



Scuola Politecnica e delle Scienze di Base

|  |
| --- |
|  |

|  |
| --- |
| Elaborato in ***Software Architecture Design***  ***YARD MANAGEMENT SYSTEM***  Anno Accademico 2020/2021 |

|  |
| --- |
| Candidati: Matricola:    **Fabio Martone M63001142**  **Antonio Landolfi M63001134** |
|  |
| **Lorenzo Savarese M63001240** |
| **Benito Visone M63001177** |
|  |
|  |

**Indice**

[Introduzione 3](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065572)

[Capitolo 1: 5](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065573)

[1.1 5](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574)

[1.2 6](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574)

[1.3 7](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574)

[1.4 7](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574)

[1.4.1 9](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574)

[1.5 9](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574)

[Capitolo 2: 10](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065573)

[2.1 10](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574)

[2.2 11](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574)

[2.3 12](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574)

2.4 15

[2.5 15](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574)

[2.5.1 15](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574)

[2.5.2 16](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574)

[2.5.3 17](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574)

[2.5.4 18](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574)

[2.6 19](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574)

[Capitolo 3: 22](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065573)

[3.1 22](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574)

[3.2](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574) 24

[3.2](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574) 25

[3.2](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574) 38

[Capitolo 4: 22](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065573)

[4.1 22](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574)

[4.2](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574) 24

[4.3](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574) 25

[4.4](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065574) 3

[Conclusioni](file:///D:\Desktop\UNIVERSITA'%20FABIO\TRIENNALE\TESI\TESI.docx#_Toc380065576) 40

**INTRODUZIONE**

L’elaborato mostra lo sviluppo di un’applicazione web che permette di gestire in modo efficiente le prenotazioni per le operazioni di carico e scarico all’interno di un magazzino. Si è parlato a lungo dei software di yard management come di prodotti cruciali per massimizzare l’efficienza del reparto di logistica. Negli ultimi anni i software per il piazzale automezzi sono diventati qualcosa in più, essendo ormai ampiamente riconosciuti come strumenti in grado di aumentare anche il livello di sicurezza all’interno dell’azienda. Si tratta di un processo attraverso il quale un’azienda – di qualsiasi dimensione – **pianifica, gestisce e organizza** i flussi di mezzi e di persone in entrata e in uscita dai relativi depositi.

Il principio fondante della gestione del magazzino e delle scorte è, come detto, quello dell’efficienza. Ogni elemento, ogni risorsa, ogni gesto necessita di un coordinamento studiato in funzione di precisi obiettivi di business. Il focus si sposta quindi dalle dinamiche interne del magazzino per assumere poi una visione più vasta relativa alla logistica. È necessario che ad ogni azione compiuta per il magazzino corrisponda una traccia precisa da documentare. Solo in questo modo diventa possibile studiare i movimenti all’interno del magazzino per sistematizzarli in modo che rispondano a criteri di efficienza che riguardano l’intera organizzazione.

Abbiamo sviluppato il seguente sistema allo scopo di soddisfare la necessità di voler regolamentare l’accesso al magazzino, per garantire una maggiore efficienza del processo di movimentazione merci e aumentare il livello di sicurezza riducendo il numero di accessi contemporanei.

**CAPITOLO I**

**RANDOM TESTING**

* 1. **SCRUM**

Scrum è un framework agile per la gestione del ciclo di sviluppo del software, iterativo ed incrementale, concepito per gestire progetti e prodotti software o applicazioni di sviluppo, creato e sviluppato da Ken Schwaber e Jeff Sutherland.

I fondatori di Scrum hanno scritto una guida per rendere chiari i principi da seguire e spiegare nel dettaglio quello che Scrum rappresenta:

«Scrum è un framework di processo utilizzato dai primi anni Novanta per gestire lo sviluppo di prodotti complessi. Scrum non è un processo o una tecnica per costruire prodotti ma piuttosto è un framework all’interno del quale è possibile utilizzare vari processi e tecniche. Scrum rende chiara l'efficacia relativa del proprio product management e delle proprie pratiche di sviluppo così da poterle migliorare.»

Scrum si basa sulla teoria del controllo empirico dei processi, o empirismo. L’empirismo afferma che la conoscenza deriva dall’esperienza e che le decisioni si basano su ciò che si conosce. I pilastri che sostengono ogni implementazione del controllo empirico di processo sono: trasparenza, ispezione e adattamento.

Scrum prevede tre fasi:

1. Una **fase iniziale** di pianificazione volta a definire gli obiettivi generali e alla progettazione dell’architettura del software.
2. Una serie di cicli di **Sprint** ognuno dei quali fornisce in output un incremento del sistema. Gli sprint hanno una durata costante (breve, dalle due alle sei settimane) durante il progetto ed all’inizio di ciascuno di essi viene selezionato un insieme di funzionalità da sviluppare entro il tempo stabilito.
3. Una **fase di chiusura** in cui si completa la documentazione tramite guide e manuali utente e si impacchetta il “progetto”.

Il framework è costituito dal Team Scrum, dai Ruoli, dagli Eventi, dagli Artefati e dalle Regole ad essi associati.

* + 1. **Team Scrum**

Il team Scrum è costituito da un product owner, ovvero un componente che si occupa di gestire il lavoro da svolgere cercando di massimizzare il valore del prodotto, uno Scrum Master responsabile di sostenere Scrum e di aiutare tutti i componenti del team a rispettare le sue regole e le sue pratiche e di un team di sviluppo costituito da un numero limitato di persone che lavorano ad ogni sprint al fine di consegnare un incremento di prodotto rilasciabile ed eseguibile. Riguardo il team di sviluppo Scrum sostiene che i gruppi di persone devono organizzarsi, non è previsto un “capo” in quanto tutti devono avere le stesse responsabilità.

* + 1. **Artefatti di Scrum**

L’intero processo porta alla creazione di alcuni artefatti tra cui un **product backlog** e degli **sprint backlog**, uno per ogni ciclo di sprint. Il product backlog è una lista di lavori da portare a termine per realizza il prodotto mentre lo sprint backlog viene creato scegliendo all’inizio di ogni ciclo di sprint, solo alcuni degli item presenti nel product backlog.

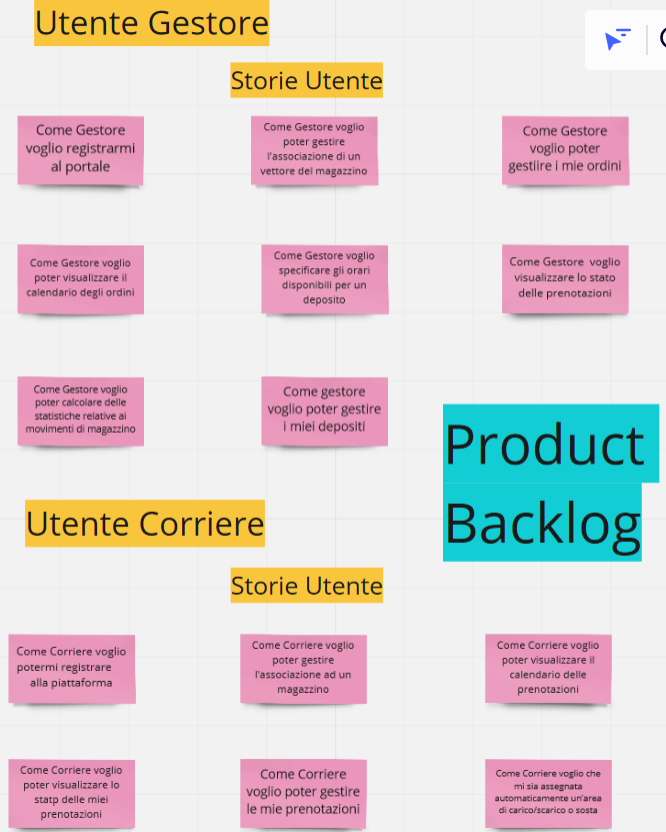
* + 1. **Eventi di Scrum**

Gli eventi previsti sono utilizzati in Scrum per creare regolarità e ridurre al minimo la necessità di riunioni non definite da Scrum stesso. Scrum utilizza eventi time-box, in modo che ogni evento abbia una durata massima. Questo assicura che una quantità appropriata di tempo sia trascorsa pianificando, senza permettere l'introduzione di sprechi nel processo di pianificazione.

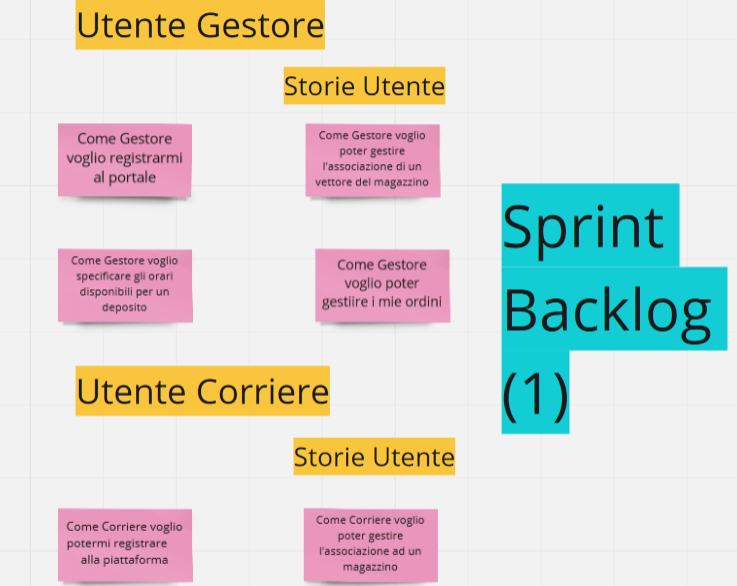
Un possibile esempio è rappresentato dal **Daily Scrum** una riunione giornaliera di al più 15 minuti che consente di ottimizzare la collaborazione tra i membri del team e massimizzare le prestazioni ispezionando il lavoro effettuato dall’ultimo Daily Scrum e pianificando le successive mosse. Anche lo Sprint Review è un evento fondamentale in Scrum, un meeting informale tenuto alla fine di uno sprint in cui lo scrum team e gli stakeholders collaborano per ispezionare l’incremento realizzato e, quando necessario, adattare il Product Backlog in base alle modifiche o alle aggiunte richieste dagli stakeholders stessi.

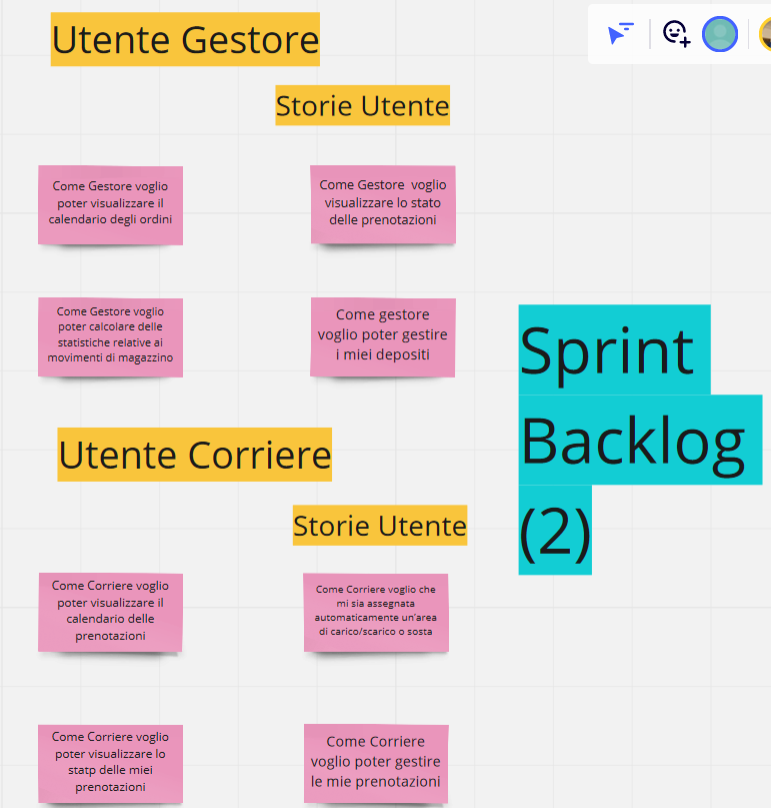
* + 1. **Scrum nel progetto**

Per lo sviluppo del nostro software abbiamo adottato SCRUM, seguendo i principi e le metodologie previste dal framework. Il team di sviluppo è composto da 4 membri tra i quali uno Scrum Master. Sono stati effettuati due cicli di sprint ognuno dei quali della durata di una settimana. Durante la fase iniziale è stato definito un Product Backlog che riportiamo:



All’inizio di ogni Sprint è stato realizzato uno Sprint Backlog con il quale abbiamo definito gli item da portare a termine durante l’iterazione. La scelta è stata effettuata considerando per ogni storia utente una priorità univoca ed una stima della mole di lavoro (in giorni). Di seguito presentiamo le loro rappresentazioni:





Una volta iniziato lo sprint, ogni giorno di prima mattina abbiamo svolto un Daily Scrum di 15 minuti in cui ci si aggiornava sul lavoro svolto e si dividevano a coppie i compiti da svolgere durante la giornata. Inoltre, alla fine delle due iterazioni veniva effettuata una sprint review per valutare l’incremento prodotto e per ragionare sull’andamento del progetto e su possibili modifiche del product backlog.

Le pratiche agili utilizzate sono:

1. **Continuous Integration:** Consiste nell'allineamento frequente (ovvero "molte volte al giorno") dagli ambienti di lavoro degli sviluppatori verso l'ambiente condiviso (mainline).
2. **Pair Programming:** una tecnica agile di sviluppo del software nella quale due programmatori lavorano insieme a una postazione di lavoro
3. **Sviluppo Iterativo.**
4. **Timeboxing:** durata degli eventi fissa e stabilita a priori.
   * 1. **Scrum nel progetto**

Per la realizzazione del sistema sono stati utilizzati alcuni tool nei diversi ambiti:

1. **Google Docs:** per la realizzazione di documentazione online.
2. **Miro:** per la realizzazione del product backlog e degli sprint backlog.
3. **Microsoft WhiteBoard:** utilizzato come lavagna durante le Daily Scrum tenute online.
4. **Visual Paradigm:** per la realizzazione dei diagrammi in linguaggio UML.
5. **Eclipse:** come ambiente di sviluppo.
6. **Github:** per la gestione del repository di progetto.
   1. **Descrizione dei requisiti e vincoli di progetto**

Il sistema deve distinguere due diverse tipologie di utente: un gestore del magazzino e un corriere. Entrambi devono avere la possibilità di potersi identificare all’interno del sistema mediante delle credenziali d’accesso.

Il gestore deve poter regolamentare gli accessi al deposito e definire le fasce orarie prenotabili dai corrieri e le aree utilizzabili dagli stessi. Le aree si distinguono in base alla tipologia che può essere di carico, di scarico o di sosta.

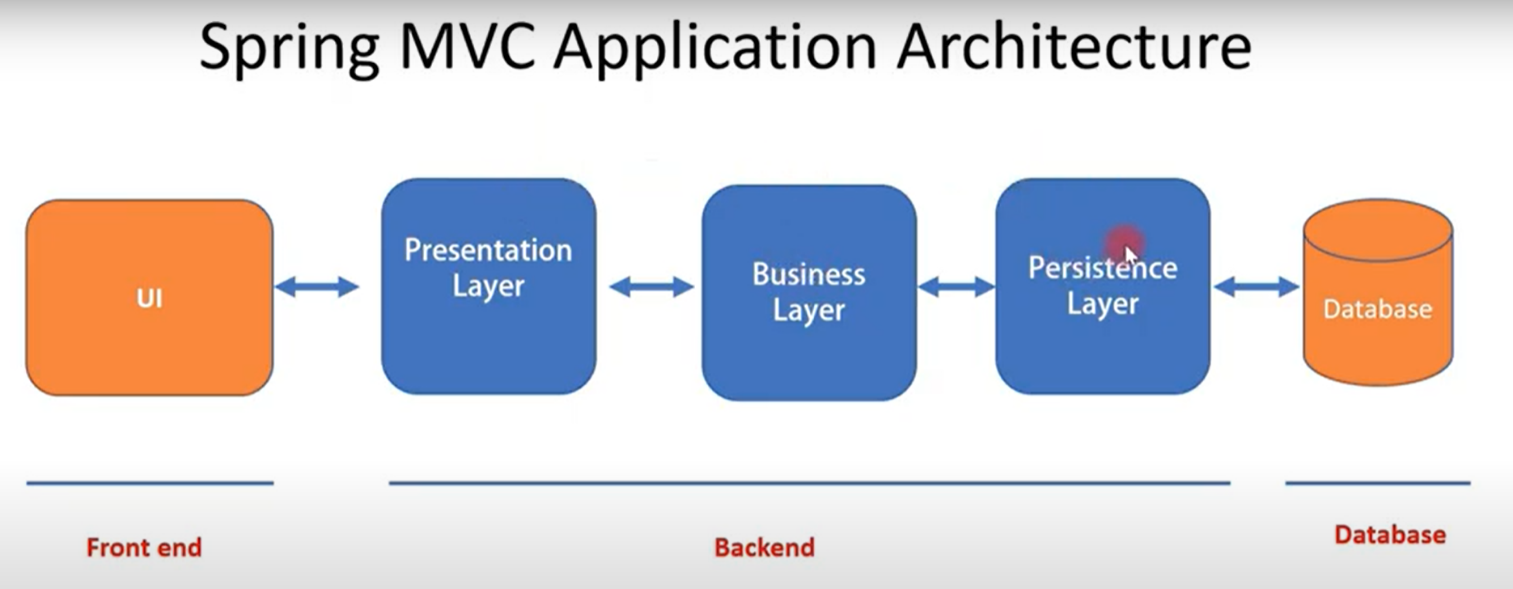
Il gestore del magazzino deve poter inserire degli ordini che si distinguono in ordini di carico e ordini di scarico. Una volta inserito l'ordine il sistema deve generare una chiave che lo identifica univocamente. Nel caso del carico, tramite un sistema di notifica il corriere segnalato dal gestore del magazzino deve ricevere, tramite e-mail, la chiave con la quale può prenotare l'accesso al deposito. Nel caso dello scarico invece, l' e-mail viene inviata al fornitore dal quale il gestore sta acquistando che contatta un corriere al quale comunica la chiave. I corrieri possono poi scaricare la ricevuta di prenotazione, che contiene un codice QR, necessario per l'ingresso e l’uscita dal magazzino.

* Il gestore deve anche avere la possibilità di disdire un ordine.
* Inoltre, deve essere possibile per il gestore la visualizzazione del calendario delle prenotazioni e lo stato di ciascuna di esse.
* Per ogni prenotazione il gestore deve poter modificarne lo stato, aggiornando orario d'ingresso e d’uscita del mezzo dal deposito.
* Il gestore del magazzino deve avere la possibilità di specificare le fasce orarie disponibili per l’accesso al magazzino ed inserire il numero e le tipologie di aree utilizzabili.
* Un corriere deve poter richiedere di essere associato ad un magazzino; viceversa, un gestore deve poter accettare o rifiutare tale richiesta.
* Il corriere può prenotarsi ad un qualsiasi ordine solo se in possesso della relativa chiave e se associato al relativo magazzino, scegliendo una fascia oraria tra quelle libere.
* Il corriere deve anche avere la possibilità di verificare lo stato di tutte le prenotazioni effettuate ed eventualmente di disdirne una.
* Il sistema deve garantire al corriere che si presenta al deposito l’assegnazione automatica di un’area (carico, scarico, sosta) in base alla disponibilità ed in base alla puntualità del trasportatore.
* Il sistema deve notificare le parti interessate in caso di modifica dello stato di una prenotazione. In particolare, il gestore del magazzino viene notificato quando il corriere che si è prenotato per un suo ordine ha annullato la prenotazione. Viceversa, un corriere viene notificato quando un gestore del magazzino annulla un ordine relativo ad una sua prenotazione.
* L’applicazione necessita di comunicare con sistemi esterni, in particolare deve comunicare con un sistema di lettura di codice QR, posto all'ingresso e all'uscita del deposito, il quale deve trasmettere l'identificativo dell'ordine e l'orario di ingresso/uscita al nostro sistema. In ogni caso l'applicazione deve permettere di inserire manualmente questi dati, da parte del gestore, nel caso in cui non fosse disponibile il lettore QR.
* Una funzionalità da prevedere per il sistema è quella del calcolo di alcune statistiche di magazzino, che permettano di monitorare l’andamento delle varie prenotazioni, in particolare:

1. Tempo medio di permanenza di un vettore all’interno del magazzino;
2. Fasce orarie più affollate;
3. Giorni più affollati;
4. Aree di carico e scarico più/meno performanti.

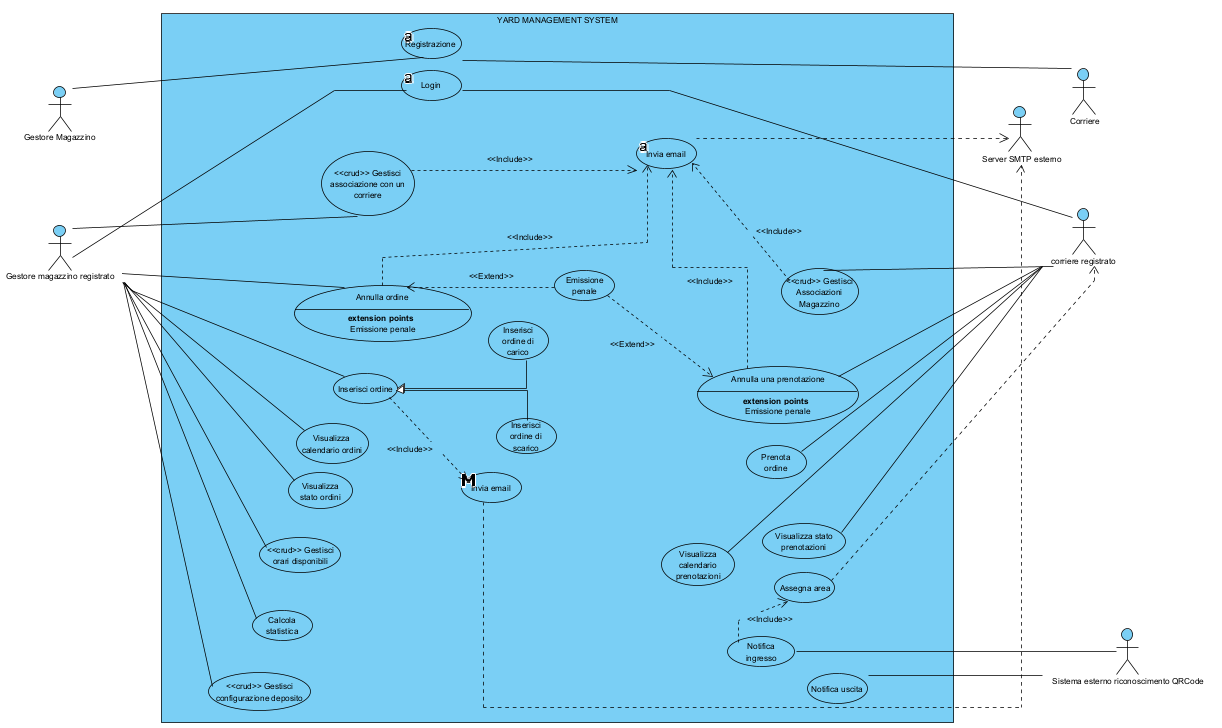
* La risposta del sistema deve essere tempestiva per non causare rallentamenti in fase di associazione area-vettore. Il sistema deve garantire che i dati sensibili non siano condivisi tra diversi utenti. Il sistema deve risultare semplice da utilizzare. É richiesto inoltre che possa essere utilizzato sia su mobile che su personal computer, accedendovi mediante un web browser. L’applicazione deve essere sviluppata in un tempo limite di un mese. Bisogna adottare un processo di sviluppo SCRUM o UP.
  1. **Documentazione di avvio del progetto**

L’architettura del software è stata modellata seguendo il pattern Model View Controller proposto dal framework Spring MVC che abbiamo utilizzato per lo sviluppo:



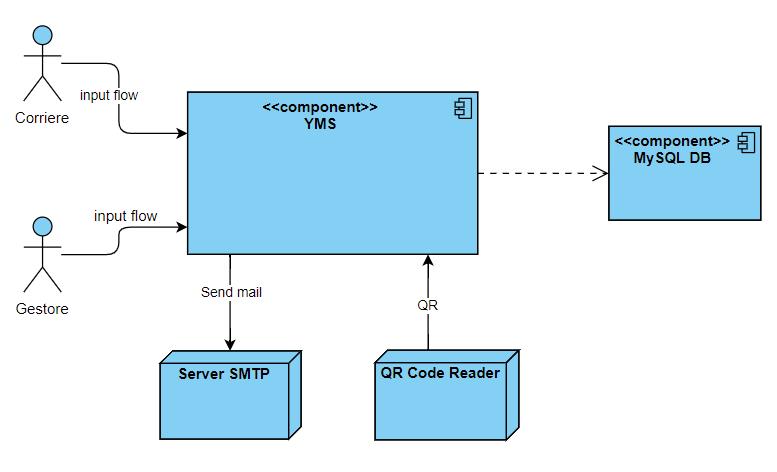
Nell’immagine, il Persistence Layer contiene il modello costruito a partire dal Database. Le informazioni presenti nel modello vengono elaborate dal Business Layer, che implementa la logica di business dell’applicativo, fornendo al Presentation Layer i risultati delle elaborazioni. Infine, quest’ultimo strato si occupa di riorganizzare le informazioni ricevute in una vista che fornisce all’interfaccia utente.

Per comprendere meglio il sistema e le sue funzionalità abbiamo modellato i requisiti mediante un diagramma dei casi d’uso:



Il diagramma chiarisce anche quanti e quali sono gli attori che interagiscono con il sistema: oltre al gestore del magazzino e al corriere, il sistema si interfaccia anche con un Server SMTP esterno per l’invio di e-mail e con un sistema esterno di riconoscimento di codice QR che dovrà essere utilizzato per autorizzare l’ingresso dei corrieri nel magazzino.

L’interazione del SuD con l’esterno può essere visualizzata meglio osservando il diagramma di contesto:

[](https://eu-prod.asyncgw.teams.microsoft.com/v1/objects/0-weu-d10-22500d6dba37c94eb4641bc72ad04e89/views/imgo)

Il sistema riceve input dal corriere, dal gestore e dal lettore di codice QR, mentre invia segnali di controllo al Server SMTP per delegare l’invio di mail. Inoltre, il diagramma mette in risalto anche una relazione di dipendenza tra il sistema e il Database.

**CAPITOLO II**

**PROGETTAZIONE**

* 1. **Stile Client/Server**

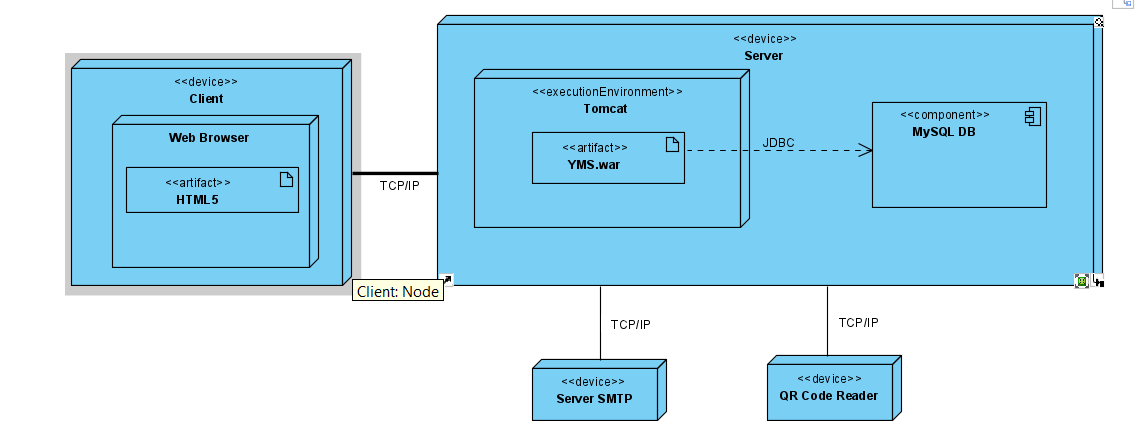
Lo sviluppo di soluzioni informatiche complesse ha reso possibile nel corso degli anni il consolidamento di un patrimonio di conoscenze ed esperienze estremamente significativo. In particolare, sono stati identificati una serie di schemi e stili ricorrenti nella progettazione e costruzione di sistemi complessi. Termini come “architetture client-server” sono entrati nel gergo quotidiano di tutti gli addetti e utenti di tecnologie informatiche. Nella produzione del software, sono stati sviluppati diversi tipi di schemi e stili, in funzione del livello di astrazione al quale ci si pone nello studiare un sistema informatico complesso.

* **Stili architetturali:** uno stile architetturale è una collezione di decisioni di design architetturale (definite da un nome) che sono applicabili in un dato contesto di sviluppo, vincolano le decisioni di design architetturale che sono specifiche per un particolare sistema all’interno di quel contesto e raggiungono qualità vantaggiose in ciascun sistema risultante.
* **Design pattern:** un pattern architetturale è una collezione di decisioni di progetto architetturali con un nome ben definito che sono applicabili ad un problema di design ricorrente e sono parametrici per renderli applicabili in differenti contesti in cui il problema appare.

Per la realizzazione del nostro sistema è stato scelto uno stile Client/Server. Questo è simbolo di un'architettura nella quale genericamente un computer client o terminale si connette ad un server per la fruizione di un certo servizio, quale ad esempio la condivisione di una certa risorsa hardware/software con altri client, appoggiandosi alla sottostante architettura protocollare. Più semplicemente la presenza di un server permette ad un certo numero di client di condividerne le risorse, lasciando che sia il server a gestire gli accessi alle risorse per evitare conflitti di utilizzazione.

L’idea di base è di avere due componenti dell’architettura con due ruoli diversi: un (o più) server che fornisce servizi a più istanze di sottosistemi detti client e i client che chiamano il server per invocare un servizio ed ottenere un risultato. Nel nostro sistema i componenti interagiscono tramite le operazioni CRUD di http (GET, POST, PUT, DELETE). La motivazione relativa alla scelta di tale stile risiede nella facile distribuzione dei componenti e nella scalabilità dell’architettura che ben si presta allo sviluppo di Web Application. Infatti, i service provider non sono consapevoli dei client per cui possono essere aggiunti nuovi client in modo arbitrario. I client sono totalmente indipendenti tra loro e ciò permette di aggiungere, modificare o eliminare client senza dover modificare il resto dei componenti. I client dipendono esclusivamente dai server provider che possono essere aggiunti dinamicamente per sostenere carichi di richieste crescenti. Lo svantaggio di tale stile risiede nella gestione della banda e nel calo delle prestazioni all’aumentare del numero di client.

Al fine di mostrare e documentare come i componenti siano stati distribuiti sui nodi fisici è stato realizzato il seguente diagramma di deploy:



Il seguente diagramma mostra che Client e Server sono presenti su nodi fisici differenti. In particolare, il client accede ai servizi offerti dal server interfacciandosi con delle pagine HTML e comunica con il server tramite il protocollo di rete TCP/IP. Il server è realizzato tramite l’artefatto relativo all’applicativo realizzato, YMS.war, collocato all’interno dell’ambiente d’esecuzione **Apache Tomcat** (o semplicemente Tomcat), un server web (nella forma di contenitore servlet) open source sviluppato dalla Apache Software Foundation. Implementa le specifiche JavaServer Pages (JSP) e servlet, fornendo quindi una piattaforma software per l'esecuzione di applicazioni web sviluppate in linguaggio Java. Il server comunica, sempre tramite protocollo TCP/IP, con un server SMTP per l’invio di e-mail e con un sistema esterno di lettura di codici QR per ricevere le informazioni relative ad entrate e uscite di corrieri al magazzino. È stata anche mostrata la dipendenza da un database MySQL alla quale si connette tramite JDBC(Java DataBase Connectivity), un connettore per databse che consente l’accesso e la gestione della persistenza dei dati sulle basi di dati da qualsiasi programma scritto in linguaggio Java, indipendentemente dal tipo di DBMS utilizzato.

* 1. **Pattern MVC**

Lo stile non definisce tutti gli aspetti dell’architettura, ma solo quelli di high-level. L’architettura del sistema viene definita da altre scelte di progetto come quella relative al pattern architetturale. Nel nostro caso è stato scelto il pattern MVC (Model-View-Controller). Il Model-View-Controller è un pattern architetturale molto diffuso nello sviluppo di sistemi software, in particolare nell'ambito della programmazione orientata agli oggetti e in applicazioni web, in grado di separare la logica di presentazione dei dati dalla logica di business e dalla logica di gestione dei dati. Nacque dalla necessità di visualizzare dati generici tramite interfaccia grafica, mediante l’utilizzo di rappresentazioni diverse dei dati stessi.

Il pattern è basato sulla separazione dei compiti fra i componenti software che interpretano tre ruoli principali:

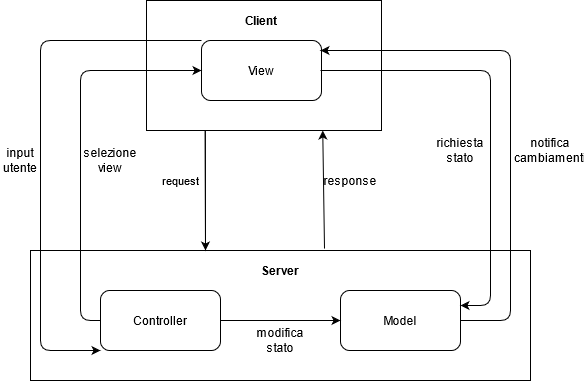
* Il model fornisce i metodi per accedere ai dati utili all'applicazione;
* La view visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti;
* Il controller riceve i comandi dell'utente (in genere attraverso il view) e li attua modificando lo stato degli altri due componenti.

View e Model sono separati utilizzando un protocollo di sottoscrizione-notifica mentre il controller trasforma gli input utente sulla UI in azioni del Model implementando la logica di controllo dell’applicazione. Dal punto di vista implementativo il model contiene le classi le cui istanze rappresentano i dati da visualizzare e manipolare, la view contiene oggetti usati per rendere in output, nella UI, i dati contenuti nel model, nel nostro caso le pagine html, il controller contiene gli oggetti che controlleranno e gestiranno l’interazione sia con il livello view che con il model. Inoltre, il model non ha alcuna dipendenza dagli altri due componenti.

La scelta di questo pattern è stata mossa dai vantaggi ottenuti separando logica di buisness, presentazione dei dati e gestione dei dati:

* L’indipendenza delle varie componenti permette un’efficiente suddivisione del lavoro tra i diversi componenti del team.
* Esiste la possibilità di scrivere viste e controllori diversi, utilizzando lo stesso modello di accesso ai dati e quindi di riutilizzare parte del codice già steso in precedenza. Infatti, la separazione di model e view permette a molte view di usare gli stessi oggetti del modello.
* È di facile supporto per nuovi tipi di client poiché è sufficiente scrivere nuove view e alcuni controller e utilizzare i soliti oggetti del model.
* La possibilità di avere un controllore separato dal resto dell’applicazione rende la progettazione più semplice e permette di concentrare gli sforzi sulla logica del funzionamento.

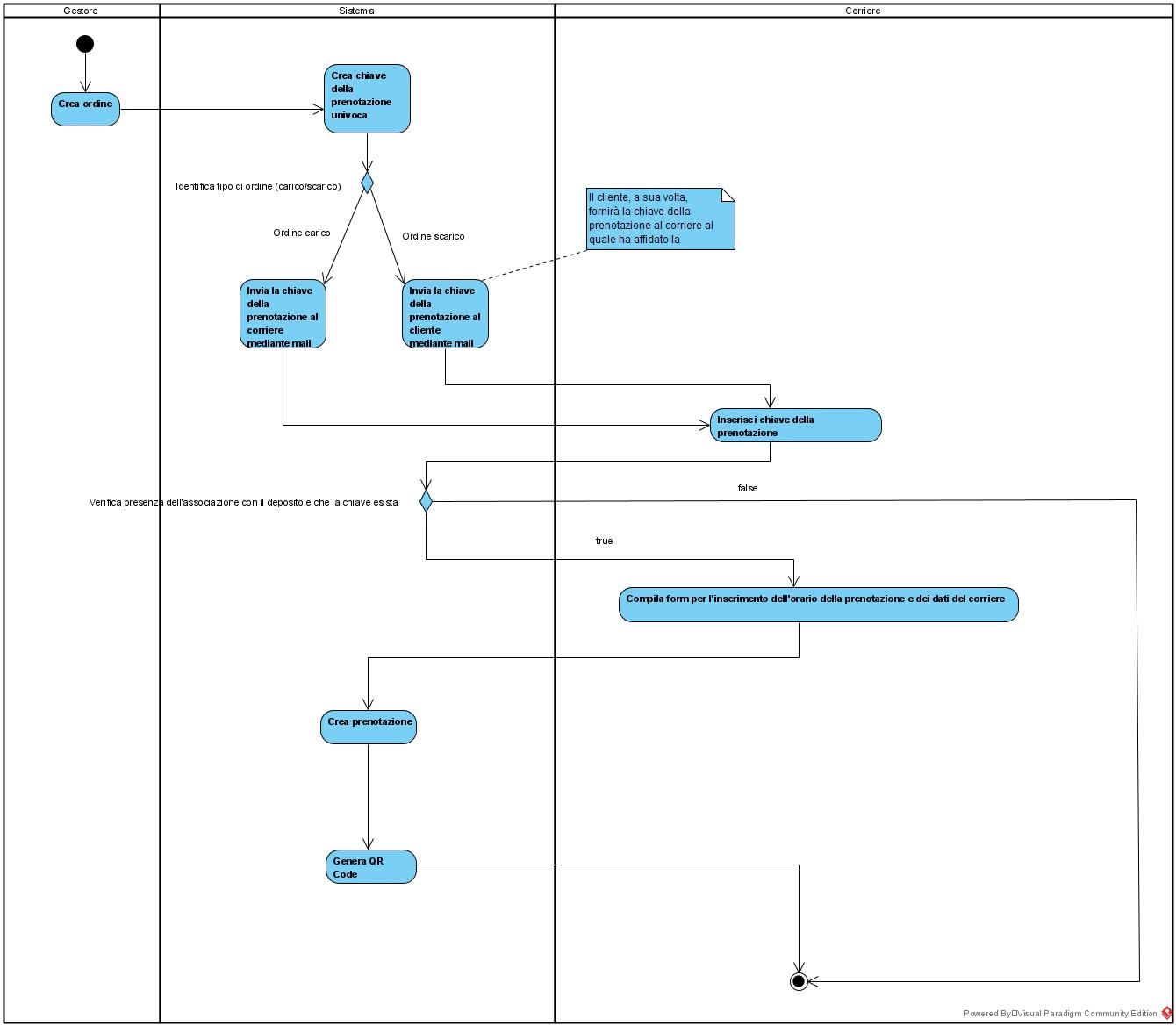
Un’ulteriore motivazione è dettata dal fatto che lo stile client/server si presta perfettamente ad una soluzione utilizzando MVC. L’architettura client/server realizzata è di tipo 2-tiered, la logica di presentazione e su un singolo tier contenuto nel client mentre logica applicativa e di gestione/accesso ai dati in un altro tier contenuto nel server (così come mostrato precedentemente nel diagramma di deploy).



* 1. **Diagrammi di progetto**

La fase di progettazione prevede la documentazione delle scelte di progetto e di conseguenza numerosi diagrammi, solitamente scritti in linguaggio UML, che forniscono diverse viste architetturali del sistema. Tramite Github abbiamo messo a disposizione nella sezione wiki tutta la documentazione, compresa i [diagrammi di progetto](https://github.com/FabioMartone/YardManagementSystem-/wiki/progetto).

Meritano particolare attenzione i diagrammi di attività con i quali è possibile comprendere alcune dinamiche relative alle funzionalità implementate. In particolare, il seguente diagramma presenta le attività da portare a termine al fine di eseguire la funzionalità “effettua prenotazione” e vuole mettere in risalto il fatto che la prenotazione non può avvenire se prima non ci si associa al deposito legato all’ordine da prenotare e soprattutto se prima non ha aver ricevuto una chiave univoca tramite email automatica inviata dal sistema al momento della creazione dell’ordine:



* 1. **Scelte tecnologiche**

Come framework di sviluppo abbiamo scelto Spring. Infatti, sebbene le peculiarità basilari di Spring possano essere adottate in qualsiasi applicazione Java, esistono numerose estensioni per la costruzione di applicazioni web-based (applicazioni web), tra cui Spring Boot.

Per produrre e testare in ambiente Java App, Spring Boot è il modo più semplice ed immediato. Prima della creazione del progetto Spring Boot, tutte le applicazioni basate su Spring avevano bisogno di un web server, come Tomcat, Jetty o Undertow, per essere eseguite. Con Spring Boot è possibile creare una applicazione avente un metodo main che lancia l'intera applicazione web, compreso il web server integrato. In più, con Spring Boot possono essere avviate tutte le applicazioni di Spring, le relative funzionalità e le conseguenti possibilità di configurazione di ambiente. Un approccio che, senza ombra di dubbio, esalta la semplicità d’uso, tanto più che il plug in non richiede codici o configurazioni XML di sorta. Spring è alla portata della maggior parte di tutti coloro che vi si cimentano, anche per la prima volta, pur non essendo una piattaforma prettamente visuale.

Nel file di configurazione pom.xml abbiamo specificato delle dipendenze della nostra applicazione da alcuni starter che ci permettono di utilizzare varie utility, tra cui:

* Spring Data JPA, parte della più vasta famiglia Spring Data, che ci ha facilitato l’implementazione di repository JPA-based semplificando la scrittura di query;
* Spring Security, che ci ha fornito semplici metodi per implementare autenticazione e varie autorizzazioni per la nostra applicazione Java. Ad oggi, è considerato lo standard de-facto per la messa in sicurezza di applicazioni Spring-based;
* Spring Web Services, che si concentra sulla creazione di servizi Web document-driven e che facilita lo sviluppo di servizi SOAP, permettendo la creazione di servizi web flessibili usando uno dei tanti modi per manipolare i carichi XML e distribuirli tra più oggetti;
* Spring Boot Thymeleaf, un’integrazione del motore di template Java Thymeleaf in Spring Boot, che fornisce numerose semplificazioni nella costruzione di file HTML;
* Spring Boot Test, che fornisce librerie e utility a support del testing dell’applicazione;
* Java Mail e servizi di supporto messi a disposizione dal framework di Spring, per semplificare l’implementazione di invio e-mail;
* Spring Boot DevTools, che mira a ottimizzare i tempi di sviluppo mentre si lavora all’applicazione, conferendo l’abilità di effettuare rapidamente setup e esecuzione di servizi.
* MySQL Connectors, che fornisce l’accesso ad una serie di driver standard-based per JDBC, ODBC, ecc… che consentono di sviluppare database app in un linguaggio a scelta dello sviluppatore. In più, una libreria nativa C permette agli sviluppatori di integrare MySQL direttamente nelle loro applicazioni.
* Spring Boot Maven Plugin, che consente di impacchettare archivi jar o war eseguibili, eseguire applicazioni Spring Boot, generare informazioni di build e avviare l’esecuzione dell’applicazione Spring Boot prima di eseguire i test d’integrazione.

Come sistema di gestione del database relazionale è stato scelto MySQL per una serie di motivi. In primo luogo, MySQL è un software Open Source, per cui è possibile per tutti scaricarlo e usarlo in maniera gratuita. Inoltre, il Server del database MySQL è molto veloce, affidabile, scalabile e facile da utilizzare. Il Server MySQL può essere mandato in esecuzione comodamente su un desktop o laptop, affiancato da altre applicazioni, web server e così via, non richiedendo alcuna attenzione. Se si decide di dedicare un intera macchina a MySQL, è possibile configurare alcuni parametri per ottimizzare l’utilizzo della memoria, la potenza della CPU e la capacità di I/O disponibile. MySQL può anche scalare a cluster di macchine interconnesse tra loro. Sebbene sia costantemente sotto sviluppo, il Server MySQL offre oggi un ampio e ricco set di funzioni; la sua connettività, velocità e sicurezza lo hanno reso ottimo nella gestione dell’accesso ai database in Internet.

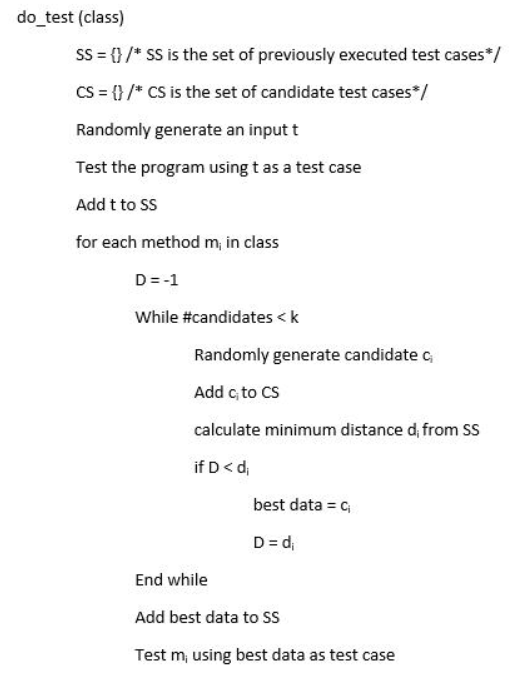
Infine, come strumento di build automation è stato scelto Maven. Maven usa un costrutto conosciuto come Project Object Model (POM): un file XML che descrive le dipendenze fra il progetto e le varie versioni di librerie necessarie nonché le dipendenze fra di esse. In questo modo si separano le librerie dalla directory di progetto utilizzando questo file descrittivo per definirne le relazioni. Inoltre, Maven effettua automaticamente il download di librerie Java e plug-in Maven dai vari repository definiti scaricandoli in locale o in un repository centralizzato lato sviluppo. Questo permette di recuperare in modo uniforme i vari file JAR e di poter spostare il progetto indipendentemente da un ambiente all'altro avendo la sicurezza di utilizzare sempre le stesse versioni delle librerie. Simile per certi versi a strumenti precedenti, come ad esempio Apache Ant, si differenzia però da quest'ultimo per quanto concerne la compilazione del codice. Con questo strumento infatti non è più necessaria la compilazione totale del codice, ma viene fatto uso di una struttura di progetto standardizzata su template definita *archetype*. Il vantaggio derivante dall’utilizzo di questo tool è da subito evidente: se generalmente per sviluppare un software sono necessarie numerose fasi (compilazione in codice binario, packaging dei binari, esecuzione di test per garantire il funzionamento del software, deployment sui sistemi e documentazione relativa al progetto portato a termine), con la build automation l’intero processo viene automatizzato, riducendo il carico di lavoro del programmatore e diminuendo le possibilità di errore da parte dello stesso.

**CAPITOLO III**

**IMPLEMENTAZIONE**

**3.1 STFCS**

In questo capitolo si vuole introdurre l’implementazione di uno degli approcci ad ART precedentemente analizzati, ovvero la strategia STFCS. L’algoritmo è stato implementato in linguaggio JAVA. Di seguito è riportato la sua implementazione in pseudocodice:



É possibile osservare dallo pseudocodice che è stato realizzato un unico componente che, al tempo stesso, esegue i compiti definiti precedentemente dai componenti *random-candidate-set-construction* e *test-case-selection* ovvero la generazione dell’insieme dei test candidati e la selezione del candidato migliore in termini di distanza. Per la sua realizzazione sono stati implementati diversi metodi all’interno di una classe java denominata art. Tra questi:

1. Metodi per la generazione casuale di input di test di tipo stringa, intero, double, vettore di interi e di double, booleano.
2. Un metodo per verificare se un input di test è presente, o meno, all’interno di una lista; entrambi i parametri sono passati come input. Restituisce un booleano.
3. Un metodo che calcola la distanza tra due input di test forniti in ingresso alla funzione. Restituisce un double.
4. Un metodo che genera un insieme di input di test candidati, di dimensione fissata e passata come parametro di ingresso, e seleziona quello con distanza minima rispetto agli elementi contenuti nella lista dei casi di test già selezionati. Restituisce una lista di oggetti di tipo Type.
5. Un metodo che calcola l’efficienza dell’algoritmo, valutata come il numero di casi di test eseguiti prima di rilevare un fallimento.
6. Un metodo che calcola l’efficacia dell’algoritmo, intesa come numero totale di fallimenti rilevati.
7. Il metodo do\_test, che esegue il test sfruttando la funzione descritta al punto 4.

Quest’ultima rappresenta il cuore dello strumento, poiché è quella che implementa l’algoritmo di FSCS. Per realizzare il metodo abbiamo utilizzato la libreria di Reflection di Java, che permette a una funzione di esaminare e/o modificare il suo comportamento a tempo d’esecuzione, in particolare la struttura del suo codice. In linea generale, il comportamento del metodo può essere riassunto come segue: ricevuta una classe in ingresso, do\_test accede ai metodi della stessa mediante la funzione getDeclaredMethods (appartenente alla libreria di Reflection) e li salva in una lista. Ogni metodo dovrà poi essere testato. Quindi do\_test genera casualmente i parametri da fornire in ingresso al metodo da testare, utilizzando la funzione select\_best\_data che sceglie, in base ai principi dell’FSCS e ai tipi dei parametri di ingresso del metodo da testare, la sequenza di ingresso migliore. Infine, applicando il metodo invoke, il metodo sotto test inizia la propria esecuzione con i parametri di ingresso generati nella fase precedente.

Similmente sono state implementate le funzioni efficiencyMeasure e effectivenessMeasure:

* La prima, per calcolare la F-measure, incrementa una variabile contatore ogni volta che il metodo testato finisce la sua esecuzione correttamente, e la decrementa (restituendone il valore in uscita e poi terminando) quando viene rilevato un fallimento. É importante osservare che, nel nostro studio, i fallimenti indicano le eccezioni non gestite nel codice.
* La seconda similmente, incrementa una variabile contatore quando viene rilevato un fallimento, restituendola come parametro d’uscita alla fine della sua esecuzione.

Un’altra funzione che contribuisce all’implementazione complessiva è la funzione select\_best\_data. In base al valore di paramTypes che riceve in ingresso, in cui vengono specificati i tipi dei parametri che la funzione da testare deve ricevere, genera casualmente i dati di test del candidato. Se il candidato è già presente nell’insieme dei test già eseguiti o nell’insieme dei casi di test candidati, i dati di test vengono rigenerati. Altrimenti, il candidato viene aggiunto alla lista dei test candidati. Quindi, il metodo individua la distanza minima tra ogni test candidato e l’insieme dei casi di test già eseguiti e sceglie il caso di test caratterizzato dalla distanza massima tra queste.

Per calcolare la distanza è stata implementata la funzione calculateDistance. Il calcolo è banale nel caso di input di tipo intero e double: in questo caso, è stata utilizzata la distanza euclidea. Il calcolo si complica con input di tipo vettore (sia di interi che di double). Infatti, abbiamo prima dovuto discriminare due differenti casistiche: il caso di vettori caratterizzati dalla stessa dimensione, e il caso di vettori con diversa dimensione. Nel primo caso, abbiamo potuto utilizzare la normale distanza euclidea, mentre nel secondo abbiamo introdotto un meccanismo che, nel calcolo della distanza, tenga conto sia della differenza del numero di elementi tra i due vettori che del loro valore. Per quanto riguarda il tipo stringa, anche in questo caso è stato ideato un meccanismo che valuta la distanza in base a varie caratteristiche delle stringhe, tra cui numero di caratteri della stringa e tipo di caratteri della stringa (maiuscolo o minuscolo, simbolo, cifra). Infine, è stato gestito banalmente il tipo booleano.

**3.2 Esempio di utilizzo**

Per facilitare l’uso del nostro strumento forniamo un esempio di utilizzo riguardante il test di un singolo programma. Ipotizziamo di voler testare un software di nome diffi. Per eseguire correttamente lo strumento è necessario che i file do\_test.jar e do\_test.bat si trovino all’interno della stessa directory e che diffi.jar (il file eseguibile del software da testare) e diffi.txt (il file che contiene le classi che si vogliono testare) si trovino all’interno di una cartella denominata jars e situata all’interno della stessa directory del file do\_test.jar. Nel nostro caso, i file diffi.jar e diffi.txt si trovano nel path “C:\Users\fabio\ART\jars”. Si vuole testare il programma diffi per un tempo pari a 30 minuti. Inoltre, i file do\_test.jar (executable jar file dello strumento) e do\_test.bat (il file che permette il corretto utilizzo dello strumento) si trovano all’interno dello stesso path “C:\Users\fabio\ART”. Dunque, da linea di comando:

1. Spostarsi nella directory relativa al percorso in cui è presente il file do\_test.jar ed il file do\_test.bat:

cd C:\Users\fabio\ART

1. Runnare il seguente comando:

start do\_test.bat diffi 1800

I risultati dell’esecuzione potranno essere osservati in una cartella denominata come il programma testato (seguita da una serie di parametri che garantiscono l’unicità della stessa) presente nella stessa directory che contiene il file do\_test.jar.

Il punto di partenza del processo è il file batch do\_test.bat che è stato sviluppato su Windows 10. Per conoscere la sintassi e avviare il processo di test per tutte le applicazioni disponibili, è possibile utilizzare il file batchtest\_do\_test.bat che apre in sequenza una serie di sessioni di test sulle 4 applicazioni considerate. Utilizzando il file batchtest\_randoop.bat è possibile effettuare la stessa attività di testing utilizzando, però, lo strumento Randoop. Dunque, da linea di comando:

spostarsi nella directory dove è presente il file batchtest\_do\_test.bat ed eseguirlo:

batchtest\_do\_test

Un’alternativa consiste nello spostarsi fisicamente nella suddetta directory e cliccare sul file batchtest\_do\_test per far avviare automaticamente l’esecuzione dello script.

**3.3 Analisi dei dati di copertura**

Per verificare la corretta esecuzione e le prestazioni dello strumento realizzato, si è pensato di testare quattro programmi Java e di ricavarne i relativi dati di copertura. Inoltre, lo stesso tipo di test è stato effettuato tramite l’utilizzo di Randoop e sono stati confrontati i dati di copertura ottenuti rispetto a quelli precedentemente calcolati con il nostro strumento.

Per analizzare in maniera approfondita l’efficacia del nostro strumento è necessario valutare i risultati ottenuti tramite i report generati dallo strumento EMMA.

EMMA è un toolkit open source che permette di misurare la copertura del codice Java . Si distingue dagli altri strumenti in quanto fornisce una serie di funzionalità molto importanti: supporto per lo sviluppo di software aziendale su larga scala garantendo velocità nello sviluppo individuale. Utilizzando questo tool, ogni sviluppatore del team può ottenere la copertura del codice in modo gratuito e veloce.

È stata effettuata un’attività di testing, tramite lo strumento realizzato, adottando come criterio di terminazione un criterio temporale. Nello specifico, sono stati considerati quattro programmi differenti, ognuno dei quali è stato testato per trenta minuti ininterrottamente. Tramite Emma abbiamo ottenuto i risultati di copertura categorizzati in questo modo:

1. Un insieme di report, uno per ogni test effettuato durante l’esecuzione, che forniscono il livello di copertura cumulativo ottenuto sommando le coperture dei test effettuati fino a quello considerato. Con un apposito programma, html\_reader, abbiamo raggruppato i risultati in un file csv e tramite l’ausilio di Matlab abbiamo generato un grafico per ognuno dei software in modo da analizzare l’andamento della copertura ottenuta.
2. Un report finale attraverso il quale possiamo osservare il livello di copertura raggiunto sommando tutti i risultati di copertura ottenuti da ciascun test.

Una prima analisi si concentra sul confronto dei risultati di copertura ottenuti attraverso il nostro strumento e quelli relativi al tool Randoop. Questo è uno strumento open source che permette di eseguire random testing in modo intelligente, a partire da una analisi del codice, limitata particolarmente ai metodi e ai loro parametri. I test sono costituiti da sequenze casuali di chiamate di metodi su oggetti e sono generati come script Junit, quindi possono essere rieseguiti. Il risultato originale del test è codificato come asserzione. Permette, inoltre, di eseguire test di regressione. Può essere utilizzato per generare test per un metodo (o per tutti i metodi di una classe o di un package).

Come è possibile osservare dai report generati da EMMA, i risultati cumulativi ottenuti dipendono dal particolare programma analizzato. Di seguito, un confronto tra i risultati ottenuti con il nostro strumento e quelli di Randoop, per ogni software testato.

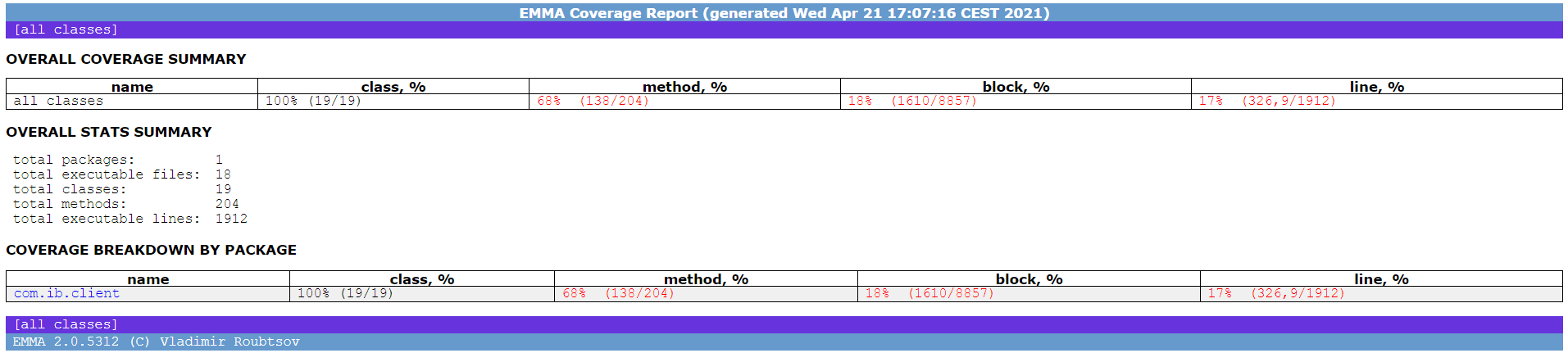


Figura 5 - Nostro\_strumento\_tullibee\_test

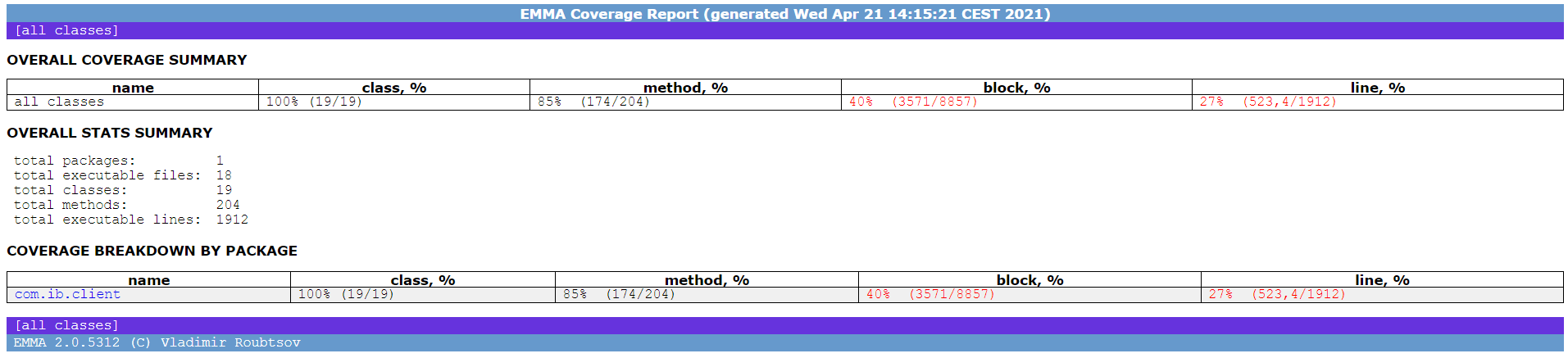


Figura 6 - Randoop\_tullibee\_test

Questi report riguardano l’unico programma il cui livello di copertura ottenuto tramite il nostro strumento è minore (almeno del 10%) rispetto a quello di Randoop.

Attraverso un’analisi più approfondita si è scoperto che il motivo di tale differenza risiede soprattutto nel fatto che il nostro strumento non è in grado di gestire oggetti relativi a classi definite dal software testato. In particolare, quindi, non solo non vengono eseguiti tutti i metodi che presentano tra i parametri di input alcuni tipi non gestiti ma anche tutti i metodi che sarebbe in grado di eseguire che, però, appartengono ad una classe di cui non si sanno gestire i parametri di ingresso al/ai costruttore/i. L’impossibilità di chiamare correttamente un costruttore della classe, non permette in alcun modo di eseguire il metodo tramite le proprietà di Reflection. Nonostante ciò, i risultati di Randoop non sono così lontani dai nostri perché nella maggior parte dei software analizzati la quantità di codice relativa ai metodi di prima è limitata.

Di seguito riportiamo alcune differenze relative ai report precedentemente proposti:

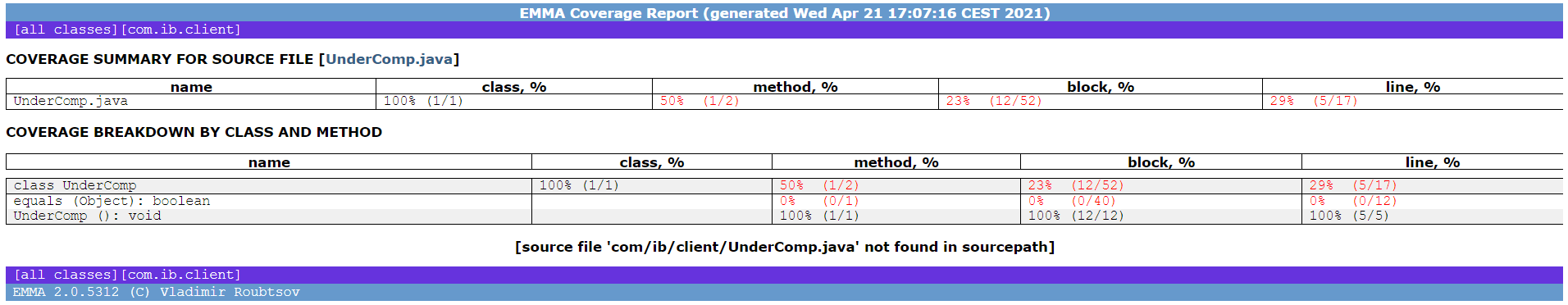


Figura 9 – nostro\_strumento\_tullibee\_class

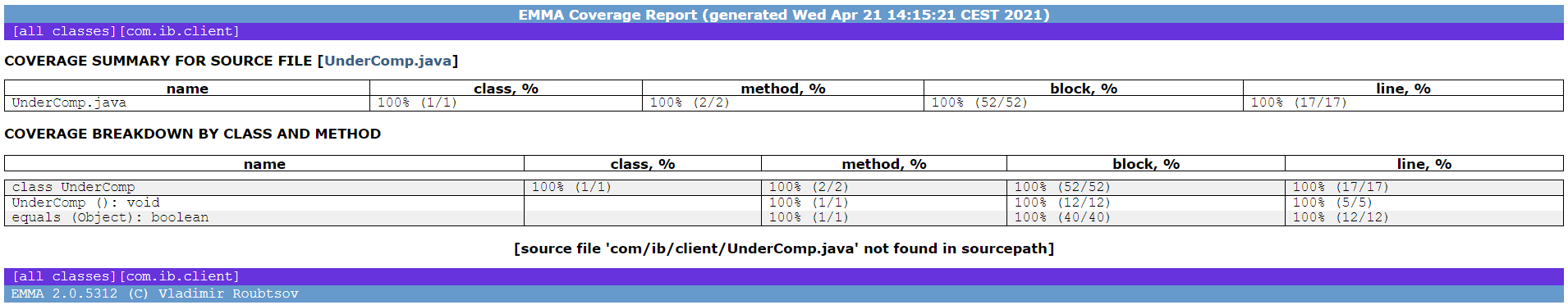


Figura 10 - Randoop\_tullibee\_class

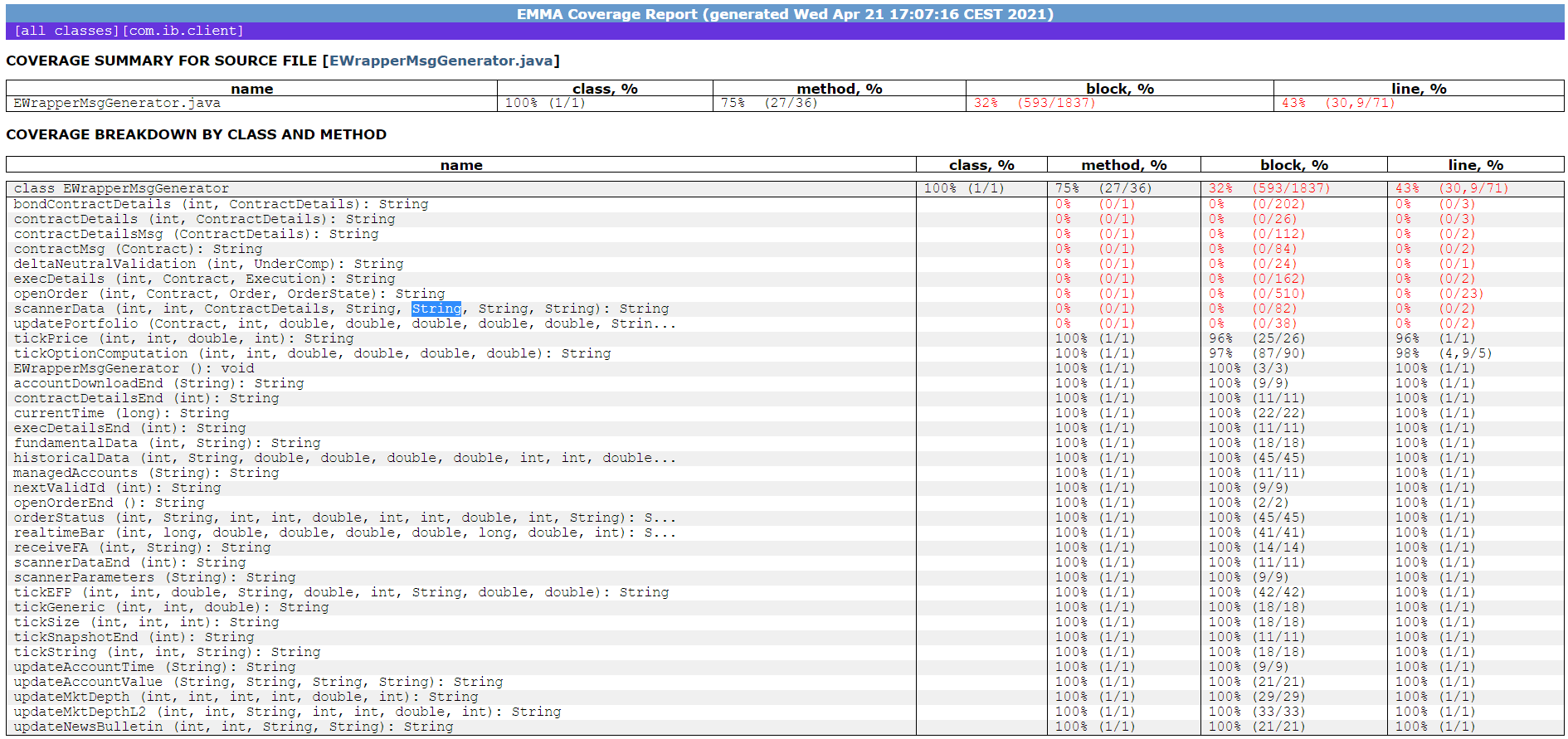


Figura 9 – nostro\_strumento\_tullibee\_class

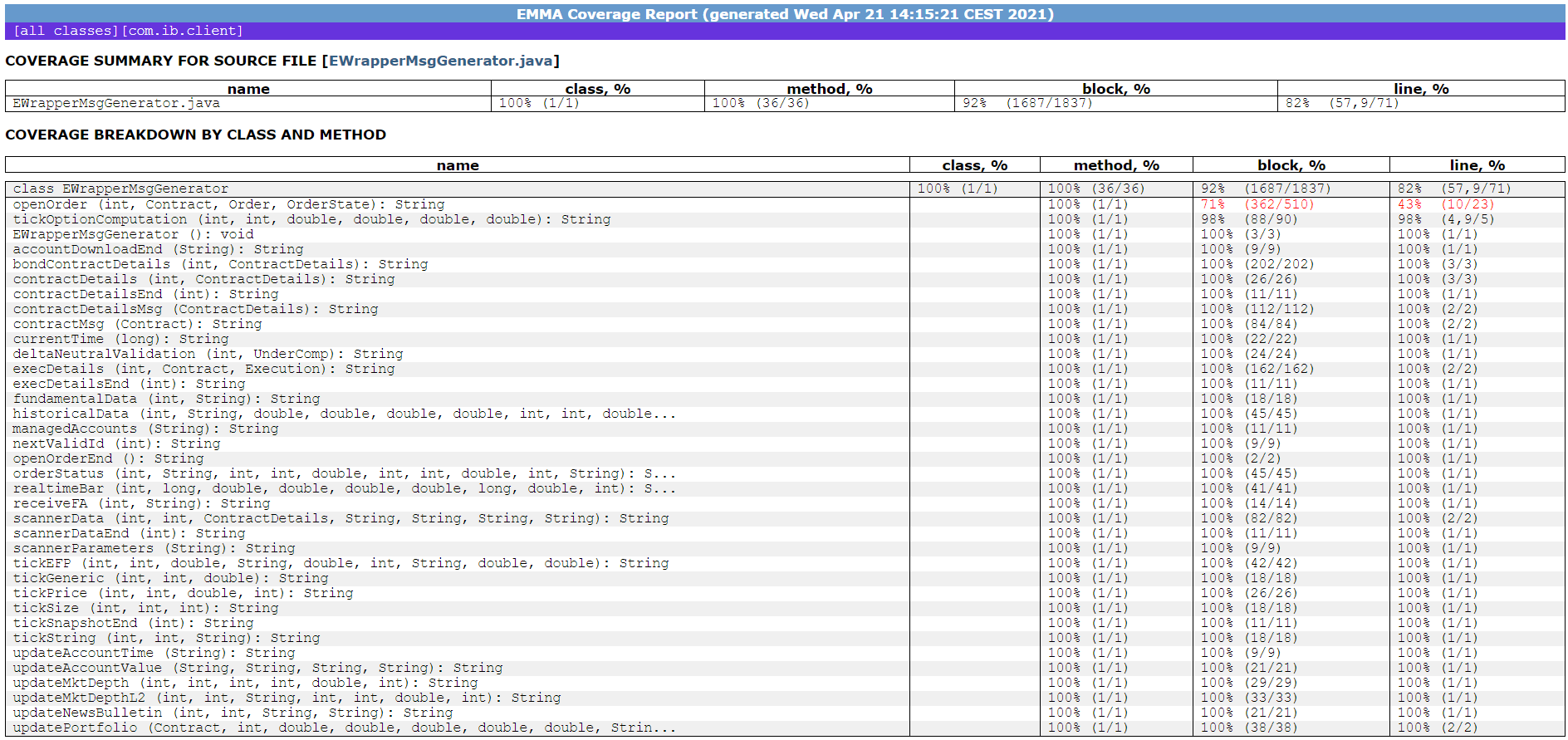


Figura 10 - Randoop\_tullibee\_class

Queste prime due immagini permettono di osservare la prima problematica presentata ovvero quella relativi ai metodi non eseguiti per via della presenza di tipi di parametri di input non gestiti. In particolare, osserviamo che i metodi equals della classe UnderComp e i primi nove metodi relativi alla classe EWrapperMsgGenerator non vengono eseguiti perché ricevono come parametro di ingresso un oggetto di una classe definito dal programma tullibee, mentre gli altri metodi vengono correttamente eseguiti (tra questi notiamo la presenza dei costruttori). Questa difficoltà è invece superata da Randoop che esegue il 100% dei metodi della classe.

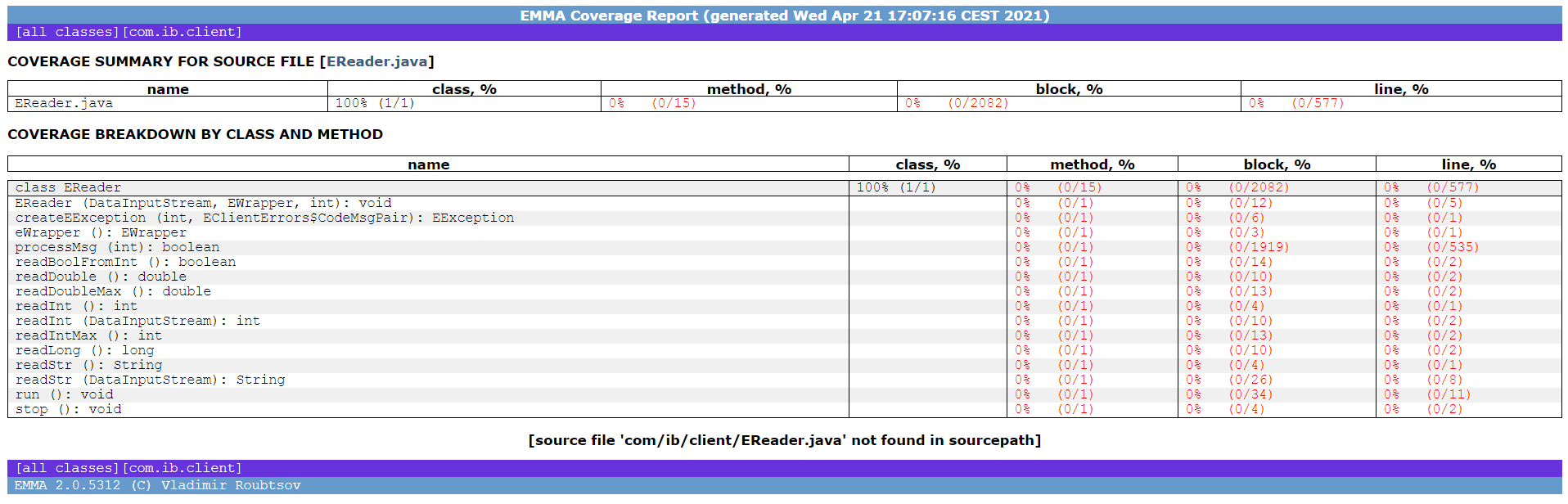


Figura 9 – nostro\_strumento\_tullibee\_class

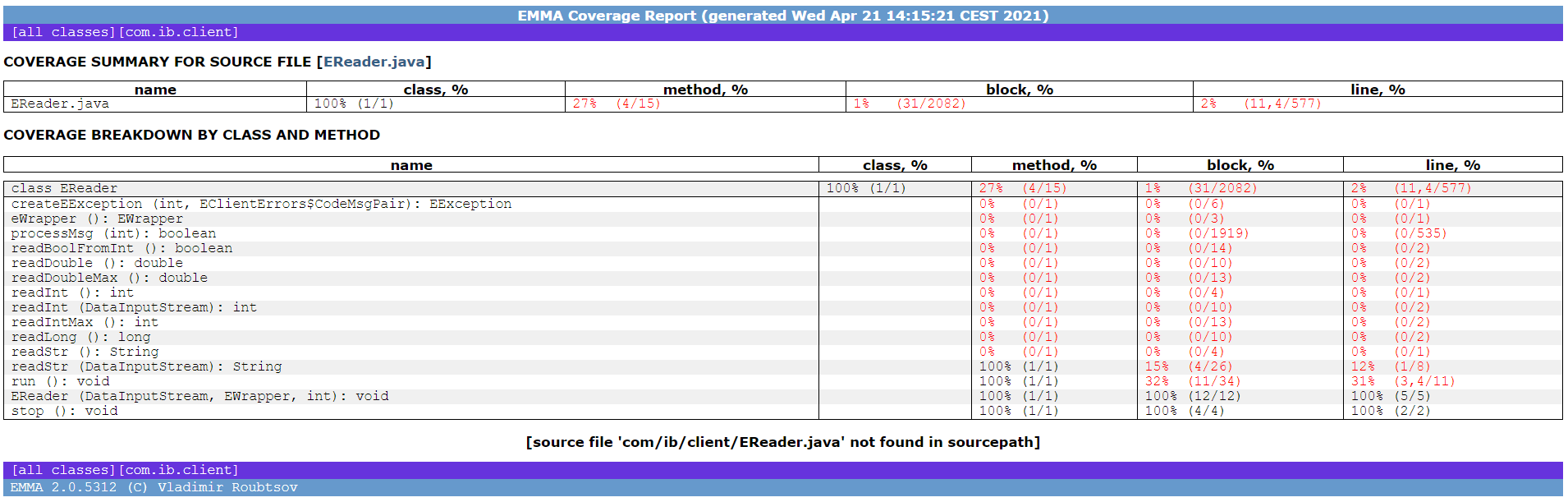


Figura 10 - Randoop\_tullibee\_class

Queste immagini mettono in risalto la seconda problematica presentata. Nel caso della classe EReader di tullibee non è stato infatti possibile eseguire alcun metodo in quanto l’unico costruttore della classe riceveva come parametro di input un oggetto della classe DataInputStream ed uno della classe EWrapper che il nostro strumento non è in grado di gestire. In questo caso Randoop riesce a coprire una quantità di codice maggiore che poi va ad influire sul risultato di copertura finale.

I successivi report sono relativi a livelli di copertura simili tra quelli ottenuti con il nostro strumento e quelli di Randoop. É possibile osservare che in questo caso la problematica riscontrata nei primi report è presente ma in quantità minore. Inoltre, relativamente a questi specifici programmi anche Randoop ha avuto difficoltà nell’esecuzione di determinati metodi.

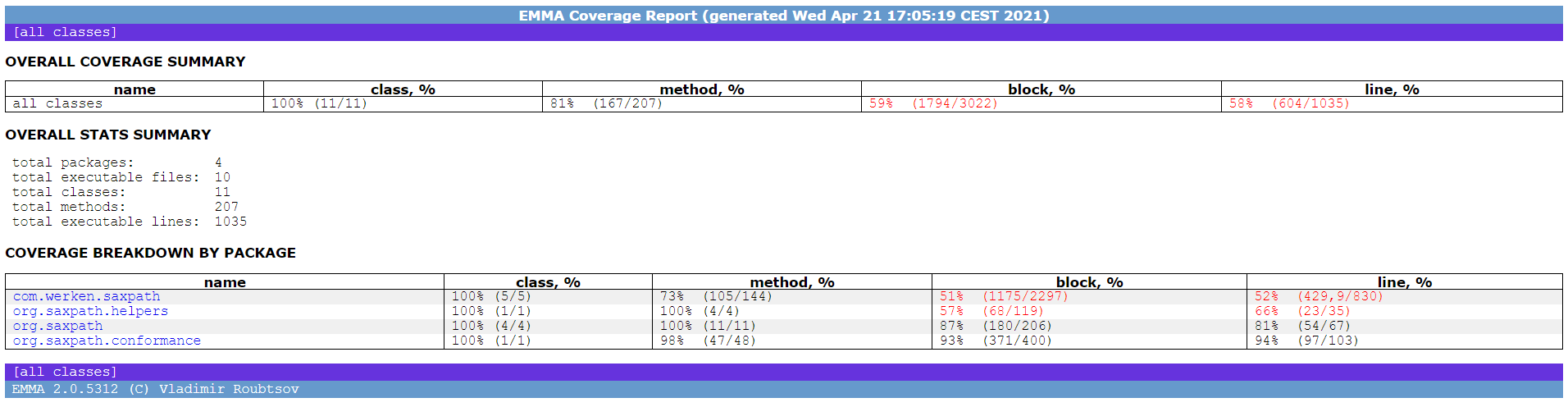


Figura 17 - nostro\_strumento\_saxpath\_test

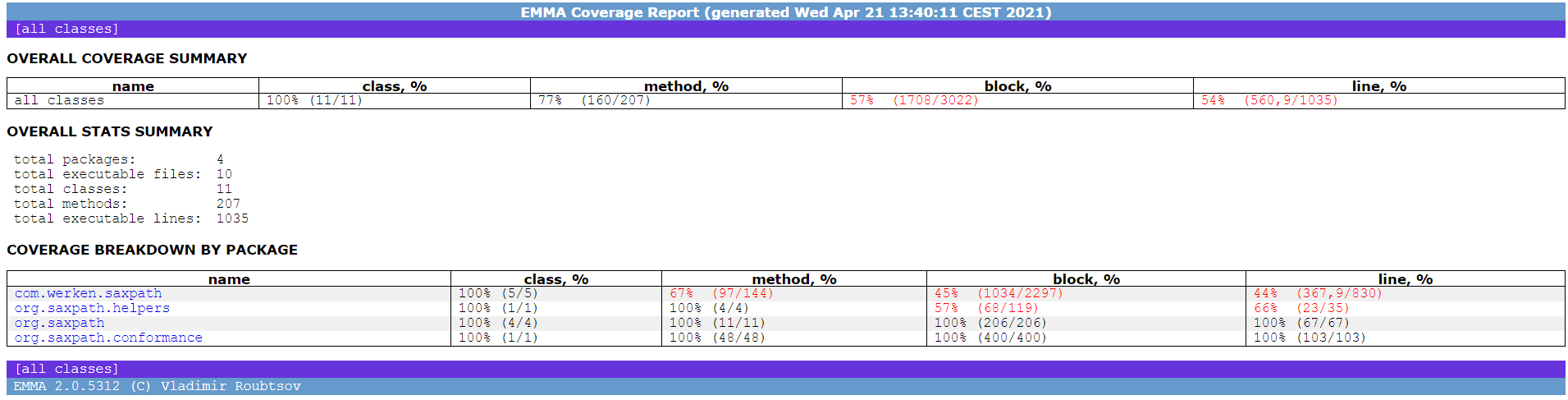


Figura 18 - Randoop\_saxpath\_test

Non è possibile comprendere le problematiche riscontrate da Randoop tramite l’analisi dei report ma con tutta probabilità queste risiedono in problemi di accessibilità. Situazioni identiche vengono evidenziate anche dai report del nostro strumento. In prima analisi questi problemi vengono riscontrati proprio relativamente a quei metodi e quelle classi che il nostro strumento ha difficoltà a gestire, per questo i risultati di copertura sono così simili. Inoltre, un’altra possibile motivazione di tale somiglianza risiede nel fatto che la maggior parte delle classi di questi programmi presentano metodi i cui tipi dei parametri di ingresso sono gestibili dal nostro tool, per cui, generando casualmente i diversi casi di test, gli strumenti riescono ad eseguire la stessa quantità di codice. Di seguito riportiamo alcuni esempi:

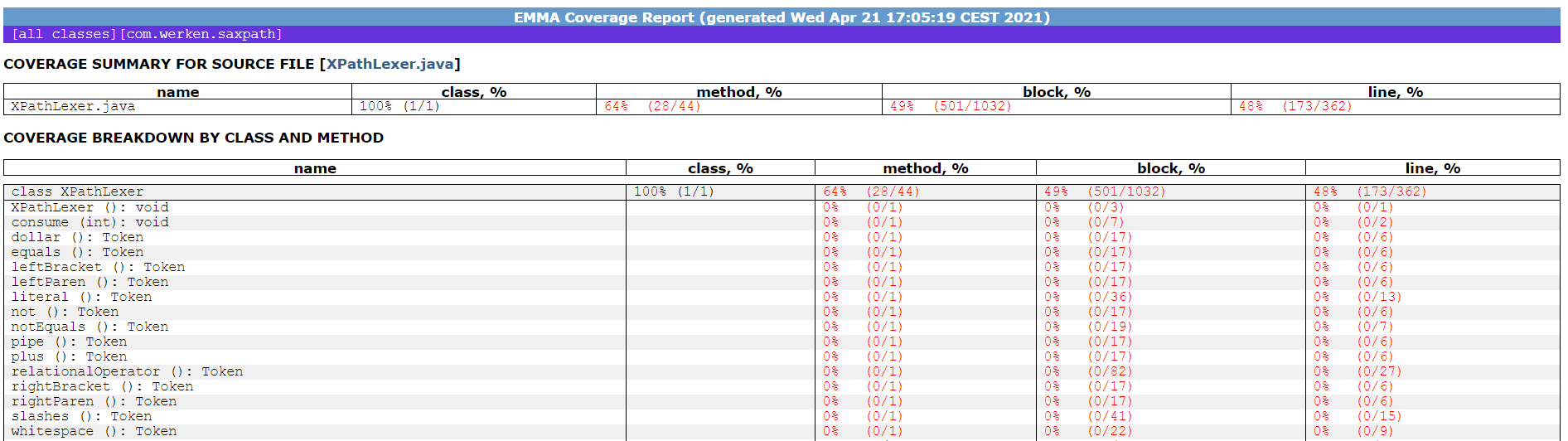


Figura 19 - nostro\_strumento\_saxpath\_class

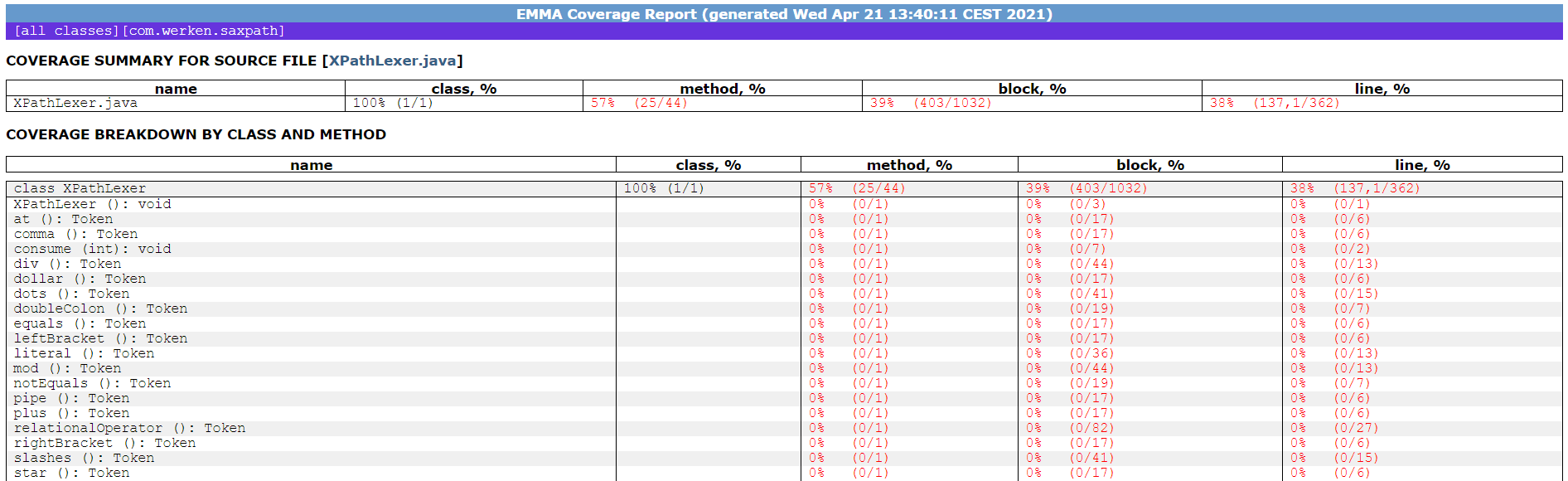


Figura 20 - Randoop\_saxpath\_class

In questo caso mostriamo la prima situazione relativamente alla classe XPathLexer, ovvero quella in cui per problemi (probabilmente) di accessibilità nessuno dei due strumenti riesce ad eseguire parte dei metodi di alcune classi.

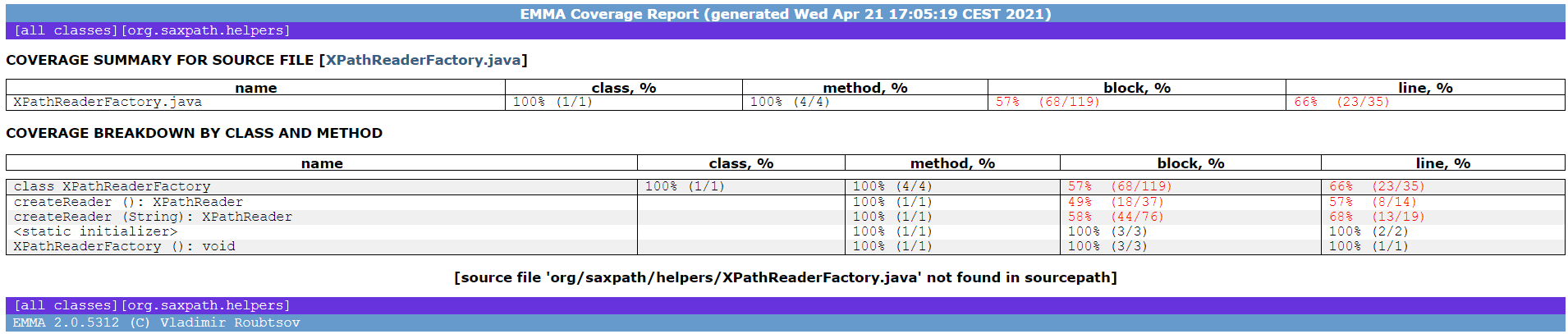


Figura 23 - nostro\_strumento\_saxpath\_class

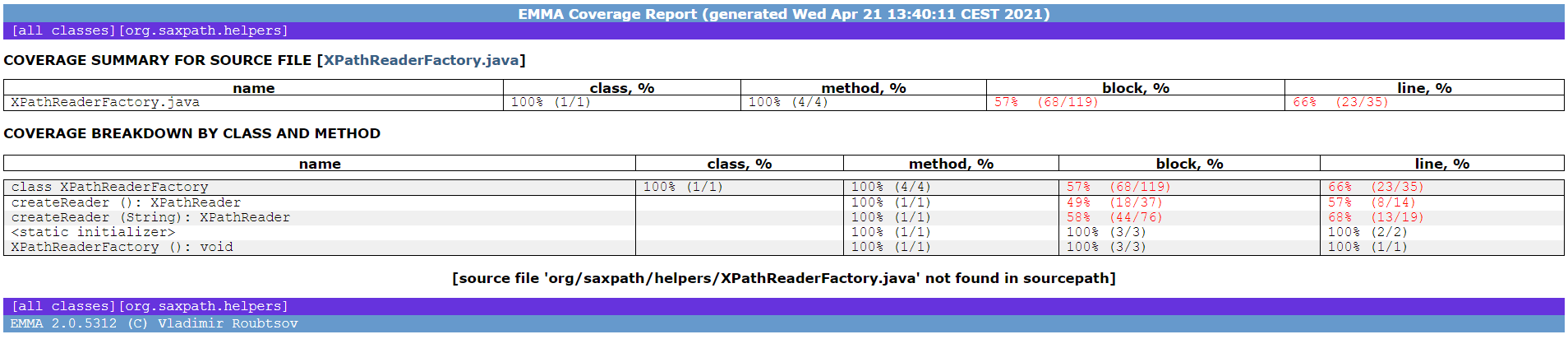


Figura 24 - Randoop\_saxpath\_class

In questo caso, invece, presentiamo la seconda situazione ovvero quella relativa alla classe XpathReaderFactory di saxpath e quindi a metodi che il nostro strumento riesce ad eseguire e per cui i risultati di copertura ottenuti sono molto vicini a quelli di Randoop.

Infine, presentiamo i casi in cui il nostro strumento esegue un numero linee di codice significativamente maggiore rispetto a quello eseguito da Randoop:

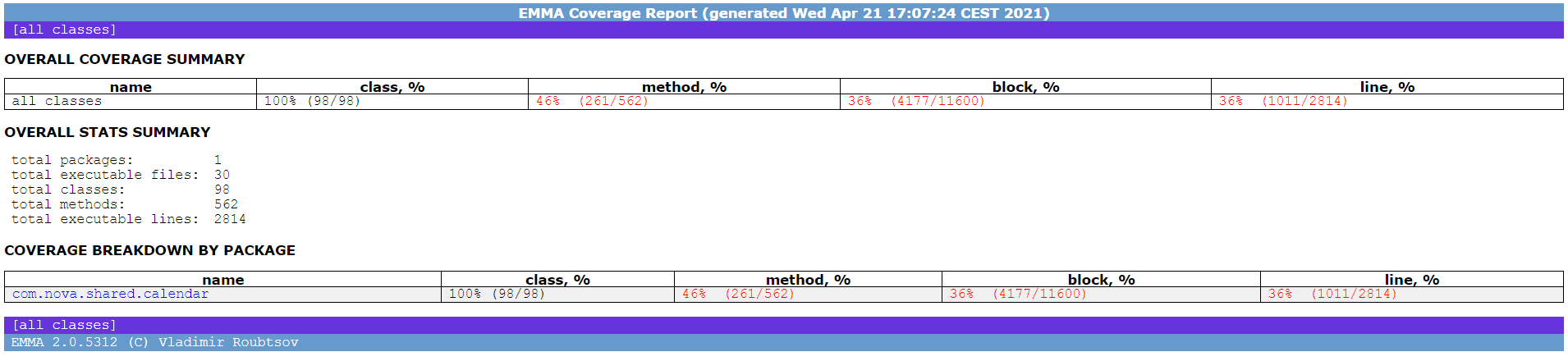


Figura 25 - nostro\_strumento\_calendar\_test

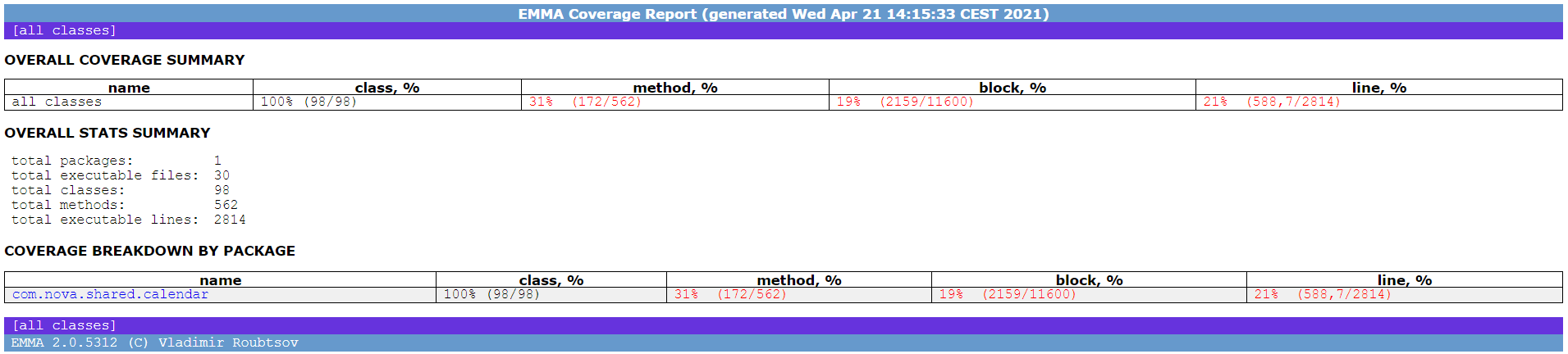


Figura 26 - Randoop\_calendar\_test

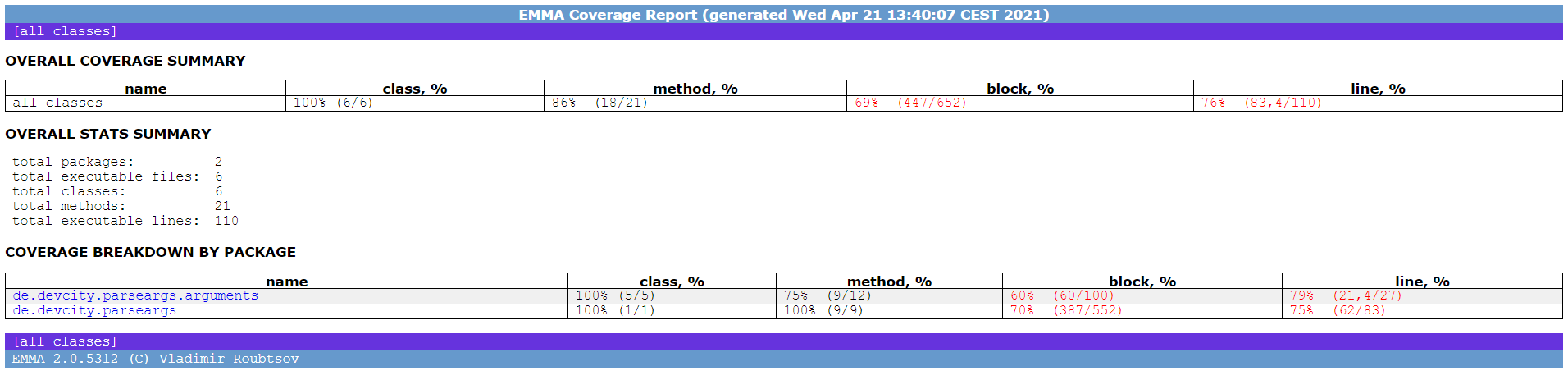


Figura 27 - nostro\_strumento\_dcparseargs \_test

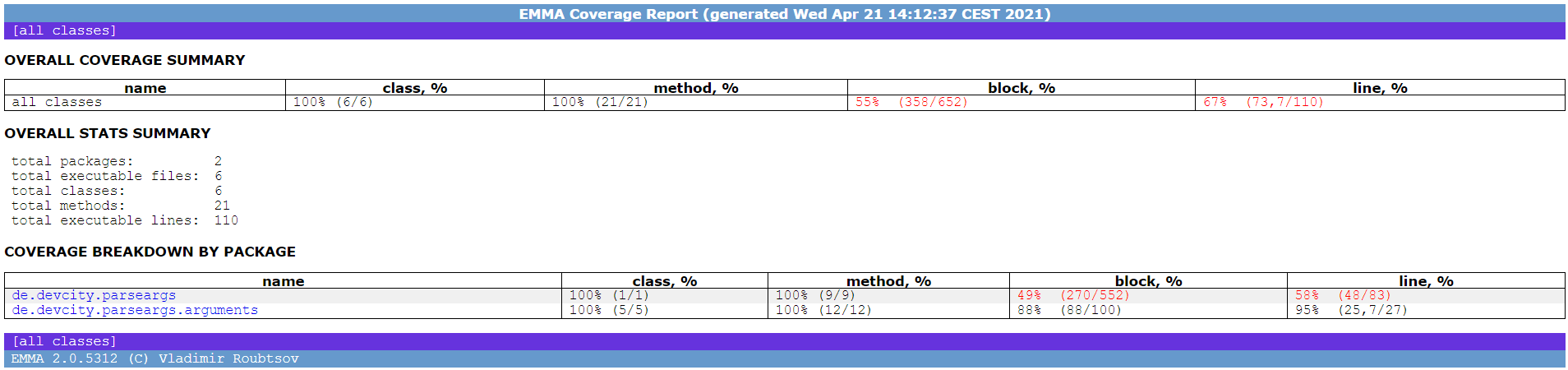


Figura 28 - Randoop\_dcparseargs\_test

Le motivazioni principali sono state trovate nel fatto che, il criterio di diversificazione dei casi di test generati basati sulla distanza è stato più efficace rispetto al criterio intelligente utilizzato da Randoop e da qui una copertura più estesa del codice delle diverse classi.

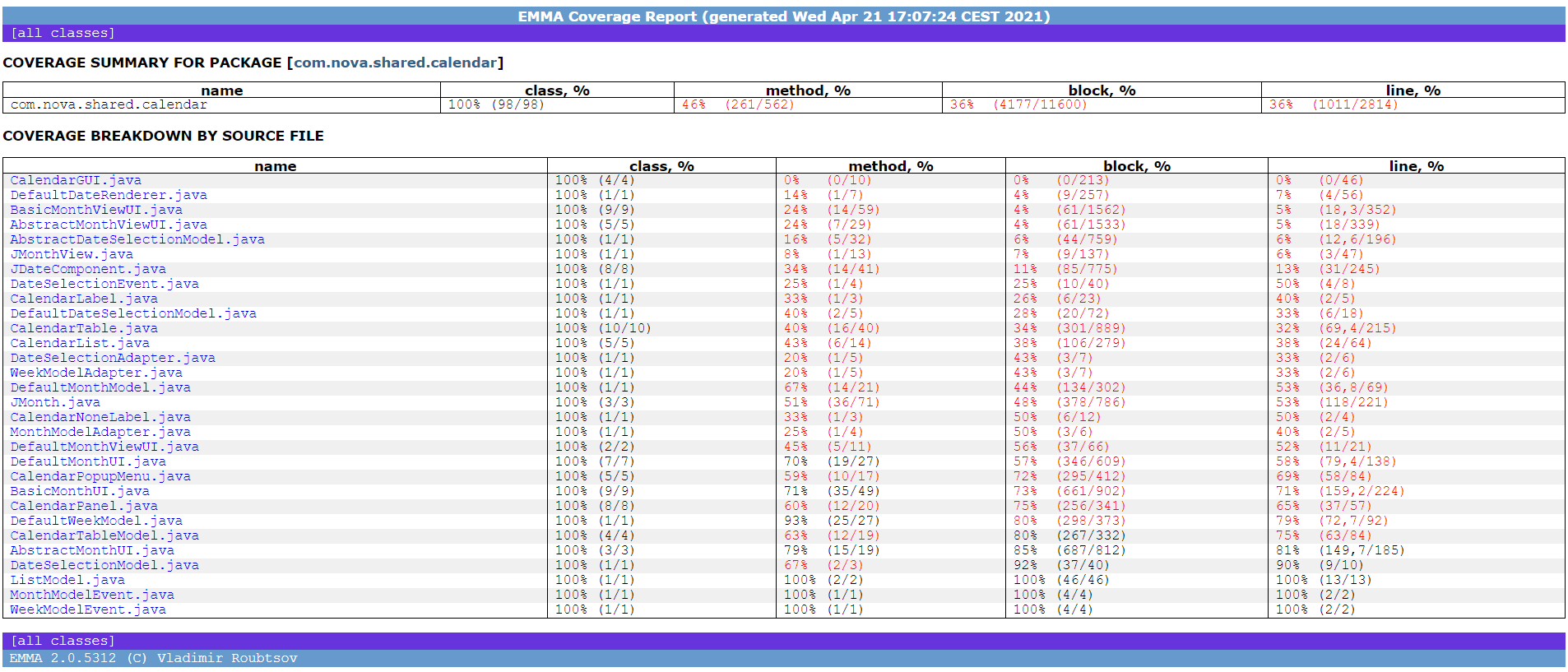


Figura 27 - nostro\_strumento\_calendar\_class

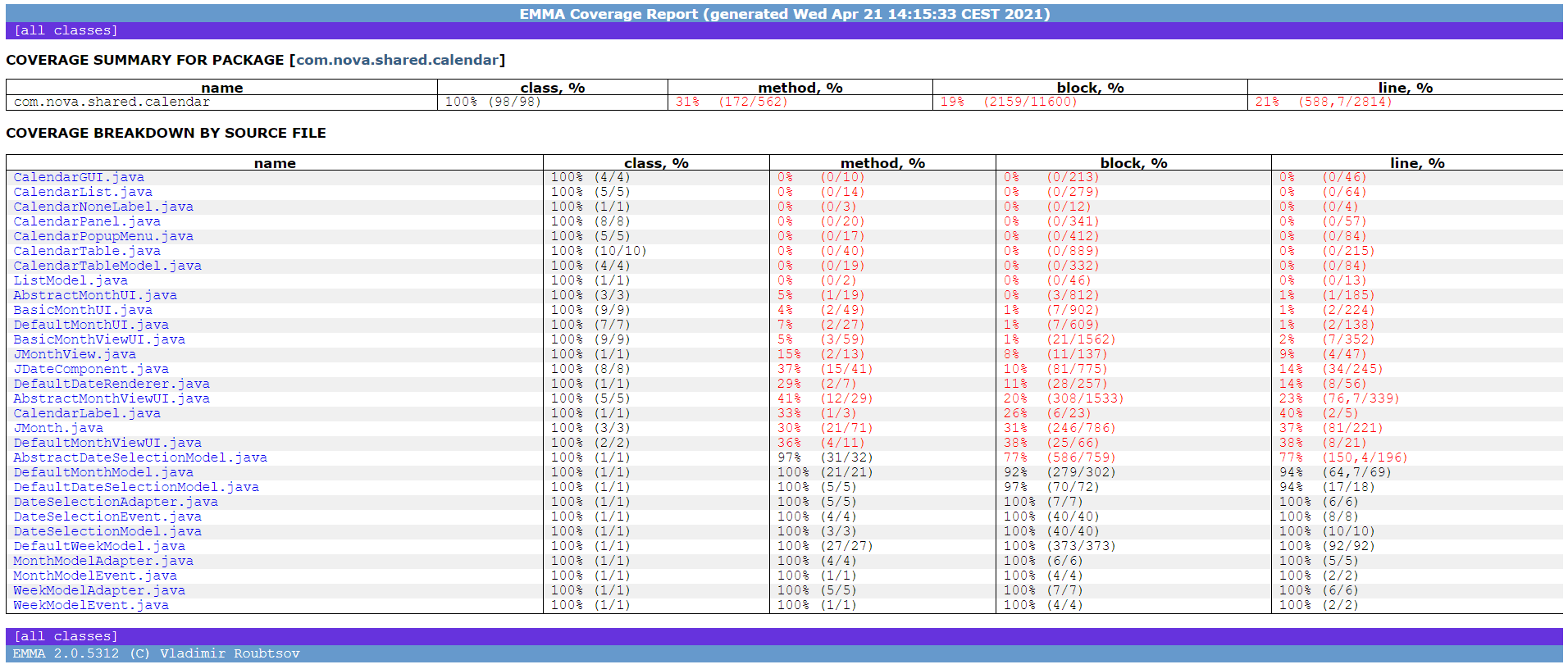


Figura 28 - Randoop\_calendar\_class

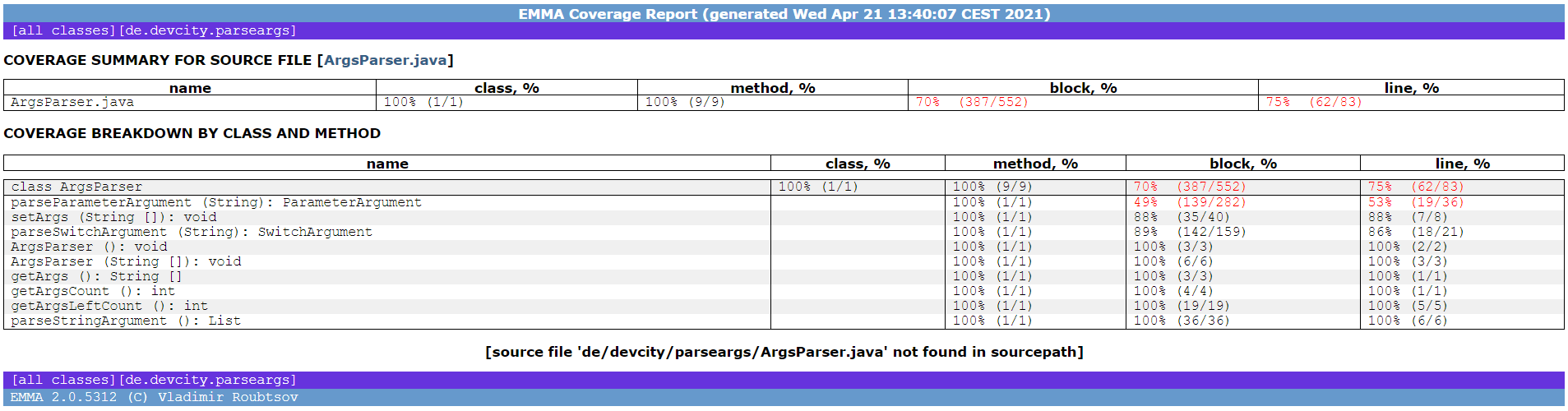


Figura 27 - nostro\_strumento\_dcparseargs\_class

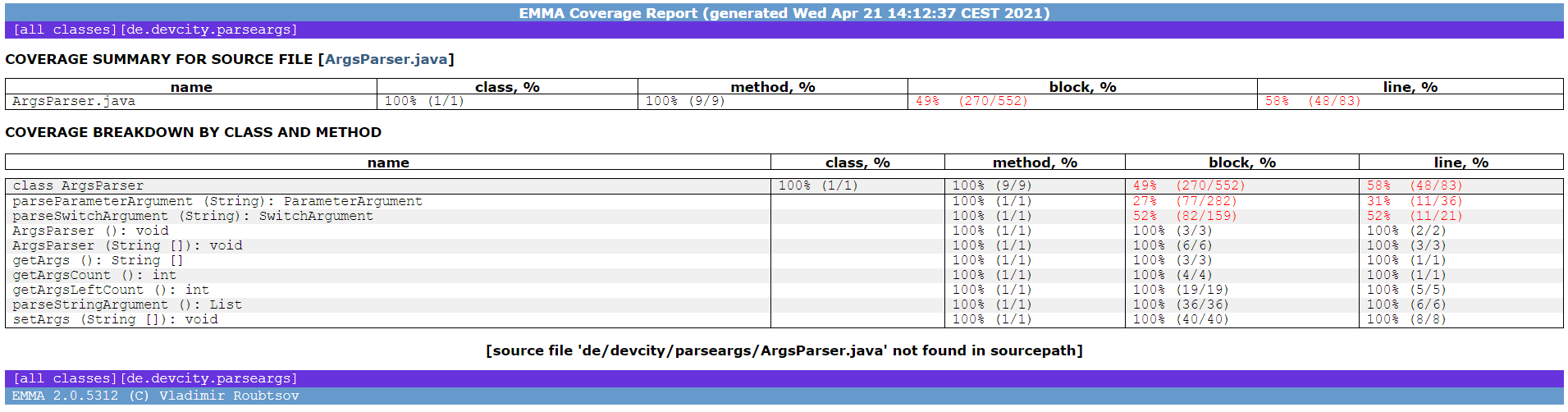


Figura 28 - Randoop\_dcparseargs\_class

Analizziamo l’andamento grafico relativo ad ogni attività di testing effettuata. Anche in questo caso possiamo categorizzare i test in diversi insiemi: un insieme il cui andamento grafico della misura di copertura varia con bassa frequenza (o non varia addirittura) ed un altro per cui la copertura ottenuta varia con più frequenza durante l’esecuzione dei test:

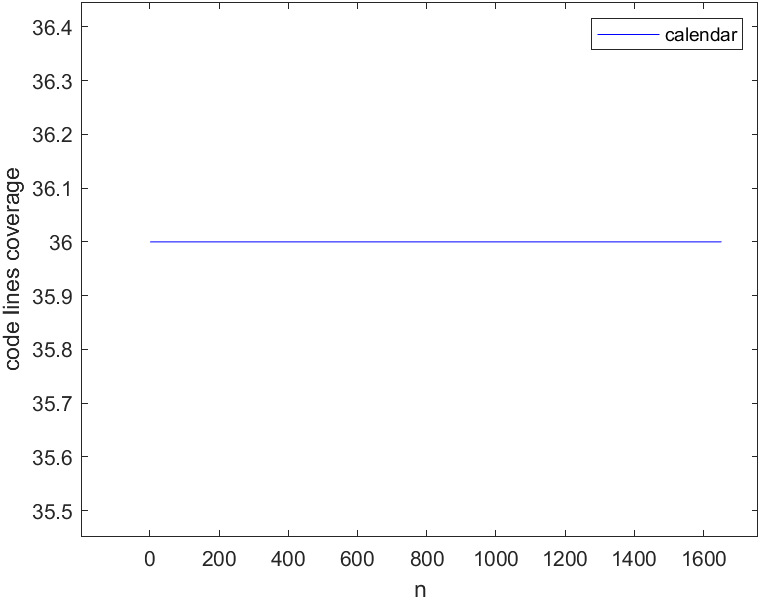
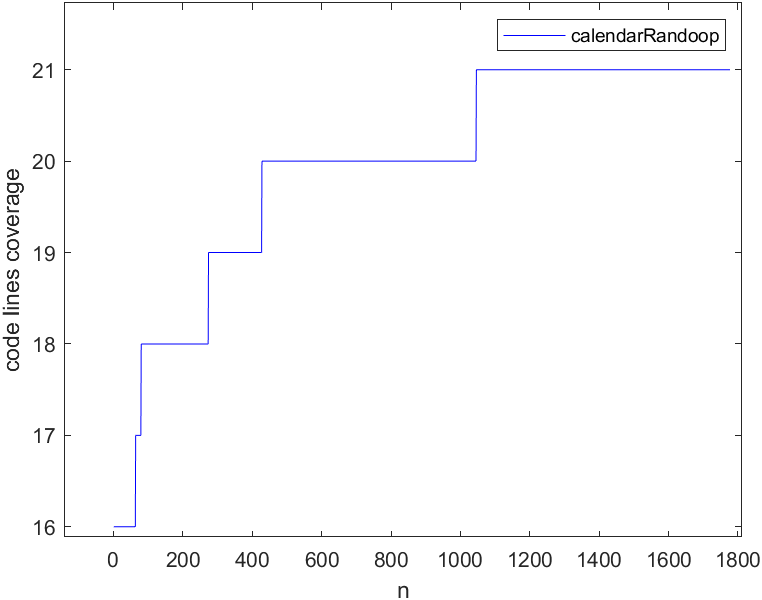
 

Figura 31 - nostro\_strumento Figura 32 - Randoop

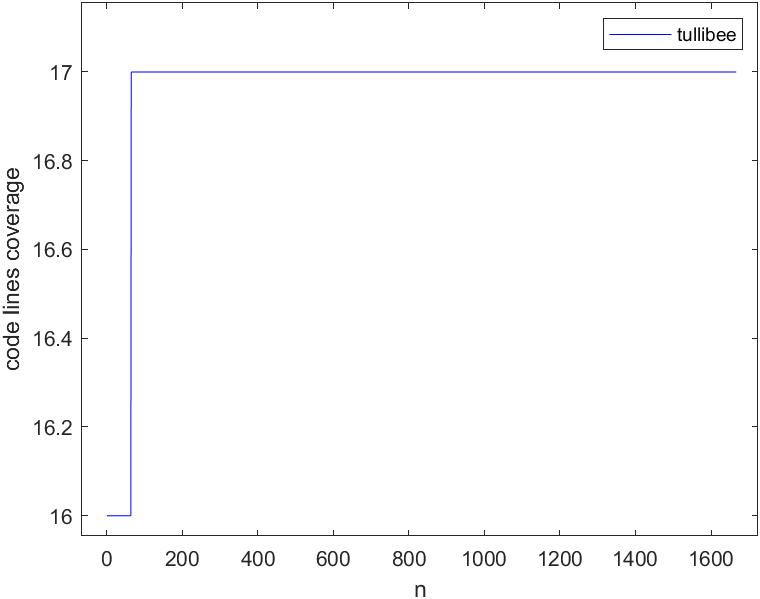
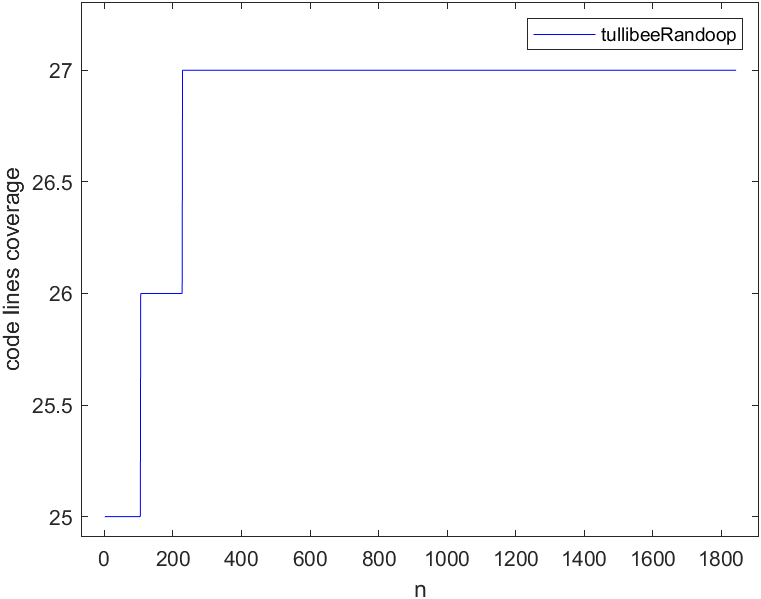
 

Figura 37 - nostro\_strumento Figura 38 - Randoop

I primi grafici presentati sono relativi ad un andamento del livello di copertura, ottenuto dal nostro strumento, pressochè costante. Questo può dipendere da diversi fattori. Una possibile motivazione potrebbe risiedere nel criterio di distanza adottato. Infatti, relativamente a questi programmi è possibile che la differenza dei casi di test generati in funzione del valore di distanza calcolato sia poco significativa ai fini della scoperta di nuove linee di codice. In alcuni casi, la maggior parte di linee di codice sono presenti nel corpo dei metodi che il nostro strumento non è in grado di eseguire, per cui il livello di copertura non riesce a crescere all’aumentare del numero di casi di test eseguiti. Infine, questo andamento potrebbe essere dovuto dalla complessità dei metodi per cui input di test generati casualmente risultano inefficaci.

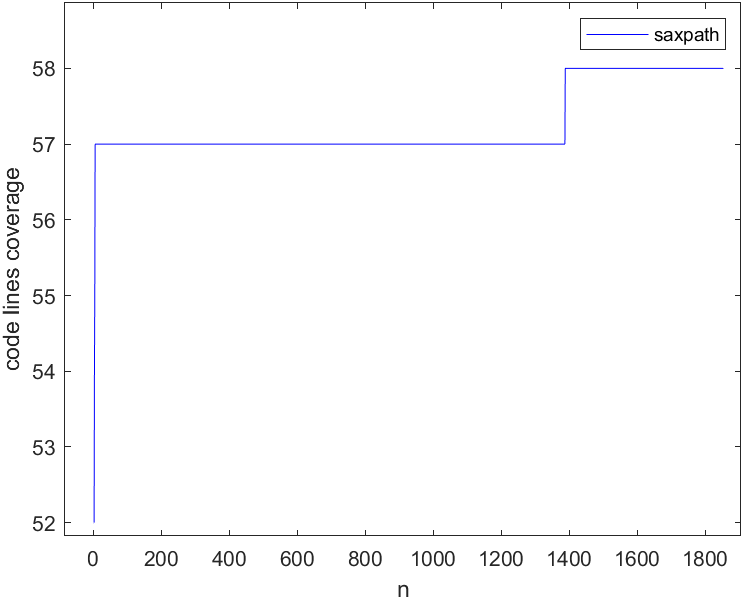
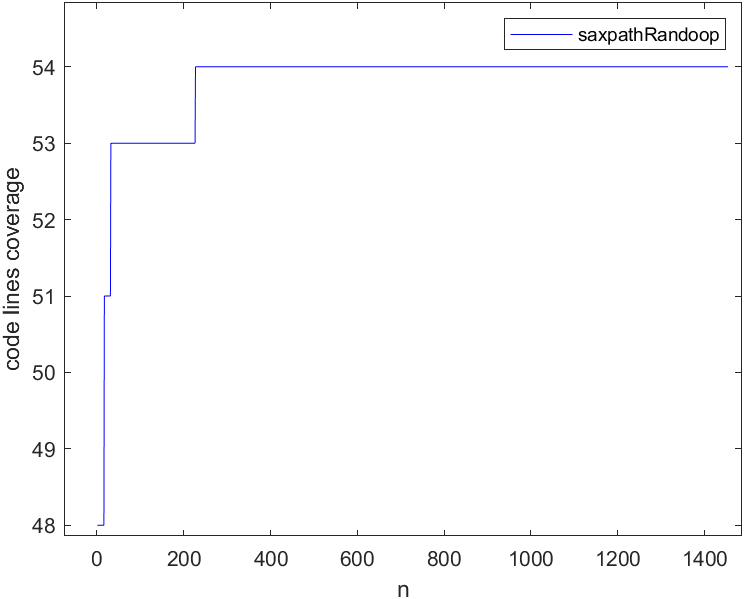
 

Figura 39 - nostro\_strumento Figura 40 - Randoop

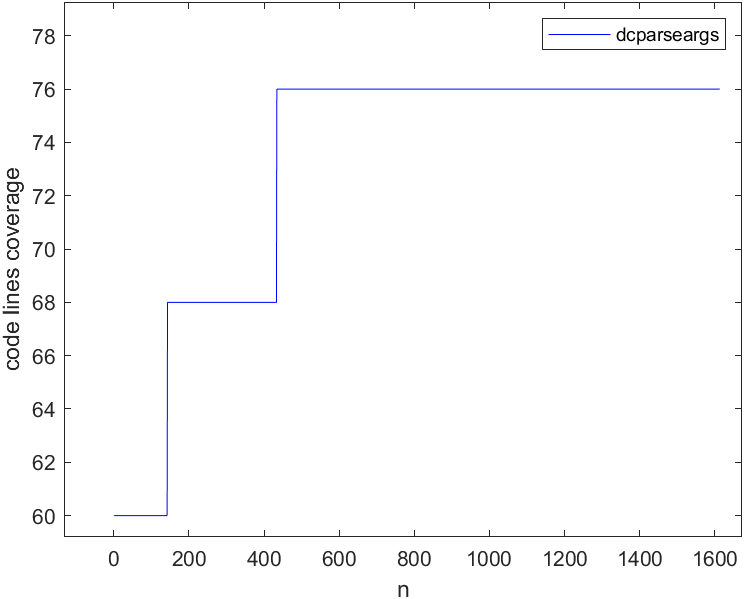
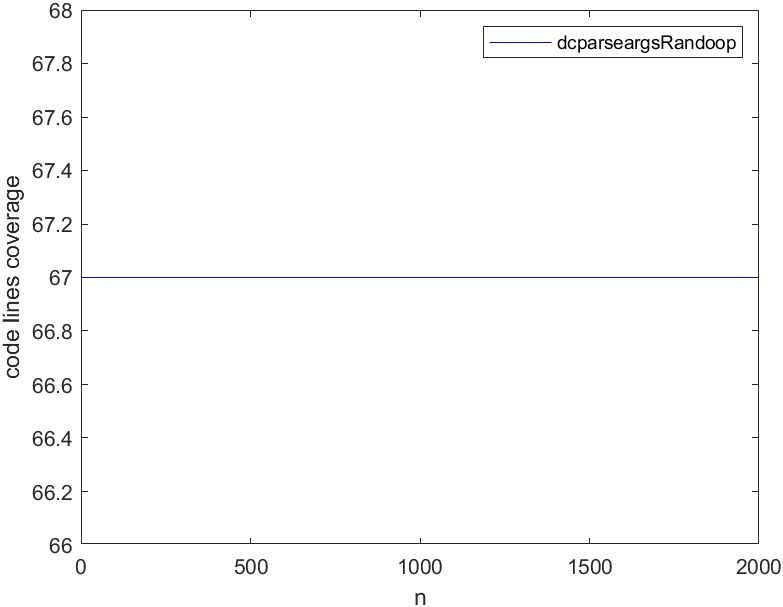
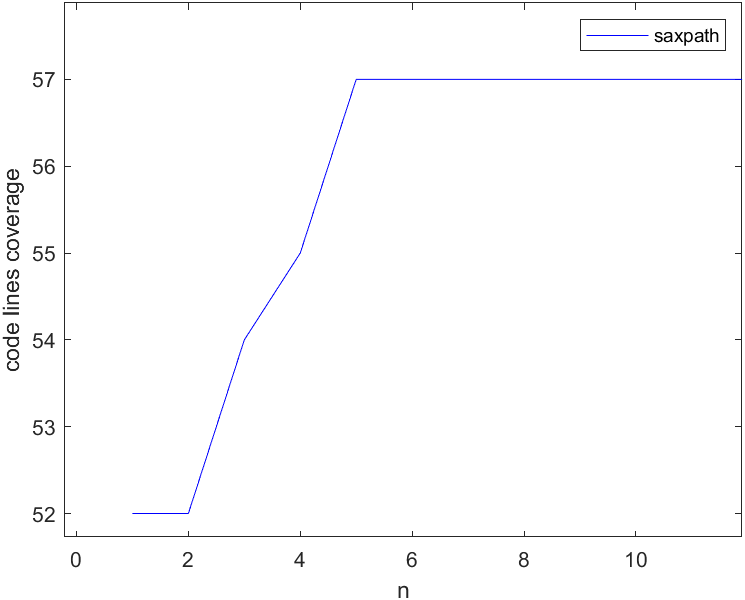
 

Figura 34 - nostro\_strumento Figura 35 – Randoop

Al contrario, i seguenti risultati mostrano che per alcuni tipi di programma il nostro strumento riesce ad ottenere un andamento che varia più velocemente, in particolare per il programma saxpath, considerato che dal grafico non è possibile apprezzare del tutto le variazioni poiché il numero di casi di test generati è molto elevato. A dimostrazione di quanto appena detto, riportiamo un focus sull’andamento relativo ai primi 10 casi di test eseguiti su saxpath, utilizzando il nostro strumento:



Ciò potrebbe significare che in questi casi particolari il criterio di distanza permette di generare casi di test significativamente differenti e che probabilmente i metodi eseguiti non erano caratterizzati da una complessità troppo elevata. In riferimento a dcparseargs, notiamo come i risultati ottenuti con Randoop portino ad una copertura minore, raggiunta tramite un andamento costante.

**3.4 Confronto con tecniche sofisticate**

Una versione di ART più performante potrebbe nascere sfruttando tecniche e metodologie più sofisticate, ad esempio tra quelle proposte dall’Intelligenza Artificiale. É già noto l’impiego di alcune di queste tecniche in alcune attività del testing: basti pensare alla validazione automatizzata delle interfacce utente, al testing delle API o allo ‘spidering’, cioè l’attività di scrittura automatica di casi di test. Alcune compagnie stanno utilizzando strumenti basati sull’IA anche per risolvere la classica problematica del testing che consiste nel trovare il minimo numero di test da eseguire per provare se un cambiamento introdotto in precedenza nel testing abbia giovato o meno al sistema.

In particolare, in uno studio denominato “Q-learning strategies for action selection in the TESTAR

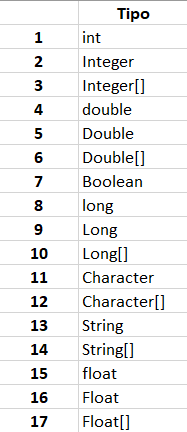
automated testing tool”, condotto all’ Università Politecnica di Valencia, “si valuta il Q-learning come metaeuristica per la selezione di azioni” nel contesto di TESTAR, dimostrando che, con un’opportuna scelta dei parametri di ingresso dell’algoritmo, le prestazioni migliorano rispetto ad attività di testing casuali come il monkey testing, con il quale viene adoperato un confronto. Il Q-learning rappresenta una tecnica di apprendimento per rinforzo in cui un agente, in uno stato s, deve scegliere un’azione da compiere all’interno di un insieme A (l’insieme delle azioni disponibili in quello stato) che ne determina la transizione di stato. Ogni azione intrapresa dall’agente gli fornisce una ‘ricompensa’, un punteggio numerico che misura l’utilità dell’azione eseguita in quel determinato stato. L’obiettivo del Q-learning è massimizzare il suo punteggio totale, “imparando” di volta in volta quale sia l’azione migliore da eseguire in ciascun stato. Uno dei suoi maggiori punti di rilievo consiste nell'abilità di comparare l'utilità aspettata delle azioni disponibili senza richiedere un modello dell'ambiente, motivo per cui viene definita come una tecnica ‘model-free’.

È evidente che il nostro strumento non sia dotato di tale complessità, adoperando tecniche ben lontane da quelle proposte dal Q-learning e dall’IA in generale. Infatti, lo strumento, che inizialmente genera input casuali, si limita a sfruttare il concetto di distanza per scegliere quali siano i casi di test da eseguire. La semplicità del nostro strumento ha comunque un grande vantaggio: la possibilità di giustificare facilmente le scelte intraprese dallo strumento e l’immediatezza di alcune osservazioni. In strumenti più complessi basati sull’IA ciò non è possibile, dovendo considerare una moltitudine di fattori in più che influenzano il loro comportamento. Questa caratteristica ci ha permesso di portare avanti le nostre analisi in maniera abbastanza lineare, evitando di impiegare complesse tecniche di reverse-engineering per comprendere il comportamento dello strumento.

**Conclusioni**

Come è stato possibile osservare nei precedenti paragrafi, il nostro strumento mostra alcuni evidenti limiti a cui però si può porre rimedio tramite determinate modifiche. Innanzitutto, risulta essere poco efficace in termine di linee di codice coperte per alcuni dei programmi testati. Sono diverse le situazioni che contribuiscono al problema:

1. Lo strumento realizzato è in grado di gestire solo alcuni tipi prestabiliti, tra cui la maggior parte dei tipi primitivi messi a disposizione dal linguaggio Java ed altri che sono dipendenti da questi. Di seguito una tabella contenente tutti i tipi gestiti:



È evidente che lo strumento non è in grado di gestire alcun tipo di oggetto appartenente ad una classe del programma da testare (non predefinita). Sarebbe possibile, dunque, risolvere o quantomeno rimediare al problema relativo a quei programmi per cui il livello di copertura raggiunto è basso, tramite la gestione di questi tipi. L’implementazione di tale meccanismo porterebbe al vantaggio di ottenere, probabilmente, una migliore copertura del codice. Sicuramente il nostro strumento risulterebbe più robusto e permetterebbe di testare in modo più efficace diversi tipi di software. Allo stesso tempo si introdurrebbe complessità nello strumento di testing e di conseguenza maggior overhead.

Un ulteriore problema è dato dal fatto che l’andamento dell’attività di testing, in termini di linee di codice coperte, risulta costante o lentamente variabile per alcuni programmi. Ciò significa che solo alcuni dei test effettuati eseguono nuove righe di codice e dunque che la maggior parte di essi sono inutili perché non evidenziano alcun malfunzionamento. Ciò potrebbe essere dovuto, come sottolineato in precedenza, alla scarsa complessità dello strumento e alla significatività dei casi di test generati, in base al criterio di distanza adottato. In generale si potrebbe pensare di effettuare alcune modifiche al criterio per verificare se lo strumento riesce a coprire un numero maggiore di linee di codice. Dunque, andrebbe sostituita la funzione calculate\_distance con un’altra più sofisticata che possa esprimere in modo più adeguato il concetto di diversità. L’obiettivo è aumentare la diversità dei casi di test eseguiti in modo da migliorare significativamente la quantità di codice scoperta. Ovviamente, non è detto che un’eventuale modifica possa portare a miglioramenti per via della forte dipendenza dallo specifico software analizzato.

Ad esempio, nello strumento proposto, la distanza tra due dati di input di tipo String dipende dalla differenza tra i diversi caratteri delle due stringhe e dalla differenza tra le lunghezze delle due stringhe. In particolare, il primo valore (quello relativo alla differenza tra i caratteri) ha un peso minore, infatti è moltiplicato per una costante pari a 0,1. In programmi in cui testare differenti lunghezze delle stringhe in input non è importante, otteremmo un livello di copertura basso. Si potrebbe dare maggior peso ai caratteri delle stringhe o valutare un criterio più generale che riesca a garantire buone performance per un qualsiasi tipo di software.

Sulla base delle osservazioni empiriche secondo cui le regioni di fallimento sono contigue, il test Adaptive combina la selezione casuale dei candidati con un processo di filtraggio per incoraggiare una diffusione uniforme dei casi di test in tutto il dominio di input. Studi sperimentali hanno dimostrato che ART può rilevare i fallimenti utilizzando fino al 50% di casi di test in meno rispetto ai test casuali. In quanto tale, riteniamo che rappresenti un'alternativa efficace ed efficiente al test casuale in molte applicazioni. Il successo di ART illustra il potenziale dell'approccio dei test basati sui fallimenti e l'impatto e l'importanza che la diversità ha sull'efficacia delle suite di test.

**Bibliografia**

1. Ilinca Ciupa, Andreas Leitner, Manuel Oriol, and Bertrand Meyer. 2008. ARTOO: adaptive random testing for object-oriented software. In Proceedings of the 30th international conference on Software engineering (ICSE '08). Association for Computing Machinery, New York, NY, USA, 71–80. DOI: https://doi.org/10.1145/1368088.1368099
2. Tsong Yueh Chen, Fei-Ching Kuo, Robert G. Merkel, T.H. Tse, Adaptive Random Testing: The ART of test case diversity, Journal of Systems and Software, Volume 83, Issue 1, 2010, Pages 60-66, ISSN 0164-1212, <https://doi.org/10.1016/j.jss.2009.02.022>
3. Hamlet, R. (2002). Random Testing. In Encyclopedia of Software Engineering, J.J. Marciniak (Ed.). <https://doi.org/10.1002/0471028959.sof268>
4. Andrea Arcuri and Lionel Briand. 2011. Adaptive random testing: an illusion of effectiveness? In Proceedings of the 2011 International Symposium on Software Testing and Analysis (ISSTA '11). Association for Computing Machinery, New York, NY, USA, 265–275. DOI: https://doi.org/10.1145/2001420.2001452
5. R. Huang, W. Sun, Y. Xu, H. Chen, D. Towey and X. Xia, "A Survey on Adaptive Random Testing," in IEEE Transactions on Software Engineering, doi: 10.1109/TSE.2019.2942921
6. Kuo, Fei-Ching & Chen, Tsong & Liu, Huai & Chan, Wing. (2008). Enhancing adaptive random testing for programs with high dimensional input domains or failure-unrelated parameters. Software Quality Journal. 16. 303-327. 10.1007/s11219-008-9047-6
7. Porfirio Tramontana, Domenico Amalfitano, Nicola Amatucci, Atif Memon, and Anna Rita Fasolino. 2019. Developing and Evaluating Objective Termination Criteria for Random Testing. ACM Trans. Softw. Eng. Methodol. 28, 3, Article 17 (August 2019), 52 pages. DOI: https://doi.org/10.1145/3339836
8. Michael J. Maher: Advances in Computer Science - ASIAN 2004, Higher-Level Decision Making, 9th Asian Computing Science Conference, Dedicated to Jean-Louis Lassez, on the Occasion of His 5th Cycle Birthday, Chiang Mai, Thailand, December 8-10, 2004, Proceedings. Lecture Notes in Computer Science 3321, Springer 2004, ISBN 3-540-24087-X
9. R. Huang, W. Sun, Y. Xu, H. Chen, D. Towey and X. Xia, "A Survey on Adaptive Random Testing," in IEEE Transactions on Software Engineering, doi: 10.1109/TSE.2019.2942921.
10. <https://en.wikipedia.org/wiki/Random_testing>
11. <https://sites.google.com/view/porfiriotramontana/research-activities/random-testing-termination-criteria>
12. Anna I. Esparcia-Alcàzar, Francisco Almenar, Mirella Martìnez, Urko Rueda, and Tanja E.J. Vos Q-learning strategies for action selection in the TESTAR automated testing tool