Generazione test con Randoop

Software Architecture Design

T9-G27

Aurora Maio 0001/22404

Alessia Maisto M63001552

Giuseppe Mastellone M63001436

Saverio Picarelli M63001553

AA: 2022/2023

Indice

1. Specifiche del progetto

L'applicazione deve offrire la funzionalità di generazione di vari livelli di difficoltà dei test su una data classe Java usando il Robot Randoop. Tale funzionalità riceverà in input un file di testo (classe da testare) ed i livelli richiesti, dovrà lanciare il generatore di Test Randoop, restituendo in output il codice dei casi di test generati ed i risultati dell'esecuzione. Variando i parametri di input di Randoop, è possibile generare test suite con diversi valori di efficacia: tali test suite verranno utilizzate dal Game Engine a seconda del livello di gioco raggiunto e corrisponderanno quindi a diversi robot con i quali sfidarsi.

2. Descrizione del progetto

Si vuole realizzare un'applicazione che implementi l'Educational Game "Man vs automated Testing Tools challenges": si compete contro diversi tipi di strumenti di generazione automatica di casi di test, quali ad esempio Randoop ed Evosuite. La realizzazione della Web Application per il gioco sul Testing è stata divisa in nove task. In particolare, i nostro task, il requisito specifico 9, consiste nello sviluppo di casi di test per classi Java, attraverso il Robot Randoop.

2.1 Processo di sviluppo

Per lo sviluppo della nostra architettura software, è stato usata la partica di insegnamento "Case Based Learning" (CBL) basata sull'analisi di casi di studio reali, condotta in gruppo. Dunque, 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

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.

2.2 Gestione del team

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.

2.3 Diario dell'effort del team e stima dei tempi di sviluppo dell'applicazione

La seguente tabella riporta i progressi effettuati sulla produzione dell'architettura del sistema che sono stati annotati, dai membri del team, alla fine di ogni giornata.

Data	Risultati
04/04/2023	Prima lettura dei requisiti dei vari task proposti
05/04/2023	Scelta dei tre requisiti maggiormente di interesse
07/04/2023	Analisi del requisito assegnato e organizzazione del team
12/04/2023	Definizione degli obiettivi del task e studio di fattibilità
13/04/2023	Definizione del glossario dei termini e studio tecnologie da utilizzare
15/04/2023	Studio ed installazione tecnologie da utilizzare
16/04/2023	Definizione degli attori del sistema e dei casi d'uso con la realizzazione dei relativi diagrammi
21/04/2023	Prima iterazione con eventuali feedback
27/05/2023	Raffinamento dei casi d'uso e realizzazione dei diagrammi di sequenza
03/05/2023	Realizzazione del diagramma delle attività e del diagramma delle classi
05/05/2023	Realizzazione prototipo e inizio sviluppo del codice
09/05/2023	Seconda iterazione ed eventuali feedback
10/05/2023	Correzioni dopo i feedback ricevuti e sviluppo codice
17/05/2023	Sviluppo codice
18/05/2023	Revisione documentazione e sviluppo codice
20/05/2023	Studio altre tecnologie da usare e sviluppo codice
25/05/2023	Revisione documentazione
06/06/2023	Terza iterazione con feedback ed eventuali domande
Da 10/06/2023 a 18/06/2023	Completamento del codice e revisione della documentazione

La seguente tabella, invece, mostra una stima delle ore di lavoro dedicate al completamento delle single attività dello sviluppo del software.

Attività	Totale ore
Studio fattibilità	24 h
Analisi dei requisiti	60 h
Progettazione	100 h
Implementazione	180 h
Produzione Documentazione	50 h

2.4 Strumenti adottati

Microsoft Teams è una piattaforma di comunicazione e collaborazione unificata che combina chat di lavoro
persistente, teleconferenza, condivisione di contenuti e integrazione delle applicazioni.

GitHub è un servizio di hosting per progetti software

Eclipse è un ambiente di sviluppo integrato (IDE) per lo sviluppo del software. È utilizzato principalmente per lo sviluppo di Java, ma supporta anche altri linguaggi di programmazione come C/C ++, Python e PHP

JUnit è un framework open source, usato per scrivere ed eseguire unit testing per Java.

Randoop è un generatore di casi di test per Java

Maven è uno strumento di build automation utilizzato prevalentemente nella gestione di progetti Java.

JaCoCo è una libreria per il Code Coverage in ambito Java.

Postman è una piattaforma API che consente agli sviluppatori di progettare, costruire, testare e iterare le proprie API.

Visual Paradigm è uno strumento CASE UML che supporta UML 2, SysML e la notazione di modellazione dei processi aziendali da Object Management Group.

Spring Boot è un framework per lo sviluppo di applicazioni Java basate su Spring Framework.

Jackson è una libreria Java per la manipolazione di dati in formato JSON. È ampiamente utilizzata nell'ecosistema Java per convertire oggetti Java in rappresentazioni JSON e viceversa.

Docker è un popolare software libero progettato per eseguire processi informatici in ambienti isolabili, minimali e facilmente distribuibili chiamati container Linux, con l'obiettivo di semplificare i processi di deployment di applicazioni software

3. Glossario dei Termini

Al fine di rendere chiari i concetti che tratteremo in seguito, riportiamo un glossario contenente i termini più importanti, utilizzati frequentemente nella presente documentazione.

Termine	***Descrizione***	***Sinonimo***
Test	Processo di valutazione e verifica del corretto funzionamento di un'applicazione o di un prodotto software rispetto alle aspettative.	Collaudo
Classe	Costrutto di un linguaggio di programmazione usato come modello per creare oggetti, modello comprende attributi e metodi che saranno condivisi da tutti gli oggetti creati (istanze) a partire dalla classe.	Costrutto
Randoop	Strumento open source per la generazione di casi di test utilizzati per testare i software.	-
File di testo	File che contiene solamente testo puro, ossia la codifica binaria di caratteri comprensibili a un lettore umano, come lettere, numeri, segni di punteggiatura, ecc.	Documento testuale
Game-Engine	Componente software che gestisce le funzionalità del gioco	-

4. Specifica dei requisiti

Nel corso delle iterazioni, sono stati individuati i seguenti requisiti:

1. L’applicazione deve offrire la funzionalità di generazione di vari livelli di difficoltà dei test su una data classe Java usando il Robot Randoop.
2. La funzionalità riceve in input un file di testo (classe da testare) ed i livelli richiesti.
3. Il nome della classe non deve contenere caratteri speciali
4. Il nome della classe deve essere una stringa non vuota
5. Il numero dei livelli richiesti deve essere un intero positivo maggiore di zero
6. L’applicazione dovrà lanciare il generatore di Test Randoop.

7. L’applicazione deve restituire in output il codice dei casi di test generati ed i risultati dell’esecuzione.
8. La funzionalità deve generare diversi test suite.
9. La generazione dei test deve fermarsi quando non può generare una copertura migliore.
10. L’applicazione deve selezionare un numero di test generati pari al numero di livelli scelti dall’utilizzatore.
11. La funzionalità deve valutare la copertura dei singoli test.
12. La funzionalità deve ordinare i test secondo i livelli prescelti.
13. La funzionalità deve suddividere i test in cartelle in base al livello ottenuto
14. La funzionalità deve avvisare al termine della generazione dei test
15. La funzionalità deve verificare che la classe inserita non sia vuota
16. L’applicazione deve essere altamente portabile

4.1 Requisiti funzionali

ID	Requisito	Numero frase della specifica dei requisiti
RF01	L'applicazione deve offrire la funzionalità di generazione di vari livelli di difficoltà dei test su una data classe Java usando il Robot Randoop.	1
RF02	La funzionalità riceve in input un file di testo (classe da testare) ed i livelli richiesti	2
RF03	L'applicazione dovrà lanciare il generatore di test Randoop	6
RF04	L'applicazione deve restituire in output il codice dei casi di test generati ed i risultati dell'esecuzione.	7
RF05	La funzionalità deve generare diversi test suite	8
RF06	La generazione dei deve fermarsi quando non può generare una copertura migliore.	9
RF07	L'applicazione deve selezionare un numero di test generati pari al numero di livelli scelti dall'utilizzatore.	10
RF08	La funzionalità deve valutare la copertura dei singoli test.	11
RF09	La funzionalità deve ordinare i test secondo i livelli prescelti.	12
RF10	La funzionalità deve suddividere i test in cartelle in base al livello ottenuto.	13
RF11	La funzionalità deve avvisare al termine della generazione dei test.	14
RF12	La funzionalità deve verificare che la classe inserita non sia vuota.	15

4.2 Requisiti non funzionali

ID	Requisito	Numero frase della specifica dei requisiti
RNF01	L'applicazione deve essere altamente portabile	16

4.3 Requisiti sui dati

ID	Requisito	Numero frase della specifica dei requisiti
RD01	Il nome della classe non deve contenere caratteri speciali	3
RD02	Il nome della classe deve essere una stringa non vuota	4
RD03	Il numero dei livelli richiesti deve essere un intero positivo maggiore di zero	5

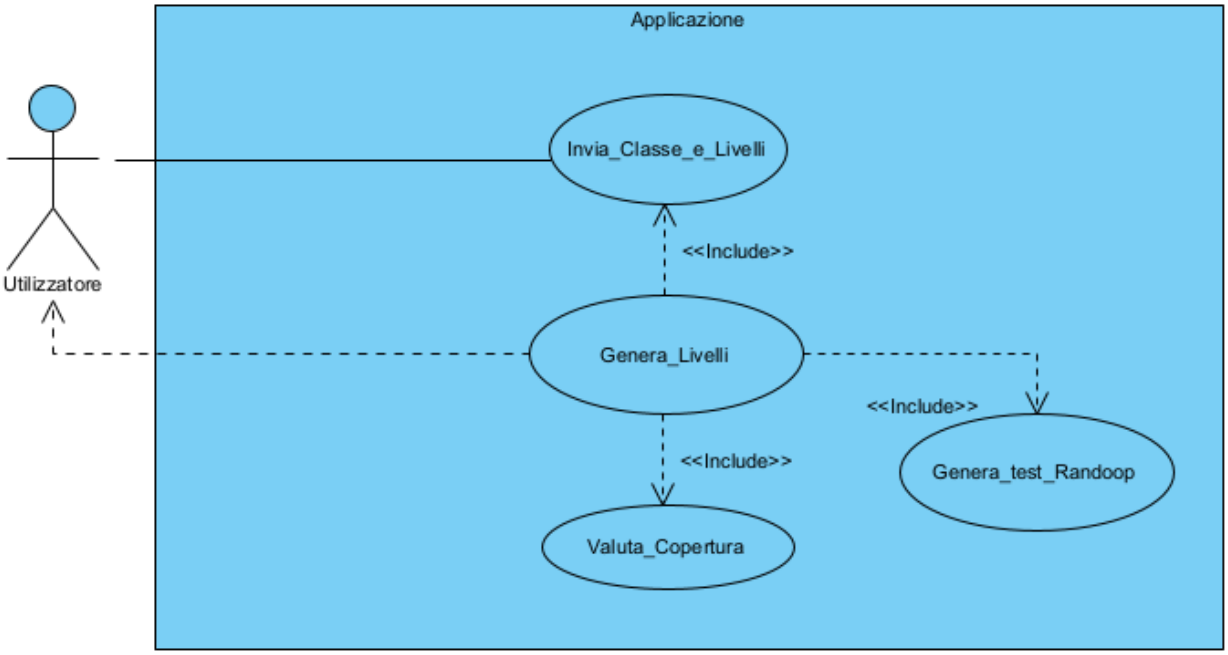
5. Analisi dei requisiti

5.1 Modellazione dei casi d'uso

La seguente tabella ritorta nel dettaglio i casi d'uso individuati e le relazioni di inclusioni tra loro:

Caso d'uso	Attori Primari	Attori Secondari	Incl. / Ext.
UC1: Invia_Classe_e_Livelli	Utilizzatore	-	Incluso in "Genera_Livelli"
UC2: Genera_Livelli	-	-	Include "Invia_Classe_e_Livelli", "Genera_test_Randoop" e "Valuta_Copertura"
UC3: Genera_test_Randoop	-	-	Incluso in "Genera_Livelli"
UC4: Valuta_Copertura	-	-	Incluso in "Genera_Livelli"

5.2 Diagramma dei casi d'uso



5.3 Scenari

Caso d'uso	**Invia_Classe_e_Livelli**
Attore primario	Utilizzatore
Attore secondario	-
Pre-Condizioni	L'utilizzatore possiede una classe Java da testare
Sequenza di eventi principale	L'utilizzatore fornisce all'applicazione la classe Java da testare ed i livelli di difficoltà da generare
Post-Condizioni	L'applicazione ha a disposizione la classe Java da testare ed i livelli di difficoltà richiesti
Casi d'uso correlati	Genera_LIvelli

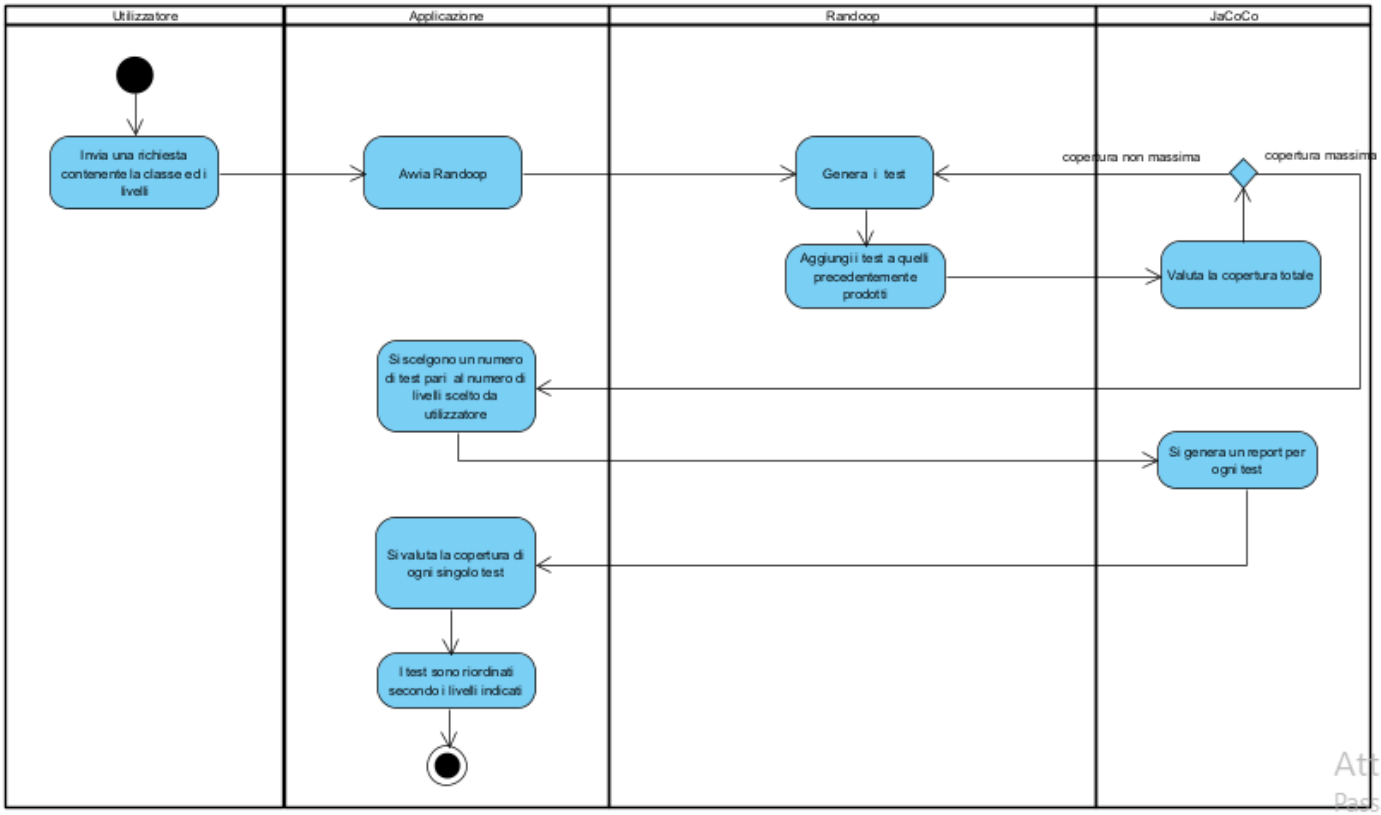
Caso d'uso	**Genera_Livelli**
Attore primario	-
Attore secondario	-
Pre-Condizioni	L'applicazione ha a disposizione la classe Java da testare ed i livelli di difficoltà richiesti dall'utilizzatore
Sequenza di eventi principale	L'applicazione genera i livelli richiesti dall'utilizzatore
Post-Condizioni	L'applicazione ha a disposizione vari livelli su cui suddividere i test
Casi d'uso correlati	Invia_Classe_e_Livelli, Genera_test_Randoop, Valuta_Copertura

Caso d'uso	**Genera_test_Randoop**
Attore primario	-
Attore secondario	-
Pre-Condizioni	Randoop ha a disposizione una classe da testare
Sequenza di eventi principale	Randoop genera i casi di test e ne valuta l'eseguibilità
Post-Condizioni	Randoop ha generato dei casi di test eseguibili
Casi d'uso correlati	Genera_Livelli

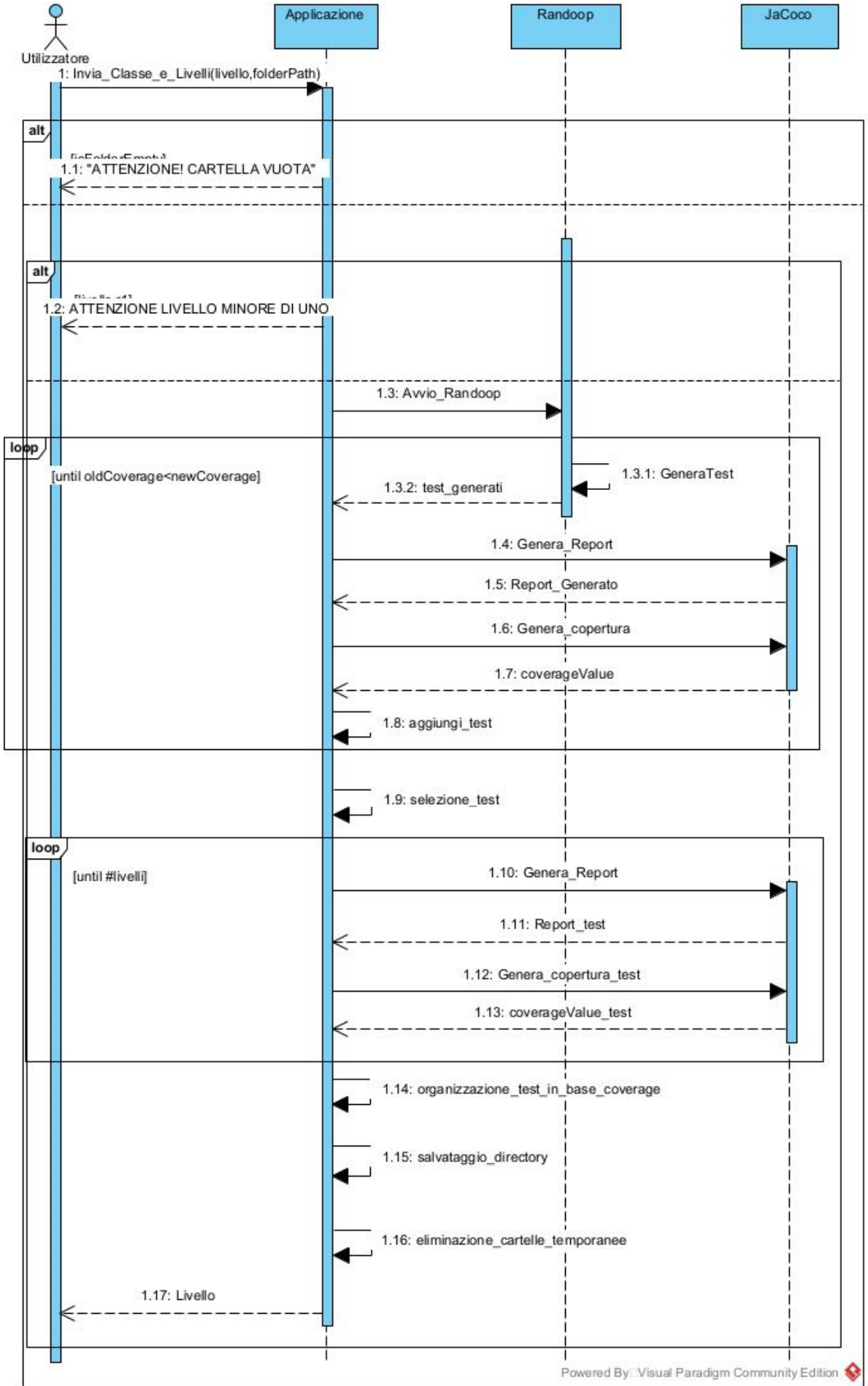
Caso d'uso	**Valuta_Copertura**
Attore primario	-
Attore secondario	-
Pre-Condizioni	Si ha a disposizione una classe di test
Sequenza di eventi principale	Il test è eseguito e JaCoCo ne valuta la copertura
Post-Condizioni	Il livello di copertura del test è disponibile
Casi d'uso correlati	Genera_Livelli

5.4 Diagramma delle attività

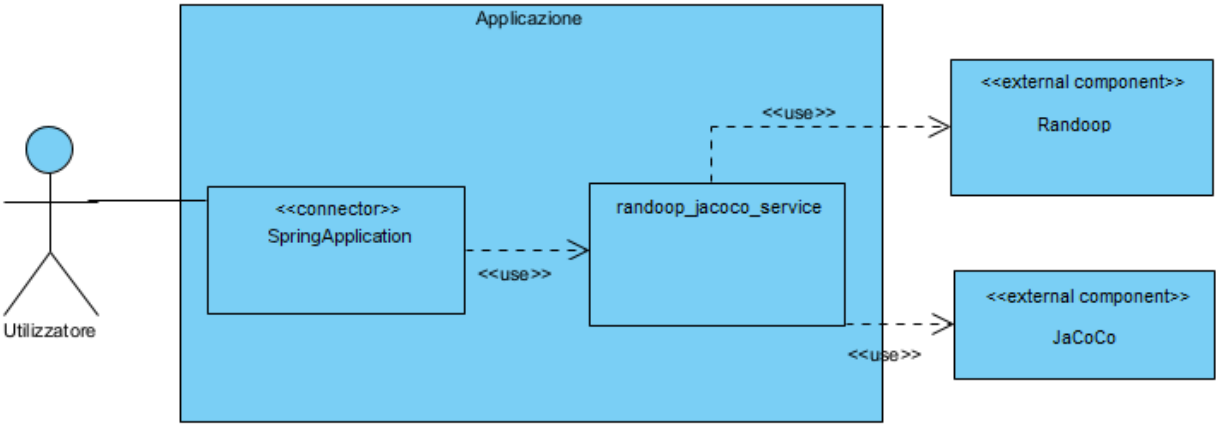
I diagrammi di attività consentono di descrivere un processo elaborativo mediante grafi orientati, in cui ogni nodo rappresenta un'attività atomica, mentre gli archi definiscono il flusso di esecuzione. Dato che i casi d'uso sono correlati, è stato deciso di progettare un unico diagramma delle attività che copra l'intero sistema.



5.5 Diagramma di sequenza



5.6 Diagramma di contesto

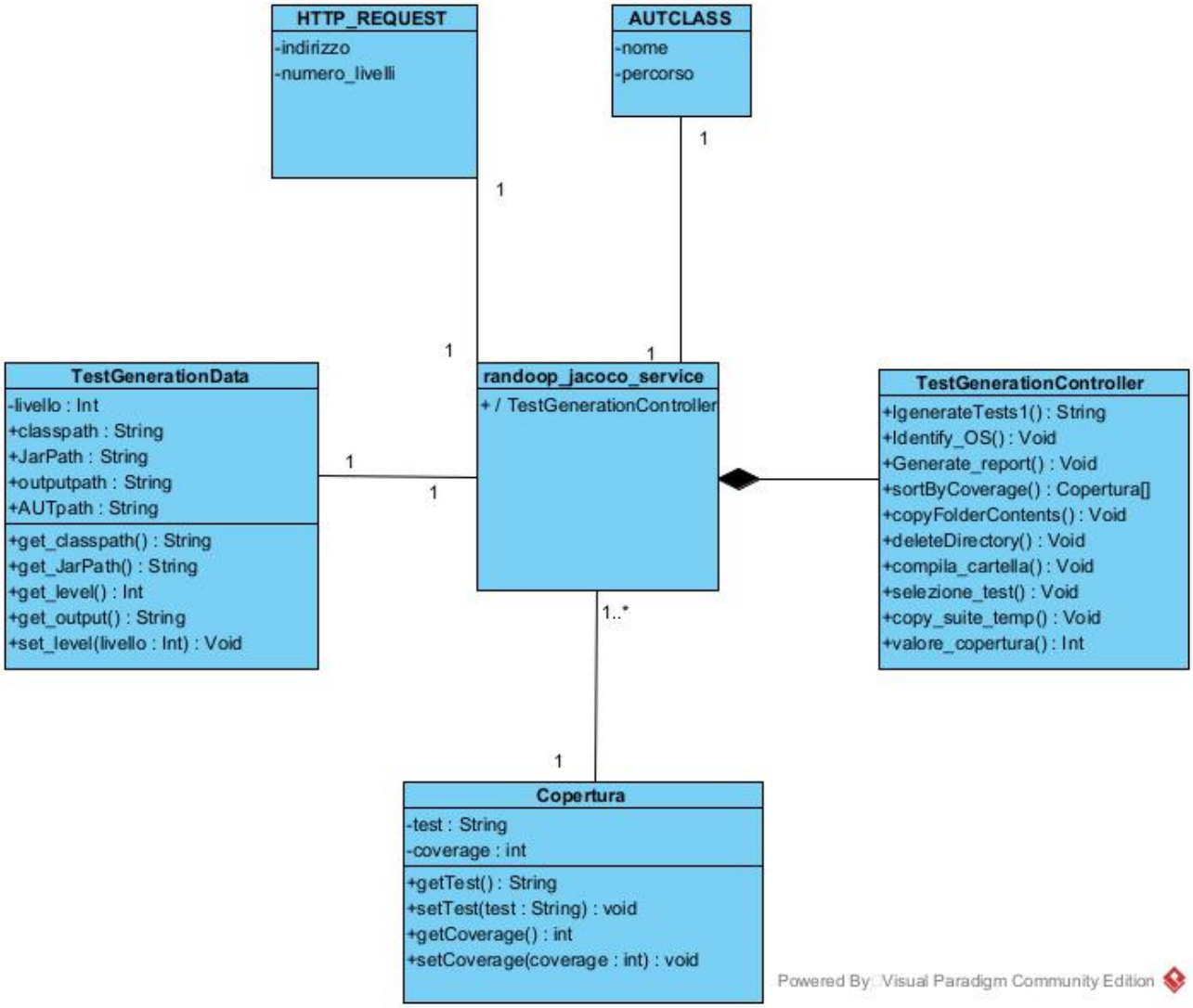


6. Architettura del software

Durante la realizzazione di un software, è fondamentale progettarne l'architettura, ovvero la struttura fondamentale ed il design del sistema software. Essa definisce gli elementi, le componenti, le relazioni e i principi che guidano la creazione e l'evoluzione di un'applicazione, e fornisce una vista ad alto livello del sistema.

Quella scelta per il software progettato è un'architettura client-server. L'architettura client-server è un modello di progettazione per la distribuzione delle applicazioni informatiche in cui il client è l'entità che richiede e utilizza i servizi o le risorse fornite dal server, ed il server è l'entità che fornisce le risorse o i servizi richiesti dal client. In questo caso, il client si interfaccia attraverso una richiesta HTTP con il server per inviare le informazioni necessarie per l'esecuzione del software. Infine, il server elabora la richiesta e restituisce una risposta al client.

6.1 Diagramma delle classi



6.2 Diagramma dei package

Per il diagramma dei package è stato scelto il modello client-server. Entrambi sono suddivisi in sotto-pacchetti. In particolare, il client è diviso in:

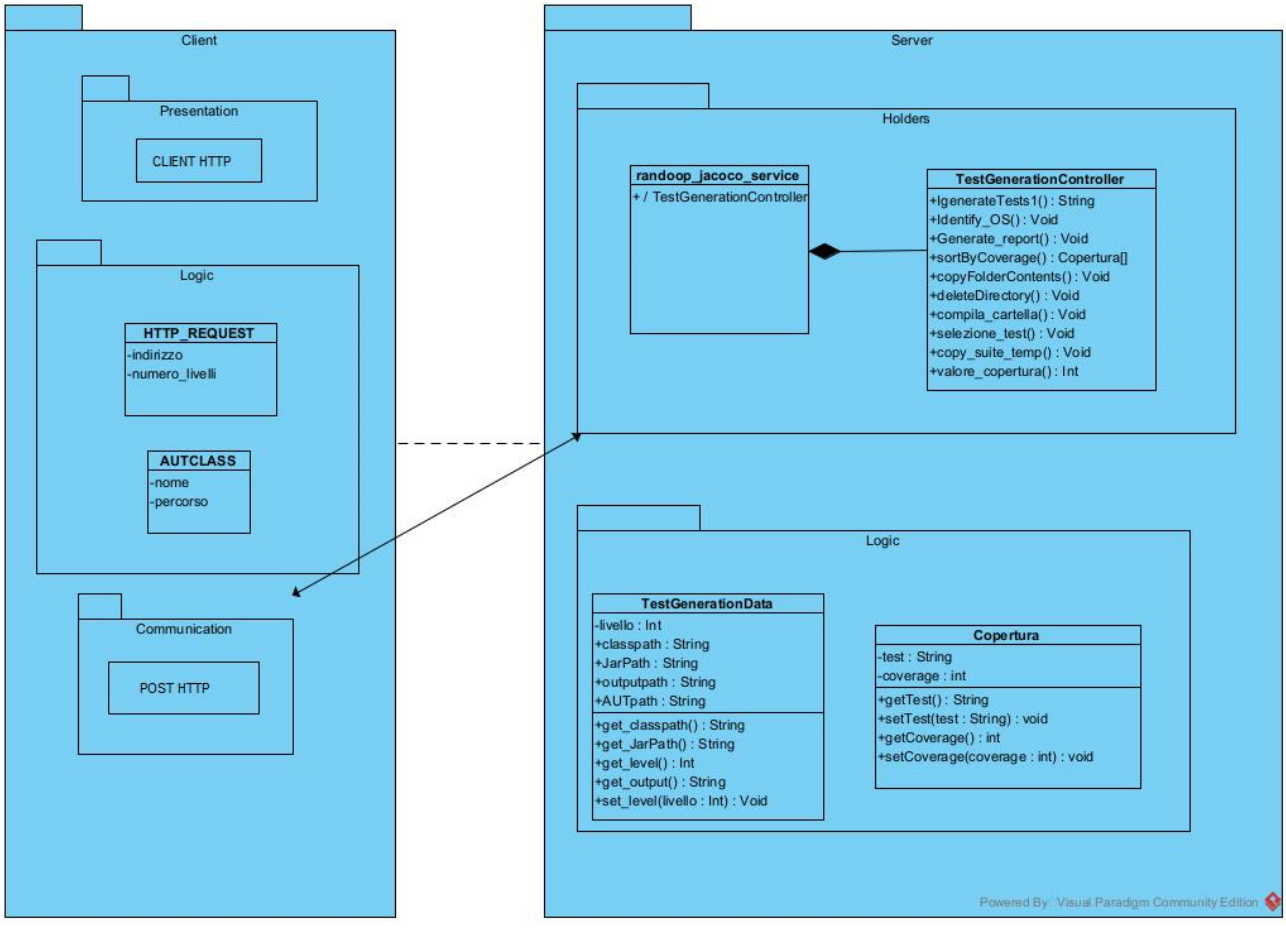
- Presentation, che contiene i componenti che gestiscono l'interfaccia utente del client, nel nostro caso un client HTTP.

- Logic, che contiene la logica del client che gestisce l'elaborazione dei dati e le operazioni specifiche dell'applicazione. Sono state inserite le classi AUTCLASS e HTTP REQUEST, in quanto gestisce la logica di presentazione del client;
- Communication, che gestisce la comunicazione tra il client e il server; nel nostro caso è una POST http.

Analogamente, il Server è stato diviso in:

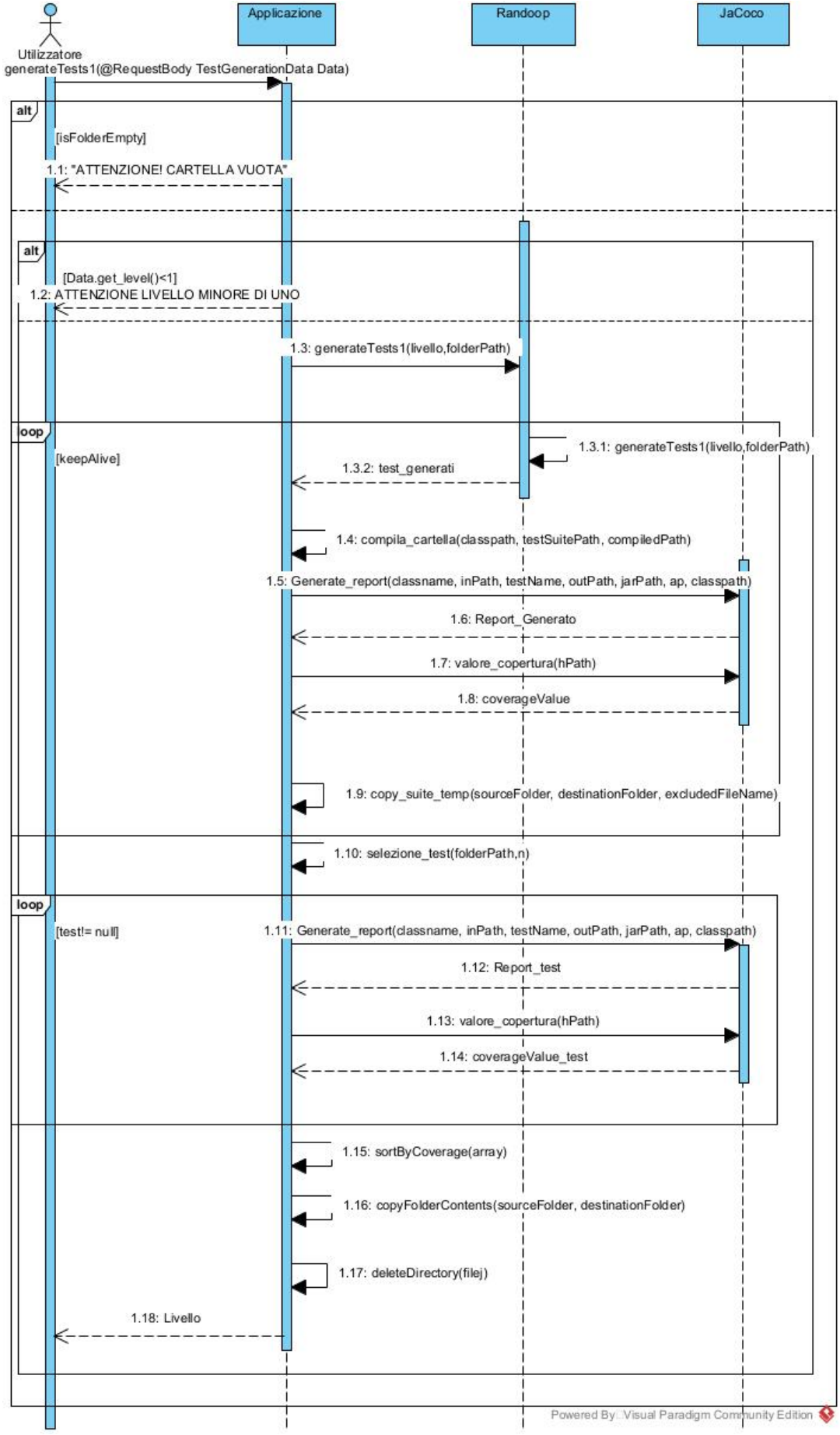
- Holders, rappresentano i componenti che detengono i dati nel client o nel server; sono state inserite le classi TestGenerationController e randoop_jacoco_service, in quanto gestisce la logica di controllo sul server;
- Logic, in cui è stata inserita la classe Copertura perchè contiene la logica di business dell'applicazione e contiene il calcolo dei risultati, e TestGenerationData.

Infine, le frecce bidirezionali tra i pacchetti comunication dal lato client e controllers dal lato server indicano la comunicazione tra client e server: il client invia richieste al server attraverso il pacchetto comunicazione, che poi vengono elaborati dai pacchetti titolari del server per fornire le risposte al client.



6.3 Diagramma di sequenza di dettaglio

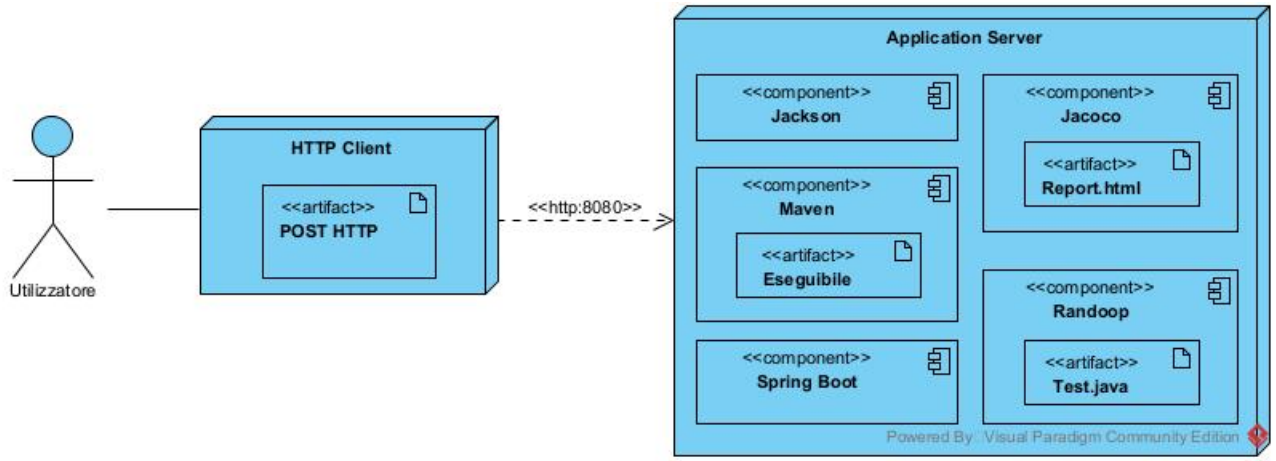
I diagrammi di sequenza consentono di visualizzare le interazioni tra l'attore ed il sistema, individuando le azioni necessarie per descrivere correttamente un caso d'uso.



7. Implementazione

7.1 Diagramma di Deployment

Il diagramma di deployment sviluppato ha la funzione di mostrare come le componenti software e hardware si dispongono fisicamente nel sistema e come le diverse entità esterne interagiscono con esso.



7.2 Analisi ed utilizzo delle tecnologie

Di seguito un approfondimento sulle diverse tecnologie utilizzate:

- **Randoop**: un software di generazione ed esecuzione di test. Il software, ricevuta in input una classe Java da testare, genera dei casi di test in maniera casuale. Successivamente l'applicazione valuta l'eseguibilità dei test generati. In particolare è stato utilizzato Randoop con i seguenti parametri:

1. --test class : nome della classe sotto test
2. --juunit-output-dir : percorso di output dei test prodotti
3. --randomseed : valore intero utilizzato come seme per la generazione casuale dei test (generato in maniera casuale ad ogni esecuzione)
4. --time-limit : tempo limite di generazione dei test (impostato a 120 s)

5. --no-error-revealing-tests : non sono prodotti test la cui esecuzione non va a buon fine (impostato a true)
6. --regression-test-basename : Prefisso dei nomi dei test generati (impostato a "RegressionTest")

• **Jacoco**: un software per la valutazione della copertura dei test eseguiti. Se avviato contestualmente all'esecuzione di un test case, genera un report.html riportante informazioni sulla copertura del codice quali, ad esempio, copertura totale, linee di codice e metodi coperti.

• **Spring**: è un framework open-source per lo sviluppo di applicazioni Java. In particolare, è stato utilizzato Spring Boot per la configurazione e la gestione del web server utilizzando API Restful.

• **Maven**: un framework per la build automation utilizzato per la gestione delle dipendenze, la compilazione e la generazione di un eseguibile del software. Le informazioni di progetto Maven sono contenute nel file "pom.xml".

• **Jackson**: Jackson è una libreria open-source per la serializzazione e deserializzazione di oggetti Java in formato JSON (JavaScript Object Notation). È stato utilizzato per gestire la conversione di oggetti in formato JSON e viceversa.

• **Docker**: per valutare la portabilità del software quindi la sua esecuzione in un ambiente isolato è stato utilizzato il software Docker. Esso crea una macchina virtuale Linux denominata "container" in cui l'applicativo può eseguire.

I container Docker consentono di isolare l'applicazione e le sue dipendenze dal sistema operativo sottostante e dagli altri contenitori presenti sulla stessa macchina garantendo la portabilità dell'applicazione. La persistenza delle informazioni prodotte dall'esecuzione è garantita dal meccanismo di montaggio delle cartelle. È possibile memorizzare i dati sia in un volume Docker sia sulla macchina host.

7.3 Testing

7.4.1 Piano di test funzionale

Di seguito, è riportato il piano di test funzionale per il caso d'uso "Invia_Classe_e_Livelli":

CLASSPATH	LIVELLO
• Stringa alfanumerica non vuota • Stringa con caratteri speciali [ERRORE] • Stringa vuota [ERRORE] • Classe inesistente[ERRORE]	• Intero positivo maggiore di zero • Intero = 0 [ERRORE] • Intero<0 [ERRORE]

Il numero di test da effettuare senza particolari vincoli è: $4 * 3 = 12$.

Introduciamo i vincoli [ERROR].

Il numero di test da eseguire per testare singolarmente i vincoli è 5 (3 per classpath, 2 per livello).

Il numero di test risultante è: (1*1) + 5 = 6

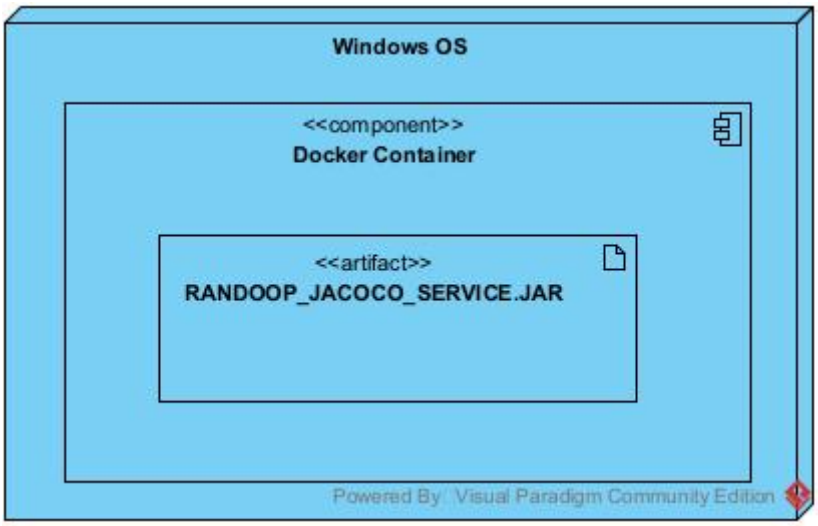
7.4.2 Test suite

Di seguito, si riporta il test suite realizzato per gli input inseriti dall'utilizzatore:

Test Case ID	Descrizione	Classi di equivalenza coperte	Pre-condizioni	Input	Output	Post-condizioni
1	Tutti gli input validi	• Classpath valida • Livello Valido	L'utilizzatore vuole inviare le informazioni	{Claspath: "Inflection", livello: "5"}	Avvio di Randoop	Informazioni acquisite
2	Test non valido: Classpath stringa con caratteri speciali	• Classpath Stringa con caratteri speciali [ERRORE] • Livello valido	L'utilizzatore vuole inviare le informazioni	{Classpath: "Infle%ction@, livello: "5"}	RandoopException	Il sistema chiede nuovamente la classe
3	Test non valido: Classpath stringa vuota	• Classpath Stringa vuota [ERRORE] • Livello valido	L'utilizzatore vuole inviare le informazioni	{Classpath: " ", livello: "5"}	RandoopException	Il sistema chiede nuovamente la classe
4	Test non valido: livello negativo	• Classpath valida • Livello<1 [ERRORE]	L'utilizzatore vuole inviare le informazioni	{Classpath: "Inflection", livelli: "-5"}	RandoopException	Il sistema chiede nuovamente il livello
5	Test non valido: livello pari a zero	Classpath valida Livello = 0 [ERRORE]	L'utilizzatore vuole inviare le informazioni	{Classpath: "Inflection", livello: "0"}	RandoopException	Il sistema chiede nuovamente il livello
6	Test non valido: cartella vuota	Classpath inesistente[ERRORE] Livello valido	L'utilizzatore vuole inviare le informazioni	{Classpath: -, livello: "5"}	RandoopException	null Il sistema chiede nuovamente l'invio della classe

8. Installazione

È stato utilizzato Docker per eseguire l'applicazione in un ambiente isolato. Docker è una piattaforma open-source che consente di creare, distribuire ed eseguire applicazioni all'interno di contenitori. È stato creato un contenitore LINUX con Java 17 che include tutto il necessario per eseguire l'applicazione. Si riporta di seguito il diagramma di deployment che descrive il contesto software di esecuzione:



8.1 Installazione

Prerequisiti: sulla macchina dev'essere preliminarmente installato Maven e Docker dev'essere in esecuzione.

Legenda:

1. *nome _immagine* = nome da dare all'immagine docker che si sta creando
2. *porto* = porto della macchina host su cui mettere in ascolto il servizio
3. *nome_container* = nome da dare al container (istanza di un'immagine) che si sta creando
4. *cartella_da_montare*: (path) deve contenere:
   I. La cartella AUTSourceCode con la classe Java da testare
   II. La cartella RobotTest che contiene la cartella vuota RandoopTest dove saranno generati i livelli di output (codice del test e report di copertura dello stesso)
   Tale cartella può essere situata sulla macchina host eseguente Docker, oppure in un volume di Docker stesso.

Avvio: Nella cartella RANDOOP_LEVEL_GENERATOR, da terminale, avviare i seguenti comandi:

```
mvn package
docker build -t nome_immagine .
```

8.2 Esecuzione

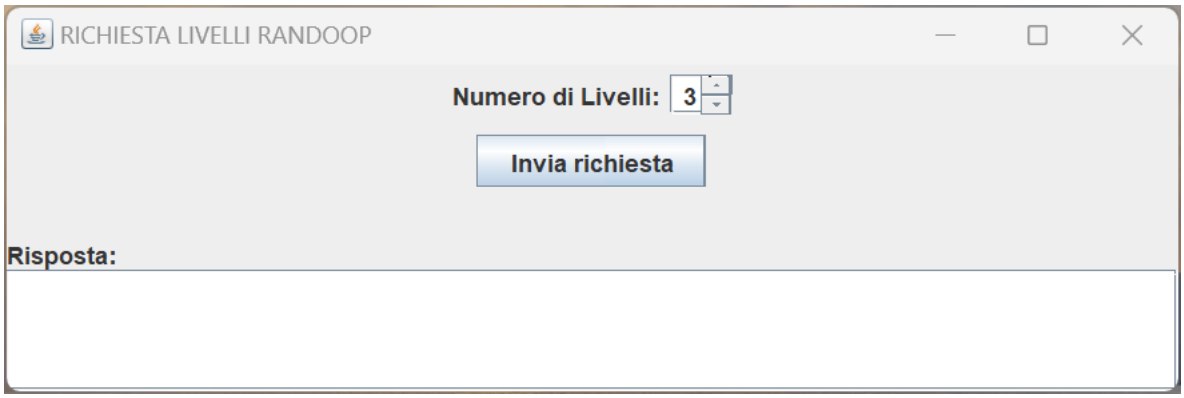
ATTENZIONE: I LIVELLI PRECEDENTEMENTE CREATI SARANNO ELIMINATI!

1) Avviare il comando

```
docker run -it -p porto:8080 --name nome_container -v cartella_da_montare:/AUTName nome_immagine
```

2) Avviare la generazione dei test utilizzando uno dei due successivi metodi:

- Utilizzando la GUI fornita (LEVEL_REQUEST_GUI.jar) e scegliendo il numero di livelli



- Inviando una POST HTTP all'indirizzo "http://localhost:8080/randoop/generate", sostituendo "porto" a 8080.

La richiesta dev'essere in formato json:

```
{
   "Livello": numero_di_livelli_da_richiedere
}
```

3) Attendere che nella cartella RandoopTest vengano generate le cartelle relative ai livelli generati. Ognuna di esse conterrà il codice del test e il rispettivo report di copertura.

```
GENERAZIONE DI 3 LIVELLI SULLA CLASSE ClasseProva
GENERAZIONE TEST RANDOOP
Esecuzione Randoop n. 1
Copertura esecuzione n. 1 = 84%
Esecuzione Randoop n. 2
Copertura esecuzione n. 2 = 87%
Esecuzione Randoop n. 3
Copertura esecuzione n. 3 = 87%
GENERAZIONE DI 3 LIVELLI
Generato livello 1 con copertura del 74%
Generato livello 2 con copertura del 79%
Generato livello 3 con copertura del 87%
ELIMINAZIONE FILE TEMPORANEI
GENERAZIONE DI 3 LIVELLI TERMINATA CON SUCCESSO
```