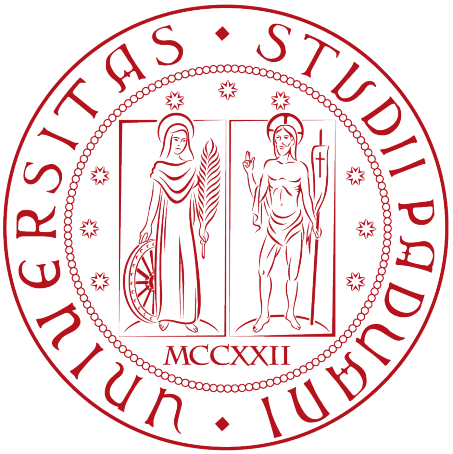
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



APAT - Android Platform Analysis Tool

Tesi di laurea triennale

Relatore

Prof. Alessandro Sperduti

Laureando

Xiaowei Wen

Anno Accademico 2019-2020

但愿日子清净, 抬头遇见的都是柔情.

"Così come in algebra due affermazioni false ne danno una vera, così spero che il prodotto dei miei insuccessi si concluda con un successo." –V. Van Gogh

Dedicato a nonna Giovanna e a tutte le persone che mi hanno aiutato a essere qui.

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa trecento ore, dal laureando Xiaowei Wen presso l'azienda Imola Informatica S.p.A. Gli obiettivi da raggiungere erano molteplici. In primo luogo era richiesto lo sviluppo di un tool in grado di automatizzare la decompilazione di un file Android Package (APK) e la ricompilazione del file. In secondo luogo era richiesta l'implementazione di alcune funzionalità di analisi del codice sorgente ottenuto dal passo precedente. Terzo e ultimo obbiettivo era quello di permettere di eseguire l'app con un proxy in un Android Virtual Device (AVD) e di monitorare il traffico di rete generando un file pcap.

"Dio benedica quelle persone che quando incroci il loro sguardo per sbaglio, sorridono."

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Sperduti, relatore della mia tesi, e Alessandro Proscia, il tutor aziendale, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

Desidero ringraziare con affetto i miei genitori per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio.

Ho desiderio di ringraziare poi Veronica, Alberto, Marco, Lorenzo, Linpeng, Tommaso, Alessandro e Giulio per tutti i bellissimi anni passati insieme, le avventure vissute e di essersi sorbiti mille delle mie lamentele.

Infine, vorrei esprimere la mia gratitudine alla famiglia Geminian e Bernardi per tutti gli aiuti ricevuti durante questi anni.

Padova, 23 Luglio 2020 Xiaowei Wen

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

In questa sezione viene svolta una breve introduzione alle idee dello stage e una breve presentazione dell'azienda Imola Informatica S.p.A.

1.1 L'idea

Le applicazioni mobili rappresentano oggi una delle principali sfide per la sicurezza informatica. La rapida diffusione degli smartphone, infatti, ha visto emergere il settore mobile come uno dei principali canali per l'erogazione di servizi provocando, di conseguenza, un aumento esponenziale degli attacchi contro tali piattaforme. Tra le minacce[11] fondamentali, distinguiamo:

- *Improper platform usage:* questa categoria di rischi comprende l'uso scorretto delle funzionalità offerte dalle piattaforme o il fallimento nell'utilizzo dei controlli di sicurezza offerti dalle piattaforme.
- *Insecure data storage:* le minacce possono essere una delle seguenti:
  - un malintenzionato è riuscito a entrare in possesso di un dispositivo in modo illecito;
  - un malware o un'applicazione ricompilata eseguono del codice non sicuro;

  Le conseguenze di questo tipo di minacce possono essere: il furto d'identità, la violazione della privacy e frode.
- *Insecure communication:* nella progettazione di un'applicazione mobile, i dati sono spesso scambiati in un'architettura client/server. Quando i dati devono essere inviati, l'attaccante potrebbe sfruttare le vulnerabilità per intercettare i dati sensibili.

- *Insecure authentication:* gli agenti di minaccia che sfruttano le vulnerabilità di autenticazione in genere lo fanno attraverso attacchi automatizzati che utilizzano strumenti disponibili o personalizzati.

- *Insufficient cryptography:* gli agenti delle minacce includono: chiunque abbia accesso fisico ai dati che sono stati crittografati in modo improprio o malware mobile che agisce per conto di un avversario. Questa vulnerabilità comporta il recupero non autorizzato d'informazioni sensibili dal dispositivo mobile e violazione della privacy.

- *Insecure authorization:* gli agenti delle minacce che sfruttano le vulnerabilità delle autorizzazioni in genere lo fanno attraverso gli attacchi automatici che utilizzano strumenti disponibili o personalizzati.

- *Client code quality:* include le entità che possono trasmettere in input dati non attendibili alle chiamate di metodo effettuate all'interno del codice mobile. Tale categoria di problemi non rappresenta in sè una grave problematica di sicurezza, ma potrebbe provocare vulnerabilità indesiderate.

- *Code tampering:* un utente malintenzionato sfrutta la modifica del codice tramite forme dannose delle applicazioni ospitante negli app store delle terze parti. L'utente malintenzionato può anche indurre l'utente a installare l'app tramite attacchi di phishing.
  In genere, per sfruttare questo tipo di vulnerabilità, un utente malintenzionato eseguirà le seguenti operazioni: apportare modifiche direttamente al file binario principale del pacchetto dell'app ospitante, apportare modifiche binarie dirette alle risorse, e reindirizzare o sostituire le API di sistema per intercettare ed eseguire il codice esterno dannoso.

- *Reverse engineering:* un utente malintenzionato in genere scarica l'app di destinazione da un app store e la analizza nel proprio ambiente locale utilizzando una suite di strumenti. Un malintenzionato deve eseguire un'analisi del file binario core finale per determinare la tabella di stringhe originale, il codice sorgente, le librerie, gli algoritmi e le risorse incorporate nell'app.

- *Extraneous Functionality:* un utente malintenzionato cerca di comprendere le funzionalità estranee all'interno di un'app mobile al fine di scoprire funzionalità nascoste nei sistemi di back-end. L'attaccante in genere sfrutta funzionalità estranee direttamente dai propri sistemi senza alcun coinvolgimento da parte degli utenti finali.

Le applicazioni mobili soffrono spesso di debolezze intrinseche dovute al design dell'applicazione e al suo sviluppo, debolezze che possono prevedere la memorizzazione d'informazioni sensibili sul device, possibilità di modificare l'applicazione (mediante repackaging dell'app), o la possibilità di un semplice reverse engineering.

La sicurezza è diventata un fattore critico sulle piattaforme mobile. Tradizionalmente, essa viene declinata secondo la cosiddetta triade CIA[12] che, in italiano, è possibile tradurre come Confidenzialità, Integrità e Disponibilità (dall'inglese Availability) di un sistema. Sui dispositivi mobile garantire il rispetto di tali proprietà è assai più complesso rispetto ai sistemi tradizionali, per via della loro eterogeneità, diffusione e distribuzione, per questi motivi è necessario ripensare alla sicurezza dei sistemi per garantire la protezione degli utenti, delle loro risorse e delle piattaforme.

La sicurezza diviene sempre più un fattore fondamentale da prevedere in tutte le fasi del ciclo di vita del software e non relegata alle sole fasi post rilascio: quindi è fondamentale ridefinire il concetto di Software Development Lifecycle in Secure Software Development Lifecycle[9], con una connotazione più orientata alla sicurezza in tutte le sue fasi, ovvero:

- *Analisi:* durante l'analisi bisogna individuare quali sono i rischi che il software è sottoposto, quindi individuare i requisiti che il software deve soddisfare per contrastare i potenziali attacchi;
- *Progettazione:* nel corso della fase di progettazione è necessario prestare attenzione nell'adozione delle librerie esterne, assicurandosi che quest'ultime non abbiano falle di sicurezza che minano l'integrità dell'intero software. Inoltre, è necessario adottare gli standard di sicurezza consolidati;
- *Sviluppo:* in questa fase le azioni[8] che si possono adottare sono molteplici, per esempio:
  - validare gli input;
  - controllare l'accesso;
  - evitare le chiavi hardcoded;
  - evitare di salvare le password in chiaro;
  - gestire gli utenti, le sessioni e i permessi;
  - standardizzare la documentazione in modo da facilitare la manutenzione del codice;
  - adottare la pratica del code review;

  - adottare il principio Keep It Simple Stupid (KISS);
  - adottare la pratica dello Static Application Security Testing (SAST) e Software Component Analysis (SCA).

- *Test:* durante la fase di test è necessario invece adottare la Dynamic Application Security Testing (DAST);
- *Deployment e Maintenance:* si tratta di messa in produzione del software sviluppato, quindi bisogna effettuare l'analisi della sicurezza e configurare sia il software che l'hardware sul quale è in esecuzione in modo corretto.

1.2 Introduzione al progetto

Lo scopo del progetto è quello di creare un tool di analisi che possa essere usato durante le fasi di codifica e di test in modo da aiutare l'utilizzatore a individuare le possibili problematiche riguardanti la sicurezza. Le funzionalità richieste sono le seguenti:

1. decompilazione di un file apk;
2. ricompilazione del file apk e la conseguente firma;
3. decodifica dei file Dalvik Executable (DEX) in codice sorgente *java*;
4. dump dell'area di storage dell'applicazione (compresi i file json, xml e database SQLite);
5. avvio dell'Android emulator device con le opzioni di proxy e permettere la registrazione delle attività di rete;
6. analisi del codice sorgente e dei file ottenuti dal passo 4 e infine estrazione dei dati dai file di database generando, al termine dell'operazione di analisi, un file di resoconto con i risultati dell'analisi.

1.3 L'azienda

Imola Informatica è una società indipendente di consulenza IT che entra in gioco ogni volta in cui una azienda pubblica o privata vuole migliorare i propri servizi. Innovare i propri processi di lavoro, gli approcci di management per cogliere le opportunità business offerte dalla trasformazione digitale. È a servizio dei principali

gruppi finanziari e assicurativi e ogni giorno è a fianco di grandi aziende e piccole startup nel gestire il cambiamento tecnologico e culturale. Fa parte di una rete che condivide l'idea di fare innovazione a misura delle persone, delle imprese e della collettività. Sono impegnati per lo sviluppo consapevole di comunità locali e smart cities.

1.4 Risultati

Lo stage si è svolto dal 4 maggio 2020 al 26 giugno 2020, la durata complessiva è di 312 ore lavorative.
Gli obiettivi fissati nella sezione §1.2 sono stati tutti raggiunti, il tutor aziendale Alessandro Proscia si è mostrato soddisfatto del prodotto finale. Durante lo stage ho sviluppato un software per l'analisi dei file APK che è in grado di decompilare l'APK, analizzare le risorse contenute al suo interno, ricompilarlo, installarlo su un AVD e quindi monitorare il traffico di rete generato dall'applicazione tutto questo in modo automatico. Inoltre, genera un PDF che contiene i risultati ottenuti durante l'analisi.
Il processo di analisi, testato su quattro applicazione, viene eseguito mediamente in 131.69 secondi, ma esiste una consistente differenza in base alla dimensione dell'applicazione e al linguaggio utilizzato per la sua realizzazione. Questo divario è causato dagli strumenti di terze di parti impiegati, per cui non è stato possibile ridurla.
Per la sua realizzazione sono state utilizzate le seguenti tecnologie:

- CFR: decompilatore Java;
- APKTool: decompilatore APK;
- Dex2Jar: decompilatore Dex verso il formato jar;
- XPath: linguaggio per estrarre le informazioni dal file AndroidManifest.xml;
- Java: linguaggio di programmazione utilizzato per la realizzazione del tool;
- Android Emulator: emulatore dei dispositivi Android, utilizzato per eseguire l'applicazione.

1.5 Organizzazione del testo

Il primo capitolo dà un'introduzione generale all'azienda ospitante e l'idea fondante del progetto di stage;

Il secondo capitolo approfondisce gli obiettivi del progetto con i relativi requisiti e pianificazione;

Il terzo capitolo approfondisce l'analisi dei requisiti dello strumento;

Il quarto capitolo approfondisce la progettazione e la codifica dello strumento;

Il quinto capitolo approfondisce la verifica e la validazione dello strumento.

Il sesto capitolo descrive le opinioni personali del laureando.

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*[g];
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Descrizione dello stage

Questo capitolo contiene l'introduzione del progetto, l'analisi dei rischi, la specifica degli obiettivi e la pianificazione dei lavori.

2.1 Modalità di svolgimento

Le attività di stage sono state svolte in modalità tele-lavoro sia a causa dell'emergenza sanitaria che per motivi di distanza geografica, tuttavia le interazioni con il tutor aziendale sono avvenute con la frequenza necessaria. Le modalità, invece, sono state le seguenti:

- Telegram, per scambio dei messaggi;
- Jitsi meet, per le video conferenze.

Per la pianificazione settimanale del lavoro segue invece la sezione della pianificazione.

2.2 Analisi preventiva dei rischi

Durante la fase di analisi iniziale sono stati individuati alcuni possibili rischi a cui si potrà andare incontro. Si è quindi proceduto a elaborare delle possibili soluzioni per far fronte a tali rischi.

1. Tecnologie/framework sconosciute
Descrizione: Durante lo svolgimento dello stage, è l'uso di alcune tecnologie o di framework finora mai viste dallo stagista..

Soluzione: È stato programmato un periodo di auto-formazione riguardante le tecnologie che si prevede di utilizzare. Inoltre, il tutor aziendale si è reso disponibile ad aiutare lo stagista in caso di necessità.

2. Guasti hardware
Descrizione: È possibile che il computer assegnato possa incorrere in guasti hardware, rischiando così di causare rallentamenti o addirittura perdita del lavoro.
Soluzione: Il lavoro svolto verrà versionato nel sistema di versionamento dell'azienda.

3. Incomprensioni o scelte non ottimali
Descrizione: A causa dell'inesperienza può accadere che le attività da svolgere siano fraintese o valutate erroneamente, causando la realizzazione di un prodotto non consono alle aspettative dell'azienda.
Soluzione: Le scelte progettuali fondamentali sono svolte in concordanza con il tutor aziendale.

2.3 Requisiti e obiettivi

Lo scopo dello stage è la realizzazione di un tool di analisi delle applicazioni Android che permetta di automatizzare le operazioni di reverse engineering dei pacchetti delle app mobile (APK), effettuando nell'ordine:

1. decompilazione sorgenti;
2. analisi dei file sorgenti ottenuti dalla decompilazione;
3. generazione di un report dell'analisi;
4. repackaging dell'applicativo;
5. installazione dell'APK ricompilato su un AVD;
6. avvio dell'AVD con opzioni di proxy;
7. creazione di un file pcap che contiene i dettagli delle attività di rete;
8. firma dell'APK ottenuto dal repackaging.

Al termine dell'esecuzione dovranno essere rese disponibili informazioni quali:

- sorgenti decompilati;
- stringhe estratte dai sorgenti per l'individuazione di chiavi hard coded;

- contenuto della storage area dell'app;
- stringhe estratte dalla storage area dell'app per l'inviduazione d'informazioni sensibili;
- file di trace per le operazioni di rete effettuate in un file pcap.

2.4 Pianificazione

Durata in ore	**Descrizione dell'attività**
40	Studio e approfondimento delle tecnologie di sviluppo
40	Analisi dei requisiti
40	Progettazione
100	Stesura del codice
60	Collaudo e risoluzione bug
40	Analisi delle performance e assestamento dei risultati
Totale ore	**320**

Capitolo 3

Analisi dei requisiti

In questa sezione viene presentata l'analisi dei requisiti, comprensivo dei casi d'uso, requisiti e il tracciamento di questi ultimi. Per lo studio dei casi di utilizzo del prodotto sono stati creati dei diagrammi. I diagrammi dei casi d'uso (in inglese Use Case Diagram*) sono diagrammi di tipo Unified Modelling Language (UML) dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso.*

3.1 Attori

L'unico utilizzatore del software è identificato come **attore generico** e ha il permesso di effettuare tutte le operazioni offerte dallo strumento.

3.2 Casi d'uso

Di seguito sono elencati i casi d'uso individuati dallo stagista durante la fase dell'analisi dei requisiti.

UC-1 Selezione file

- **attori:** utente generico;
- **descrizione:** serve per permettere all'utente di selezionare il file APK da analizzare;
- **pre-condizioni:** l'utente ha avviato il tool;
- **post-condizioni:** l'utente ha selezionato un file di estensione APK valido;

- **flusso degli eventi principali:**
  - l'utente seleziona il file;
  - l'utente conferma la selezione del file.

UC-2 Avvio decompilazione

- **attori:** utente generico;
- **descrizione:** l'utente deve avviare la decompilazione del file APK;
- **pre-condizioni:** l'utente ha selezionato con successo un file APK;
- **post-condizioni:** la decompilazione è avvenuta con successo;
- **flusso degli eventi principali:**
  - l'utente avvia la decompilazione;
  - l'utente visualizza il messaggio della decompilazione avvenuta con successo;
- **Estensione:**
  - UC-3 Visualizzazione errore di decompilazione;

UC-3 Visualizzazione errore di decompilazione

- **attori:** utente generico;
- **descrizione:** la decompilazione del file APK potrebbe generare degli errori;
- **pre-condizioni:** l'utente ha avviato la decompilazione dell'APK;
- **post-condizioni:** l'utente ha visualizzato il messaggio d'errore;
- **flusso degli eventi principali:**
  - l'utente visualizza il messaggio di errore;

UC-4 Installazione APK ricompilato

- **attori:** utente generico;
- **descrizione:** permette all'utente d'installare l'apk, decompilato, modificato e ricompilato, su un AVD;

- **pre-condizioni:** la decompilazione dell'APK è stata eseguita con successo;
- **post-condizioni:** l'APK è stato installato sull'AVD con successo;
- **flusso degli eventi principali:**
  - l'utente seleziona un'AVD presente sul proprio computer;
  - l'utente avvia l'installazione dell'APK ricompilato;
  - l'utente visualizza un messaggio che segnala l'installazione avvenuta con successo.

UC-4.1 Selezione AVD

- **attori:** utente generico;
- **descrizione:** serve per permettere all'utente di selezionare un'AVD presente nel proprio computer;
- **pre-condizioni:** l'utente ha avviato lo strumento;
- **post-condizioni:** l'utente ha selezionato l'AVD da avviare;
- **flusso degli eventi principali:**
  - l'utente visualizza un elenco delle AVD presenti nel proprio computer;
  - l'utente seleziona un'AVD;
  - l'utente conferma la selezione;
  - l'AVD selezionato si è avviato con successo.
- **Estensione**
  - UC-5 Visualizzazione messaggio nessun AVD rilevato;

UC-5 Visualizzazione messaggio nessun AVD rilevato

- **attori:** utente generico;
- **descrizione:** quando non sono presenti nessun AVD o il tool non è riuscito a rilevarne, viene mostrato un messaggio all'utente;
- **pre-condizioni:** l'utente ha aperto il tool;
- **post-condizioni:** l'utente ha visualizzato il messaggio;

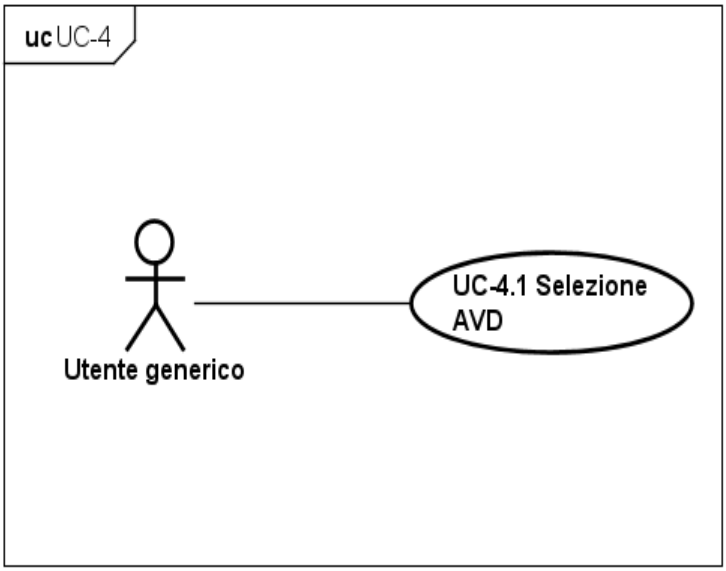


Figura 3.1: Sottocaso d'uso UC-4.1 Selezione AVD.

- **flusso degli eventi principali:**
  - l'utente visualizza il messaggio;

UC-6 Errore durante avvio dell'AVD

- **attori:** utente generico;
- **descrizione:** serve a mostrare all'utente gli eventuali errori durante l'avvio dell'AVD;
- **pre-condizioni:** l'utente ha selezionato un'AVD e ha confermato l'avvio;
- **post-condizioni:** l'utente ha visualizzato il messaggio d'errore;
- **flusso degli eventi principali:**
  - l'utente visualizza il messaggio d'errore;

UC-7 Dump dello storage interno

- **attori:** utente generico;
- **descrizione:** nel caso l'utente volesse una copia dei dati interni dell'applicativo, ha bisogno di fare il dump;

- **pre-condizioni:** l'installazione dell'APK modificato è andata a buon fine;
- **post-condizioni:** l'area di storage dell'app è stata copiata con successo;
- **flusso degli eventi principali:**
  - l'utente seleziona la voce "copia i dati interni";
  - l'utente seleziona il path dove collocare i dati.

UC-8 Decodifica del codice

- **attori:** utente generico;
- **descrizione:** durante la decompilazione dell'APK vengono creati dei file .dex che contengono il codice sorgente dell'APK, e questo caso d'uso serve per permettere all'utente di ottenere il codice sorgente;
- **pre-condizioni:** la decompilazione è avvenuta con successo;
- **post-condizioni:** l'utente ha ottenuto una copia del codice sorgente in java;
- **flusso degli eventi principali:**
  - l'utente ha selezionato la funzionalità di decodifica dei dex;
  - l'utente seleziona il percorso dove posizionare il codice sorgente ottenuto;
  - l'utente ha salvato il codice sorgente ottenuto.

UC-8.1 Avvio decodifica

- **attori:** utente generico;
- **descrizione:** serve all'utente per avviare la decodifica dei file .dex;
- **pre-condizioni:** la decompilazione dell'APK è avvenuta correttamente;
- **post-condizioni:** la decodifica è avvenuta con successo;
- **flusso degli eventi principali:**
  - l'utente seleziona la funzionalità di decodifica dei file .dex;

UC-8.2 Salvataggio del codice decodificato

- **attori:** utente generico;
- **descrizione:** serve all'utente per salvare i codici sorgenti decodificati;
- **pre-condizioni:** la decompilazione è avvenuta con successo;
- **post-condizioni:** i file con i codici sorgenti sono stati salvati correttamente;
- **flusso degli eventi principali:**
  - l'utente seleziona la voce "salva file decodificati";
  - l'utente seleziona la posizione dove vuole salvare i file;
  - i file vengono salvati correttamente.

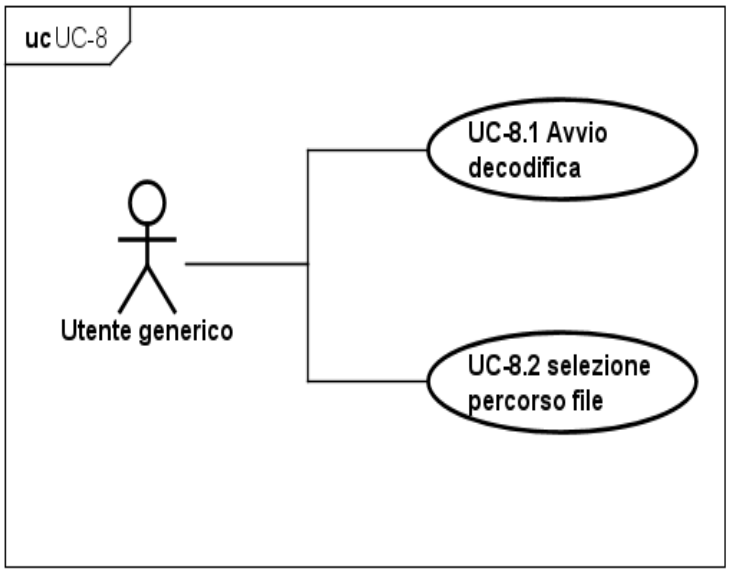


Figura 3.2: Caso d'uso UC-8 Decodifica codici .dex.

UC-9 Visualizzazione errore decodifica

- **attori:** utente generico;
- **descrizione:** durante la decodifica dei .dex possono sorgere molteplici errori;
- **pre-condizioni:** l'utente ha selezionato la funzionalità di decodifica del codice .dex;

- **post-condizioni:** l'utente ha visualizzato il messaggio di errore durante la decodifica;
- **flusso degli eventi principali:**
  - l'utente ha visualizzato il messaggio d'errore;

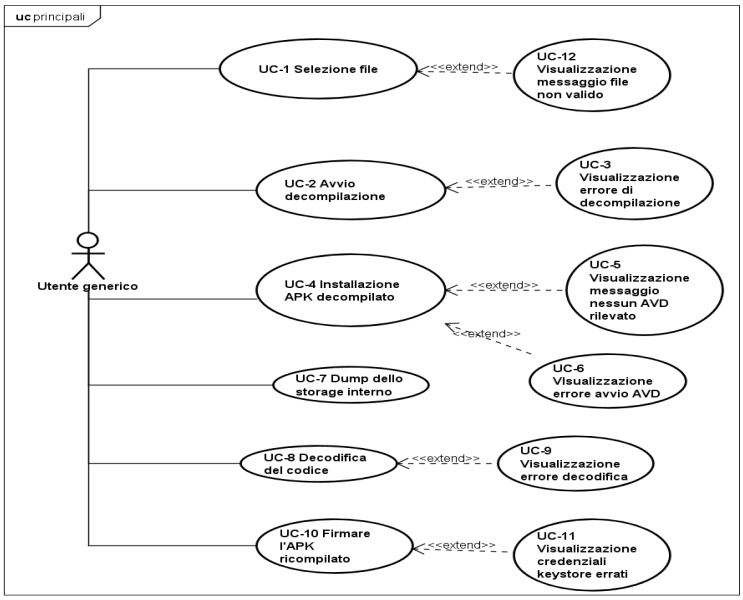


Figura 3.3: Casi d'uso da 1 a 12.

UC-10 Firmare l'APK ricompilato

- **attori:** utente generico;
- **descrizione:** dopo la ricompilazione si può firmare l'APK;
- **pre-condizioni:** la ricompilazione dell'APK è avvenuta correttamente;
- **post-condizioni:** l'APK è stata firmato correttamente;
- **flusso degli eventi principali:**
  - UC-10.1 selezione keystore;
  - UC-10.3 inserimento alias;
  - UC-10.4 inserimento password.
- **Estensione**
  - UC-11 Visualizzazione credenziali keystore errate;

UC-10.1 Selezione keystore

- **attori:** utente generico;
- **descrizione:** l'utente deve selezionare il keystore da utilizzare per firmare l'APK
- **pre-condizioni:** la ricompilazione dell'APK è avvenuta correttamente;
- **post-condizioni:** è stato selezionato un file di tipo keystore corretto (estensione: **.jks**);
- **flusso degli eventi principali:**
  - l'utente seleziona la funzionalità per selezionare il keystore;
  - l'utente seleziona il keystore;
- **flussi secondari**
  - UC-10.2 visualizzazione messaggio che mostra che il file selezionato non è valido;

UC-10.2 Visualizzazione messaggio file selezionato non valido

- **attori:** utente generico;
- **descrizione:** l'utente, al quale è stato chiesto di selezionare un file di tipo keystore, potrebbe selezionare un file non valido;
- **pre-condizioni:** l'utente ha selezionato un file;
- **post-condizioni:** il messaggio di errore è stato mostrato;
- **flusso degli eventi principali:**
  - l'utente visualizza il messaggio d'errore.

UC-10.3 Inserimento Alias

- **attori:** utente generico;
- **descrizione:** l'utente deve inserire l'alias della chiave da utilizzare durante la firma dell'APK;
- **pre-condizioni:** l'utente ha selezionato un keystore valido;

- **post-condizioni:** l'utente ha inserito l'alias da utilizzare;
- **flusso degli eventi principali:**
  - l'utente inserisce l'alias della chiave da utilizzare per firmare l'APK;

UC-10.4 Inserimento password

- **attori:** utente generico;
- **descrizione:** l'utente deve inserire la password del keystore da utilizzare durante la firma dell'APK;
- **pre-condizioni:** l'utente ha selezionato un keystore valido;
- **post-condizioni:** l'utente ha inserito la password da utilizzare;
- **flusso degli eventi principali:**
  - l'utente inserisce la password;

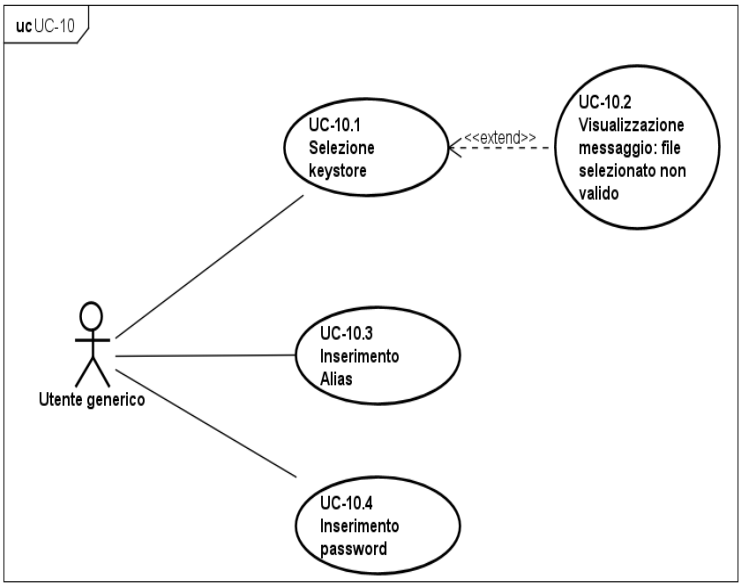


Figura 3.4: Sottocasi d'uso del caso d'uso UC-10.

UC-11 Visualizzazione credenziali keystore errati

- **attori:** utente generico;
- **descrizione:** per firmare l'APK l'utente deve inserire delle credenziali, e quando quest'ultime sono errate viene mostrato un messaggio di errore;

- **pre-condizioni:** l'utente ha selezionato la funzionalità per ricompilare l'APK;
- **post-condizioni:** l'utente ha visualizzato il messaggio di errore;
- **flusso degli eventi principali:**
  - l'utente ha visualizzato l'errore;

UC-12 Visualizzazione messaggio file non valido

- **attori:** utente generico;
- **descrizione:** quando l'utente seleziona un file che non ha l'estensione APK un messaggio deve essere mostrato all'utente;
- **pre-condizioni:** l'utente ha selezionato un file non APK;
- **post-condizioni:** l'utente ha visualizzato il messaggio d'errore;
- **flusso degli eventi principali:**
  - l'utente visualizza il messaggio d'errore.

UC-13 Decodifica dei file *.dex* in file *.java*

- **attori:** utente generico;
- **descrizione:** dall'APK ricompilato si ottengono dei file *.dex* che possono essere convertiti in *.class* e quindi in *.java*;
- **pre-condizioni:** la decompilazione dell'APK è andata a buon fine;
- **post-condizioni:** la decodifica dei file *.dex* in *.java* è andata a buon fine;
- **flusso degli eventi principali:**
  - l'utente seleziona la voce "Decodifica Dex".

UC-14 Analisi del codice *.java*

- **attori:** attore generico;
- **descrizione:** dopo aver ottenuto i file *.java* eseguendo il caso d'uso UC-13, si potrà effettuare dell'analisi sul codice;
- **pre-condizioni:** la decodifica dei file *.dex* in *.java* è andata a buon fine;

- **post-condizioni:** viene generato un pdf con i risultati dell'analisi;
- **flusso degli eventi principali:**
  - l'utente seleziona la voce "analyze";
  - selezione delle opzioni di analisi;
  - il file viene salvato nel percorso specificato dall'utente.
- **Estensione:**
  - UC-14.1 Selezione path per salvare i risultati;
  - UC-14.2 Elencare i file analizzati;
  - UC-14.3 Trovare le stringhe hard-coded;
  - UC-14.4 Selezionare un file black list;
  - UC-14.5 Analizzare i dati dello storage interno;
  - UC-14.6 Selezionare un file di white list.

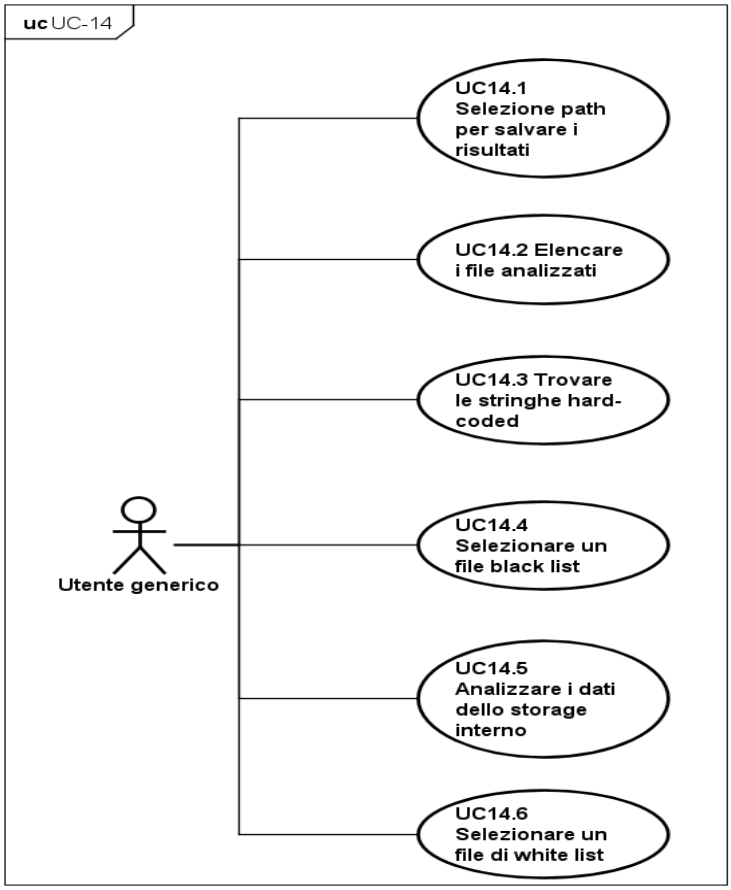


Figura 3.5: Sottocasi d'uso del caso d'uso 14.

UC-14.1 Selezione path per salvare i risultati

- **attori:** attore generico;
- **descrizione:** l'utente deve specificare il path di destinazione del pdf che contiene i risultati dell'analisi;
- **pre-condizioni:** l'utente ha selezionato l'opzione di analisi del tool;
- **post-condizioni:** l'utente ha specificato il path di destinazione del pdf;
- **flusso degli eventi principali:**
  - l'utente seleziona il path di destinazione.

UC-14.2 Elencare i file analizzati

- **attori:** attore generico;
- **descrizione:** l'utente specifica che il pdf risultante deve contenere una lista dei file analizzati;
- **pre-condizioni:** l'utente ha selezionato l'opzione di analisi;
- **post-condizioni:** l'utente ha selezionato l'opzione di elencare i file analizzati;
- **flusso degli eventi principali:**
  - l'utente seleziona l'opzione di elencare i file analizzati.

UC-14.3 Trovare le stringhe hard-coded

- **attori:** attore generico;
- **descrizione:** l'utente specifica che il pdf risultante deve contenere tutte le stringhe hard-coded presenti nei codici sorgenti analizzati;
- **pre-condizioni:** l'utente ha selezionato l'opzione di analisi;
- **post-condizioni:**; l'utente ha selezionato l'opzione per trovare le stringhe hard-coded presenti nei codici sorgenti;
- **flusso degli eventi principali:**
  - l'utente seleziona la voce Trovare le stringhe hard-coded.

UC-14.4 Selezionare un file blacklist

- **attori:** attore generico;
- **descrizione:** l'utente seleziona un file di blacklist;
- **pre-condizioni:** l'utente ha selezionato l'opzione di analisi;
- **post-condizioni:** l'utente ha selezionato un file di blacklist;
- **flusso degli eventi principali:**
  - l'utente inserisce il file path verso il file di blacklist.

UC-14.5 Analizzare i dati dello storage interno

- **attori:** attore generico;
- **descrizione:** l'utente seleziona la voce analisi dei dati presenti nell'area dello storage dell'app;
- **pre-condizioni:** l'utente ha selezionato l'opzione di analisi;
- **post-condizioni:** l'utente ha selezionato la voce di analisi dei dati dati dell'area dello storage;
- **flusso degli eventi principali:**
  - l'utente ha selezionato la voce analisi dei dati presenti nell'area dello storage dell'app.

UC-14.6 Selezionare un file whitelist

- **attori:** attore generico;
- **descrizione:** l'utente seleziona un file di whitelist;
- **pre-condizioni:** l'utente ha selezionato l'opzione di analisi;
- **post-condizioni:** l'utente ha selezionato un file di whitelist;
- **flusso degli eventi principali:**
  - l'utente inserisce il file path verso il file di whitelist.

UC-15 avvio AVD

- **attori:** attore generico;
- **descrizione:** serve per avviare un'avd per poter eseguire l'applicazione;
- **pre-condizioni:** il tool è stato avviato correttamente e nel sistema è presente almeno un AVD;
- **post-condizioni:** l'AVD è stato avviato correttamente;
- **flusso degli eventi principali:**
  - l'attore visualizza l'elenco delle AVD presenti nel sistema;
  - l'attore seleziona un'AVD che vuole avviare;
  - l'attore seleziona se utilizzare un proxy da impostare nell'AVD;
  - l'attore seleziona la voce avvia AVD.
- **estensione:** UC-5 Visualizzazione messaggio nessun AVD rilevato.

UC-16 avvio AVD senza proxy

- **attori:** attore generico;
- **descrizione:** quando l'attore decide che non vuole avviare l'AVD modificando le impostazioni di proxy, viene eseguito questo caso d'uso;
- **pre-condizioni:** l'attore ha avviato correttamente il tool e nel sistema è presente almeno un'AVD;
- **post-condizioni:** l'AVD è stato avviato correttamente con le impostazioni di proxy di default;
- **flusso degli eventi principali:**
  - l'attore visualizza l'elenco delle AVD presenti nel sistema;
  - l'attore seleziona un'AVD che vuole avviare;
  - l'attore seleziona di non utilizzare un proxy da impostare nell'AVD;
  - l'attore seleziona la voce avvia AVD.

UC-17 avvio AVD con proxy

- **attori:** attore generico;
- **descrizione:** quando l'attore decide che non vuole avviare l'AVD modificando le impostazioni di proxy, viene eseguito questo caso d'uso;
- **pre-condizioni:** l'attore ha avviato correttamente il tool e nel sistema è presente almeno un'AVD;
- **post-condizioni:** l'AVD è stato avviato correttamente con le impostazioni di proxy inseriti;
- **flusso degli eventi principali:**
  - l'attore visualizza l'elenco delle AVD presenti nel sistema;
  - l'attore seleziona un'AVD che vuole avviare;
  - l'attore seleziona di utilizzare un proxy da impostare nell'AVD;
  - l'attore inserisce le informazioni del proxy, per esempio: *localhost:8080*;
  - l'attore seleziona la voce avvia AVD.

UC-17.1 Selezione voce Avvia con proxy

- **attori:** attore generico;
- **descrizione:** l'utente vuole avviare l'AVD con le opzioni di proxy;
- **pre-condizioni:** il tool è stato avviato correttamente ed è riuscito a rilevare gli AVD presenti nel sistema;
- **post-condizioni:** l'opzione di proxy è stata selezionata;
- **flusso degli eventi principali:**
  - l'utente seleziona la voce per avviare l'AVD.

UC-17.2 selezione voce avvio AVD

- **attori:** attore generico;
- **descrizione:** l'utente avvia l'AVD;
- **pre-condizioni:** l'utente ha selezionato le opzioni per avviare l'AVD;

- **post-condizioni:** il tool avvia l'AVD;
- **flusso degli eventi principali:**
  - l'utente seleziona la voce avvio l'AVD.

UC-17.3 inserimento indirizzo e porta proxy

- **attori:** attore generico;
- **descrizione:** serve all'attore per inserire i dati del proxy;
- **pre-condizioni:** l'utente ha selezionato l'opzione di avviare l'AVD con il proxy;
- **post-condizioni:** i dati sono stati inseriti correttamente;
- **flusso degli eventi principali:**
  - l'utente inserisce l'indirizzo IP del server proxy;
  - l'utente inserisce il numero della porta del server proxy.

UC-17.4 conferma dei dati

- **attori:** attore generico;
- **descrizione:** serve all'utente per confermare i dati inseriti per avviare l'AVD;
- **pre-condizioni:** l'utente ha inserito le opzioni di proxy correttamente;
- **post-condizioni:** l'AVD è stato avviato correttamente con le opzioni di proxy;
- **flusso degli eventi principali:**
  - l'utente conferma le informazioni inserite e avvia l'AVD.

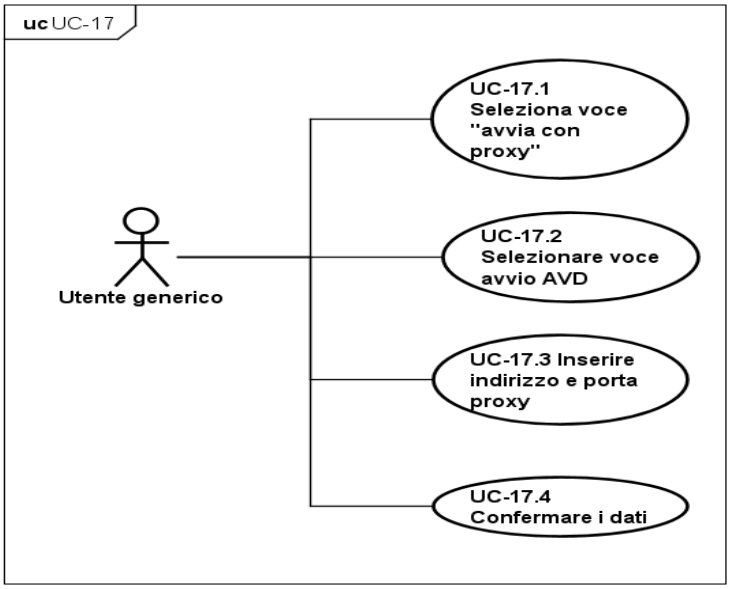


Figura 3.6: Sottocasi d'uso del caso d'uso 17.

UC-18 Inizio registrazione traffico di rete

- **attori:** attore generico;
- **descrizione:** quando l'utente vuole registrare le attività di rete effettuate dall'AVD eseguendo l'applicazione, può utilizzare questa funzionalità;
- **pre-condizioni:** l'attore ha avviato l'applicazione;
- **post-condizioni:** l'attore ha registrato le attività di rete e ottenuto un file pcap;
- **flusso degli eventi principali:**
  - l'attore seleziona la voce "Start record!";
  - l'attore successivamente seleziona la voce "Stop Record".

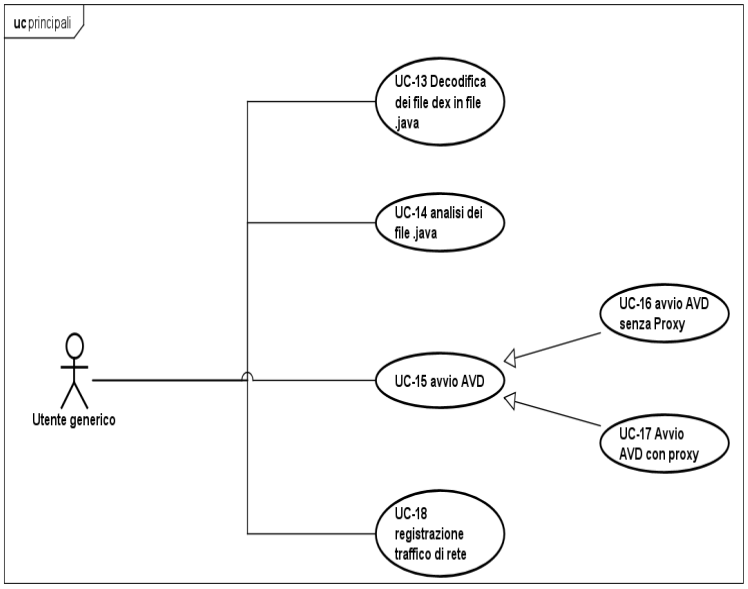


Figura 3.7: Casi d'uso da 13 a 18.

3.3 Tracciamento dei requisiti

3.3.1 Classificazione

Di seguito sono riportati i requisiti individuati durante l'attività di analisi. Tali requisiti sono stati individuati dai casi d'uso, dal piano di lavoro e dai colloqui con il tutor aziendale *Alessandro Proscia*. I requisiti individuati sono stati divisi in:

- **Requisiti funzionali:** insieme di requisiti che definiscono le azioni fondamentali che devono avvenire in grado di processare un input e di generare un output;
- **Requisiti dichiarativi:** insieme di requisiti che rappresentano un vincolo di natura realizzativa, normativa o contrattuale;
- **Requisiti qualitativi:** insieme di requisiti che garantiscono una certa qualità al prodotto e che indicano le best practice usate per la realizzazione.

Inoltre, a ogni requisito è stata assegnata un'importanza:

- **Obbligatori:** requisito al quale non si può rinunciare, indispensabile per il corretto funzionamento del prodotto;
- **Desiderabile:** requisito non necessario ma che porta valore aggiunto al prodotto;

- **Facoltativo:** requisito che risulta essere relativamente utile oppure contrattabile con il proponente in un momento successivo.

3.3.2 Requisiti funzionali

Di seguiti sono elencati i requisiti derivanti dai casi d'uso individuati dalla sezione precedente.

ID	Descrizione	Fonte
R-1-F-O	Il tool deve permettere di selezionare un file APK.	UC-1
R-1.1-F-O	Il tool deve permettere di mostrare un messaggio di errore se file selezionato non è valido.	UC-1
R-2-F-O	Il tool deve permettere di avviare la decompilazione dell'APK selezionato.	UC-2
R-2.1-F-O	Il tool deve permettere di visualizzare un messaggio di decompilazione avvenuto con successo.	UC-2
R-2.2-F-O	Il tool deve aggiungere il flag android:debuggable="true" nel AndroidManifest.xml del decompilato.	UC-2
R-3-F-O	Il tool deve permettere di visualizzare il messaggio di errore quando la decompilazione non è terminata con successo.	UC-3
R-4-F-O	Il tool deve permettere d'installare l'APK decompilato e modificato su un AVD.	UC-4
R-4.1-F-D	Il tool deve permettere di ricompilare l'APK decompilato.	UC-4
R-4.2-F-D	Il tool deve permettere di avviare l'applicazione.	UC-4
R-4.3-F-O	Il tool deve permettere di selezionare un'AVD presente nel sistema.	UC-4
R-4.4-F-O	Il tool deve permettere di visualizzare l'elenco delle AVD presenti nel sistema operativo.	UC-4

R-4.5-F-O	Il tool deve permettere di confermare la selezione dell'AVD.	UC-4
R-5-F-O	Il tool deve permettere di visualizzare il messaggio quando non sono stati rilevati nessun AVD.	UC-5
R-6-F-O	Il tool deve permettere di visualizzare il messaggio di errore se l'AVD selezionato non si è avviato correttamente.	UC-6
R-7-F-O	Il tool deve permettere di fare il dump dello storage interno dell'applicazione.	UC-7
R-7.1-F-O	Il tool deve permettere di selezionare il path dove collocare i dati copiati.	UC-7
R-8-F-D	Il tool deve permettere di visualizzare i dex ottenuti dalla decompilazione.	UC-8
R-8.1-F-O	Il tool deve permettere di visualizzare il messaggio quando la decodifica è avvenuta con successo.	UC-8
R-8.2-F-O	Il tool deve permettere di far selezionare il dex che l'utente vuole decodificare.	UC-8
R-8.3-F-O	Il tool deve permettere di confermare la selezione del dex da decodificare.	UC-8
R-8.4-F-O	Il tool deve permettere di far selezionare il path di dove salvare i file ottenuti dalla decodifica.	UC-8
R-9-F-O	Il tool deve permettere di visualizzare il messaggio d'errore se la decodifica non è andata a buon fine.	UC-9
R-10-F-F	Il tool deve permettere di firmare l'APK ricompilato.	UC-10
R-10.1-F-F	Il tool deve permettere di selezionare un file di tipo keystore.	UC-10
R-10.2-F-F	Il tool deve permettere di mostrare un messaggio di errore quando viene selezionato un file non keystore.	UC-10

R-10.3-F-F	Il tool deve permettere d'inserire l'alias della chiave da utilizzare.	UC-10
R-10.4-F-F	Il tool deve permettere d'inserire la password del keystore da utilizzare.	UC-10
R-11-F-F	Il tool deve permettere di mostrare dei messaggi quando le credenziali inserite non sono corrette.	UC-11
R-12-F-O	Il tool deve permettere di mostrare dei messaggi quando è stato selezionato un file non valido.	UC-12
R-13-F-O	Il tool deve permettere di decodificare i file *.dex* in *.java*	UC-13
R-14-F-O	Il tool deve permettere di effettuare dell'analisi del codice java	UC-14
R-14.1-F-O	Il tool deve permettere di selezionare diverse opzioni di analisi	UC-14.1
R-14.2-F-O	Il tool deve permettere di selezionare il path dove collocare i risultati dell'analisi.	UC-14.2
R-15-F-D	Il tool deve permettere di avviare un'AVD presente nel sistema.	UC-15
R-15.1-F-D	Il tool deve permettere di visualizzare l'elenco delle AVD presenti nel sistema operativo.	UC-15
R-15.2-F-D	Il tool deve permettere di selezionare un'AVD presente nel sistema.	UC-15
R-15.3-F-D	Il tool deve permettere di specificare se avviare l'AVD con opzione di proxy.	UC-15
R-16-F-D	Il tool deve permettere d'inserire le informazioni per il proxy.	UC-17
R-16.1-F-D	Il tool deve permettere d'inserire l'indirizzo del server proxy.	UC-17
R-16.2-F-D	Il tool deve permettere d'inserire il numero di porta del server proxy.	UC-17
R-17-F-D	Il tool deve permettere di registrare il traffico di rete.	UC-18

R-17.1-F-D	Il tool deve generare il file che contiene i dettagli del traffico di rete.	UC-18

Tabella 3.1: Requisiti funzionali.

3.3.3 Requisiti di vincolo

ID	Descrizione	Fonte
R-1-V-D	Il tool può essere un tool da righe di comando	Piano di lavoro.
R-2-V-D	Il tool può essere un tool dotato di GUI	Piano di lavoro.
R-3-V-D	Il tool può essere sviluppato in JAVA	Piano di lavoro.
R-4-V-F	Il tool può essere sviluppato utilizzando i linguaggi funzionali.	Piano di lavoro.

Tabella 3.2: Requisiti di vincolo.

3.3.4 Requisiti qualitativi

ID	Descrizione	Fonte
R-1-Q-O	Il codice sorgente deve essere versionato col sistema di versionamento dell'azienda ospitante.	Piano di lavoro
R-2-Q-O	Il codice sorgente deve essere sotto licenza GPL v3.	Piano di lavoro
R-3-Q-O	Il code coverage dei test unitari deve superare 80%.	Piano di lavoro
R-4-Q-D	Il code coverage dei test unitari deve essere 90%.	Piano di lavoro

Tabella 3.3: Requisiti qualitativi.

3.4 Tracciamento fonte - requisiti

Nella seguente tabella sono elencati i casi d'uso con i relativi requisiti derivanti da quest'ultimo.

Fonte	**Requisito**
UC-1	• R-1-F-O • R-1.1-F-O
UC-2	• R-2-F-O • R-2.1-F-O • R-2.2-F-O
UC-3	• R-3-F-O
UC-4	• R-4-F-O • R-4.1-F-D • R-4.2-F-D • R-4.3-F-O • R-4.4-F-O • R-4.5-F-O • R-4.6-F-O
UC-5	• R-6-F-O

UC-6	• R-7-F-O
UC-7	• R-8-F-O • R-8.1-F-O
UC-8	• R-9-F-D • R-9.1-F-O • R-9.2-F-O • R-9.3-F-O
UC-8.2	• R-9.4-F-O
UC-9	• R-10-F-O
UC-10	• R-11-F-F • R-11.1-F-F • R-11.2-F-F • R-11.3-F-F • R-11.4-F-F • R-11.4-F-F

UC-11	• R-12-F-F
UC-12	• R-13-F-O
UC-13	• R-14-F-O
UC-14	• R-15-F-O • R-15.1-F-O • R-15.2-F-O
UC-15	• R-16-F-D • R-16.1-F-D • R-16.2-F-D • R-16.3-F-D
UC-17	• R-17-F-D • R-17.1-F-D • R-17.2-F-D

UC-18	• R-18-F-D • R-18.1-F-D • R-18.2-F-D • R-18.3-F-D
Piano di stage	• R-1-V-D, • R-2-V-D, • R-3-V-D, • R-4-V-F, • R-1-Q-O, • R-2-Q-O, • R-3-Q-D, • R-4-Q-O

Tabella 3.4: Tracciamento fonte - requisiti.

3.4.1 Tracciamento requisito - fonte

Nella seguente tabella sono presenti i requisiti individuati nella sezione §3.2, e per ognuno di questi viene indicato il caso d'uso dal quale deriva.

Requisito	**Fonte**
R-1-F-O	UC-1
R-1.1-F-O	UC-1
R-2-F-O	UC-2
R-2.1-F-O	UC-2
R-2.2-F-O	UC-2
R-3-F-O	UC-3
R-4-F-O	UC-4
R-4.1-F-D	UC-4

R-5-F-O	UC-4.1
R-5.1-F-O	UC-4.1
R-5.2-F-O	UC-4.1
R-5.3-F-O	UC-4.1
R-6-F-O	UC-5
R-7-F-O	UC-6
R-8-F-O	UC-7
R-8.1-F-O	UC-7
R-9-F-D	UC-8
R-9.1-F-O	UC-8
R-9.2-F-O	UC-8.1
R-9.3-F-O	UC-8.1
R-9.4-F-O	UC-8.2
R-10-F-O	UC-9
R-11-F-F	UC-10
R-11.1-F-F	UC-10.1
R-11.2-F-F	UC-10.2
R-11.3-F-F	UC-10.3
R-11.4-F-F	UC-10.4
R-12-F-F	UC-11
R-13-F-O	UC-12
R-14-F-O	UC-13
R-15-F-O	UC-14
R-15.1-F-O	UC-14
R-15.2-F-O	UC-14
R-16-F-D	UC-15
R-16.1-F-D	UC-15
R-16.2-F-D	UC-15
R-16.3-F-D	UC-15
R-17-F-D	UC-17
R-17.1-F-D	UC-17
R-17.2-F-D	UC-17
R-18-F-D	UC-18
R-18.1-F-D	UC-18

R-18.2-F-D	UC-18
R-18.3-F-D	UC-18
R-19-V-D	Piano di lavoro
R-20-V-D	Piano di lavoro
R-21-V-D	Piano di lavoro
R-22-V-F	Piano di lavoro
R-23-Q-O	Piano di lavoro
R-24-Q-O	Piano di lavoro

Tabella 3.5: Tracciamento requisiti - fonte.

Capitolo 4

Progettazione e codifica

In questo capitolo vengono presentate le tecnologie, gli strumenti, il ciclo di sviluppo del software e i design pattern utilizzati.

4.1 Tecnologie

Ecco una panoramica delle tecnologie e degli strumenti utilizzati.

Strumento Tecnologia	**Consigliato dall'azienda**	**Motivazioni della scelta**
CFR	Sì	-
ApkTool	Sì	-
Dex2Jar	Sì	-
XPath	No	La scelta di utilizzare XPath è dovuta alle necessità di dover interagire con un documento in formato XML, l'utilizzo di XPath ha semplificato l'estrazione dei dati dal documento AndroidManifest.xml.
Java	No	È stato scelto questo linguaggio di programmazione perché è ben conosciuto dallo stagista, offre una libreria per le interfacce grafiche, ed è portabile. Ciò ha consentito la creazione dello strumento anche per la piattaforma Linux.

Intellij Idea	No	Essendo un IDE molto potente, permette d'interagire con lo strumento di build automation Maven e di effettuare la compilazione del codice sorgente.
Android Emulator	Sì	-
Maven	Sì	-
Astah UML	No	Astah UML è un prodotto software per la realizzazione dei diagrammi UML già conosciuto dallo stagista in quanto è utilizzato in precedenza.
GitLab	Sì	-

Tabella 4.1: Panoramica tecnologie e strumenti utilizzati.

Di seguito, ogni tecnologia/strumento presente nella Tabella 4.1 verrà presentato con maggiori dettagli.

CFR - decompilatore java

Un decompilatore è un software che prende come input un file eseguibile e tenta di creare del codice sorgente ad alto livello che può essere ricompilato successivamente. È lo strumento che effettua l'operazione inversa del compilatore. I decompilatori solitamente non riescono a ricostruire il codice sorgente nella forma originale, ma riescono spesso a ottenere del codice offuscato. Con offuscamento del codice si intende la tecnica che viene applicata al codice sorgente o codice macchina per rendere il codice meno leggibile in modo da rendere l'operazione di reverse engineering più complicata. Nonostante ciò, il decompilatore rimane uno dei più importanti tool utilizzati nell'ambito del reverse engineering del software. CFR è il decompilatore utilizzato per trasformare il codice bytecode *.class* in codice sorgente *.java*; `http://www.benf.org/other/cfr/index.html`

Apk Tool

Apktool è un tool per effettuare il reverse engineering delle applicazioni Android. Può decodificare le risorse contenute nell'APK in una forma quasi uguale a quella

originale e può ricostruire l'APK dopo le modifiche effettuate alle risorse modificate. Nel tool APAT è stato utilizzato per poter decompattare un'APK in modo da ottenere i file che compongono l'APK originale. Apktool permette anche di ricomporre un'apk utilizzando le risorse ottenute precedentemente. https://ibotpeaches.github.io/Apktool/

Figura 4.1: Icona Apktool.

Dex2Jar

Questo strumento permette di trasformare i file *.dex* in formato *.jar* in modo che il contenuto del file possa essere letto attraverso un software di compressione, come 7Zip. https://github.com/pxb1988/dex2jar

XPath

In informatica XPath è un linguaggio, parte della famiglia XML, che permette d'individuare i nodi all'interno di un documento XML. Le espressioni XPath, a differenza delle espressioni XML, non servono a identificare la struttura di un documento, bensì a localizzarne con precisione i nodi. Nell'APAT XPath è stato utilizzato per poter estrapolare alcuni dati necessari per poter effettuare il dump dei dati dall'area di storage dell'app e ottenere le informazioni per poter decompilare solamente nel package dell'applicazione.

Java

In informatica, Java[2] è un linguaggio di programmazione ad alto livello, orientato agli oggetti e tipizzazione statica, che si appoggia sull'omonima piattaforma software di esecuzione, specificamente progettato per essere il più possibile indipendente dal sistema hardware di esecuzione. Nell'APAT è stato utilizzato per la stesura del codice, in particolare, per l'interfaccia grafica è stata utilizzata la libreria di Java, chiamata Swing.

Figura 4.2: Icona Java.

JSON

JSON (JavaScript Object Notation) è un semplice formato per lo scambio di dati. Per le persone è facile da leggere e scrivere, mentre per le macchine risulta facile generare il codice e analizzarne la sintassi.

JSON è basato su due strutture:

- un insieme di coppie nome/valore, in diversi linguaggi, questo è realizzato come un oggetto, un record, uno struct, un dizionario, una tabella hash, un elenco di chiavi o un array associativo.
- un elenco ordinato di valori, nella maggior parte dei linguaggi questo si realizza con un array, un vettore o una sequenza.

Nell'APAT è stato utilizzato per contenere i dati di stato del tool quando viene chiuso e alcuni dati che indicano i percorsi ai tool di terze parti. https://www.json.org/json-it.html

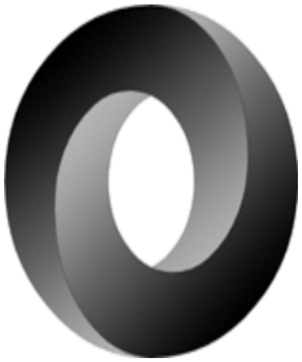


Figura 4.3: Icona JSON.

4.2 Strumenti

Intellij IDEA

Intellij IDEA è l'Integrated Development Environment (IDE) che è stato utilizzato per la scrittura del codice in Java sviluppato da JetBrains. https://www.jetbrains.com/idea/

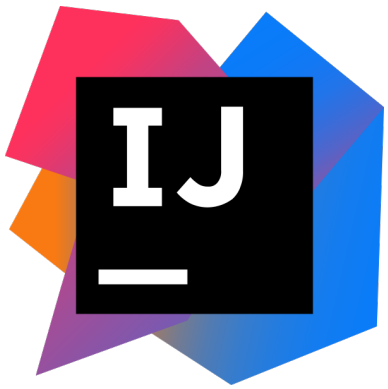


Figura 4.4: Icona Intellij Idea.

Android Emulator

Android emulator è lo strumento che permette di avviare degli emulatori Android, conosciuti anche come AVD. È stato utilizzato per eseguire l'applicazione ricompilata, per poter fare il dump dei dati presenti nell'area di storage dell'applicazione, come i file JSON, XML e SQLite, o come il file delle attività di rete. https://developer.android.com/studio/run/emulator

Figura 4.5: Icona Android Emulator.

Maven

Maven è un tool di build automation, basato sul concetto di Project Object Model (POM). Maven può gestire le build, generare i report e la documentazione da

un singolo centro d'informazione. https://maven.apache.org/

Figura 4.6: Icona Maven.

Nel caso dell'APAT è stato utilizzato per la build del tool, generando in questo modo il file jar.

Astah UML

Astah è uno strumento di modellazione UML. Permette di creare vari diagrammi UML, come quelli di classe, di package, di sequenza e di attività. https://astah.net/products/astah-community/

Figura 4.7: Icona Astah UML.

Git - GitLab

Gitlab è una piattaforma web open-source che permette la gestione di repository e di funzioni d'Issue Tracking System (ITS). Questa è un'istanza privata dell'azienda Imola Informatica S.p.A.. https://git.imolinfo.it/users/sign_in

Figura 4.8: Icona GitLab.

4.3 Ciclo di vita del software

Il modello adottato per lo sviluppo del progetto è il modello incrementale[14]. Esso permette di suddividere la durata del progetto in diversi incrementi, in ognuno dei quali sono stati definiti degli obiettivi/requisiti da raggiungere. Inoltre, tale modello di sviluppo offre i seguenti vantaggi:

- l'assegnazione delle diverse priorità agli obiettivi da raggiungere, in modo che quelli più importanti vengano raggiunti prima;
- la dimostrazione del prodotto al termine di ogni incremento in questo modo si assicura che il risultato di ogni incremento sia consono alle aspettative del cliente;
- la verifica del prodotto avviene a ogni incremento in modo da garantire la correttezza del prodotto di ogni incremento;
- in ogni incremento vengono aggiunte delle funzionalità corrette e funzionanti; in questo modo si ha la garanzia che il prodotto finale soddisferà i requisiti del cliente.

4.4 Progettazione

4.4.1 Package it.imolinfo.apat

Questo è il root package del tool, contiene altri package e le due classi elencate successivamente.

ApatLauncher: è la classe di launcher del tool, si occupa principalmente di mettere in relazione di observed-observer i componenti della vista con i dati del modello;

Utils: la classe di utilities che ha delle funzioni statiche utilizzate nel tool per evitare la ripetizione del codice.

4.4.2 Package it.imolinfo.apat.controller

Questo è il package che contiene la classe Controller e ciò che riguarda l'interazione del controller con il filesystem.

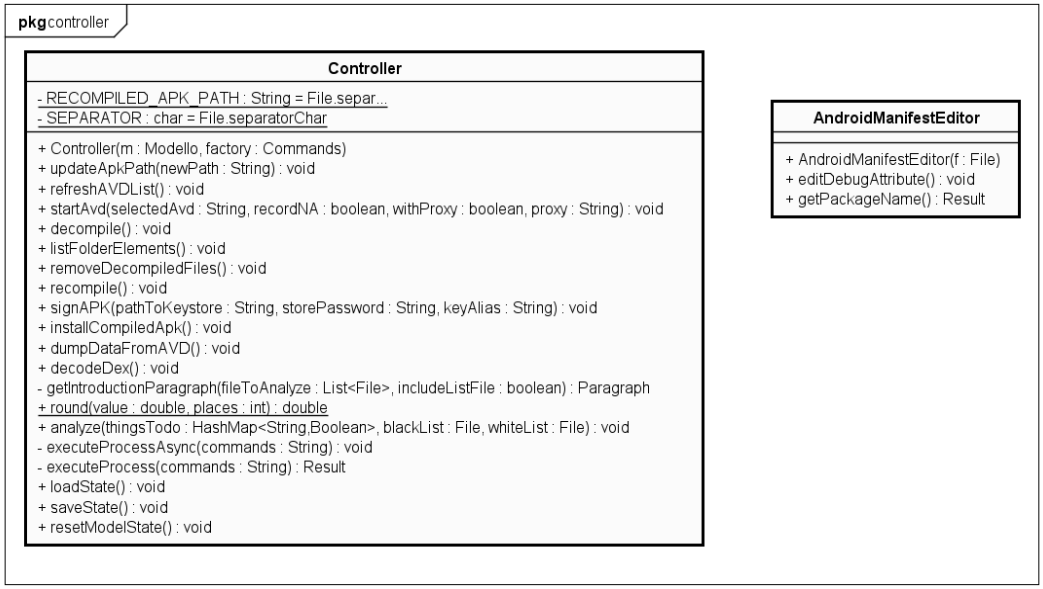


Figura 4.9: Package Controller.

AndroidManifestEditor.java: è la classe che si occupa, dato un file di tipo AndroidManifest, di modificare e/o aggiungere il tag *debuggable* impostando il suo valore a *true*;

Controller: è il controller del MVC, si occupa di effettuare le operazioni di decompilazione, ricompilazione, decodifica e analisi dei file;

4.4.3 Package it.imolinfo.apat.model

Questo package contiene le classi che riguardano il modello del pattern Model View Controller.

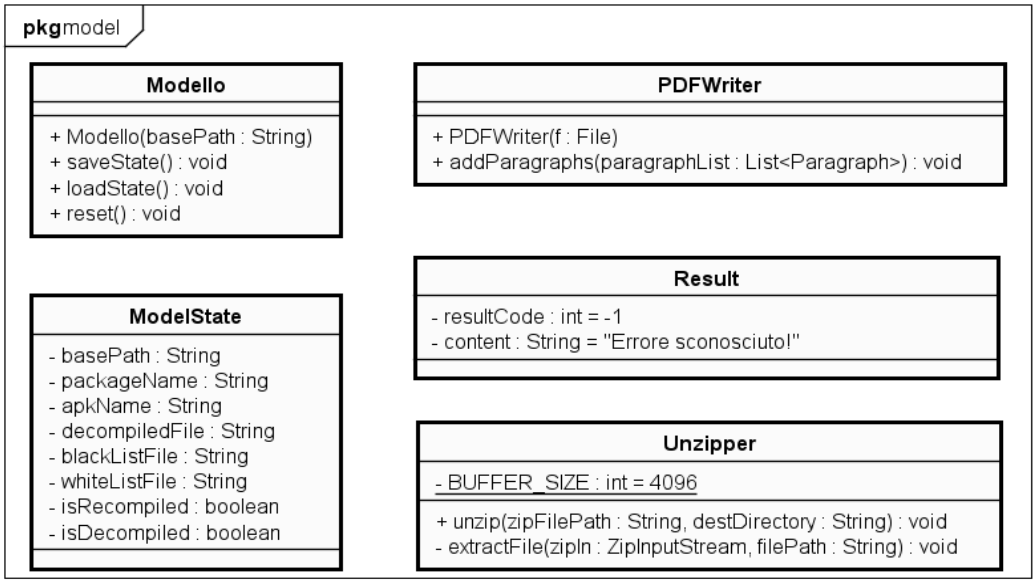


Figura 4.10: Package Model.

Modello: la classe che contiene i dati utili al corretto funzionamento del tool;

ModelState: è la classe che viene usata per poter salvare lo stato del funzionamento del tool. Lo stato viene salvato quando viene chiuso il tool, e viene ricaricato al successivo avvio;

PDFWriter: è la classe wrapper che si occupa della creazione del file PDF per salvare i risultati dell'analisi;

Result: è la classe di messaggio che viene restituita quando il tool interagisce con il file system;

Unzipper: è la classe si occupa di decomprimere i file .zip;

Dumper: è la classe che si occupa di effettuare il dump dei dati da un file con estensione *db*;

4.4.4 Package it.imolinfo.apat.pattern

Questo package contiene dei sotto-package ognuno dei quali rappresenta un design pattern utilizzato nello sviluppo del tool.
Il package analyzer è composto dalle seguenti classi:

analyzer.Analyze: è l'interfaccia di base del pattern Decorator.

analyzer.BaseAnalyzer: è la classe dell'oggetto base che viene decorato.

analyzer.BaseAnalyzeDecorator: è la classe astratta del decorator di base che implementa l'interfaccia Analyze, e ha un metodo astratto *doAnalysis()* che deve essere implementato dai decorator concreti.

analyzer.DumpDataBase: è il decorator che si occupa di estrarre i contenuti dei file *.db* scaricati dall'area di storage dell'app.

analyzer.DumpedFilesAnalyzer: è il decorator che si occupa di analizzare i file dell'area di storage dell'applicazione con estensione *XML* e *JSON*. Principalmente, legge il contenuto di tali file, e in base ad una whitelist, seleziona quali risultati restituire al chiamante.

analyzer.LambdaCounter: è il decorator che conta il numero di lambda presenti per ogni classe di codice *.java* decompilato.

analyzer.StringFinder: è il decorator che analizza i file di tipo *.java*, ed estrae le stringhe hardcoded, può essere utilizzato insieme a un blacklist delle stringhe che devono essere ignorate.

Il package observer è composto da seguenti classi:

observer.Observable: la classe parametrizzata T che può essere osservata;

observer.Observer: l'interfaccia parametrizzata che ha il ruolo dell'observer;

Il package factory method è composto da seguenti classi:

cliCommandFactory.Commands: è l'interfaccia che contiene i metodi, dove ognuno dei quali deve generare delle istruzioni per la linea di comando.

cliCommandFactory.CommandFactory: è la classe astratta che implementa la precedente interfaccia, con un costruttore di default che richiede un path di base.

cliCommandFactory.UnixCommandFactory: è l'implementazione della classe astratta CommandFactory per il sistema UNIX.

cliCommandFactory.WindowsCommandFactory: è l'implementazione della classe astratta CommandFactory per il sistema Windows.

4.4.5 Package it.imolinfo.apat.view

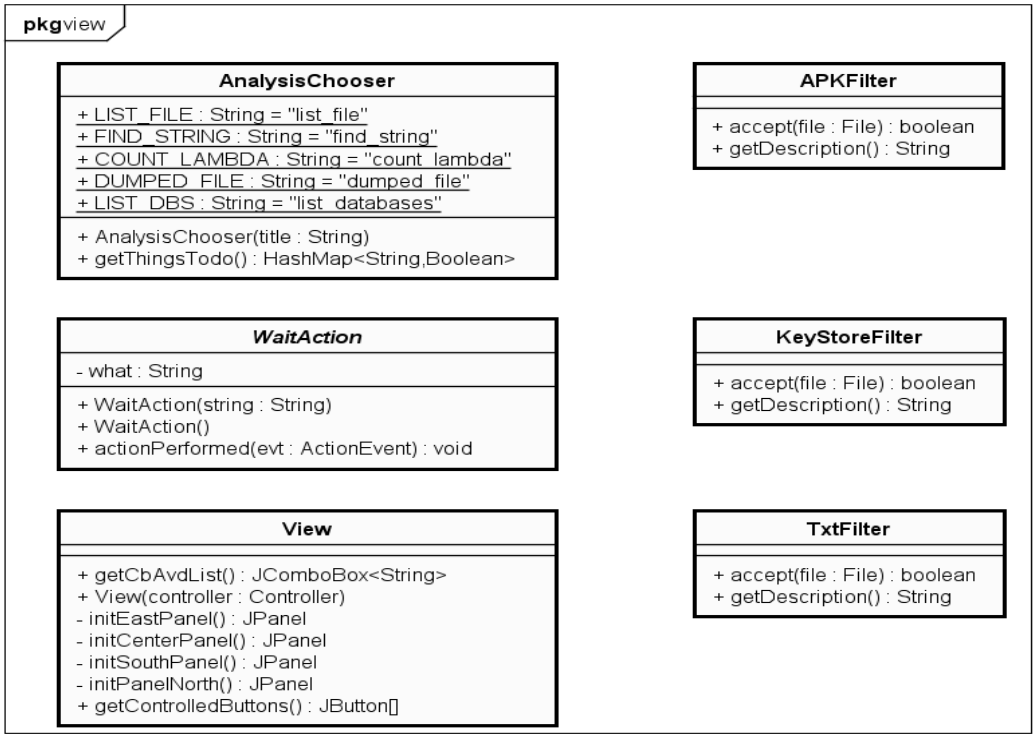


Figura 4.11: Package View.

AnalysisChooser: è la finestra che mostra le opzioni di analisi.

View: è la finestra principale, dove permette di selezionare il file *apk* da decompilare ed analizzare.

WaitAction: è una classe astratta che a sua volta eredita dalla classe Action e permette di eseguire un processo mostrando la barra del caricamento.

ApkFilter: è l'implementazione dell'interfaccia **FileFilter** che accetta solo i file di tipo *APK*.

KeyStoreFilter: è l'implementazione dell'interfaccia **FileFilter** che accetta solo i file di tipo *JKS*.

TextFilter: è l'implementazione dell'interfaccia **FileFilter** che accetta solo i file di tipo *txt*.

4.5 Design Pattern utilizzati

Di seguito, sono elencati i design pattern utilizzati durante la creazione del tool. Per ognuno di questi, verrà fornito il diagramma delle classi e una descrizione.

4.5.1 Model View Controller

Come pattern architetturale è stato adottato il pattern Model-View-Controller, poiché è in grado di separare la logica di presentazione dei dati dalla logica di business.
L'applicazione di tale pattern avviene attraverso il pattern Observer. Più nello specifico, le singole componenti della vista osservano i valori della classe Modello, in questo modo, quando il valore di un determinato attributo viene modificato, il nuovo valore viene notificato alla componente della vista.

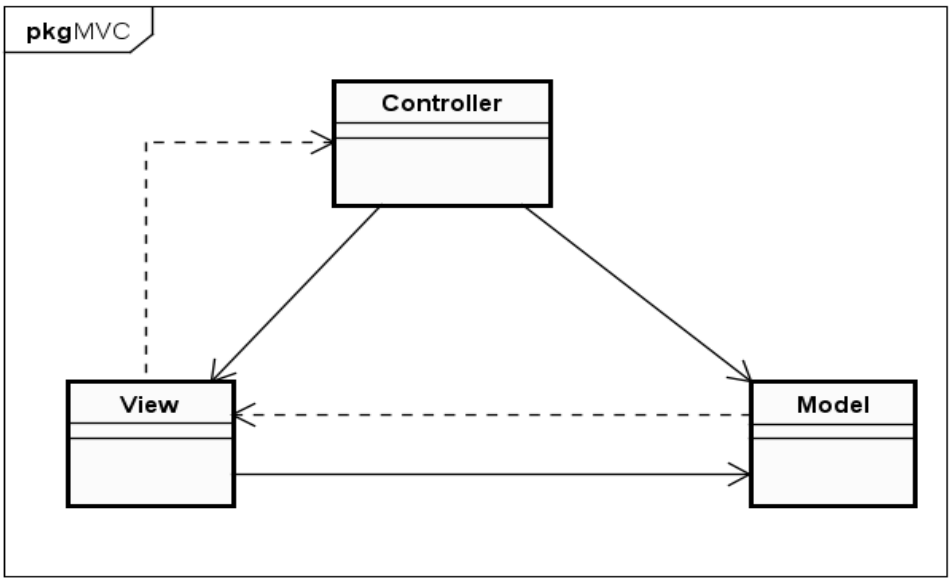


Figura 4.12: Diagramma delle classi del pattern MVC.

4.5.2 Observer

Il pattern observer[7] permette di definire una dipendenza uno a molti fra gli oggetti, in modo tale che se un oggetto cambia il proprio stato, tutti gli oggetti dipendenti da questo siano notificati e aggiornati automaticamente. L'implementazione di tale pattern avviene attraverso una classe Observable e un'interfaccia Observer entrambe template. Per modificare il valore dell'observable è sufficiente utilizzare il metodo setValue(newValue:T): in questo modo, su ogni elemento della lista observers viene invocato il metodo onChanged passando il nuovo valore. Tale pattern viene applicato nel tool perché i componenti delle viste possano osservare i contenuti del modello.

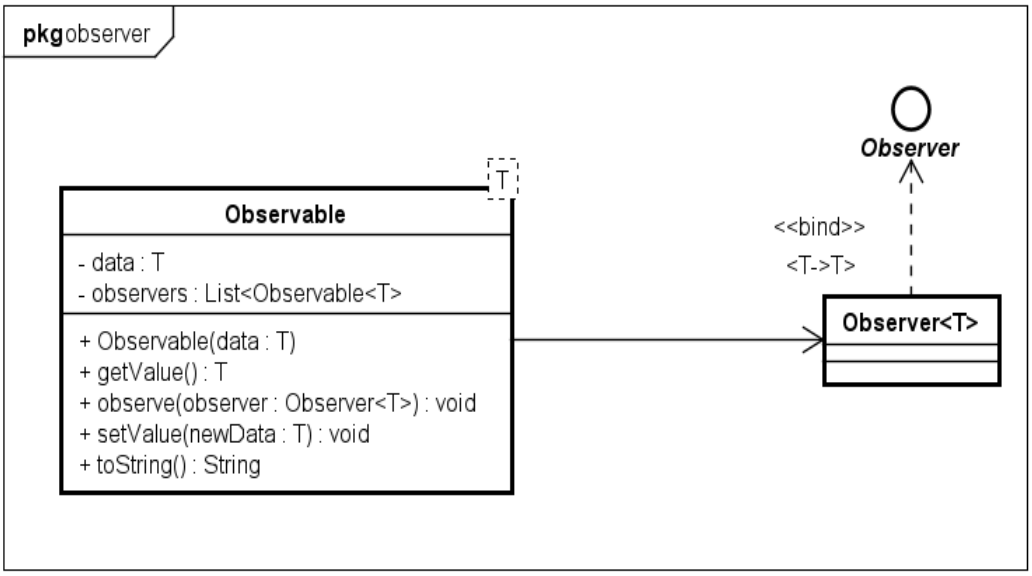


Figura 4.13: Diagramma delle classi del pattern Observer.

4.5.3 Decorator

Il pattern decorator[6] è stato utilizzato per creare la parte del tool che si occupa di effettuare l'analisi del codice decompilato e dei file scaricati dall'area di storage dell'applicazione presenti nell'Android Emulator Device.
Il vantaggio ottenuto per aver utilizzato questo pattern è quello di poter "decorare" l'oggetto base di analisi, ovvero l'istanza della classe *BaseAnalyzer* con i vari decorator. Un altro importante vantaggio è quello di poter aggiungere altre funzionalità di analisi facilmente nel tool. È, infatti, sufficiente creare un'altra sottoclasse della classe *BaseAnalyzeDecorator* quindi implementare il metodo astratto corrispondente.

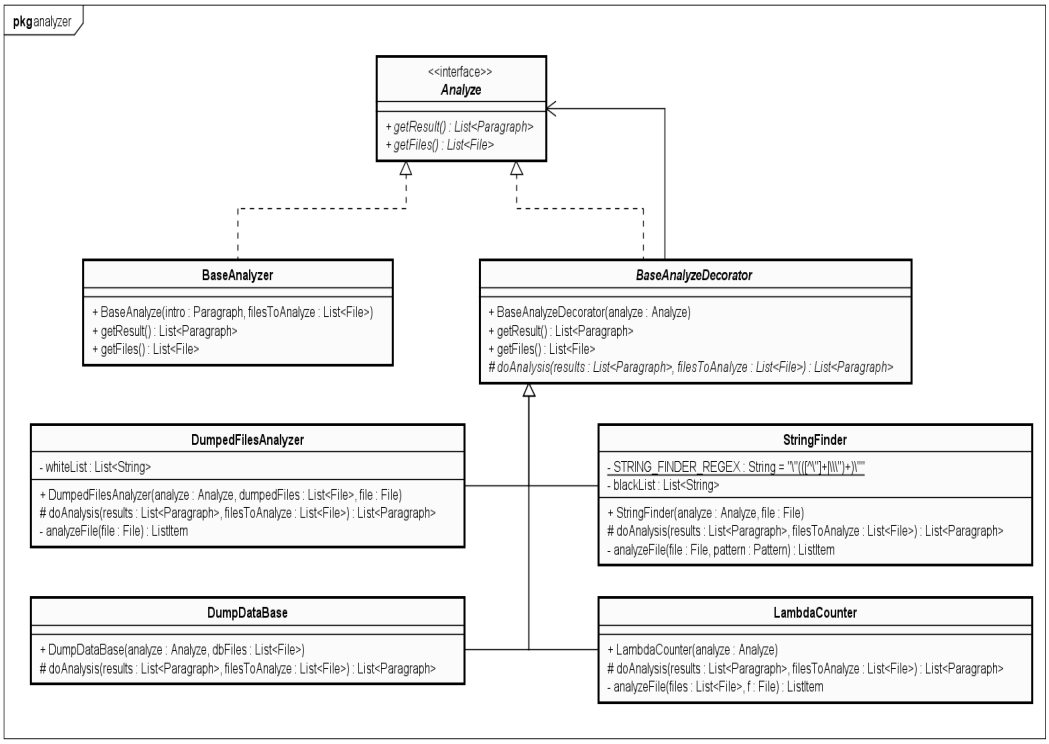


Figura 4.14: Diagramma delle classi del pattern Decorator.

4.5.4 Factory Method

Il pattern Factory Method[5] è stato utilizzato per poter fornire al controller dei comandi da eseguire nei processi, essendo i comandi dipendente dal sistema operativo. Sono state create due classi, una per Windows chiamata *WindowsCommandFactory* e una per Linux chiamata *LinuxCommandFactory*. Allo start-up del tool, viene letta dalla JVM una variabile chiamata *os.name*, e dipendentemente da valore letto, viene deciso quale delle sottoclassi istanziare per il corretto funzionamento del tool.

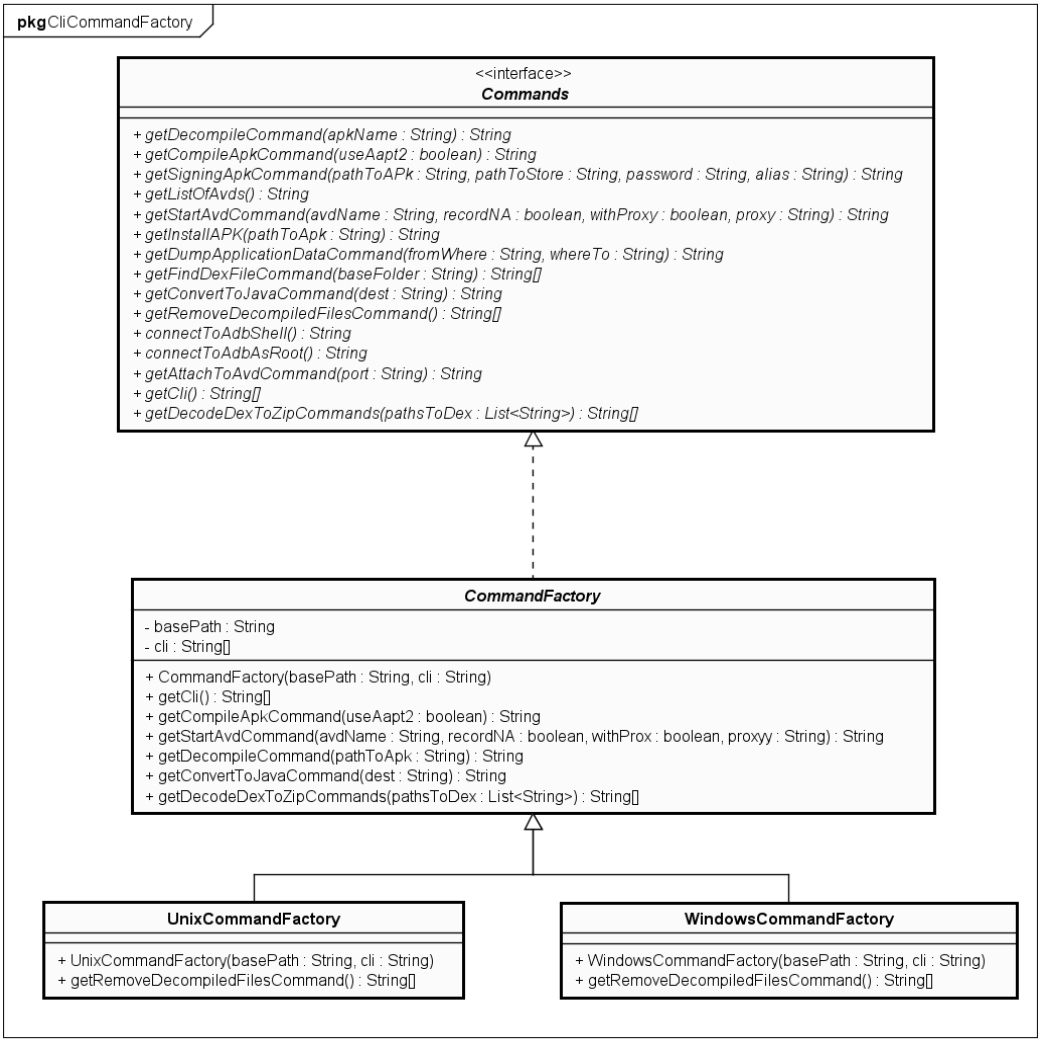


Figura 4.15: Diagramma delle classi del pattern Factory Method.

4.6 Codifica

Per la fase di codifica sono state adottate le seguenti convenzioni:

- il nome delle classi seguirà la notazione a CamelCase, un esempio corretto è: *WindowsCommandFactory*;
- il nome degli attributi seguirà sempre la notazione a CamelCase, ma con la prima lettera in minuscolo, un esempio corretto è: *commandsFactory*;
- ove possibile, si devono utilizzare le lambda espressioni;
- la gestione delle eccezioni deve essere fatta sempre dal chiamante;
- per ogni classe e ogni metodo deve essere fornita una descrizione in JavaDoc.

4.6.1 Inizializzazione View

Il seguente segmento di codice è necessario per creare la relazione di osservazione che esiste tra la View e il Model. Nella View è presente un'etichetta che mostra il file APK selezionato da decompilare. Questa componente della view osserva l'attributo apkName del modello, e una volta selezionato il file, viene ottenuto il path del file APK e aggiornato il modello, l'etichetta riceverà la notifica con il path aggiornato.

Codice 4.1: Inizializzazione relazione di osservazione View e Model

```
model.getApkName().observe(newValue -> {
    view.getPackageInfo().setText(newValue);
    view.getBtnStartDecompile().setEnabled(!newValue.
        equalsIgnoreCase(""));
});
model.getAvdList().observe(lis -> view.getCbAvdList().setModel
   (new DefaultComboBoxModel<>(lis)));
model.getIsDecompiled().observe(newValue -> {
    JButton[] controlledButtons = view.getControlledButtons();
    for (JButton controlledButton : controlledButtons)
        controlledButton.setEnabled(newValue);
});
```

Nella View è presente un'eticchetta che mostra il file APK selezionato da decompilare, questa componente della view osserva l'attributo *apkName* del modello, e una volta selezionato il file, viene ottenuto il path del file APK e aggiornato il modello; quindi l'etichetta riceverà la notifica con il path aggiornato.

4.6.2 Analisi del codice e generazione del pdf

Dopo aver decompilato un file *.apk*, e decodificato i file *dex* l'utente può scegliere di effettuare l'operazione di analisi offerta dal tool. Per questa funzionalità si deve istanziare un oggetto di tipo BaseAnalyzer, e in base alle opzioni di analisi che l'utente ha selezionato dalla vista, decorare l'oggetto base.

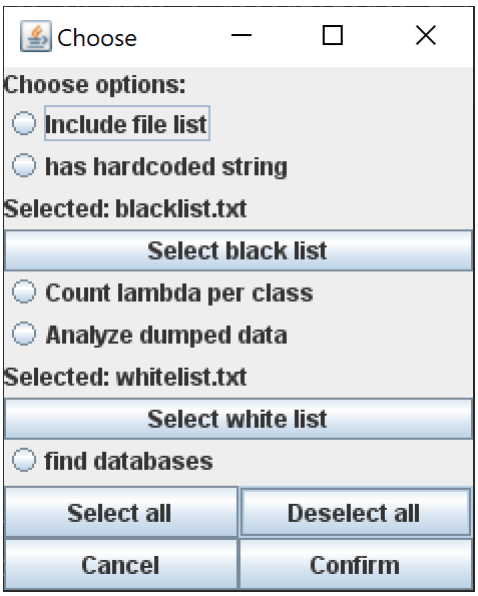


Figura 4.16: Schermata delle opzioni dell'analisi

Di seguito è presente il codice che si occupa di "decorare" l'oggetto base, effettuare l'analisi e quindi generare un file pdf contenente il risultato:

Codice 4.2: Decorator

```
public void analyze(Map< String, Boolean > thingsTodo, File
    blackList, File whiteList) {
Analyze analyze = new BaseAnalyzer(thingsTodo.get(
    AnalysisChooser.LIST_FILE), decompiledFiles);
...
// se analyzeString==true, allora analizzo le stringhe
if (thingsTodo.get(AnalysisChooser.FIND_STRING))
     analyze = new StringFinder(analyze, blackList);
if (thingsTodo.get(AnalysisChooser.COUNT_LAMBDA))
     analyze = new LambdaCounter(analyze);
if (thingsTodo.get(AnalysisChooser.DUMPED_FILE))
     analyze = new DumpedFilesAnalyzer(analyze, whiteList);
if (thingsTodo.get(AnalysisChooser.LIST_DBS))
     analyze = new DumpDataBase(analyze);
```

```
PDFWriter pdfWriter = new PDFWriter(resultFile);
pdfWriter.addParagraphs(analyze.getResult());
}
```

Con l'utilizzo del pattern decorator è estremamente semplice aggiungere un nuovo tipo di analisi, poiché è sufficiente aggiungere nella map *thingsToDo* al momento della chiamata del metodo *analyze*, una nuova classe che eredita dalla classe base dei decorator implementandone il metodo astratto e aggiungere due righe d'istruzione per decorare l'oggetto base con il decorator nuovo.

4.6.3 Istanziazione di Commands

Per il corretto funzionamento del tool è necessario che all'avvio venga instanziata una delle seguenti classi:

- WindowsCommandFactory;
- LinuxCommandFactory.

Il seguente segmento di codice risolve tale problema:

Codice 4.3: Creazione dell'istanza di Commands in base al S.O.

```
Commands commands;
String osInfo = System.getProperty("os.name");
if (containsIgnoreCase(osInfo, "windows")) {
    commands = new WindowsCommandFactory(model.getBasePath().
       getValue(), "cmd.exe", "/c");
} else {
    commands = new UnixCommandFactory(model.getBasePath().
       getValue(), "bash", "-c");
}
Controller controller = new Controller(model, commands);
```

Capitolo 5

Verifica e validazione

In questa sezione vengono presentate le varie attività di verifica e validazione per garantire la qualità del tool. Le attività svolte sono analisi statica, test unitari, test d'integrazione, test di sistema e di validazione.

5.1 Analisi statica

L'analisi statica del codice[1] è l'analisi del software che è effettuata senza eseguire il codice. Si pone in contrasto con l'analisi dinamica che invece richiede l'esecuzione del programma. Il termine è spesso utilizzato per indicare l'analisi eseguita da tool automatici che nel caso del coinvolgimento dell'essere umano diventa code review. Essa può essere di due tipi, walkthrough e inspection. Il tool utilizzato per l'analisi statica automatizzata è un plugin di Maven chiamato checkstyle[3] che è capace di generare un report indicando la presenza o meno del codice che non è conforme alle convenzioni definite dal programmatore.

In questo progetto sono state definite le seguenti regole:

- complessità delle espressioni booleane: al massimo 3;
- complessità ciclomatica delle funzioni: al massimo 16;
- lunghezza massima di ogni riga: 150 caratteri;
- lunghezza massima di ogni metodo: 100 righe;
- lunghezza massima di ogni file: 1000 righe;
- presenza dei blocchi di catch vuoi: vietata.

Il risultato dell'analisi è stato positivo, poiché non presenta nessuna riga del codice che non rispetti le regole sopracitate.

5.2 Test unitari

In ingegneria del software, per test d'unità si intende l'attività di testing delle singole unità del software. Per unità si intende normalmente il componente più piccolo con funzionamento autonomo. Dipendentemente dal tipo di linguaggio di programmazione, l'unità può essere una funzione, una classe o un metodo. Come le altre forme di test, i test d'unità possono essere completamente manuali o automatici. Specialmente nel caso dello unit testing automatico, lo sviluppo dei test case può essere considerato parte integrante dell'attività di sviluppo.
Nel caso del progetto Android Package Analysis Tool (APAT) i test unitari sono stati fatti utilizzando il framework JUnit4[10] integrato con il tool di build automation.

5.2.1 Specifica dei test

Di seguito sono riportati i test d'unità che verificano il corretto funzionamento delle singole unità.

ID	Descrizione	Stato[1]
TU-1	Verificare che il file AndroidManifest.xml venga modificato correttamente.	I
TU-2	Verificare che dal file AndroidManifest.xml venga estratto il package corretto.	I
TU-3	Verificare che il nuovo path dell'APK venga aggiornato correttamente.	I
TU-4	Verificare che la lista delle AVD venga ottenuta correttamente.	I
TU-5	Verificare che l'AVD venga avviato con i parametri corretti.	I
TU-6	Verificare che i file dex vengano decompilati correttamente.	I
TU-7	Verificare che l'elenco dei file presenti nella cartella ./tmp sia corretto.	I
TU-8	Verificare che i file presenti nella cartella ./tmp vengano rimossi correttamente.	I
TU-9	Verificare che la ricompilazione dell'APK avvenga correttamente.	I

[1] I: Implementato, NI: Non Implementato

TU-10	Verificare che il signing dell'APK ricompilato avvenga correttamente.	I
TU-11	Verificare che l'apk venga installato correttamente nell'AVD.	I
TU-12	Verificare che i dati scaricati dall'AVD siano quelli presenti nell'areas di storage dell'applicazione.	I
TU-13	Verificare che i file DEX vengano decompilati correttamente.	I
TU-14	Verificare che il pdf generato sia corretto.	I
TU-15	Verificare che lo stato del tool caricato sia corretto.	I
TU-16	Verificare che lo stato del tool venga salvato correttamente.	I
TU-17	Verificare che lo stato del tool venga resettato correttamente.	I
TU-18	Verificare che i paragrafi vengano inseriti nel pdf correttamente.	I
TU-19	Verificare che il file zip venga estratto correttamente.	I
TU-20	Verificare che il paragrafo generato sia corretto rispetto all'elenco dei file dato.	I
TU-21	Verificare che il paragrafo generato sia corretto rispetto all'elenco dei file dato.	I
TU-22	Verificare che il paragrafo generato sia corretto rispetto all'elenco dei file dato.	I
TU-23	Verificare che il paragrafo generato sia corretto rispetto all'elenco dei file dato.	I
TU-24	Verificare che il comando generato per decompilare l'APK sia corretto.	I
TU-25	Verificare che il comando generato per ricompilare l'APK sia corretto.	I
TU-26	Verificare che il comando generato per firmare l'APK ricompilato sia corretto.	I
TU-27	Verificare che il comando generato per ottenere l'elenco delle AVD sia corretto.	I

TU-28	Verificare che il comando generato per avviare l'AVD sia corretto.	I
TU-29	Verificare che il comando generato per installare l'APK sull'AVD sia corretto.	I
TU-30	Verificare che il comando generato per scaricare l'area di storage dell'app installato sia corretto.	I
TU-31	Verificare che il comando generato per convertire i file .class in .java sia corretto.	I
TU-32	Verificare che il comando generato per rimuovere i file decompilati sia corretto.	I
TU-33	Verificare che il comando generato per connettersi all'AVD come root sia corretto.	I
TU-34	Verificare che il comando generato per connettersi all'AVD normalmente sia corretto.	I
TU-35	Verificare che il comando generato per ottenere i parametri del terminale sia corretto.	I
TU-36	Verificare che la mappa restituita contenga esattamente le scelte dell'utente.	I
TU-37	Verificare che vengano accettati solo i file di tipo APK.	I
TU-38	Verificare che la descrizione restituita sia corretta.	I
TU-39	Verificare che vengano accettati solo i file di tipo JKS.	I
TU-40	Verificare che la descrizione restituita sia corretta.	I
TU-41	Verificare che vengano accettati solo i file di tipo TXT.	I
TU-42	Verificare che la descrizione restituita sia corretta.	I
TU-43	Verificare che la stringa restituita sia corretta.	I
TU-44	Verificare che l'elenco dei file ottenuto sia corretto.	I
TU-45	Verificare che l'arrotondamento avviene correttamente.	I

Tabella 5.1: Test d'unità.

5.2.2 Tracciamento

La seguente tabella associa ogni singolo test unitario all'unità verificata.

ID	Componente
TU-1	AndroidManifest.editDebugAttribute()
TU-2	AndroidManifest.getPackageName()
TU-3	Controller.updateApkPath()
TU-4	Controller.refreshAVDList()
TU-5	Controller.startAvd()
TU-6	Controller.decompile()
TU-7	Controller.listFolderElements()
TU-8	Controller.removeDecompiledFiles()
TU-9	Controller.recompile()
TU-10	Controller.signAPK()
TU-11	Controller.installCompiledApk()
TU-12	Controller.dumpDataFromAVD()
TU-13	Controller.decodeDex()
TU-14	Controller.analyze()
TU-15	Controller.loadState()
TU-16	Controller.saveState()
TU-17	Controller.resetModelState()
TU-18	PDFWriter.addParagraphs()
TU-19	Unzipper.unzip()
TU-20	DumpDataBase.doAnalysis()
TU-21	DumpedFilesAnalyzer.doAnalysis()
TU-22	LambdaCounter.doAnalysis()
TU-23	StringFinder.doAnalysis()
TU-24	Commands.getDecompileCommand()
TU-25	Commands.getCompileApkCommand()
TU-26	Commands.getSigningApkCommand()
TU-27	Commands.getListOfAvds()
TU-28	Commands.getStartAvdCommand()
TU-29	Commands.getInstallAPK()
TU-30	Commands.getDumpApplicationDataCommand()
TU-31	Commands.getConvertToJavaCommand()

TU-32	Commands.getRemoveDecompiledFilesCommand()
TU-33	Commands.connectToAdbAsRoot()
TU-34	Commands.getAttachToAvdCommand()
TU-35	Commands.getCli()
TU-36	Analysis.chooser()
TU-37	APKFilter.accept()
TU-38	APKFilter.getDescription()
TU-39	KeystoreFilter.accept()
TU-40	KeystoreFilter.getDescription()
TU-41	TxtFilter.accept()
TU-42	TxtFilter.getDescription()
TU-43	Utils.getDate()
TU-44	Utils.listAllFiles()
TU-45	Utils.round()

Tabella 5.2: Tracciamento dei test d'unità.

I test sopraelencati sono stati superati con successo, con una code coverage del *84.2%*.

5.3 Test d'integrazione

In ingegneria del software con test d'integrazione si indicano i test di livello intermedio e abitualmente seguono temporalmente quelli di unità e precedono quelli di sistema. Infatti, tali test vengono eseguiti quando due o più unità già testate vengono aggregate in una struttura più grande, rappresentando l'estensione logica del test di unità.

5.3.1 Specifica dei test

Di seguito sono riportati i test d'integrazione che verificano il corretto funzionamento tra le unità.

ID	Descrizione	Stato[2]
TI-1	Verificare che la view riesca a ottenere correttamente la lista delle AVD presenti nel sistema dal modello.	I
TI-2	Verificare che la view riesca a ottenere correttamente il path di base dal modello.	I
TI-3	Verificare che la view riesca a ottenere correttamente il percorso verso il file APK da decompilare dal modello.	I
TI-4	Verificare che la view riesca a ottenere correttamente l'elenco dei file presenti nella cartella tmp dal modello.	I
TI-5	Verificare che la view riesca a ottenere correttamente il percorso verso il file whitelist dal modello.	I
TI-6	Verificare che la view riesca a ottenere correttamente il percorso verso il file blacklist dal modello.	I
TI-7	Verificare che la view riesca a ottenere correttamente le informazioni di proxy dal modello.	I
TI-8	Verificare che il controller riesca a ottenere correttamente lo stato *isRecompiled* dal modello.	I

[2]I: Implementato, NI: Non Implementato

TI-9	Verificare che il controller riesca a ottenere correttamente lo stato *isDecompiled* dal modello.	I
TI-10	Verificare che il controller riesca a ottenere correttamente lo stato *isDecoded* dal modello.	I
TI-11	Verificare che il controller riesca a ottenere correttamente il path di base dal modello.	I
TI-12	Verificare che il controller riesca a ottenere correttamente il percorso verso il file APK da decompilare dal modello.	I
TI-13	Verificare che il controller riesca a ottenere correttamente l'elenco dei file presenti nella cartella tmp dal modello.	I
TI-14	Verificare che il controller riesca a ottenere correttamente il percorso verso il file whitelist dal modello.	I
TI-15	Verificare che il controller riesca a ottenere correttamente il percorso verso il file blacklist dal modello.	I
TI-16	Verificare che il controller riesca a ottenere correttamente le informazioni di proxy dal modello.	I
TI-17	Verificare che il controller riesca a ottenere correttamente i comandi corretti dal CommandFactory.	I
TI-18	Verificare che il controller riesca a eseguire correttamente i comandi ottenuti dal CommandFactory.	I
TI-19	Verificare che il controller riesca a ottenere correttamente i risultati dal LambdaFinder.	I
TI-20	Verificare che il controller riesca a ottenere correttamente i risultati dal StringFinder.	I
TI-21	Verificare che il controller riesca a ottenere correttamente i risultati dal DumpDataBase.	I
TI-22	Verificare che il controller riesca a ottenere correttamente i risultati dal DumpedFileAnalyzer.	I

TI-23	Verificare che il controller riesca a ottenere correttamente i risultati finali dal BaseAnalyzer.	I
TI-24	Verificare che il controller riesca a ottenere correttamente il path dell'APK dalla view.	I
TI-25	Verificare che il controller riesca a ottenere correttamente le opzioni per avviare AVD dalla view.	I
TI-26	Verificare che il controller riesca a ottenere correttamente le opzioni per effettuare il signing dell'APK dalla view.	I
TI-27	Verificare che il controller riesca a ottenere correttamente i path dei package da decodificare in java dalla view.	I
TI-28	Verificare che il controller riesca a ottenere correttamente la mappa delle analisi da effettuare dalla view.	I
TI-29	Verificare che il controller riesca a ottenere correttamente la blacklist selezionato per effettuare l'analisi dalla view.	I
TI-30	Verificare che il controller riesca a ottenere correttamente la whitelist selezionato per effettuare l'analisi dalla view.	I
TI-31	Verificare che il modello riesca a ricaricare correttamente i dati di state dal sistema.	I
TI-32	Verificare che il modello riesca a salvare correttamente i dati di state dal sistema.	I
TI-33	Verificare che il tool riesca a ottenere il path verso CFR dal sistema.	I
TI-34	Verificare che il tool riesca a ottenere il path verso APKTool dal sistema.	I
TI-35	Verificare che il tool riesca a ottenere il path verso Jarsigner dal sistema.	I

Tabella 5.3: Test d'integrazione.

I test d'integrazione sono superati effettuando tutte le operazioni che coinvol-

gono le due parti interessati, il totale corretto funzionamento del tool, dimostra il superamento dei test d'integrazione.

5.4 Test di sistema

In ingegneria del software il test di sistema è un procedimento, parte del ciclo di vita del software, utilizzato per individuare le carenze di correttezza, completezza e affidabilità delle componenti software in corso di sviluppo. Consiste nell'esecuzione del software da parte del collaudatore per verificare le funzionalità offerte del software in relazione con i requisiti.

5.4.1 Specifica dei test

Di seguito sono riportati i test di sistema che verificano il corretto funzionamento del sistema.

ID	Requisito	Descrizione	Stato[3]
TS-1	R-1-F-O	Verificare che il tool permetta di selezionare solo i file APK.	I
TS-2	R-1.1-F-O	Verificare che venga mostrato un messaggio di errore quando il file selezionato non è APK.	I
TS-3	R-2-F-O	Verificare che il file APK venga decompilato correttamente.	I
TS-4	R-2.1-F-O	Verificare che venga mostrato un messaggio quando l'APK è stato decompilato correttamente.	I
TS-5	R-2.2-F-O	Verificare che nel file AndroidManifest.xml venga aggiunto l'attributo debuggable correttamente.	I
TS-6	R-3-F-O	Verificare che venga mostrato il messaggio d'errore quando la decompilazione non è andato a buon fine.	I
TS-7	R-4-F-O	Verificare che l'applicazione ricompilato venga installata correttamente nell'AVD selezionato.	I

[3]I: Implementato, NI: Non Implementato

TS-8	R-4.1-F-D	Verificare che l'APK venga ricompilata correttamente.	I
TS-9	R-4.2-F-D	Verificare che venga avviata l'applicazione.	I
TS-10	R-4.3-F-O	Verificare che il tool permetta di selezionare un AVD presente nel sistema.	I
TS-11	R-4.4-F-O	Verificare che venga visualizzata l'elenco degli AVD presenti nel sistema.	I
TS-12	R-4.5-F-O	Verificare che il tool permetta di confermare la selezione dell'AVD.	I
TS-13	R-5-F-O	Verificare che venga mostrato un messaggio quando non sono stati rilevati nessun AVD nel sistema.	I
TS-14	R-6-F-O	Verificare che deve mostrare un messaggio d'errore quando l'AVD non è stato avviato correttamente.	I
TS-15	R-7-F-O	Verificare che vengano effettivamente scaricati i file dall'area di storage dell'applicazione.	I
TS-16	R-7.1-F-O	Verificare che sia permesso all'utente di selezionare il path dove collocare i file scaricati dall'area di storage.	I
TS-17	R-8-F-D	Verificare che vengano decodificati i file dex.	I
TS-18	R-8.1-F-O	Verificare che venga visualizzato il messaggio quando la decodifica dei file dex è avvenuta con successo.	I
TS-19	R-8.2-F-O	Verificare che venga fatta selezionare i file dex da decodificare.	I
TS-20	R-8.3-F-O	Verificare che confermata la selezione dei file dex.	I
TS-21	R-8.4-F-O	Verificare che sia permessa la selezione dei path di destinazione dei file decodificati.	I

TS-22	R-9-F-O	Verificare che venga mostrato il messaggio d'errore quando la decodificato dei file dex non è andato a buon fine.	I
TS-23	R-10-F-F	Verificare che il tool permetta di firmare l'apk ricompilato.	I
TS-24	R-10.1-F-F	Verificare che il tool permetta di selezionare un file di tipo keystore.	I
TS-25	R-10.2-F-F	Verificare che venga mostrato un messaggio d'errore quando il file selezionato non è di tipo keystore.	I
TS-26	R-10.3-F-F	Verificare che il tool permetta l'inserimento dell'alias della chiave da utilizzare.	I
TS-27	R-10.4-F-F	Verificare che il tool permetta l'inserimento della password del keystore da utilizzare.	I
TS-28	R-11-F-F	Verificare che venga mostrato il messaggio d'errore quando i dati inseriti non sono corretti.	I
TS-29	R-12-F-O	Verificare che venga mostrato un messaggio quando il file selezionato non è valido.	I
TS-30	R-13-F-O	Verificare che il tool permetta di decodificare i file dex in .java.	I
TS-31	R-14-F-O	Verificare che il tool permetta di effettuare l'analisi del codice java.	I
TS-32	R-14.1-F-O	Verificare che il tool permetta di selezionare diversi tipi di analisi.	I
TS-33	R-14.2-F-O	Verificare che il tool permetta di selezionare la destinazione per il file di report.	I
TS-34	R-15-F-D	Verificare che il tool permetta di avviare un AVD presente nel sistema.	I
TS-35	R-15.1-F-D	Verificare che il tool permetta di selezionare l'AVD presente nel sistema.	I

TS-36	R-15.2-F-D	Verificare che il tool permetta di visualizzare l'elenco degli AVD presenti nel sistema.	I
TS-37	R-16-F-D	Verificare che il tool permetta d'inserire le opzioni di proxy.	I
TS-38	R-16.1-F-D	Verificare che il tool permetta d'inserire l'indirizzo del server proxy.	I
TS-39	R-16.2-F-D	Verificare che il tool permetta d'inserire il numero di porta del server proxy.	I
TS-40	R-17-F-D	Verificare che il tool registri il traffico di rete.	I
TS-41	R-17.1-F-D	Verificare che il tool generi un file di report al termine delle operazioni effettuate.	I

Tabella 5.4: Test di sistema.

I test di sistema sono stati effettuati insieme al tutor aziendale, il quale si è ritenuto soddisfatto dei risultati ottenuti.

Capitolo 6

Conclusioni

In questa sezione, verrà presentato il consuntivo finale dello stage/progetto, lo stato di raggiungimento degli obiettivi, le conoscenze acquisite e la valutazione personale dello stagista.

6.1 Consuntivo finale

Il lavoro è stato svolto nei tempi previsti. Il numero di ore pianificato per ognuna delle fasi di sviluppo è stato sufficiente. Nella seguente tabella sono riportate le attività svolte con le ore previste e, tra parentesi, quelle effettive.

Attività	**Descrizione**	**Previste** (**Effettive**)
Formazione	Formazione sulle tecnologie.	40 (35)
Pianificazione	Pianificazione delle attività da svolgere.	40 (38)
Analisi dei requisiti	Individuazione dei casi d'uso, dei requisiti e creazione delle tabelle per il tracciamento dei requisiti.	40 (37)

Progettazione	Progettazione del tool APAT includendo: • progettazione della logica per la creazione dei comandi; • progettazione della logica dell'esecuzione dei comandi generati; • progettazione della logica per effettuare l'analisi del codice sorgente; • progettazione dell'interfaccia grafica.	60 (50)
Codifica	Codifica del tool APAT includendo: • codifica della logica per la creazione dei comandi; • codifica della logica dell'esecuzione dei comandi generati; • codifica della logica per effettuare l'analisi del codice sorgente; • codifica dell'interfaccia grafica.	100 (118)
Test	Analisi statica, stesura dei test automatici, test manuali del tool.	20 (22)
Collaudo	User Acceptance Test, verifica soddisfacimento requisiti.	20 (20)

Tabella 6.1: Attività svolte.

La pianificazione è stata svolta in maniera sufficientemente precisa. In questo modo, le attività sono state svolte nei tempi previsti della durata dello stage. Nonostante si fossero presentate alcune difficoltà il lavoro non ha subito gravi ritardi grazie all'aiuto del tutor aziendale.

6.2 Raggiungimento degli obiettivi

Gli obiettivi dello stage erano molteplici, ognuno dei quali aveva una priorità. Nella seguente tabella vengono presentati gli obiettivi, le priorità e gli stati di completamento.

Obiettivo	Priorità	Stato
Decompilazione sorgenti.	Obbligatorio	Soddisfatto
Repackaging dell'applicativo.	Obbligatorio	Soddisfatto
Analisi dei file sorgenti ottenuti dalla decompilazione.	Obbligatorio	Soddisfatto
Firma dell'APK ottenuto dal repackaging.	Obbligatorio	Soddisfatto
Installazione dell'APK ricompilato.	Desiderabile	Soddisfatto
Dump dei dati dall'AVD.	Obbligatorio	Soddisfatto
Generazione di un report dell'analisi.	Obbligatorio	Soddisfatto
Creazione di un file pcap che contiene i dettagli delle attività di rete.	Desiderabile	Soddisfatto
Avvio dell'AVD con opzioni di proxy.	Facoltativo	Soddisfatto

Gli obiettivi dello stage fissati nel piano di lavoro sono tutti soddisfatti.
Di seguito vengono presentati i requisiti individuati nella sezione dei requisiti con lo stato di completamento per ognuno di essi.

Requisito	Descrizione	Stato di completamento
R-1-F-O	Il tool deve permettere di selezionare un file APK.	Completato
R-2-F-O	Il tool deve permettere di avviare la decompilazione dell'APK selezionato.	Completato
R-3-F-O	Il tool deve permettere di visualizzare il messaggio di errore quando la decompilazione non è terminato con successo.	Completato
R-4-F-O	Il tool deve permettere d'installare l'APK decompilato e modificato su un AVD.	Completato

R-5-F-O	Il tool deve permettere di visualizzare il messaggio quando non sono stati rilevati nessun AVD.	Completato
R-6-F-O	Il tool deve permettere di visualizzare il messaggio di errore se l'AVD selezionato non si è avviato correttamente.	Completato
R-7-F-O	Il tool deve permettere di fare il dump dello storage interna dell'applicazione.	Completato
R-8-F-D	Il tool deve permettere di visualizzare i dex ottenuti dalla decompilazione.	Completato
R-9-F-O	Il tool deve permettere di visualizzare il messaggio d'errore se la decodifica non è andato a buon fine.	Completato
R-10-F-F	Il tool deve permettere di firmare l'APK ricompilato.	Completato
R-11-F-F	Il tool deve permettere di mostrare dei messaggi quando le credenziali inseriti non sono corretti.	Completato
R-12-F-O	Il tool deve permettere di mostrare dei messaggi quando è stato selezionato un file non valido.	Completato
R-13-F-O	Il tool deve permettere di decodificare i file .dex in .java.	Completato
R-14-F-O	Il tool deve permettere di permettere di effettuare dell'analisi del codice java.	Completato
R-15-F-D	Il tool deve permettere di avviare un'AVD presente nel sistema.	Completato
R-16-F-D	Il tool deve permettere d'inserire le informazioni per il proxy.	Completato
R-17-F-D	Il tool deve permettere di registrare il traffico di rete.	Completato
R-1-V-D	Il tool può essere un tool da righe di comando.	Non Completatoc
R-2-V-D	Il tool può essere un tool dotato di GUI.	Completato
R-3-V-D	Il tool può essere sviluppato in JAVA.	Completato

R-4-V-F	Il tool può essere sviluppato utilizzando i linguaggi funzionali.	Completato
R-1-Q-O	Il codice sorgente deve essere versionato col sistema di versionamento dell' azienda ospitante.	Completato
R-2-Q-O	Il codice sorgente deve essere sotto licenza GPLv3.	Completato
R-3-Q-O	Il code coverage dei test unitari deve superare 80%.	Completato
R-4-Q-O	Il code coverage dei test unitari deve essere 90%.	Completato

Tabella 6.3: Stato completamento requisiti.

I requisiti della precedente tabella, individuati durante l'analisi dei requisiti, sono stati soddisfatti tutti.
L'azienda si è mostrata soddisfatta delle funzionalità offerte dal tool e anche delle prestazioni in termini di tempo impiegato per effettuare le operazioni. Di seguito viene fatta una comparativa dei tempi tra diversi file APK. La prima è un'applicazione per home-banking prodotta dall'azienda Imola Informatica S.p.A per un suo cliente. La seconda è l'applicazione Stalker realizzata dallo stagista e dal suo gruppo durante il progetto d'Ingegneria del Software, la terza e la quarta sono due applicazioni Android base ma sviluppate con linguaggi di programmazione differenti.

	XYZ Bank	**Stalker**	**App base 1**	**App base 2**
Decompile	25.06	47.06	16.57	17.29
Recompile	60.46	97.37	30.65	39.39
Decode	24.05	46.42	15.33	23.59
Analyze	7.20	7.06	3.55	4.56
Clean up	11.93	30.76	5.93	8.52
Totale	128.70	228.67	78.03	91.35
Linguaggio	Kotlin	Java	Kotlin	Java

Tabella 6.4: Tempi di esecuzione del tool in secondi.

Come si nota dalla Tabella 6.4 esiste una differenza sostanziale nella durata per le applicazioni sviluppate in Java e quelle in Kotlin dovuta al tool esterno. Più nello specifico, per le operazioni di decompile, recompile e decode la differenza è più notevole, mentre per l'attività di analisi la differenza è meno evidente. I dati sono stati misurati manualmente e siccome le prime due applicazioni hanno sostanziali differenze, per vari motivi, tra cui la differenza del linguaggio di programmazione utilizzato, la dimensione dell'applicazione in termine di Source Line Of Code (SLOC) e di file, non possono essere presi per la comparazione. La terza e la quarta applicazione sono due versioni dello stesso progetto messi a disposizione da Android Studio con l'unica differenza del linguaggio di programmazione utilizzato, e il tempo impiegato per effettuare le varie operazioni sono significativamente differenti, con la differenza di circa *15.58%* che è dovuta ai tool esterni che il APAT utilizza, per cui non è stato possibile migliorare la prestazione da questo punto di vista.

6.3 Prodotti realizzati

Lo stage ha portato alla realizzazione del tool APAT che permette di analizzare il codice sorgente di un'applicazione e individuare eventuali problemi di sicurezza. Il tool, scritto in Java, è composto da 3216 righe di codice, suddivise in 52 file, e dipende da due file di configurazione in formato JSON.

Per la progettazione del tool sono stati realizzati sei diagrammi delle classi e dei package costituendo di conseguenza il documento *Manuale dello sviluppatore*. Per maggiori dettagli consultare la sezione §4.4.

Per l'analisi dei requisiti sono stati realizzati sette diagrammi dei casi d'uso componendo così il documento *Analisi dei requisiti*. Per maggiori dettagli consultare la sezione §3.

Lo strumento APAT è composto da quattro viste, quella principale in cui sono presenti le principali funzionalità disponibili e altre tre viste che richiedono l'inserimento dei dati.

La seguente lista riassume quanto detto:

- **Linee di codice:** 3216;
- **Numero di classi:** 52;

- **Numero di file di configurazione:** 2;
- **Diagrammi dei casi d'uso:** 7;
- **Diagrammi delle classi:** 6;
- **Numero di viste:** 4;
- **Documenti realizzati:** 2.

6.4 Conoscenze acquisite

Durante lo svolgimento dello stage sono state acquisite delle nuove conoscenze, come:

- la tecnica del reverse engineering applicata alle applicazioni Android;
- l'analisi del codice sorgente, dei dati presenti nell'area di storage e nei database SQLite possono rivelare dati sensibili e dati personali;
- applicazione di alcuni dei design pattern visti a lezione;
- libreria Swing di Java;
- alcuni aspetti di Maven, tool di build automation;
- utilizzo dell'AVD per monitorare le attività di rete dell'applicazione.

6.5 Valutazione personale

Ritengo che concludere un percorso formativo come quello del corso di laurea in Informatica con uno stage aziendale sia molto interessante e utile, poiché è un'occasione importante per mettere in pratica le conoscenze teoriche apprese durante i corsi d'insegnamento, adattando il proprio metodo di lavoro alla realtà aziendale. Inoltre, quest'esperienza mi ha permesso di entrare in contatto con gli esperti del settore informatico, che mi hanno fatto conoscere molti aspetti del mondo lavorativo che prima di allora ignoravo completamente. Infine, penso che questo stage sia stato costruttivo sia nella formazione professionale che personale e ho apprezzato molto l'ospitalità e la disponibilità dell'azienda Imola Informatica S.p.A. e del tutor aziendale Alessandro Proscia che mi ha fornito tutti gli aiuti necessari per portare a termine il progetto.

Glossario

Android Studio è l'IDE ufficiale sviluppato e mantenuto da Google Inc. È basato sul software di JetBrains Intellij IDEA ed è progettato specificamente per lo sviluppo di applicazioni Android. Nasce per sostituire gli Android Development Tools (ADT) di Eclipse. 76, 79

APAT è un tool che, dato un file APK, può effettuare una serie di operazioni di decompilazioni, decodifica, ricompilazione e l'analisi del codice sorgente generando al termine un file di report. 81

APK indica un file Android Package. Questo formato di file, una variante del formato .JAR, è utilizzato per la distribuzione e l'installazione di componenti in dotazione sulla piattaforma per dispositivi mobili Android. 81

AVD è una macchina virtuale fornita da Google che permette di avere un'istanza di qualsiasi versione del sistema operativo Android in esecuzione nei computer per poter testare le applicazioni android senza dover utilizzare uno smartphone. 81

Build automation è il processo di automatizzazione della creazione del build software e i processi inclusi sono: compilazione del codice sorgente in codice binario, packaging del codice binario ed esecuzione dei test automatici. 40, 43, 58, 79

Code coverage è la metrica utilizzata per descrivere il grado con il quale il codice sorgente è stato eseguito dai test. Un programma con un'alta percentuale di code coverage è meno probabile che contenga degli errori. 62, 79

Complessità ciclomatica è una metrica utilizzata per misurare la complessità di un programma. Misura direttamente il numero di cammini linearmente indipendenti attraverso il grafo di controllo di flusso. La complessità ciclomatica è calcolata utilizzando il grafo di controllo di flusso del programma: i nodi

del grafo corrispondono a gruppi indivisibili d'istruzioni, mentre gli archi connettono due nodi se il secondo gruppo d'istruzioni può essere eseguito immediatamente dopo il primo gruppo. La complessità ciclomatica può, inoltre, essere applicata a singole funzioni, moduli, metodi o classi di un programma[4] . 57, 79

DAST è una tecnologia che è capace di trovare vulnerabilità eseguendo un'applicazione. Questo metodo è altamente scalabile, facilmente e velocemente integrabile. Il punto debole del DAST è che ha bisogno della configurazione esperta e potrebbe produrre sia falsi positivi che negativi. 81

DEX sono file eseguibili per le Dalvik virtual machine, e questi file DEX vengono utilizzati per inizializzare ed eseguire applicazioni sviluppate per il sistema operativo mobile Android. 81

IDE è un software che, in fase di programmazione, aiuta i programmatori nello sviluppo del codice sorgente di un programma. Spesso l'IDE aiuta lo sviluppatore segnalando errori di sintassi del codice direttamente nella fase di scrittura, oltre a tutta una serie di strumenti e funzionalità di supporto alla fase di sviluppo e debugging. 81

Inspection è una pratica dell'analisi statica che viene usata per rivelare la presenza di difetti. Questa ricerca viene fatta in modo mirato solitamente in seguito al presentarsi di un errore. Per poter effettuare l'inspection bisogna aver prima stilato una lista di controllo che contiene i punti del codice con più probabilità di contenere un errore. Inoltre, l'aggiornamento di quest'ultima è estremamente importante. 57, 79

ITS è un computer software che gestisce e contiene la lista delle issue (problemi), gli ITS sono comunemente usato dagli servizi clienti per la gestione delle problematiche dei clienti o dei dipendenti. Un ITS è simile a un bug-tracker. È uno strumento che facilita la gestione del processo di sviluppo e di gestione dei cambiamenti attraverso la gestione di attività (work item) diverse (analisi dei requisiti, sviluppo, test, bug ecc.). 81

KISS il principio utilizzato nel mondo dell'informatica per indicare che la maggior parte dei sistemi semplici hanno una prestazione maggiore rispetto ai sistemi complicati. La semplicità deve essere un obiettivo da perseguire nella

progettazione e la complessità aggiuntiva non necessaria deve essere evitata. 81

pcap I file con estensione .pcap contengono dati acquisiti dai programmi di tracciamento del pacchetto di trasmissione dati. I dati sono in forma grezza, il che significa esattamente la stessa forma in cui è stato catturato. I file sono spesso definiti come file di traccia o file ossei. Salvare i pacchetti catturati usando il formato pcap può essere fatto da molti tipi di app, chiamati sniffer. Questi programmi quindi analizzano i dati acquisiti come richiesto dall'utente, applicando la filtrazione e l'elaborazione appropriate. v, 8, 73, 80

Repackaging In ingegneria del software, il termine repackaging indica il processo di creazione del pacchetto d'installazione a seguito di un processo di trasformazione contraria. 3, 80

Reverse engineering è una tecnica di analisi delle funzioni, degli impieghi, della collocazione, dell'aspetto progettuale, geometrico e materiale di un manufatto o di un oggetto che è stato rivenuto. Lo scopo è quello di produrre un altro oggetto che abbia un funzionamento analogo o migliore, o più adatto al contesto in cui ci si trova. 3, 80

SAST Static Application Security Testing[15] è una tecnologia che è frequentemente utilizzata come un tool di analisi del codice sorgente. Il metodo analizza il codice sorgente dal punto di vista della vulnerabilità anche se potrebbe produrre alcuni falsi positivi ma per la maggior parte delle implementazioni richiede l'accesso al codice sorgente, complicate configurazioni e alta potenza di calcolo. 81

SCA è una parte dell'industria del software relativamente nuova. Questi strumenti sono spesso costruiti assemblando componenti di terze parti o open-source, integrate con codice con il codice originale[13]. 81

SLOC è una metrica software che misura le dimensioni di un software basandosi sul numero di linee di codice sorgente. Questo metodo di misura viene utilizzato per stabilire la complessità di un software e per stimare le risorse necessarie per la produzione e il mantenimento del software. Se il software è di grandi dimensioni, possono essere utilizzate anche le unità di misura KLOC (1 000 LOC) e MLOC (1 000 000 LOC). 81

swing è una libreria di Java che permette di creare interfaccia grafica. 41, 80

UML in ingegneria del software *UML, Unified Modeling Language* (dall'inglese linguaggio di modellazione unificato) è un linguaggio di modellazione e specifica basato sul paradigma object-oriented. L'*UML* svolge un'importantissima funzione di "lingua franca" nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa tale linguaggio per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico. 81

Walkthrough è una pratica dell'analisi statica che viene usata per rivelare la presenza di difetti. Richiede la rilettura ad ampio spettro del codice con attenzione, senza discriminare le parti meno significative. 57, 80

XML (acronimo di eXtensible Markup Language) è un metalinguaggio per la definizione dei linguaggi di markup, ovvero un linguaggio marcatore basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo. Il nome indica che si tratta di un linguaggio marcatore estendibile, in quanto permettere di creare tag personalizzati. 41, 80

Acronimi

APAT Android Package Analysis Tool. 58

APK Android Package. v

AVD Android Virtual Device. v, 43

DAST Dynamic Application Security Testing. 4

DEX Dalvik Executable. 4

IDE Integrated Development Environment. 43

ITS Issue Tracking System. 44

KISS Keep It Simple Stupid. 4

SAST Static Application Security Testing. 4

SCA Software Component Analysis. 4

SLOC Source Line Of Code. 76

UML Unified Modelling Language. 11, 44

Bibliografia

Riferimenti bibliografici

[2] J. Bloch. *Effective Java*. Pearson, 2009 (cit. a p. 41).

[5] *Design Patterns*. Pearson, 2002. Cap. 3.3, p. 107 (cit. a p. 53).

[6] *Design Patterns*. Pearson, 2002. Cap. 4.4, p. 175 (cit. a p. 52).

[7] *Design Patterns*. Pearson, 2002. Cap. 5.7, p. 295 (cit. a p. 51).

[14] Ian Sommerville. *Ingegneria del Software*. Pearson. Cap. 2.1.2, p. 38 (cit. a p. 45).

Siti web consultati

[1] *Analisi Statica*. URL: https://en.wikipedia.org/wiki/Static_program_analysis (cit. a p. 57).

[3] *Check Style plugin*. URL: https://maven.apache.org/plugins/maven-checkstyle-plugin/ (cit. a p. 57).

[4] *Complessità ciclomatica*. URL: https://it.wikipedia.org/wiki/Complessit%C3%A0_ciclomatica (cit. a p. 80).

[8] Code Dx. INC. *Secure coding best practices*. URL: https://codedx.com/blog/10-secure-coding-best-practices/a (cit. a p. 3).

[9] Imola Informatica. *Secure Software Developmente Lifecycle*. URL: https://blog.imolainformatica.it/wp-content/uploads/2018/12/SSDLC-tech_report.pdf (cit. a p. 3).

[10] *JUnit4*. URL: https://junit.org/junit4/ (cit. a p. 58).

[11] *Owasp*. URL: https://owasp.org/www-project-mobile-top-10/ (cit. a p. 1).

[13] *Software Component Analysis*. URL: https://owasp.org/www-community/Component_Analysis (cit. a p. 81).

[15] *Static Application Security Testing*. URL: https://en.wikipedia.org/wiki/Application_security (cit. a p. 81).

Articoli consultati

[12] Spyridon Samonas e David Coss. «THE CIA STRIKES BACK: REDEFINING CONFIDENTIALITY, INTEGRITY AND AVAILABILITY IN SECURITY.» In: *Journal of Information System Security* 10.3 (2014) (cit. a p. 3).