

## Iterazione 3: Requisito R4

Dicembre 2023

### 1. Documentazione

Il requisito R4 è il seguente:

Il Task T1, che consente all'amministratore di caricare una nuova classe da testare, interagisce con T8 e T9 per richiedere la generazione dei test ai 2 Robot. Tali test sono poi salvati nei volumi condivisi T8 e T9. T1 ha successivamente la responsabilità di salvare nel database di T4 i dati di sintesi sulla classe, sui Robot disponibili per essa, e i relativi livelli dei Test disponibili per ogni Robot. Per fare ciò T1 analizza il File system e deduce tali informazioni, per poi salvarle in T4. Verificare la dinamica di questo comportamento di T1 e prevederne una variante (scenario alternativo) che consenta di precaricare nel file system condiviso i Test dei Robot che siano già stati generati e di salvare in T4 i dati di sintesi di questi test.

Il requisito è stato modificato leggermente aggiungendo la possibilità di precaricare non solo i test ma anche la classe relativa in modo da poter giocare subito.

#### a. User Story

Come amministratore voglio inserire una nuova classe e i relativi test generati da EvoSuite e Randoop

**GIVEN:** L'amministratore ha accesso alla sezione di gestione delle classi.

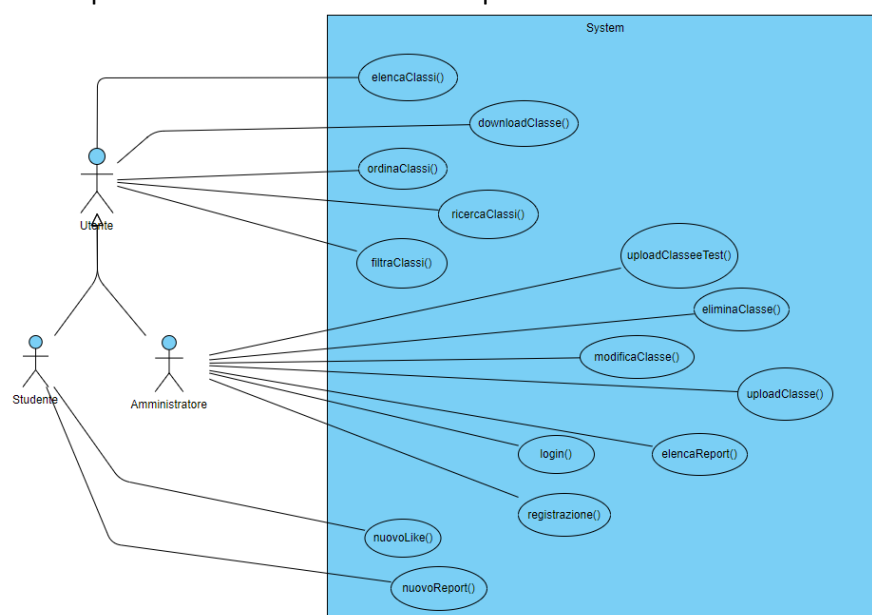
**WHEN:** L'amministratore inserisce i dettagli della nuova classe e dei test e clicca sul pulsante di conferma.

**THEN:** L'elenco delle classi e dei robot da sfidare viene aggiornato.

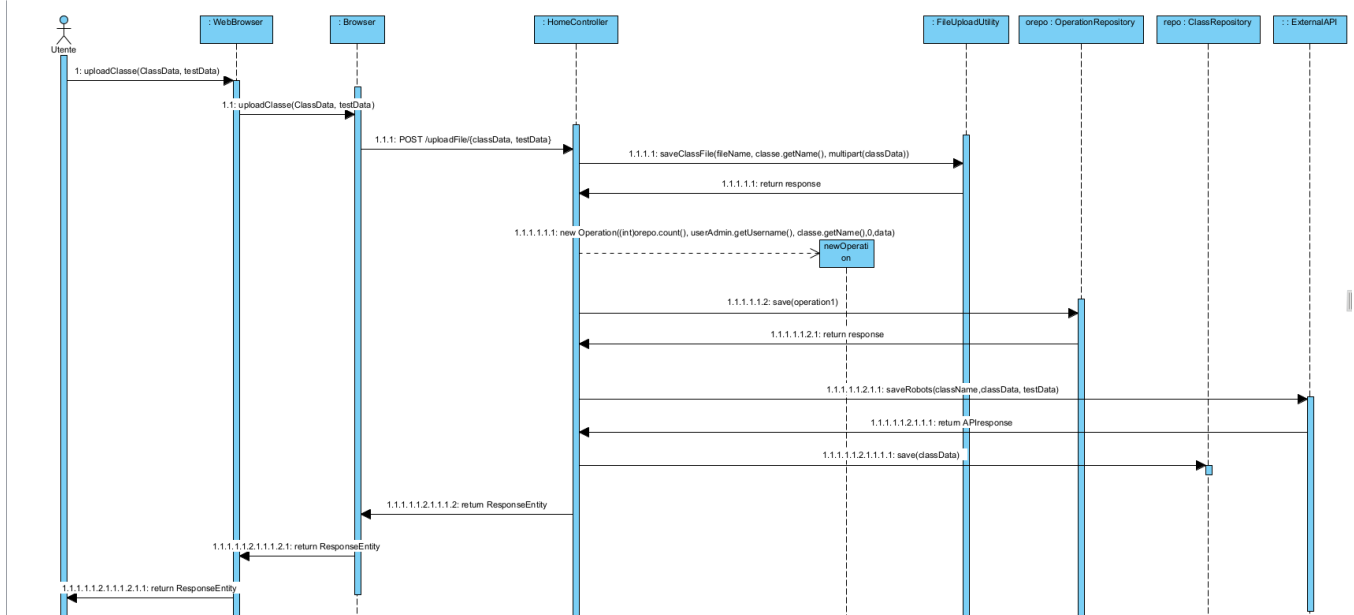
**AND:** Gli utenti possono visualizzare e scaricare il codice sorgente della nuova classe.

#### b. Use Case Diagram di T1 aggiornato

È stata aggiunta la presenza di un nuovo caso d'uso per la nuova funzionalità:



### c. Sequence Diagram



## 2. Refactoring del metodo generateAndSaveRobots

Prima di passare alla nuova funzionalità è stato eseguito un ulteriore refactoring (rispetto alla seconda iterazione) del codice del precedente scenario di caricamento dei test in modo da mantenere una coerenza tra il vecchio e il nuovo codice. In particolare, per migliorare la leggibilità del codice, la funzione è stata divisa in ulteriori funzioni e ognuna di queste rappresenta una fase specifica del caricamento della classe e la generazione dei test. In generale tutte le funzioni create sono riusate più volte all'interno del codice favorendo il riuso e la manutenibilità dello stesso.

Il codice della funzione è il seguente:

```

public static void generateAndSaveRobots(String fileName, String className, MultipartFile classFile) throws IOException {

    // RANDOOP - T9
    Path directory = Paths.get("/VolumeT9/app/FolderTree/" + className + "/" + className + "SourceCode");
    caricaFile(fileName, directory, classFile);

    //Randoop T9
    // creazione del processo esterno di generazione dei test
    ProcessBuilder processBuilder = new ProcessBuilder();

    // con command si configura il comando del processo esterno per eseguire il file
    // JAR 'Task9-G19-0.0.1-SNAPSHOT.jar'
    // l'esecuzione avviene attraverso la JVM di Java.
    // Il parametro "-jar" specifica l'esecuzione di un file JAR.
    processBuilder.command("java", "-jar", "Task9-G19-0.0.1-SNAPSHOT.jar");

    // La directory di lavoro per il processo esterno viene impostata su
    // "/VolumeT9/app/" utilizzando
    // questo metodo garantisce che il processo lavori nella directory desiderata
    processBuilder.directory(new File("/VolumeT9/app/"));

    // linea di debug--potremmo anche commentarla
    System.out.println("Prova");

    // si avvia il processo
    Process process = processBuilder.start();

    //Legge l'output del processo appena creato
    outputProcess(process);

    // Crea un oggetto File che rappresenta il percorso della directory contenente i
    // risultati
    // della generazione di robot da Randoop. Il percorso è costruito in base all'ID
    // della classe di test 'className'.
    File resultsDir = new File("/VolumeT9/app/FolderTree/" + className + "/RobotTest/RandoopTest");

    // Inizializza la variabile 'liv' a 0, rappresentante il massimo livello di
    // robot prodotti da Randoop.
    // Questo valore sarà aggiornato successivamente durante l'analisi dei
    // risultati.
    int liv = 0; //livelli di robot prodotti da randoop

    File results [] = resultsDir.listFiles();

```

```

// Itera attraverso tutti i file nella directory dei risultati della generazione
// di robot da Randoop.
for(File result : results) {

    // Calcola la copertura delle linee per ciascun file XML di copertura estraendo
    // il valore dal file XML 'coveragetot.xml' nella directory corrispondente.
    int score = LineCoverage(result.getAbsolutePath() + "/coveragetot.xml");

    System.out.println(result.toString().substring(result.toString().length() - 7, result.toString().length() - 5));

    // Estrae il livello numerico dall'ultimo tratto del nome della directory,
    // basandosi sulla convenzione specifica naming. Nella convenzione attuale:
    // Directory = 0xlivello -> livello = 0x
    int livello = Integer.parseInt(result.toString().substring(result.toString().length() - 7, result.toString().length() - 5));

    System.out.println("La copertura del livello " + String.valueOf(livello) + " è: " + String.valueOf(score));

    saveT4(score, livello, className);

    // Se il livello del robot generato è superiore al livello massimo attuale,
    // aggiorna il livello massimo.
    if(livello > liv)
        liv = livello;
}

// Il seguente codice è l'adattamento ad evosuite del codice appena visto, i
// passaggi sono gli stessi
// EVOSUITE - T8
// TODO: RICHIEDE AGGIUSTAMENTI IN T8
Path directoryE = Paths.get("/VolumeT8/FolderTreeEvo/" + className + "/" + className + "SourceCode");

caricaFile(fileName, directoryE, classFile);

ProcessBuilder processBuilderE = new ProcessBuilder();

processBuilderE.command("bash", "robot_generazione.sh", className, "\\\"", "/VolumeT9/app/FolderTree/" + className + "/" + className + "SourceCode", String.valueOf(liv));
processBuilderE.directory(new File("/VolumeT8/Prototipo2.0/"));

Process processE = processBuilderE.start();

outputProcess(processE);

File resultsDirE = new File("/VolumeT8/FolderTreeEvo/" + className + "/RobotTest/EvoSuiteTest");

```

```

File resultsE [] = resultsDirE.listFiles();
for(File result : resultsE) {
    int score = LineCoverage(result.getAbsolutePath() + "/TestReport/statistics.csv");

    System.out.println(result.toString().substring(result.toString().length() - 7, result.toString().length() - 5));
    int livello = Integer.parseInt(result.toString().substring(result.toString().length() - 7, result.toString().length() - 5));

    System.out.println("La copertura del livello " + String.valueOf(livello) + " è: " + String.valueOf(score));

    saveT4(score, livello, className);
}

```

### a. Caricamento della classe nel Filesystem

Questa fase è racchiusa nella funzione **caricaFile**.

```
public static void caricaFile(String fileName, Path directory, MultipartFile file) throws IOException{
    try {
        // Verifica se la directory esiste già
        if (!Files.exists(directory)) {
            // Crea la directory
            Files.createDirectories(directory);
            System.out.println("La directory è stata creata con successo.");
        } else {
            System.out.println("La directory esiste già.");
        }
    } catch (Exception e) {
        System.out.println("Errore durante la creazione della directory: " + e.getMessage());
    }
    // Legge l'input stream del file caricato e lo copia nella directory specificata
    try (InputStream inputStream = file.getInputStream()) {
        // Risolve il percorso completo del file all'interno della directory
        // specificata.
        // Viene utilizzato il metodo 'directory.resolve(fileNameClass)' per ottenere il
        // percorso completo
        // del file all'interno della directory 'directory'. Questo percorso completo
        // sarà utilizzato
        // successivamente per copiare il contenuto dell'input stream del file nella
        // posizione desiderata.
        Path filePath = directory.resolve(fileName);
        System.out.println(filePath.toString());

        // copio il contenuto dell'input stream nel file di destinazione
        // l'ultimo parametro di questa funzione indica che se il file già esiste deve
        // essere sostituito
        Files.copy(inputStream, filePath, StandardCopyOption.REPLACE_EXISTING);

        // chiusura dell'input stream dopo aver completato la copia
        inputStream.close();
    }
}
```

### b. Generazione dei test: lettura dell'output del processo

Questa fase è racchiusa nella funzione **outputProcess**. Legge e stampa sulla console di sistema.

```
public static void outputProcess(Process process) throws IOException{
    // Legge l'output del processo esterno tramite un BufferedReader, che a sua
    // volta usa
    // un InputStreamReader per convertire i byte in caratteri. Il metodo
    // 'process.getInputStream()'
    // restituisce lo stream di input del processo esterno.
    BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
    String line;

    // All'interno del loop viene letta ogni linea disponibile finché il processo
    // continua a produrre output.
    while ((line = reader.readLine()) != null)
        System.out.println(line);

    // funzionamento analogo al precedente, invece di leggere l'output leggiamo gli
    // errori
    reader = new BufferedReader(new InputStreamReader(process.getErrorStream()));
    while ((line = reader.readLine()) != null)
        System.out.println(line);

    try {
        // Attende che il processo termini e restituisce il codice di uscita
        int exitCode = process.waitFor();

        System.out.println("ERRORE CODE: " + exitCode);
    } catch (InterruptedException e) {
        System.out.println(e);
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

### c. Salvataggio dei dati nel Task T4

Questa fase è racchiusa nella funzione **saveT4**. Comunica con il task T4 per salvare tutte le informazioni relative ai test condotti dai Robot a valle della valutazione dei risultati.

```
public static void saveT4(int score, int livello, String className) throws IOException{
    // Configurazione di un client HTTP
    HttpClient httpClient = HttpClientBuilder.create().build();

    // Creazione di un oggetto HttpPost con l'URL "http://t4-g18-app-1:3000/robots"
    HttpPost httpPost = new HttpPost("http://t4-g18-app-1:3000/robots");

    // Creazione di un array JSON per contenere le informazioni sui robot generati
    JSONArray arr = new JSONArray();

    // Creazione di un oggetto JSON per rappresentare un singolo robot generato
    JSONObject rob = new JSONObject();

    // L'array JSON viene utilizzato per raggruppare gli oggetti JSON che
    // rappresentano le informazioni sui robot generati.
    // L'array arr contiene una serie di oggetti rob, ognuno dei quali rappresenta
    // le caratteristiche di un robot specifico generato da Randoop.

    // Aggiunge al robot l'informazione relativa al punteggio convertito in stringa
    rob.put("scores", String.valueOf(score));

    // aggiunge al robot l'informazione relativa a quale robot è stato utilizzato,
    // in questo caso randoop
    rob.put("type", "randoop");

    // aggiunge al robot l'informazione riguardante il livello di difficoltà
    // convertito in stringa
    rob.put("difficulty", String.valueOf(livello));

    // aggiunge al robot l'informazione relativa all'id della classe di test
    rob.put("testClassId", className);

    // Aggiunge l'oggetto robot all'array JSON
    arr.put(rob);

    // Crea un oggetto JSON principale contenente l'array di robot
    JSONObject obj = new JSONObject();

    // inserimento dell'array di robot all'interno dell'oggetto
    obj.put("robots", arr);

    // Crea un'entità JSON utilizzando il contenuto dell'oggetto JSON principale.
    StringEntity jsonEntity = new StringEntity(obj.toString(), ContentType.APPLICATION_JSON);

    // Configura la richiesta POST con l'entità JSON creata
    httpPost.setEntity(jsonEntity);

    // esegue la richiesta ed ottiene la risposta
    HttpResponse response = httpClient.execute(httpPost);
}
```

### 3. Struttura dei file prodotti da Randoop:

Prima di passare alla nuova funzionalità è stato necessario capire sotto quale forma caricare tutti i file dei test e che struttura hanno i risultati della generazione. La struttura è la seguente:

```
+01level
+02level
+03level
|
|
|
+0xlevel
```

Per rendere semplice e veloce l'inserimento dei file dal front-end, abbiamo deciso di rendere possibile il caricamento di questa struttura all'interno di un file .zip che verrà successivamente scompattato dal

back-end. Quest'ultimo, inoltre, eseguirà anche il *rename* dell'archivio in modo da rendere il file trattabile dalla funzione che estrae i file dall'archivio. La convenzione decisa per il nome dei file zip caricati è:

**|nomeClasse|Test|nomeRobot|.zip**

Inoltre, durante lo sviluppo della nuova funzione, ci siamo accorti che non era possibile caricare file superiori a una certa dimensione imposta da Spring di default.

Abbiamo quindi modificato i file **default.conf** aggiungendo:

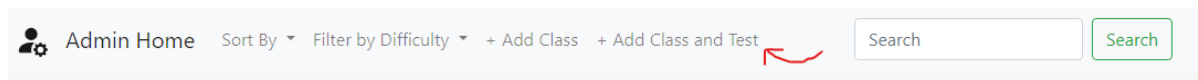
```
client_max_body_size 5M;  
client_body_timeout 60s;
```

e **application.properties**:

```
spring.servlet.multipart.max-file-size=5MB  
spring.servlet.multipart.max-request-size=5MB
```

#### 4. Modifiche alla pagina home\_adm:

È stato aggiunto un nuovo elemento nella barra di navigazione della home per gli admin in modo da non creare confusione nel form già esistente per il caricamento e la generazione di test.




Il codice aggiunto è il seguente:

```
<li class="nav-item">  
  <a class="nav-link" href="/uploadClasseAndTest">+ Add Class and Test</a>  
</li>
```

#### 5. Pagina uploadClasseAndTest:

La nuova pagina appare molto simile alla precedente con l'aggiunta dell'input del file zip per i test.

Test upload



Class name

Date

Difficulty Beginner

Select the difficulty level of this item

Description

Category 1

Category 2

Category 3

Upload your class (java): Scegli il file | Nessun file scelto

Upload your test Randoop (zip): Scegli il file | Nessun file scelto

Upload your test Evosuite (zip): Scegli il file | Nessun file scelto

Upload class and tests

Il codice aggiunto rispetto alla pagina già esistente è il seguente:

```
<div class="form-group mt-3">
  <label class="mr-2">Upload your test Randoop (.zip) :</label>
  <input type="file" name="test" id="testInput">
</div>

<div class="form-group mt-3">
  <label class="mr-2">Upload your test Evosuite (.zip) :</label>
  <input type="file" name="testEvo" id="testInputEvo">
</div>
```

```
<div class="form-group mt-3">
  <button type="button" onclick="uploadTest()" class="btn btn-success">Upload class and tests</button>
</div>
```

```
<script>
  //Si aggancia al form tramite ID
  const formTest = document.getElementById('formId');

  //Funzione per l'upload della classe e dei relativi test
  function uploadTest(event) {

    //Creazione delle costanti che rappresentano i dati input al form
    const name = document.getElementById('className').value;
    const date = document.getElementById('date').value;
    const difficulty = document.getElementById('difficulty').value;
    const code_Url = "" ;
    const description = document.getElementById('description').value;
    const category = [
      document.getElementById('category1').value,
      document.getElementById('category2').value,
      document.getElementById('category3').value
    ];

    //Costante che rappresenta il file .java in input della classe
    const classInput = document.getElementById('fileInput');
    const file = classInput.files[0];
    //Costante che rappresenta il file .zip in input dei test Randoop
    const testInput = document.getElementById('testInput');
    const test = testInput.files[0];
    //Costante che rappresenta il file .zip in input dei test Randoop
    const testInputEvo = document.getElementById('testInputEvo');
    const testEvo = testInputEvo.files[0];

    //Costante che rappresenta l'insieme dei dati del form e i file in input
    const formData = new FormData();
    formData.append('file', file);
    formData.append('model', JSON.stringify({
      name: name,
      date: date,
      difficulty: difficulty,
      code_Url: code_Url,
      description: description,
      category: category
    }));
    formData.append('test', test);
    formData.append('testEvo', testEvo);

    //chiamata HTTP alla funzione uploadTest in HomeController con metodo POST e dati del form come body
    fetch('/uploadTest', {
      method: 'POST',
      body: formData
    })
    .then(response => response.json())
    .then(data => {
      console.log('Success:', data);
      window.location.href = "/home_adm";
      // Aggiungi qui il codice per gestire la risposta dal server
    })
    .catch((error) => {
      console.error('Error:', error);
      // Aggiungi qui il codice per gestire gli errori
    });
  }
</script>
```

## 6. Modifiche al controller:

Per mappare la nuova pagina e collegarla al controller sono state fatte le seguenti modifiche:

### a. default.conf:

```
1. location ~
   ^/(loginAdmin|registraAdmin|home_adm|modificaClasse|orderBydate|Dfilterby.+|orderbyname|Reports|uploadClasse|uploadClasseAndTest|reportClasse|delete|getLikes|uploadFile|uploadTest|home|t1) {
2.     include /etc/nginx/includes/proxy.conf;
3.     proxy_pass http://manvsclass-controller-1:8080;
4. }
```

### b. HomeController.java:

```
@GetMapping("/uploadClasseAndTest")
public String showUploadClasseAndTest() {
    return "uploadClasseAndTest";
}
```

Utilizza l'annotazione `@GetMapping` per mappare una richiesta HTTP GET all' endpoint `/uploadClasseAndTest`.

Inoltre, è stato aggiunto un nuovo metodo per la richiesta HTTP POST di caricamento dei file della classe e dei test:

```
@PostMapping("/uploadTest")
@ResponseBody
public ResponseEntity<FileUploadResponse> uploadTest(@RequestParam("file") MultipartFile classFile, @RequestParam("model") String model, @RequestParam("test") MultipartFile testFile,
    @RequestParam("testEvo") MultipartFile testFileEvo) throws IOException {

    //Legge i metadati della classe della parte "model" del body HTTP e li salva in un oggetto ClasseUF
    ObjectMapper mapper = new ObjectMapper();
    ClassUF classe = mapper.readValue(model, ClassUF.class);

    //Salva il nome del file della classe caricato
    String fileNameClass = StringUtils.cleanPath(classFile.getOriginalFilename());
    long size = classFile.getSize();

    //Salva la classe nel filesystem condiviso
    FileUploadUtil.saveClassFile(fileNameClass, classe.getName(), classFile);

    //Salva i test nel filesystem condiviso
    String fileNameTest = StringUtils.cleanPath(testFile.getOriginalFilename());
    String fileNameTestEvo = StringUtils.cleanPath(testFileEvo.getOriginalFilename());
    RobotUtil.saveRobots(fileNameClass, fileNameTest, fileNameTestEvo, classe.getName(), classFile, testFile, testFileEvo);

    FileUploadResponse response = new FileUploadResponse();
    response.setFileName(fileNameClass);
    response.setSize(size);
    response.setDownloadUri("/downloadFile");

    //Setta data di caricamento e percorso di download della classe
    classe.setUri("Files-Upload/" + classe.getName() + "/" + fileNameClass);
    classe.setDate(today.toString());

    //Creazione dell'oggetto riguardante l'operazione appena fatta
    LocalDate currentDate = LocalDate.now();
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
    String data = currentDate.format(formatter);
    Operation operation1 = new Operation((int) orepo.count(), userAdmin.getUsername(), classe.getName() + " con Robot", 0, data);

    //Salva i dati sull'operazione fatta nel database
    orepo.save(operation1);
    //Salva i dati sulla classe nel database
    repo.save(classe);
    return new ResponseEntity<>(response, HttpStatus.OK);
}
```



## 7. Metodo saveRobots:

Sfruttando il refactoring del metodo **generateAndSaveRobots** della classe **RobotUtil**, è stato scritto un nuovo metodo per soddisfare il requisito R4. Il file .zip viene caricato all'interno del filesystem condiviso del robot e subito dopo viene rinominato il file in modo da uniformarsi alla convenzione scelta (3), vengono estratte le cartelle con la struttura vista (3) e infine viene cancellato l'archivio. Il resto dell'elaborazione rimane pressoché identica. Di seguito vediamo il codice:

### a. CARICAMENTO CLASSE E TEST + CONTROLLO SUL NOME DELLE ZIP:

```
/*CARICAMENTO CLASSE NEI VOLUMI T8 E T9*/
Path directory = Paths.get("/VolumeT9/app/FolderTree/" + className + "/" + className + "SourceCode");
Path directoryEvo = Paths.get("/VolumeT8/FolderTreeEvo/" + className + "/" + className + "SourceCode");
caricaFile(fileNameClass, directory, classFile);
caricaFile(fileNameClass, directoryEvo, classFile);

/*CARICAMENTO TEST */
//Randoop
Path directoryTest = Paths.get("/VolumeT9/app/FolderTree/" + className + "/RobotTest/RandoopTest");
caricaFile(fileNameTest, directoryTest, testFile);
//Evosuite
Path directoryTestEvo = Paths.get("/VolumeT8/FolderTreeEvo/" + className + "/RobotTest/EvoSuiteTest");
caricaFile(fileNameTestEvo, directoryTestEvo, testFileEvo);

//Rinomina il file zip caricato secondo la convenzione attuale: |nomeClasse|TestRandoop.zip
File fileZipDir = new File("/VolumeT9/app/FolderTree/" + className + "/RobotTest/RandoopTest/");
File fileZip[] = fileZipDir.listFiles();
String nomeAttuale = fileZip[0].getAbsolutePath().toString();
String nuovoNome = "/VolumeT9/app/FolderTree/" + className + "/RobotTest/RandoopTest/" + className + "TestRandoop.zip";
File zipAttuale = new File(nomeAttuale);
File zipNuova = new File(nuovoNome);
boolean rinominato = zipAttuale.renameTo(zipNuova);

//Rinomina il file zip caricato secondo la convenzione attuale: |nomeClasse|TestEvoSuite.zip
File fileZipDirEvo = new File("/VolumeT8/FolderTreeEvo/" + className + "/RobotTest/EvoSuiteTest");
File fileZipEvo[] = fileZipDirEvo.listFiles();
String nomeAttualeEvo = fileZipEvo[0].getAbsolutePath().toString();
String nuovoNomeEvo = "/VolumeT8/FolderTreeEvo/" + className + "/RobotTest/EvoSuiteTest/" + className + "TestEvoSuite.zip";
File zipAttualeEvo = new File(nomeAttualeEvo);
File zipNuovaEvo = new File(nuovoNomeEvo);
boolean rinominatoEvo = zipAttualeEvo.renameTo(zipNuovaEvo);
```

### b. UNZIP:

```
//Estrae i test dall'archivio caricato : Randoop
String zipRandoop = "/VolumeT9/app/FolderTree/" + className + "/RobotTest/RandoopTest/" + className + "TestRandoop.zip";
File destRandoop = new File("/VolumeT9/app/FolderTree/" + className + "/RobotTest/RandoopTest/");
RobotUtil.unzip(zipRandoop, destRandoop);

//Elimina la zip dei test
Files.delete(Paths.get("/VolumeT9/app/FolderTree/" + className + "/RobotTest/RandoopTest/" + className + "TestRandoop.zip"));

//Estrae i test dall'archivio caricato : Evosuite
String zipEvo = "/VolumeT8/FolderTreeEvo/" + className + "/RobotTest/EvoSuiteTest/" + className + "TestEvoSuite.zip";
File destEvo = new File("/VolumeT8/FolderTreeEvo/" + className + "/RobotTest/EvoSuiteTest/");
RobotUtil.unzip(zipEvo, destEvo);

//Elimina la zip dei test
Files.delete(Paths.get("/VolumeT8/FolderTreeEvo/" + className + "/RobotTest/EvoSuiteTest/" + className + "TestEvoSuite.zip"));
```

### c. ELABORAZIONE RISULTATI RANDOOP:

```
/*SALVATAGGIO RISULTATI NEL TASK T4 */

// Crea un oggetto File che rappresenta il percorso della directory contenente i
// risultati
// della generazione di robot da Randoop. Il percorso è costruito in base all'ID
// della classe di test 'className'.
File resultsDir = new File("/VolumeT9/app/FolderTree/" + className + "/RobotTest/RandoopTest");

// Inizializza la variabile 'liv' a 0, rappresentante il massimo livello di
// robot prodotti da Randoop.
// Questo valore sarà aggiornato successivamente durante l'analisi dei
// risultati.
int liv = 0; // livelli di robot prodotti da randoop

File results[] = resultsDir.listFiles();

// Itera attraverso tutti i file nella directory dei risultati della generazione
// di robot da Randoop.

for (File result : results) {

    // Calcola la copertura delle linee per ciascun file XML di copertura estraendo
    // il valore dal file XML 'coveragetot.xml' nella directory corrispondente.
    int score = LineCoverage(result.getAbsolutePath() + "/coveragetot.xml");
    // Stampa le informazioni sulla copertura del livello
    System.out.println(
        result.toString().substring(result.toString().length() - 7, result.toString().length() - 5));

    // Estrae il livello numerico dall'ultimo tratto del nome della directory,
    // basandosi sulla convenzione specifica naming. Nella convenzione attuale:
    // Directory = 0xlivello -> livello = 0x
    int livello = Integer.parseInt(
        result.toString().substring(result.toString().length() - 7, result.toString().length() - 5));

    System.out.println("La copertura del livello " + String.valueOf(livello) + " è: " + String.valueOf(score));

    saveT4(score, livello, className);

    // Se il livello del robot generato è superiore al livello massimo attuale,
    // aggiorna il livello massimo.
    if (livello > liv)
        liv = livello;
}
}
```

### d. ELABORAZIONE RISULTATI EVOSUITE:

```
/*TODO: AGGIUSTAMENTI T8 PER EVOSUITE */
// Il seguente codice è l'adattamento ad evosuite del codice appena visto, i
// passaggi sono gli stessi
File resultsDirEvo = new File("/VolumeT8/FolderTreeEvo/" + className + "/RobotTest/EvoSuiteTest");

File resultsEvo [] = resultsDirEvo.listFiles();
for (File result : resultsEvo) {
    int score = LineCoverage(result.getAbsolutePath() + "/TestReport/statistics.csv");

    System.out.println(result.toString().substring(result.toString().length() - 7, result.toString().length() - 5));
    int livello = Integer.parseInt(result.toString().substring(result.toString().length() - 7, result.toString().length() - 5));

    System.out.println("La copertura del livello " + String.valueOf(livello) + " è: " + String.valueOf(score));

    saveT4(score, livello, className);
}
}
```

Per realizzare la funzione di *unzip* abbiamo fatto uso della libreria *java.util.zip*:

```
//-----FUNZIONE UNZIP
public static File newFile(File destinationDir, ZipEntry zipEntry) throws IOException {
    File destFile = new File(destinationDir, zipEntry.getName());

    String destDirPath = destinationDir.getCanonicalPath();
    String destFilePath = destFile.getCanonicalPath();

    if (!destFilePath.startsWith(destDirPath + File.separator)) {
        throw new IOException("Entry is outside of the target dir: " + zipEntry.getName());
    }

    return destFile;
}

public static void unzip(String className) throws IOException {
    String fileZip = "/VolumeT9/app/FolderTree/" + className + "/RobotTest/RandoopTest/" + className + "TestRandoop.zip";
    File destDir = new File("/VolumeT9/app/FolderTree/" + className + "/RobotTest/RandoopTest/");

    byte[] buffer = new byte[1024];
    ZipInputStream zis = new ZipInputStream(new FileInputStream(fileZip));
    ZipEntry zipEntry = zis.getNextEntry();
    while (zipEntry != null) {
        File newFile = newFile(destDir, zipEntry);
        if (zipEntry.isDirectory()) {
            if (!newFile.isDirectory() && !newFile.mkdirs()) {
                throw new IOException("Failed to create directory " + newFile);
            }
        } else {
            // fix for Windows-created archives
            File parent = newFile.getParentFile();
            if (!parent.isDirectory() && !parent.mkdirs()) {
                throw new IOException("Failed to create directory " + parent);
            }

            // write file content
            FileOutputStream fos = new FileOutputStream(newFile);
            int len;
            while ((len = zis.read(buffer)) > 0) {
                fos.write(buffer, 0, len);
            }
            fos.close();
        }
        zipEntry = zis.getNextEntry();
    }
    zis.closeEntry();
    zis.close();
}
//-----
```