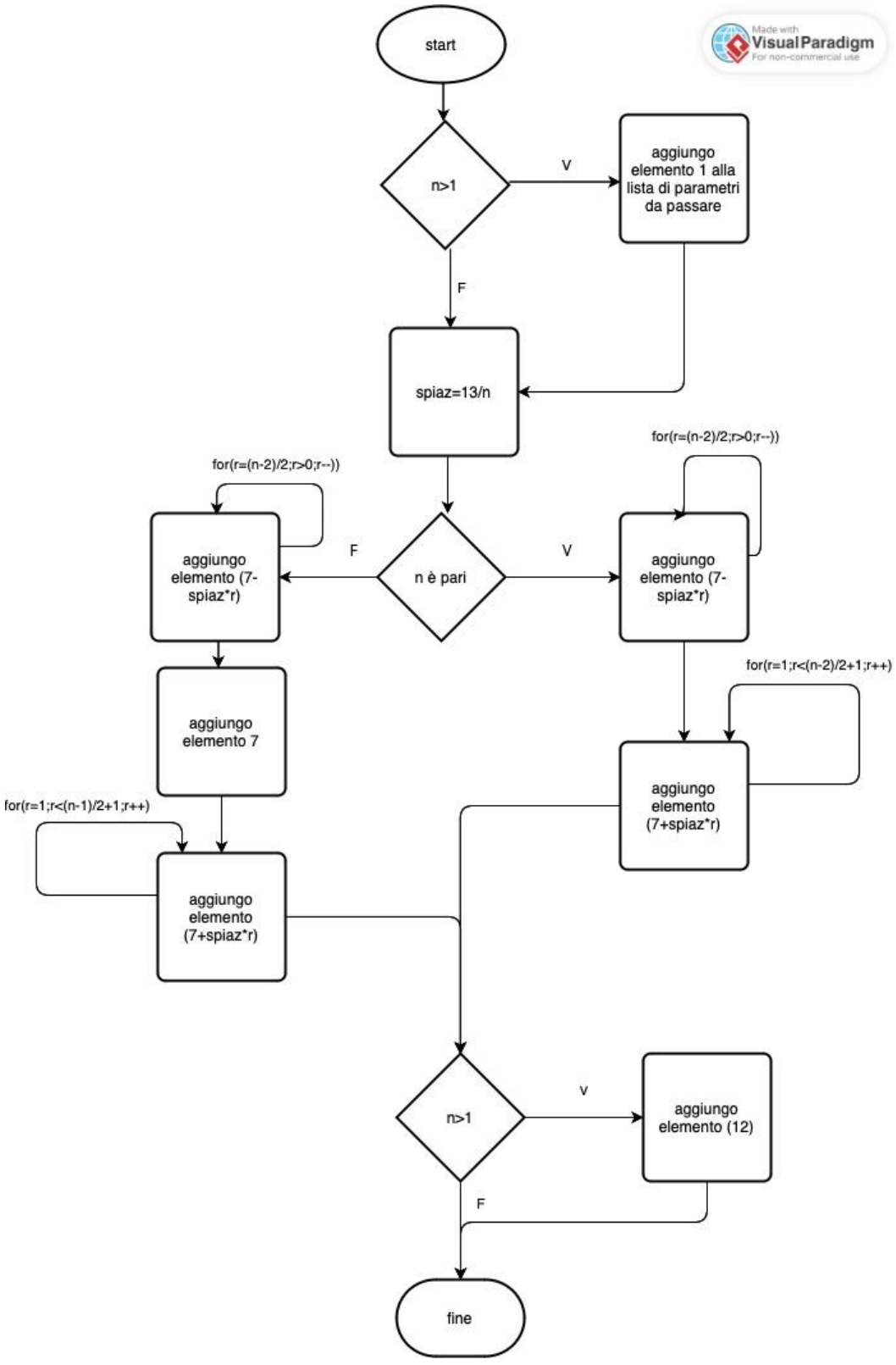


Figura 11: Flow chart

3. Documentazione di progetto

Il **Documento di Progettazione** fornisce una panoramica dettagliata dell'architettura e del design del sistema. Esso descrive in modo completo e approfondito le scelte progettuali, le strutture dei componenti, i modelli di interazione e altre informazioni pertinenti necessarie per la corretta implementazione e comprensione del sistema.

3.1 Diagramma dei componenti

Il diagramma dei componenti viene utilizzato per descrivere l'architettura software di un sistema, esso mostra i componenti del sistema e le relazioni tra di essi.

3.1.1 Misurazione utente

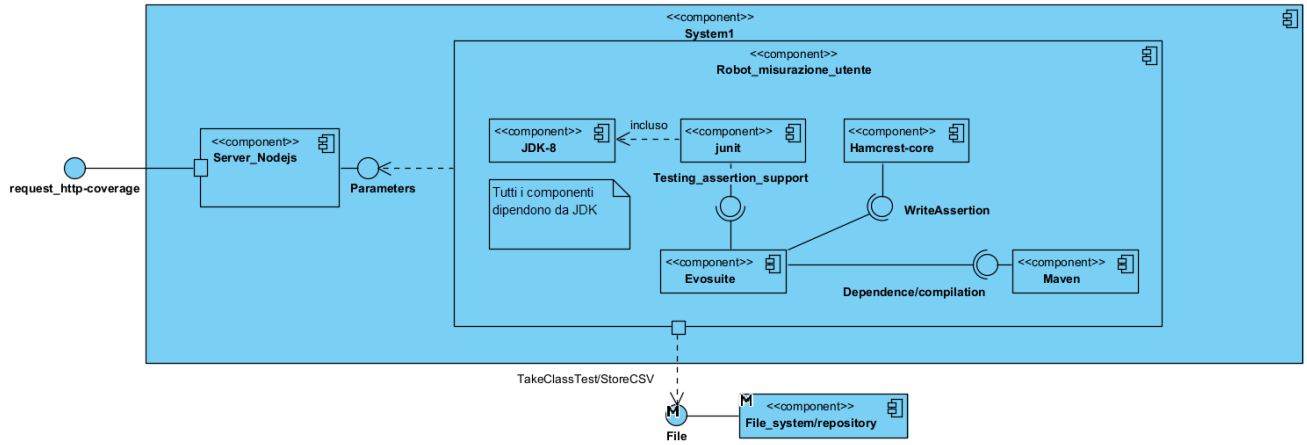


Figura 12: Diagramma dei componenti misurazione utente

3.1.2 Generazione livelli

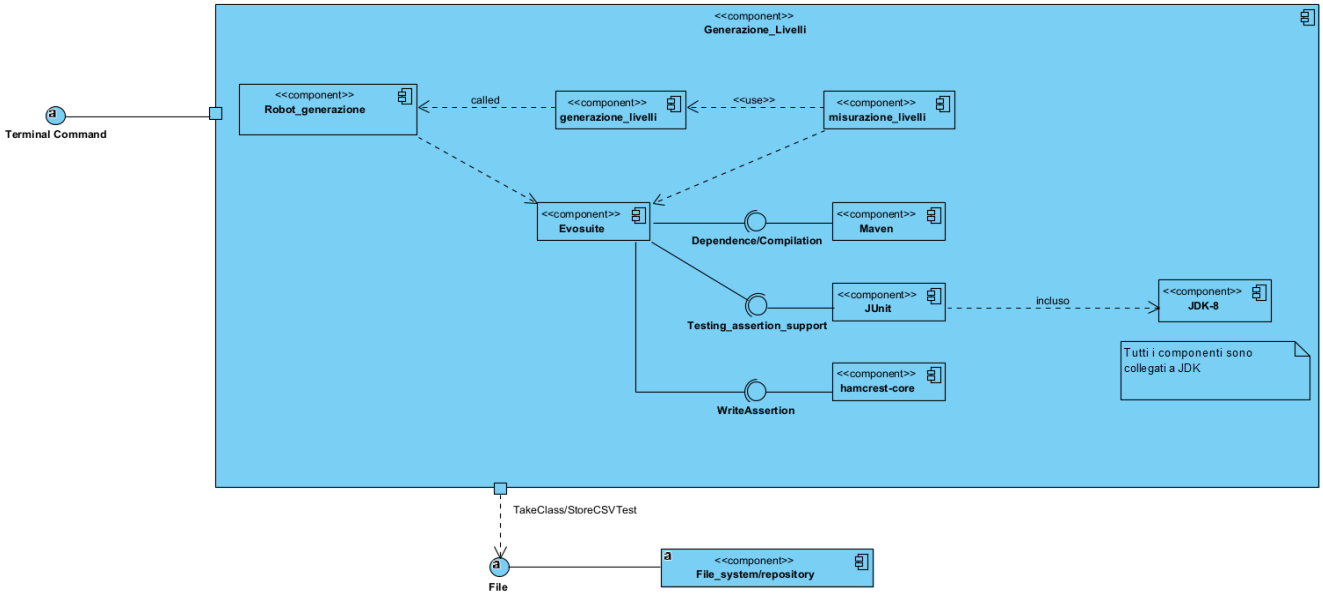


Figura 13: Diagramma dei componenti generazione livelli

3.2 Diagramma di Deployment

I **diagrammi di deployment** forniscono una visualizzazione della distribuzione fisica dei componenti del sistema, illustrando come le diverse parti del sistema sono organizzate e collocate su hardware e infrastrutture specifiche. Questi diagrammi consentono di comprendere l'ambiente di deploy del sistema e le relazioni tra i componenti hardware e software.

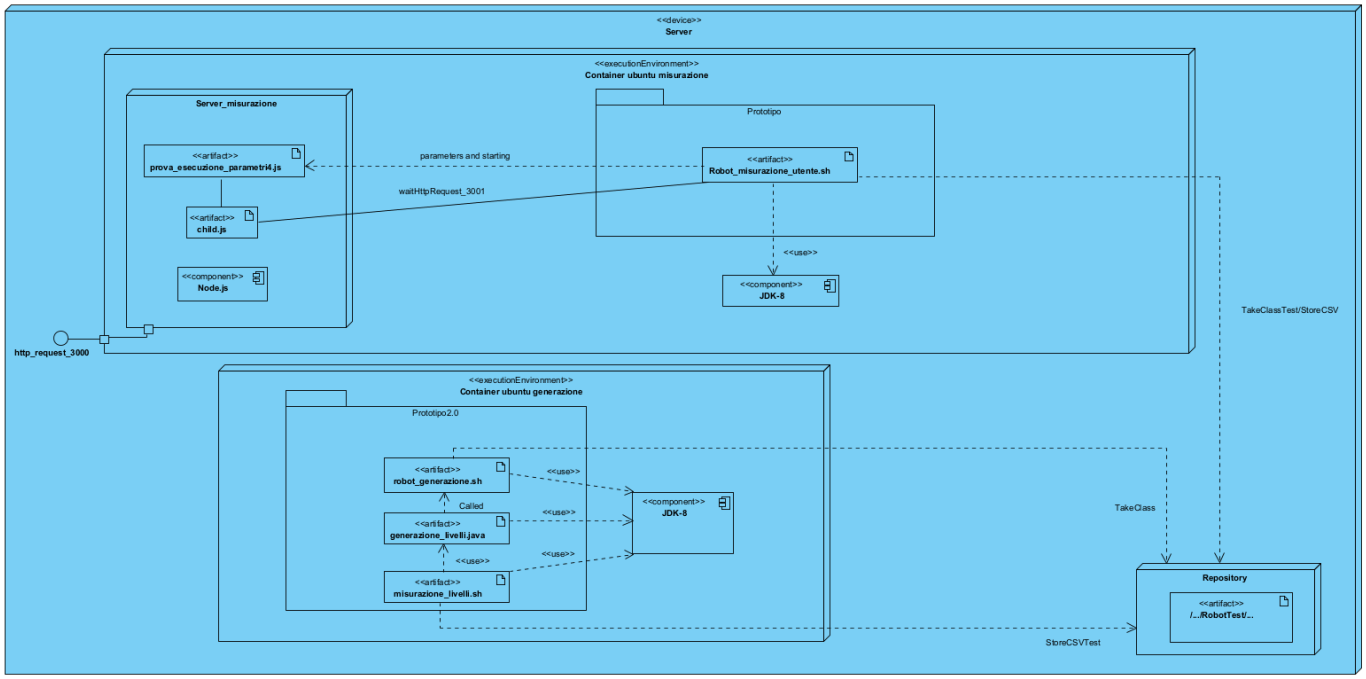


Figura 14: Diagramma di deployment

3.3 Installation view

L'**installation view** del sistema fornisce una rappresentazione visuale delle componenti e delle dipendenze necessarie per installare e configurare correttamente il sistema. Questo diagramma è uno strumento fondamentale per comprendere l'ambiente di installazione e i requisiti necessari per il corretto funzionamento del sistema.

3.3.1 Misurazione utente

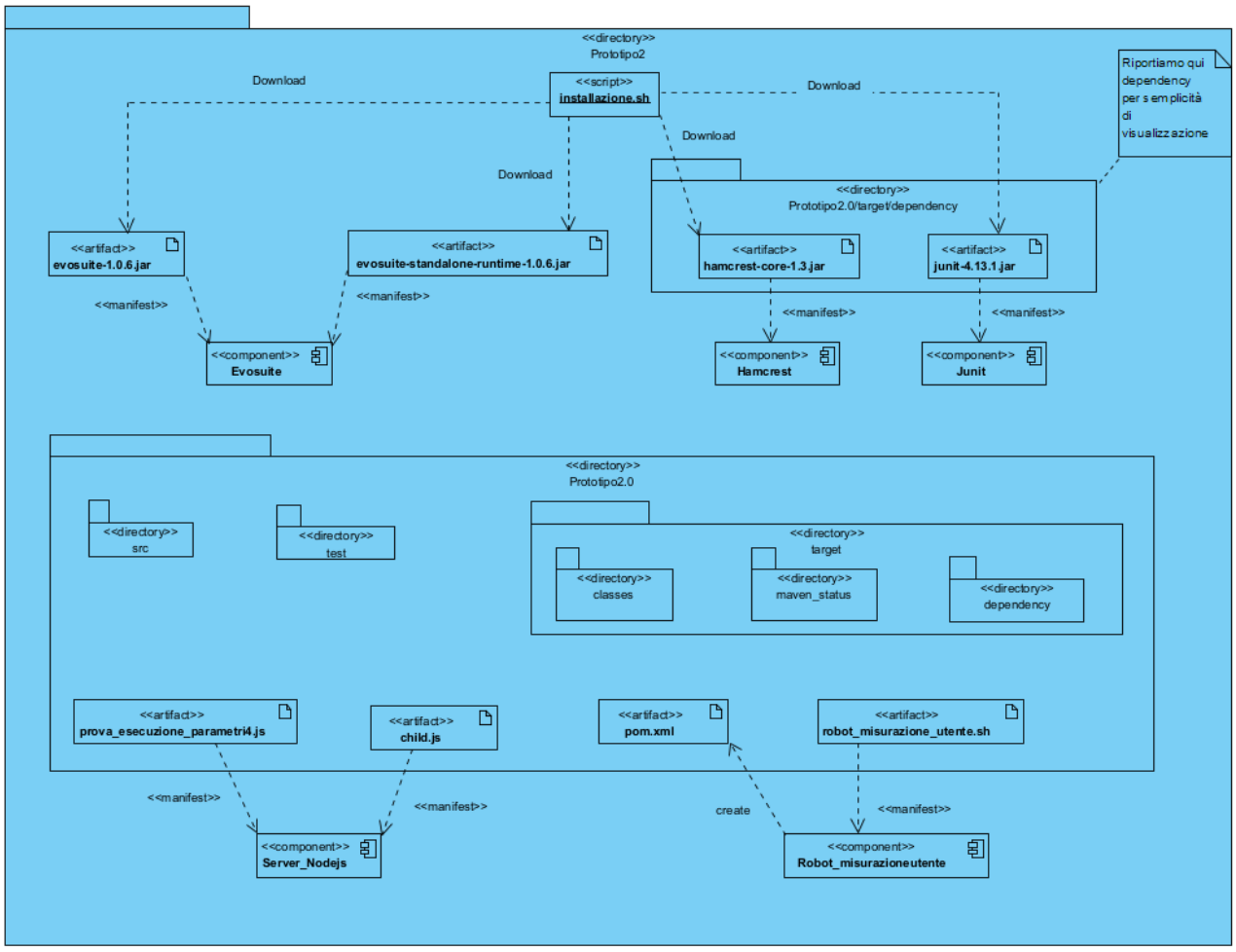


Figura 15: Installation View misurazione utente

3.3.2 Generazione livelli

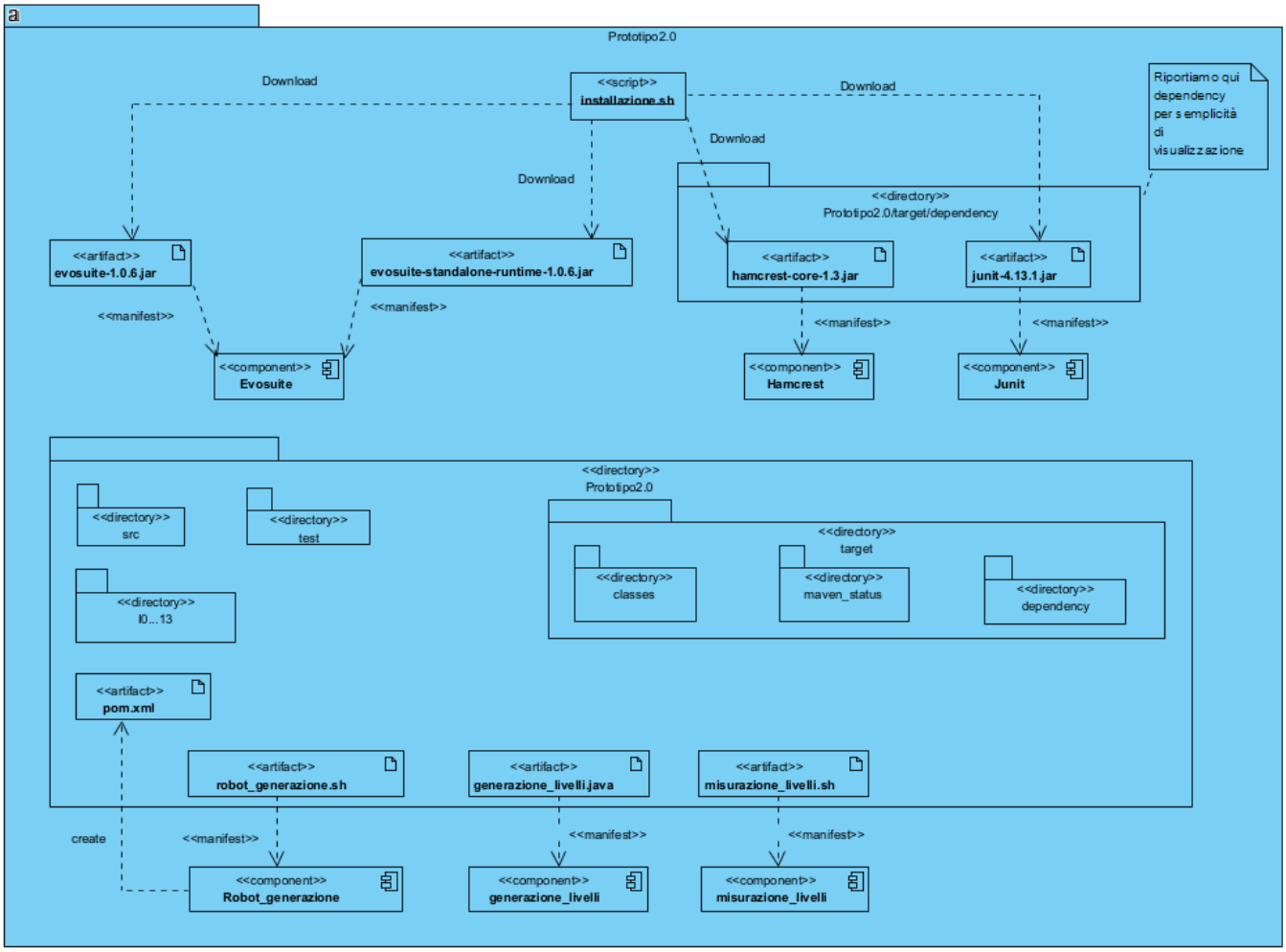


Figura 16: Installation View generazione livelli

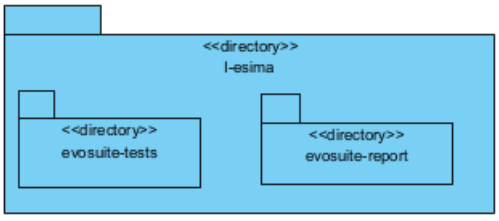


Figura 17: Directory Iterazione

3.4 Pipe and filter

"Pipe and Filter" è un pattern architetturale che prevede il flusso dei dati attraverso una serie di componenti chiamati "filtri" connessi tra loro da "tubi" (pipes). Ogni filtro è responsabile di elaborare i dati in input in base a una determinata funzione o trasformazione, e i tubi gestiscono il passaggio dei dati tra i filtri.

Questo stile architetturale è basato sul principio della separazione delle preoccupazioni (separation of concerns), in cui ogni filtro si occupa di un'attività specifica e indipendente, lavorando in modo isolato dagli altri filtri. I dati fluiscono attraverso i tubi da un filtro all'altro, consentendo una pipeline di elaborazione sequenziale.

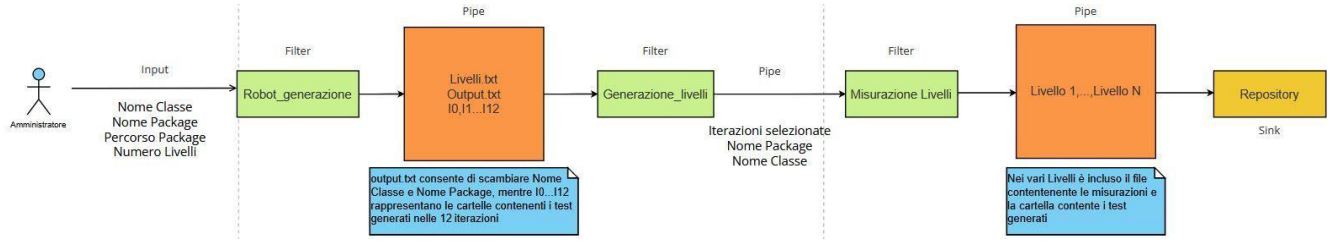


Figura 18: Pipe and Filter

Le caratteristiche principali dello stile Pipe and Filter includono:

1. Separazione dei compiti: Ogni filtro si concentra su una funzionalità specifica, come il filtraggio, la trasformazione, l'elaborazione, la validazione, ecc. Questo permette una migliore gestione della complessità e la possibilità di riutilizzare i filtri in contesti diversi.

2. Comunicazione tramite pipes: Le pipes connettono i filtri consentendo il flusso dei dati da un filtro all'altro. I filtri non sono a conoscenza degli altri filtri nella pipeline, comunicano solo attraverso le pipes.

3. Trasparenza dei dati: I filtri sono isolati l'uno dall'altro e non hanno conoscenza diretta dei dati che attraversano la pipeline. Trattano solo l'input ricevuto e producono un output, garantendo che la pipeline sia modulare e scalabile.

Questo stile architetturale è particolarmente utile quando si desidera separare le fasi di elaborazione in modo che possano essere gestite in modo indipendente e scalabile.

4. Testing

4.1 Casi di test

Test ID	Descrizione	Pre-Condizioni	Input	Output	Post-Condizioni	Esito
1	Generazione Test 6 Livelli	La classe calcolatrice si trova nel percorso specificato nell'input	Calcolatrice, Calcolatrice, /percorso_package/ 6	Statistics.csv ed evosuite_test per ogni livello	Nelle cartelle dei primi 6 livelli sono stati inseriti gli output	PASS
2	Generazione Test 5 Livelli	La classe calendario si trova nel percorso specificato nell'input	Calendario, Calendario, /percorso_ package / 5	Statistics.csv ed evosuite_test per ogni livello	Nelle cartelle dei primi 5 livelli sono stati inseriti gli output	PASS
3	Generazione Test 3 Livelli con percorso sbagliato	La classe calendario non si trova nel percorso specificato nell'input	Calendario, Calendario, /percorso_package errato/ 3	Messaggio di errore	Nelle cartelle dei primi 3 livelli non sono stati inseriti gli output	PASS
4	Generazione Test 0 livelli	La classe calcolatrice si trova nel percorso specificato nell'input	Calcolatrice, Calcolatrice, /percorso_ package / 0	Messaggio di errore: "Arithmethic Exception: / by zero"	Nelle cartelle dei livelli non sono stati inseriti gli output	PASS

5	Generazione Test 3 Livelli	La classe ImprovedTokenizer si trova nel percorso specificato nell'input	ImprovedTokenizer, ImprovedTokenizer /percorso_ package / 3	Statistics.csv ed evosuite_test per ogni livello	Nelle cartelle dei primi 3 livelli sono stati inseriti gli output	
6	Misurazione copertura utente	La classe e i test si trovano nel percorso specificato nell'input	/percorso_classe/+ /percorso_test/+ /percorso_salvataggio/	Statistic.csv	Nella cartella *report* è stato inserito l'output	PASS
7	Misurazione copertura utente	La classe non si trova nel percorso specificato nell'input	/percorso_classe/+ /percorso_test/+ /percorso_salvataggio/	Messaggio di errore	Nella cartella *report* non è stato inserito l'output	PASS
8	Misurazione copertura utente	I test non si trovano nel percorso specificato nell'input	/percorso_classe/+ /percorso_test/+ /percorso_salvataggio/	Messaggio di errore	Nella cartella *report* non è stato inserito l'output	PASS
9	Misurazione copertura utente	Il percorso di salvataggio esiste	/percorso_classe/+ /percorso_test/+ /percorso_salvataggio/	Statistics.csv	Nella cartella *report* è stato inserito l'output	PASS
10	Misurazione copertura utente	Il percorso di salvataggio non esiste	/percorso_classe/+ /percorso_test/+ /percorso_salvataggio/	Messaggio di errore	Nella cartella *report* non è stato inserito l'output	PASS

4.2 Analisi tempo

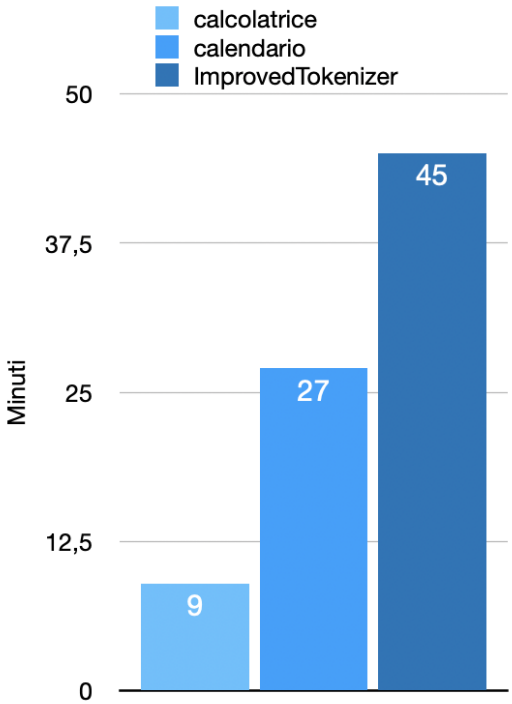


Figura 19: Analisi tempo

Nella figura 19, sono mostrati i tempi di esecuzione della generazione dei livelli sul nostro computer. Per ogni livello in più, si aggiungono 1 o 2 minuti in più al tempo. Questi tempi posso variare in base alla potenza di calcolo del PC. Abbiamo sperimentato che su calcolatori più veloci, il tempo impiegato potrebbe ridursi significativamente.

Per quanto riguarda la misurazione, i tempi sono più costanti e pari all'incirca ad un minuto.

5. Guida all'utilizzo

Questo capitolo fornisce un'esaustiva panoramica sul corretto utilizzo del sistema. Alla fine di questo capitolo si avrà una comprensione completa su come installare, configurare ed utilizzare il sistema.

5.1 Dipendenze

Questo paragrafo ha il solo scopo di illustrare i componenti utilizzati per la realizzazione del software e non fornisce ancora una guida all'installazione. Di seguito, riportiamo le dipendenze comuni ad entrambi i sistemi:

- **OpenJDK-8**
- **Java** versione 1.8
- **Evosuite** versione 1.0.6
- **JUnit** versione 4.13.1
- **hamcrest-core** versione 1.3

In particolare, per la parte di misurazione livelli, sarà necessario includere le seguenti dipendenze:

- **Nodejs** versione 19
- **npm** versione 6

Il sistema può essere eseguito sia su un computer con sistema operativo ubuntu 22.04 sia su container ubuntu docker.

5.2 Installazione

1. Prima di iniziare bisogna eseguire lo script installazione.sh che contiene i software necessari, come Evosuite, Maven e Nodejs

2. In opt/Prototipo2.0 sono presenti: i file .js per la creazione del server e uno script per la misurazione dei test scritti dall'utente

3. In opt_livelli sono presenti: lo script per la generazione dei test da parte di evosuite (robot_generazione.sh), il java che seleziona le iterazioni da utilizzare (generazione_livelli.java) e lo script per salvare le misurazioni e i test scelti nel repository comune (misurazione_livelli.sh)

5.3 Configurazione

1. Per avviare il server per la misurazione utente bisogna lanciare a linea di comando "node prova_esecuzione_parametri4.js"

2. Per avviare il server per la generazione dei livelli bisogna lanciare a linea di comando “robot_generazione”

    1. Per un corretto avvio della generazione dei test e dei livelli bisogna lanciare lo script robot_generazione.sh nel seguente modo:
    >bash robot_generazione.sh NOME_CLASSE NOME_PACKAGE PERCORSO_PACKAGE NUMERO_LIVELLI
    esempio: >bash robot_generazione.sh calcolatrice calcolatrice mnt/f/Desktop/repository/calcolatrice 3

5.4 Rotta API

Le API (Application Programming Interface) sono un insieme di strumenti che consentono agli utenti del nostro sistema di interagire con esso in modo programmato, tramite scambio di dati e richieste. Le API sono fondamentali per la nostra architettura software, poiché consentono una maggiore flessibilità e scalabilità del sistema.

Le API sono progettate per essere modulari e consentono un facile accesso ai dati attraverso richieste HTTP.

Postman è un'applicazione per il testing di API che consente agli sviluppatori di creare, testare e documentare le loro API in modo rapido ed efficiente.

Di seguito, riportiamo il formato della richiesta http:

http://ip:porta/api/percorso_classe+percorso_test+percorso_salvataggio

- ip:porta: IP e porta su cui è connesso e comunica il server. Il numero della porta scelta è stato 3000.
- Percorso_classe: Percorso della classe da testare. In questo percorso deve essere presente sia il nome della classe che il nome del package in cui essa è contenuta.
- Percorso_test: Percorso della suite di test
- Percorso_salvataggio: Percorso della cartella in cui salvare i risultati ottenuti. Il nome del file, che verrà salvato, sarà “statistics.csv”.

Esempio di richiesta:

http://127.0.1.1:3000/api/data/StudentLogin/GameId/calcolatrice/calcolatrice.java+/data/StudentLogin/GameId/TestSourceCode/calcolatrice_test.java+/data/StudentLogin/GameId/TestReport

Simuliamo la richiesta http, sopra riportata, con Postman e riportiamo, nella figura sottostante, la risposta:

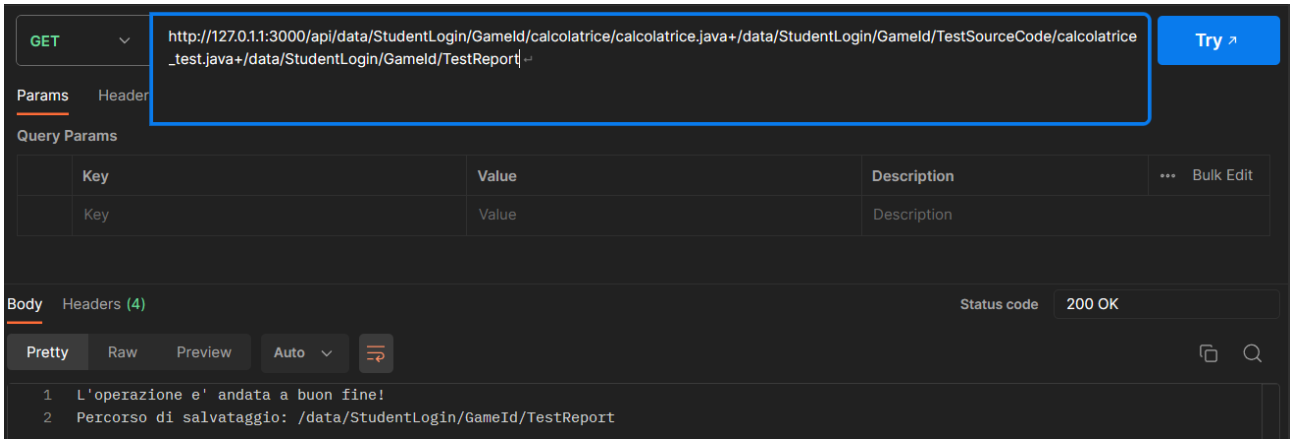


Figura 20: Postman

Per ulteriore chiarezza, riportiamo la seguente tabella indicando la descrizione, l'URL, il metodo, la risposta ed eventuali errori:

DESCRIZIONE:	L'utente richiede la misurazione della copertura della suite di test da lui indicata
URL:	*http://ip:porta/api/percorso_classe+percorso_test+percorso_salvataggio*
METODO:	GET
RISPOSTA:	L'operazione è andata a buon fine! Percorso di salvataggio: /percorso_salvataggio
ERRORI:	Timeout

5.5 Output

La seguente sezione mostra, attraverso un esempio sulla classe “calendario”, il formato del file “statistics.csv” e la specifica dei suoi parametri.

	A	B	C	D	E	F	G
1	TARGET_CLASS,criterion,Coverage,Total_Goals,Covered_Goals						
2	calendario.calendario,LINE,0.3333333333333333,48,16						
3	calendario.calendario,BRANCH,0.16091954022988506,87,14						
4	calendario.calendario,EXCEPTION,1.0,0,0						
5	calendario.calendario,WEAKMUTATION,0.2403846153846154,104,25						
6	calendario.calendario,OUTPUT,0.0,3,0						
7	calendario.calendario,METHOD,0.0,2,0						
8	calendario.calendario,METHODNOEXCEPTION,0.0,2,0						
9	calendario.calendario,CBRANCH,0.16091954022988506,87,14						
10							

Figura 21: Statistics.csv

- TARGET_CLASS: nome del package e nome della classe sotto test
- criterion: il criterio di valutazione dei test
- Total_Goals: numero totali di obiettivi da coprire
- Covered_Goals: numero di obiettivi coperti
- Coverage: Covered_Goals/ Total_Goals

6. Glossario

CFG: control flow graph

CUT: classe sotto test

LINE: per essere soddisfatto, questo criterio prevede che una test suite esegua ogni linea di codice non commentata almeno una volta.

BRANCH: una suite di test soddisfa questo criterio se tutti i rami del CFG sono valutati da almeno un caso di test: la suite di test, quindi, contiene un test per ogni nodo istruzione e, per i nodi predicato, almeno un test la cui esecuzione valuta il predicato del ramo true e, almeno uno, la cui esecuzione valuta il predicato del ramo false.

EXCEPTION: questo criterio premia le suite di test che costringono la CUT a generare più eccezioni possibili, dichiarate o non dichiarate.

WEAK MUTATION: una suite di test soddisfa questo criterio se, per ogni mutazione generata, almeno un test rileva la mutazione. Il test di mutazione debole apporta piccole modifiche al codice della classe sotto test, per controllare, se esiste un test in grado di distinguere tra l'originale e il mutante.

OUTPUT: in questo criterio i tipi degli output dei vari metodi, vengono mappati come valori astratti e una suite di test soddisfa il criterio se, per ogni metodo pubblico, almeno un test produce un valore concreto caratterizzato da ogni valore astratto.

METHOD: questo criterio richiede semplicemente che tutti i metodi della CUT vengano eseguiti almeno una volta, direttamente o indirettamente.

METHOD NO EXCEPTION: questo criterio richiede che tutti i metodi della CUT vengano chiamati in maniera diretta e che la loro esecuzione debba terminare senza generare eccezioni.

C-BRANCH (Context Branch): questo criterio forza la generazione di casi di test che richiamano in maniera diretta il metodo che contiene un ramo del CFG. Quindi, un ramo del grafo non viene considerato coperto se viene invocato da altri metodi.